CS421/621- Advance Web Application Development

Week 6

-MongoDB, Flask-

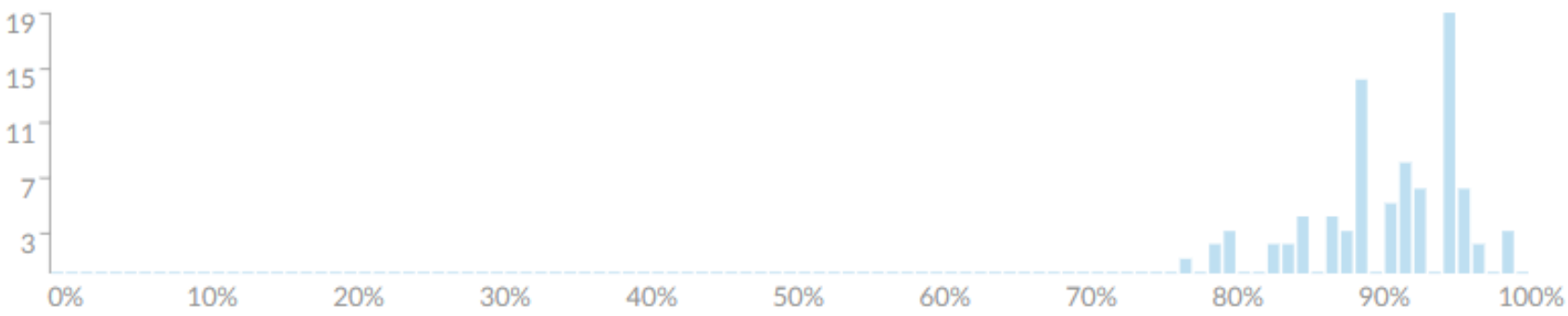
Professor : Mahmut Unan – UAB CS

Agenda

- MongoDB
- Mongoose
- Flask Framework
- Virtual Environments
- Creating the main page
- Multiple pages
- Dynamic routing
- Debugging
- Forms

Exam Results

Average Score	High Score	Low Score	Standard Deviation	Average Time
91%	99%	77%	3.71	30:43



HW3

Project Proposals

- Binary grade
- Sample project

MongoDB

- MongoDB is a document-oriented database used for high volume data storage.
- NoSQL database / Open-Source
- Each database contains collections which in turn contains documents. Each document can be different with a varying number of fields. The size and content of each document can be different from each other.
- The data model available within MongoDB allows you to represent hierarchical relationships, to store arrays, and other more complex structures more easily.

https://www.guru99.com/what-is-mongodb.html

Collection

- A group of MongoDB documents
- Documents of the same Collection can have different fields
  - The goal of the Collection is to group the documents which have the same or similar purpose together

Documents

- Set of key-value pairs
- Dynamic Schema

RDBMS	MongoDB
Database	Database
Table	Collection
Tuple/Row	Document
column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default key _id provided by MongoDB itself)
Database Server and Client	
mysqld/Oracle	mongod
mysql/sqlplus	mongo

https://www.tutorialspoint.com/mongodb/mongodb_overview.htm

Sample Document

```
{
   _id: ObjectId(7df78ad8902c)
   title: 'MongoDB Overview',
   description: 'MongoDB is no sql database',
   by: 'tutorials point',
   url: 'http://www.tutorialspoint.com',
   tags: ['mongodb', 'database', 'NoSQL'],
   likes: 100,
   comments: [
      {
         user:'user1',
         message: 'My first comment',
         dateCreated: new Date(2011,1,20,2,15),
         like: 0
      },
      {
         user:'user2',
         message: 'My second comments',
         dateCreated: new Date(2011,1,25,7,45),
         like: 5
      }
   ]
}
```

https://www.tutorialspoint.com/mongodb/mongodb_overview.htm

Why Use MongoDB?

- Document-oriented
- Ad hoc queries
- Indexing
- Replication
- Load balancing
- Cloud System Support
- No complex joins
- MongoDB is easy to scale
- …..

https://www.guru99.com/what-is-mongodb.html

Where to use MongoDB

- Big Data Management
- Data Hub
- Content Management and Blog Systems
- E-commerce type of product-based applications
- High speed / Real-time
- Dealing with location wise Geospatial data
- Mobile and Social Networking apps

Disadvantages / Not so good for…

- Where the data model is designed up front.
- Tightly coupled systems
- It can’t handle complex transactions
- It is not as strong ACID as compare to the most RDBMS systems

Who are using it ?

