Flask Guide

Meher Krishna Patel

Created on : Octorber, 2017

Last updated : May, 2020

Table of contents

Chapter 1

Flask

1.1 Introduction

In this chapter, we will learn the basics of Flask framework. More specifically, we will see the URLs, HTML templates, Forms and Sessions in Flask. Below are the required libraries for this tutorial,

```
Flask==0.12.2
Flask-WTF==0.11
Jinja2==2.7.3
MarkupSafe==0.23
WTForms==2.0.2
Werkzeug==0.10.1
itsdangerous==0.24
```

1.2 Hello World

- Below is the code for which writes the "Hello World!" on the browser. Read comments for more details,

```
# app.py

from flask import Flask

app = Flask(__name__) # application 'app' is object of class 'Flask'

# decorator 'app.route' binds the 'url' with 'function',
# i.e. url of 'home page (/)' will call function 'index' and
# 'return' value will be send back to the browser.
@app.route('/')  # root : main page
def index():
    return '<p>Hello World!</p>'

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

- Run the code as below,

```
$ python app.py
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
[...]
```

- Now, open browser and go to link 'http://127.0.0.1:5000/' and **Hello World!** will be displayed on the browser.

1.3 Render HTML

- In the below code, the 'index.html' file is called by the function 'index' (Line 13).
- Also, we can define the port number for localhost as shown in Line 18.

Note: By default, 'render_template' looks inside the folder 'template'

```
1  # app.py
2
3  from flask import Flask, render_template
4
5  app = Flask(__name__) # application 'app' is object of class 'Flask'
6
7  # decorator 'app.route' binds the 'url' with 'function',
8  # i.e. url of 'home page (/)' will call function 'index' and
9  # 'return' value will be send back to the browser.
10 @app.route('/')  # root : main page
11 def index():
12     # by default, 'render_template' looks inside the folder 'template'
13     return render_template('index.html')
14
15 if __name__ == '__main__':
16     # '0.0.0.0' = 127.0.0.1 i.e. localhost
17     # port = 5000 : we can modify it for localhost
18     app.run(host='0.0.0.0', port=5000, debug=True) # local webserver : app.run()
```

- Below is the content of 'index.html' file,

```
<!-- index.html -->

<!DOCTYPE html>
<html>
<head>
    <title>Home</title>
</head>
<body>
    <h1>Hello World!</h1>
</body>
</html>
```

- Fig. 1.1 is the output of above codes,

Fig. 1.1: Hello World!

1.4 Variables in HTML

We can use the variables in the HTML with the help of Jinja2-templates; and the pass the value of variable from the 'python code',

- Below is the HTML code, where the variable 'sub' is used at Line 10. Note that variables are written betwee {{ }}.

```
<!-- subject.html -->

<!DOCTYPE html>
<html>
<head>
    <title>Subject</title>
</head>
<body>
    <h1> Subject selection </h1>
    <p> Recommended subject : <b> {{sub}} </b></p>
</body>
</html>
```

- Below is the Python code where the value for the variable 'sub' is passed through Line 19. Further, the value of the 'id' is provided by the 'url' e.g. if url is 'subjects/1', then 'id' will be set to '1' by Line 17.

```
# app.py

from flask import Flask, render_template

app = Flask(__name__) # application 'app' is object of class 'Flask'

subjects = ["English", "Dance", "Maths", "Geography"]

# decorator 'app.route' binds the 'url' with 'function',
# i.e. url of 'home page (/)' will call function 'index' and
# 'return' value will be send back to the browser.
@app.route('/')  # root : main page
def index():
    # by default, 'render_template' looks inside the folder 'template'
    return render_template('index.html')

@app.route('/subject/<int:id>')
def subject(id):
    return render_template('subject.html', sub="subjects[id]")


if __name__ == '__main__':
    # '0.0.0.0' = 127.0.0.1 i.e. localhost
    # port = 5000 : we can modify it for localhost
    app.run(host='0.0.0.0', port=5000, debug=True) # local webserver : app.run()
```

- Below is the output for link 'http://127.0.0.1:5000/subject/1',

```
Subject selection

Recommended subject : Dance
```

1.5 Navigation using 'url_for'

It is better to link the functions to URL using the method 'url_for' (instead of hard-coded link). In this way, if we decide to change the URL-name, then we need not to change it in the 'html' page; the 'url_for' will automatically render the new URL.

- Lets add one more method 'interest' as shown in Lines 8 and 24-26.

