### Milestone:Setting Up Application Environment

### Activity :Create Flask Project

Flask is a lightweight WSGI web application framework. It is designed to make getting started quick and easy. This will even make it easier by showing a step-by-step guide to creating a Python Flask web application with auto-generated desired data to get the application started.

### Objectives:

1. Utilize Python 3 and Flask to create a web application
2. Create view functions for handling requests to specific URLs (routes)
3. Utilize the Jinja templating engine to add logic when generating HTML
4. Process form data in a view function using the request proxy
5. Store and retrieve data in a session for use between requests
6. Write tests using pytest
7. Utilize static files (sylesheets, images) for enhancing the look and feel of the app
8. Create flash messages for displaying information to the user
9. Log messages to a file to understand the operations of the web app

### Microframework

Flask is often referred to as a microframework. It is designed to keep the core of the application simple and scalable.

Instead of an abstraction layer for database support, Flask supports extensions to add such capabilities to the application.

Prerequisites:

* Python3.
* To start, let's create a simple "hello world" application. First, create a new project. From the terminal or command line create a new directory:

1. mkdir myproject
2. cd myproject

Inside the project directory, create a virtual environment for the project. You can check virtualenv for more information. First, install **virtualenv**, create one, activate it, and install Flask: pip install virtualenv

#Create virtualenv

python3 -m venv venv

#Create virtualenv for windows

py -3 -m venv venv

#Activate virualenv:

. venv/bin/activate

#Activate virualenv for windows

venv\Scripts\activate

#Install Flask on the enviroment

pip install flask

The project setup is now ready to start adding functionalities. To start, let's create a new file in the same directory with the following content and name it main.py.

1

from flask import Flask

2

​

3

app = Flask(\_\_name\_\_)

4

​

5

@app.route('/')

6

def hello():

7

return "Hello World!"

8

​

We can now run the application by telling the terminal which application to run by using the following commands:

1

#Tell the terminal what application to run

2

export FLASK\_APP=main.py

3

#Tell the terminal what application to run for windows

4

set FLASK\_APP=main.py

5

#Run the application

6

flask run

The result should be like the following, telling you that the application is running on <http://127.0.0.1:5000/>. Navigating to this page should return "Hello World”

Hello, world output

After completing the first part of the article, let's move to the next part and add functionality to our project. To start, we will download [SQLALCHEMY](https://www.sqlalchemy.org/) to our project, which is an open source SQL toolkit and object-relational mapper for Python. We will download some other packages for Flask, such as wtforms and flask-wtforms to build our forms.

1

#To install Flask-SQLAlchemy

2

pip install pip install flask-sqlalchemy

3

#To install WTForms

4

pip install WTForms

5

#To install Flask-wtforms

6

pip install Flask-WTF

It's recommended to add the installed packages to a requirements format file. To do so, run the following command:

1

pip freeze requirements.txt

This command will generate a file called requirements.txt. It's used to store the packages, you can use the following command to install the packages from the requirements file. (this step may be skipped if you are running the code on the same machine used for the previous steps.)

1

pip install -r requirements.txt

After having the project ready for the new functionalities, let's add some magic to it. To start, let me quickly introduce the [Clowiz CodeGen](https://www.clowiz.com/code-generator/) feature from [The Cloud Wizard](https://www.clowiz.com/).

To start adding magic to the project:

1. Go to <https://www.clowiz.com/code-generator/> to start using the CodeGen feature.
2. From the technologies section press on the Python Flask logo.
3. In the metadata section, fill in the Form Name (e.g. Employee). (It can be anything you like.)
4. In the Field Name section, change the first value to e.g. Name.
5. The second one can be renamed to Email. (Change the Data Type to Email.)
6. The third one can be changed to Salary and the Data Type can be changed to Double.
7. Copy the code from the generated code section into our main.py file below the hello() method.

Now, let's define the database model, back to Clowiz AppGen, choose SQLAlchemy Model from the Generators section and copy the generated code and paste it in the  main.py  file under the form that we created.

After saving the main.py file, go to the terminal and execute the following commands to create the database:

1

python3

2

from main import db

3

db.create\_all()

The next step is to create the routes for the application back to Clowiz AppGen. Then, choose from the generators view, copy the generated code, paste it below the database model in the main.py file, and save it.

Your main.py file should look like the following:

1

from flask import Flask, escape, request

2

​

3

app = Flask(\_\_name\_\_)

4

​

5

app.config['SECRET\_KEY'] = 'any secret key'

6

​

7

@app.route('/')

8

def hello():

9

return "Hello World!"