- Castlight Health
- IBM
- Citrix
- Twitter
- T-Mobile
- Zendesk
- Sony
- BrightRoll
- Foursquare
- HTC
- InVision
- Intercom etc.
- …

https://www.dotnettricks.com/learn/mongodb/what-is-mongodb-and-why-to-use-it#:~:text=MongoDB%20is%20the%20most%20commonly,and%20model%2Dbased%20data%20structure.

Platform And Language Support

- C
- C++
- C# and .NET
- Java
- **Node.js**
- Perl
- PHP, PHP Libraries, Frameworks, and Tools.
- Python
- Ruby
- Mongoid (Ruby ODM)

https://www.dotnettricks.com/learn/mongodb/what-is-mongodb-and-why-to-use-it#:~:text=MongoDB%20is%20the%20most%20commonly,and%20model%2Dbased%20data%20structure.

Mongoose

- An object modelling package for Node
- Easily installed by using npm
  - Npm install mongoose –save
  - ***We already have it in Chapter7
- Once we install it, we can grab it in any project;
  - Var mongoose = require(‘mongoose’)
- MongoDB database connection is also pretty easy
  - mongoose.connect('mongodb://localhost/myappdatabase');

Modeling Data with Mongoose

- Mongoose is a Node.js module that serves two primary purposes.
- It works as a client for MongoDB in the same way that the node-redis module works as a client for Redis.
- Mongoose also serves as a data modeling tool, which allows us to represent documents as objects in our programs.

Models

- A data model is simply an object representation of a collection of documents in a data store.
- In addition to specifying the fields that are in every document of a collection, it adds MongoDB database operations like save and find to the associated objects.

- In Mongoose, a data model consists of a schema, which describes the structure of all of the objects that are of that type.
- For instance, suppose we wanted to create a data model for a collection of playing cards. We'd start by specifying the schema for a card—namely, explicitly declaring that every card has a rank and a suit. In our JavaScript file, this looks something like the following:

```
var CardSchema = mongoose.Schema({
"rank" : String,
"suit" : String
});
```

The allowed **`SchemaTypes`**

- String
- Number
- Boolean
- Array
- Date
- Buffer
- Mixed
- ObjectId

- Once we create the schema, building a model is very easy. By convention, we use a capital letter for data model objects:

```
var Card = mongoose.model("Card", CardSchema);
```

- Schemas can get more complicated. For example, we might build a schema for blog posts that contain dates and comments. In this example, the comments attribute represents an array of strings instead of a single one:

```
var BlogPostSchema = mongoose.Schema({
   title: String,
   body : String,
   date : Date,
   comments : [ String ]
   });
```

- Once we have a model, we can create an object of the model type very easily using JavaScript's new operator. For example, this line of code creates the ace of spades and stores it in a variable called c1:

```
var c1 = new Card({"rank":"ace",
"suit":"spades"});
```

- Great, but couldn't we have just as easily done that with this code?

```
var c2 = { "rank":"ace", "suit":"spades" };
```

Built-in functions

- The difference is that the Mongoose object allows us to interact with the database through some built-in functions!

```
// save this card to our data store
c1.save(function (err) {
if (err !== null) {
// object was not saved!
console.log(err);
} else {
console.log("the object was saved!");
}
});
```

`find` function

```
Card.find({}, function (err, cards) {
if (err !== null) {
console.log("ERROR: " + err);
// return from the function
return;
}
// if we get here, there was no error
Cards.forEach(function (card) {
// this will print all of the cards in the database
console.log (card.rank + " of " + card.suit);
});
});
```

```
Card.find({}, function (err, cards) {
if (err !== null) {
console.log("ERROR: " + err);
// return from the function
return;
}
// if we get here, there was no error
Cards.forEach(function (card) {
// this will print all of the cards in the database
console.log (card.rank + " of " + card.suit);
});
});
```