```
# app.py

from flask import Flask, render_template
```

(continues on next page)

(continued from previous page)

```
4
5  app = Flask(__name__) # application 'app' is object of class 'Flask'
6
7  subjects = ["English", "Dance", "Maths", "Geography"]
8  interests = ["Writing", "Dancing", "Logics", "Earth"]
9
10
11 # decorator 'app.route' binds the 'url' with 'function',
12 # i.e. url of 'home page (/)' will call function 'index' and
13 # 'return' value will be send back to the browser.
14 @app.route('/')  # root : main page
15 def index():
16     # by default, 'render_template' looks inside the folder 'template'
17     return render_template('index.html')
18
19 @app.route('/subject/<int:id>')
20 def subject(id):
21     return render_template('subject.html', sub=subjects[id])
22
23
24 @app.route('/interest/<int:id>')
25 def interest(id):
26     return render_template('interest.html', interest=interests[id])
27
28 if __name__ == '__main__':
29     # '0.0.0.0' = 127.0.0.1 i.e. localhost
30     # port = 5000 : we can modify it for localhost
31     app.run(host='0.0.0.0', port=5000, debug=True) # local webserver : app.run()
```

- Now, modify the 'index.html', here one url is added at Line 11, which goes to 'interest.html' page with the help of 'url_for'.

```
1  <!-- index.html -->
2
3  <!DOCTYPE html>
4  <html>
5  <head>
6      <title>Home</title>
7  </head>
8  <body>
9      <h1>Tell your interest!</h1>
10     <p> Please select one option as 'yes' from the questions ... </p>
11     <p><a href="{{ url_for('interest', id=0) }}"> Start </a></p>
12 </body>
13 </html>
```

Output of index.html page is shown below. If we press on 'Start', the page will redirect to 'interest.html/0'

```
Tell your interest!

Please select one option as 'yes' from the questions ...

Start
```

- Next, add the 'interest.html' page as below. Here, Line 13 reference back to 'index' page.

```
1  <!-- interest.html -->
2
3  <!DOCTYPE html>
4  <html>
5  <head>
```

(continues on next page)

(continued from previous page)

```
    <title>Interest</title>
</head>
<body>
    <h1> Select your interest? </h1>

    <p> {{interest}} </p>

    <p><a href="{{ url_for('index')}}"> Begin again </a></p>

</body>
</html>
```

Output of 'interest.html/0' is show below. If we press 'Begin again', the page will redirect to 'index.html'

```
Select your interest?

Writing

Begin again
```

1.6 Forms

In previous section, we created links to go from one page to another. In this section, we will add a form to get the input from the user; and based on input-value, the page will go to next page.

- In the below code, the form is added at Lines 13-19, whose output is shown in Fig. 1.2.

```
<!-- interest.html -->

<!DOCTYPE html>
<html>
<head>
    <title>Interest</title>
</head>
<body>
    <h1> Select your interest? </h1>

    <p> {{interest}} </p>

    <form method="post">
        <p>
            <input type="radio" name="choice" value="yes"> Yes <br>
            <input type="radio" name="choice" value="no"> No <br>
        </p>
        <input type="submit" value="Submit">
    </form>

    <p><a href="{{ url_for('index')}}"> Begin again </a></p>

</body>
</html>
```

- Below is the Python codes, which goes to 'interest-id' page (Lines 26-28), if choice is 'no' otherwise remains as the same page.