10

​

11

​

12

from flask\_wtf import FlaskForm

13

from wtforms import SubmitField, HiddenField, StringField, IntegerField, DecimalField

14

from wtforms.validators import Email

15

​

16

class EmployeeForm(FlaskForm):

17

id = HiddenField()

18

name = StringField('Name')

19

email = StringField('Email', validators=[Email()])

20

salary = DecimalField('Salary')

21

submit = SubmitField("Save")

22

​

23

​

24

from flask\_sqlalchemy import SQLAlchemy

25

app.config["SQLALCHEMY\_DATABASE\_URI"] = "sqlite:////tmp/employee.db"

26

db = SQLAlchemy(app)

27

​

28

class Employee(db.Model):

29

id = db.Column(db.Integer, primary\_key=True)

30

name = db.Column(db.String)

31

email = db.Column(db.String)

32

salary = db.Column(db.Numeric)

33

references = db.Column(db.String)

34

​

35

def \_\_repr\_\_(self):

36

return "(%r, %r, %r)" %(self.name,self.email,self.salary)

37

​

38

​

39

from flask import render\_template, request, flash, redirect, url\_for

40

​

41

@app.route("/employee", methods=["GET", "POST"])

42

def createEmployee():

43

form = EmployeeForm(request.form)

44

employees = Employee.query.all()

45

if form.validate\_on\_submit():

46

employee = Employee(name=form.name.data, email=form.email.data, salary=form.salary.data)

47

db.session.add(employee)

48

db.session.commit()

49

db.session.refresh(employee)

50

db.session.commit()

51

flash("Added Employee Successfully")

52

return redirect(url\_for("createEmployee"))

53

return render\_template("employee.html", title="Employee", form=form, employees=employees)

54

​

55

@app.route("/updateEmployee/<int:employee\_id>", methods=["GET", "POST"])

56

def updateEmployee(employee\_id):

57

employee = Employee.query.get(employee\_id)

58

form = EmployeeForm(request.form, obj=employee)

59

if form.validate\_on\_submit():

60

form.populate\_obj(employee)

61

db.session.commit()

62

flash("Updated Employee Successfully")

63

return redirect(url\_for("createEmployee"))

64

return render\_template("employee.html", title="Employee", form=form, employees=Employee.query.all())

65

​

66

@app.route("/deleteEmployee/<int:employee\_id>", methods=["GET", "POST"])

67

def deleteEmployee(employee\_id):

68

employee = Employee.query.get(employee\_id)

69

db.session.delete(employee)

70

db.session.commit()

71

return redirect(url\_for("createEmployee"))

72

Since we have a created a view, we should present the form to the user. Let's see how to generate the user interface.

Create a new directory in the project directory called templates by using the following commands(this folder must be called templates):

1. In the folder, create two files, one called layout.html (note, this must be called layout.html because it will be imported in the employee.html file), and one called employee.html.

1

mkdir templates

2

touch templates/layout.html

3

touch templates/employee.html

3. Go back to Clowiz AppGen, choose Layout from the Generators section, copy the generated code, and paste it in the layout.html file.

4. Change the Generators section to Template and copy the generated code to the employee.html file.

5. Save the files and run flask run in the terminal. Then, navigate to .

You should see the following and be able to add a new record.

Employee Form:

|  |  |
| --- | --- |
| Name |  |
| Salary |  |
| Email |  |

|  |
| --- |
| Save |

|  |  |  |  |
| --- | --- | --- | --- |
| Id | Name | Salary | Action |

Adding a new employee

Now fill in the form and save the values you will get the following:

Employee Form:

|  |  |
| --- | --- |
| Name | John Adams |
| Salary | 10,000.00 |
| Email | Johnadams13@gmail.com |

|  |
| --- |
| Save |

|  |  |  |  |
| --- | --- | --- | --- |
| Id | Name | Salary | Action |
| I | John adams | 10,000.00 | Update delete |

Employee successfully added

You can even update the record by pressing on the update button, changing the value you wish to update, and saving it.

Employee Form:

|  |  |
| --- | --- |
| Name | John Adams |
| Salary | 10,000.00 |
| Email | Johnadams13@gmail.com |

|  |
| --- |
| Save |

|  |  |  |  |
| --- | --- | --- | --- |
| Id | Name | Salary | Action |
| I | John adams | 10,000.00 | Update delete |

New employee displayed