- We can also update elements by finding the appropriate one (via its _id or another query) and saving it again. For instance, suppose we wanted to change all of the cards that have the suit spades to hearts:

```
Card.find({"suit" : " hearts"}, function (err, cards) {
cards.forEach(function (card) {
// update the card to spades
card.suit = "spades";
// save the updated card
card.save(function (err) {
if (err) {
// object was not saved
console.log(err);
}
});
});
});
```

- Last but not least, we can remove elements from the database by calling the remove function on the data model:

```
Card.remove({ "rank":"ace",
"suit":"spades" }, function (err) {
if (err !== null) {
// object was not successfully
removed!
console.log(err);
}
});
```

Suggested Links

- https://scotch.io/tutorials/using-mongoosejs-in-node-js-and-mongodb-applications
- https://www.mongodb.com/what-is-mongodb

- https://www.youtube.com/watch?v=pWbMrx5rVBE

Frameworks

- ASP.NET
- Django
- Spring
- Play
- Angular
- Ruby on Rails
- Symfony
- Flask
- Lamp

Virtual Environments

- Python applications mostly use packages and modules
  - Specific version of a library
- It is not possible for one Python installation to meet the requirements of every application
  - Solution: create virtual environments

- https://docs.python.org/3/tutorial/venv.html

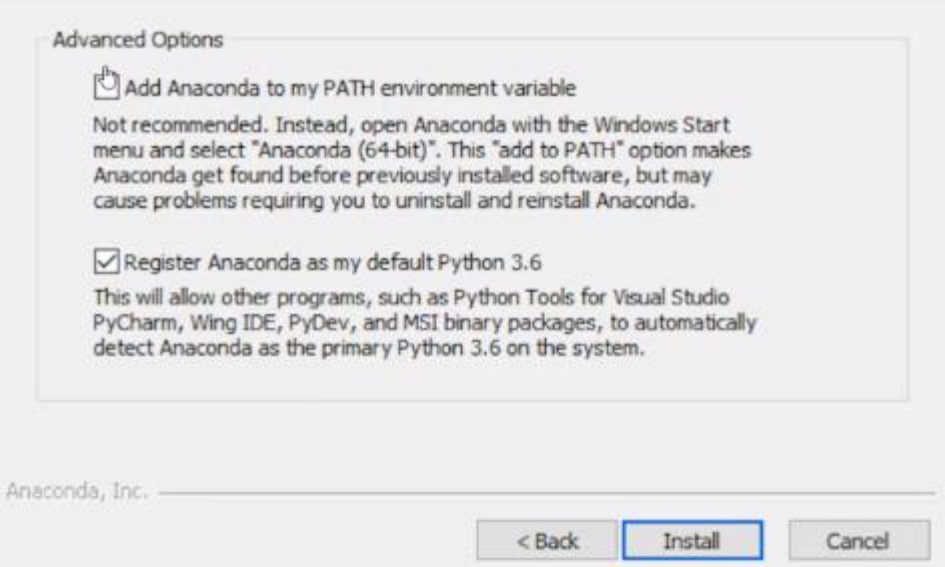
Flask

- Flask is a microframework for Python.
- While minimalist, Flask can create serious websites out of the box. It contains a development server and debugger, and includes support for Jinja2 templating, secure cookies, unit testing, and RESTful request dispatching. It has good documentation and an active community.
- Flask has become extremely popular, particularly for developers who need to provide web services on small, resource-constrained systems (e.g. running a web server on a Raspberry Pi, Drone controllers, etc.)

https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Web_frameworks

Download / Install Anaconda

- https://www.anaconda.com/distribution/#download-section
- Select add anaconda to my path environment variable (even though it is not recommended)



Python

```
C:\Users\unan\Desktop\WebApps>python
Python 3.7.3 (default, Mar 27 2019, 17:13:21) [MSC v.1915 64 bit (AMD64)] :: Anaconda, I
in32

Warning:
This Python interpreter is in a conda environment, but the environment has
not been activated.  Libraries may fail to load.  To activate this environment
please see https://conda.io/activation

Type "help", "copyright", "credits" or "license" for more information.
>>>
```

To exit the interactive prompt, you can type exit() and press Enter.

Download the requirements.txt file

- Create **week6** folder (anywhere you want)
- Go to Canvas files/code folder and download the **requirements.txt** file
- This file is a list of libraries that we will install using pip install
- Move the downloaded file in your week6 folder

```
alembic==0.9.9
blinker==1.4
chardet==3.0.4
click==6.7
Flask==1.0.2
Flask-Dance==0.14.0
Flask-DebugToolbar==0.10.1
Flask-Login==0.4.1
Flask-Migrate==2.1.1
```

https://www.udemy.com/python-and-flask-bootcamp-create-websites-using-flask/learn/lecture/9345700#overview

Conda Virtual Environment

- When you are in the **week6** folder which includes requirements.txt file
- We will create our virtual environment using conda;
  - `conda create -n firstflaskenv python=3.7`
  - Type `y` for yes
- Now, we should **activate** the environment;
  - For windows: `activate firstflaskenv`
  - For mac/linux: `source activate firstflaskenv`

*** sometimes it is: `conda activate firstflaskenv`

*****To exit : `deactivate` or `conda deactivate`

Install all the libraries

- `pip install -r requirements.txt`

- It may take some time…..

- Let’s test our environment
  - Type `python` and hit enter
  - Now you are in python environment
  - Type `import flask` and hit enter
  - If you don’t get any error, that means you installed it correctly

Exercise 1 - Hello World!

- Let’s create our first web page with flask
- Create a python file in your project folder
- Name it (I will name it main.py, feel free to give any name you want)
  - * do not name your file flask.py / it may confuse the python

Main.py file

```
from flask import Flask
app = Flask (__name__)

@app.route ('/')
def index():
    return '<h1> Hello World! </h1>'



if __name__ == '__main__':
    app.run()
```

Run hello world

- In the command window, type `python main.py` and hit enter
- Go to your browser and put the following ip address in your address bar
  - http://127.0.0.1:5000/
- You can stop your server with Control+C

Exercise 2 –Multiple Pages (routes)

- Our homepage or domain is locally represented as http://127.0.0.1:5000/

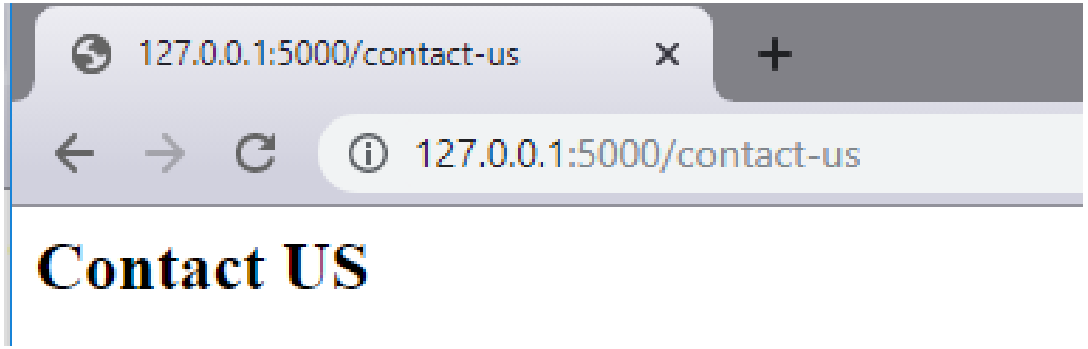
- To create multiple pages, we will use decorators;
  - @app.route (“/another-page”)

- To access this page, we will use;
  - http://127.0.0.1:5000/another-page
-

Exercise 2 –Multiple Pages / 2

```
@app.route('/contact-us')
def info():
    return "<h2> Contact US</h2>"
```

http://127.0.0.1:5000/contact-us



Dynamic Routes

- Most of the time, we want URL route extensions to be dynamic based on the situation

- For example : we may want to create a profile page for users and we want to this dynamically when the user logged in.
- Dynamic routes have 2 key aspects
  - A variable in the route **<variable>**
  - A **parameter** passed into the function

Exercise 3 – Dynamic Route

```
@app.route ('/users/<username>')
def userprofile(username):
    return "<h3> This is a profile page for {}</h1>".format(username)
```

This is a profile page for mahmut

We can modify the text

- We can use the string operators

```
@app.route ('/users/<username>')
def userprofile(username):
    return "upper case {} ".format(username.upper())
```

127.0.0.1:5000/users/mahmut

upper case MAHMUT

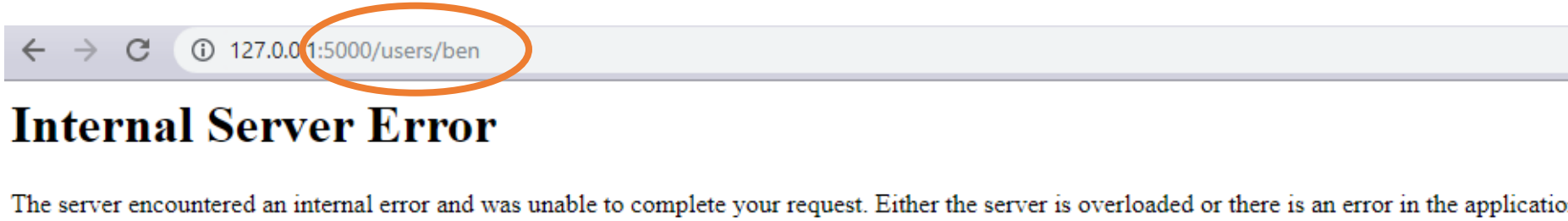
Debugging

- Let’s modify our dynamic routing code a little bit

```
@app.route ('/users/<username>')
def userprofile(username):
    return "5th character of your name is ={} ".format(username[5])
```

5th character of your name is =t

Debugging / 2



- To enable debugging, modify your app.run code;

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

Debugging / 3

Good for developing / bad for publishing

builtins.IndexError

```
IndexError: string index out of range
```

Traceback (most recent call last)

File "C:\Users\umahm\Anaconda3\envs\firstflaskenv\lib\site-packages\flask\app.py", line *2309*, in __call__

```
return self.wsgi_app(environ, start_response)
```

File "C:\Users\umahm\Anaconda3\envs\firstflaskenv\lib\site-packages\flask\app.py", line *2295*, in wsgi_app

```
response = self.handle_exception(e)
```

File "C:\Users\umahm\Anaconda3\envs\firstflaskenv\lib\site-packages\flask\app.py", line *1741*, in handle_exception

```
reraise(exc_type, exc_value, tb)
```

File "C:\Users\umahm\Anaconda3\envs\firstflaskenv\lib\site-packages\flask_compat.py", line *35*, in reraise

```
raise value
```

File "C:\Users\umahm\Anaconda3\envs\firstflaskenv\lib\site-packages\flask\app.py", line *2292*, in wsgi_app

```
response = self.full_dispatch_request()
```

File "C:\Users\umahm\Anaconda3\envs\firstflaskenv\lib\site-packages\flask\app.py", line *1815*, in full_dispatch_request

```
rv = self.handle_user_exception(e)
```

File "C:\Users\umahm\Anaconda3\envs\firstflaskenv\lib\site-packages\flask\app.py", line *1718*, in handle_user_exception

```
reraise(exc_type, exc_value, tb)
```

File "C:\Users\umahm\Anaconda3\envs\firstflaskenv\lib\site-packages\flask_compat.py", line *35*, in reraise

```
raise value
```

File "C:\Users\umahm\Anaconda3\envs\firstflaskenv\lib\site-packages\flask\app.py", line *1813*, in full_dispatch_request

```
rv = self.dispatch_request()
```

File "C:\Users\umahm\Anaconda3\envs\firstflaskenv\lib\site-packages\flask\app.py", line *1799*, in dispatch_request

```
return self.view_functions[rule.endpoint](**req.view_args)
```

File "C:\Users\umahm\Desktop\flask_examples\flask1\main.py", line *14*, in userprofile

```
return "5th character of your name is ={} ".format(username[5])
```

IndexError: string index out of range

The debugger caught an exception in your WSGI application. You can now look at the traceback which led to the error.

To switch between the interactive traceback and the plaintext one, you can click on the "Traceback" headline. From the text traceback you can also create a paste of it. For code execution mouse-over the frame you want to debug and click on the console icon on the right side.

Debugging / 4

- Go over the error message and find the console icon

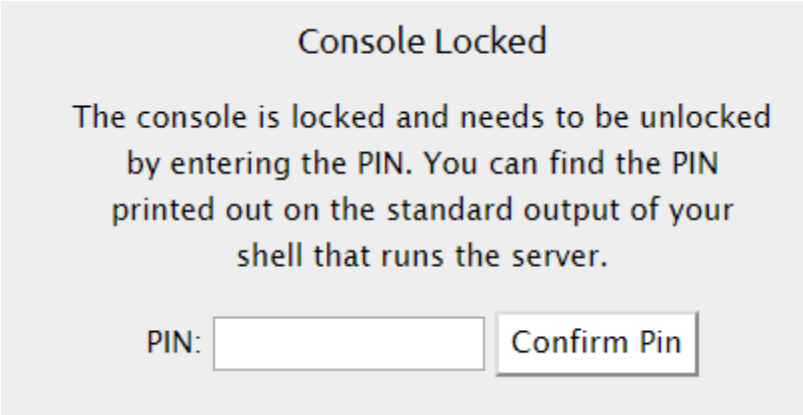
```
File "C:\Users\umahm\Desktop\flask_examples\flask1\main.py", line 14, in userprofile
  return "5th character of your name is ={} ".format(username[5])
```

- When you hit the console icon, you will get a debugger pin code in your console

```
* To enable the debugger you need to enter the security pin:
* Debugger pin code: 304-569-978
```

Debugging / 5

- Also, you will get a message box in your browser



- Put your pin code here and hit the button

```
File "C:\Users\umahm\Desktop\flask_examples\flask1\main.py", line 14, in userprofile
  return "5th character of your name is ={} ".format(username[5])
[console ready]
>>> |
```

```
>>> username
'ben'
>>>
```

Exercise 4

- Let’s update our username control code
- Check the name and apply the following;
  - If the name doesn’t end in a y, add a y to the name
    - For ex: John -> Johny or Ben -> Beny
  - If the name does end in a y, then replace it with iful instead
    - For ex: Anthony → Anthoniful or Jay → Jaiful

Template Forms

- In the HTML lectures we learned how to create HTML forms for users to supply information
- Now, we will learn how we can connect our Flask application to these forms

Exercise 5

- Let’s create **exercise5.py** file
- Create a **templates** folder
- Also create 5 different html files inside this **templates** folder;
  - **base2.html**
  - **index.html**
  - **signup.html**
  - **thank_you.html**
  - **404.html**

base2.html

- Import bootstrap into your html file (https://getbootstrap.com/docs/3.3/getting-started/)
- Insert a navigation bar
- **url_for** is going to be used to link the pages
  - Usage: Home Page

base2.html

```
<head>
  <meta charset="utf-8">
  <title> My log in system</title>
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap-theme.m
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js" integrity="sha38
</head>
<body>
 <nav class="navbar navbar-expand-lg navbar-light bg-light">
   <a class = " navbar-brand" href="{{url_for('index')}}">Home Page</a>

 </nav>
 {% block content %}

 {% endblock %}
</body>
```

index.html

```
{% extends "base2.html" %}

{% block content %}
<div class="jumbotron">
  <p> Welcome to our company page</p>
  <p> Do you want to be a member?</p>
  <a href="{{url_for('signup_form')}}"> Sign up here</a>
</div>



{% endblock %}
```

signup.html

```
{% extends "base2.html"%}
{% block content %}
<div class="jumbotron">
  <p> Welcome to sign up page</p>
  <p> Fill out the form and submit</p>
  <form  action="{{url_for('thank_you')}}"  >
  <label for="first"> First Name:</label>
  <input type="text" name="first">
  <label for="last">Last Name: </label>
  <input type="text" name="last">
  <input type="submit" value="Submit Form">
  </form>
</div>
{% endblock %}
```

thank_you.html

```
{% extends "base2.html"%}
{% block content %}

<div class="jumbotron">
  <h1> Thank you for becoming a member {{first}} {{last}}</h1>
</div>

{% endblock %}
```

404.html

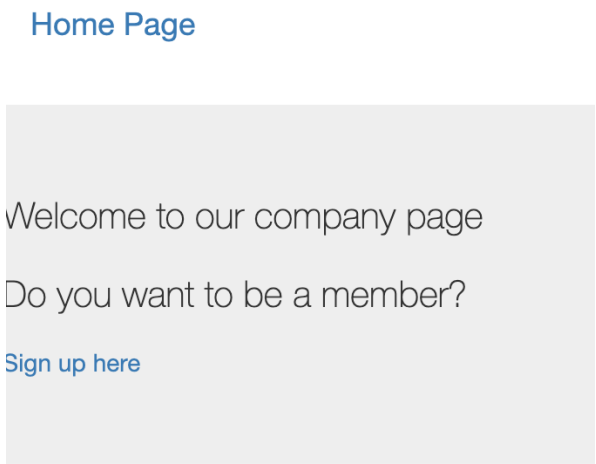
```
{% extends "base2.html"%}
{% block content %}

<div class="jumbotron">
<h3>Sorry, we couldn't find the page you were looking for.</h3>

</div>

{% endblock %}
```

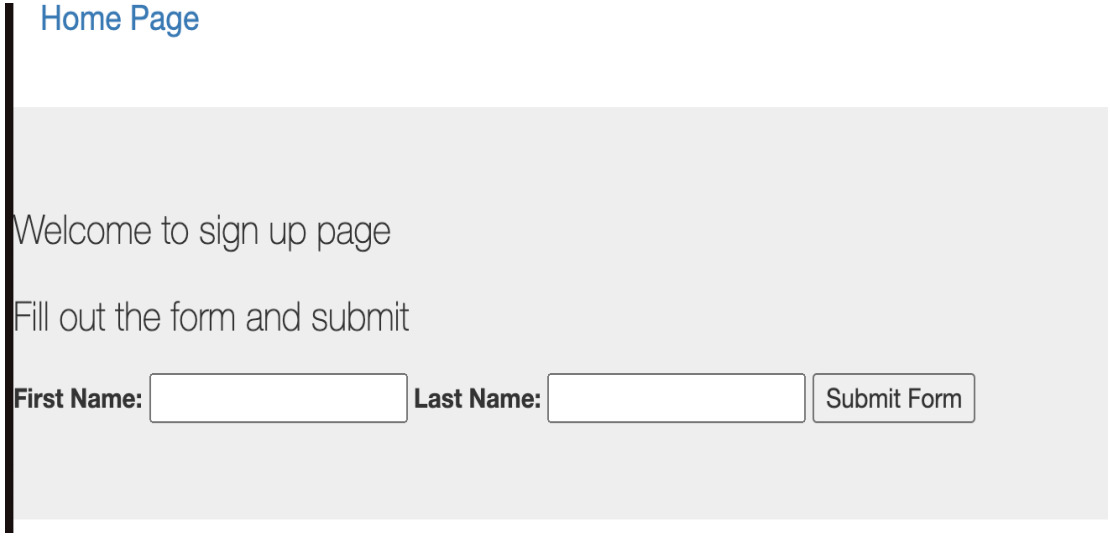
exercise5.py

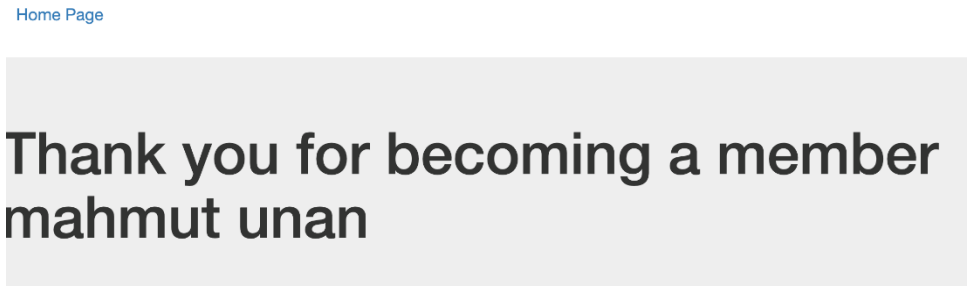
```python
from flask import Flask, render_template, request
app = Flask (__name__)

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

@app.route('/signup_form')
def signup_form ():
    return render_template ('signup.html')

@app.route('/thankyou')
def thank_you():
    first = request.args.get('first')
    last = request.args.get('last')
    return render_template('thank_you.html',first=first,last=last)

@app.errorhandler(404)
def page_not_found(e):
    return render_template('404.html'),404

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

Exercise 5

- Now you can run exercise 5







Next Week

- Flask Templates, Forms, Inheritance,Filters, and SQL databases with flask