```
# app.py

from flask import Flask, render_template, request, redirect, url_for
```

(continues on next page)

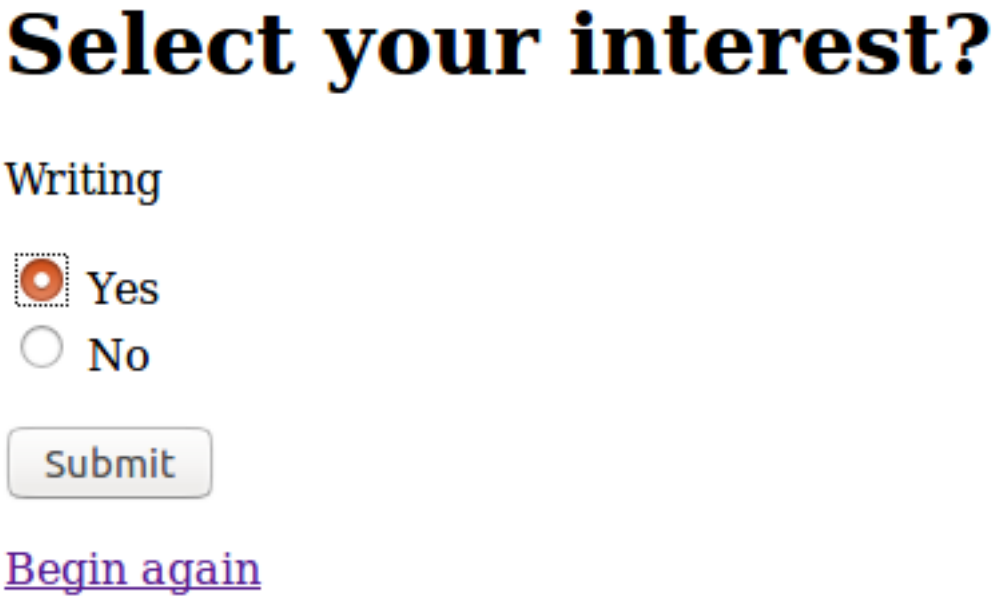

Fig. 1.2: Form

(continued from previous page)

```
4
5  app = Flask(__name__) # application 'app' is object of class 'Flask'
6
7  subjects = ["English", "Dance", "Maths", "Geography"]
8  interests = ["Writing", "Dancing", "Logics", "Earth"]
9
10
11 # decorator 'app.route' binds the 'url' with 'function',
12 # i.e. url of 'home page (/)' will call function 'index' and
13 # 'return' value will be send back to the browser.
14 @app.route('/')  # root : main page
15 def index():
16     # by default, 'render_template' looks inside the folder 'template'
17     return render_template('index.html')
18
19 @app.route('/subject/<int:id>')
20 def subject(id):
21     return render_template('subject.html', sub=subjects[id])
22
23
24 @app.route('/interest/<int:id>', methods=['GET', 'POST'])
25 def interest(id):
26     if request.method == 'POST':
27         if request.form['choice'] == 'no':
28             return redirect(url_for('interest', id=id+1))
29     return render_template('interest.html', interest=interests[id])
30
31 if __name__ == '__main__':
32     # '0.0.0.0' = 127.0.0.1 i.e. localhost
33     # port = 5000 : we can modify it for localhost
34     app.run(host='0.0.0.0', port=5000, debug=True) # local webserver : app.run()
```

1.7 More about Forms

In previous section, we have created a basic form which does not have any validation/security methods in it. In this section, we will use some built in methods for creating the form,

- Line 8 is added for avoiding the CSRF attack.
- Line 13-16 creates a class 'SelectChoiceForm' which adds the various choices for the Form.
- Line 34 creates an object of class SelectChoiceForm.
- Line 35 validates the submitted data.
- Lines 38-39 redirects the url to 'subject.html' page, if the choice is yes; otherwise Line 37 redirects the page to next interest-option.
- Lastly, the the object 'form' is send to html page through Line 40.

```
1  # app.py
2
3  from flask import Flask, render_template, request, redirect, url_for
4  from flask_wtf import Form
5  from wtforms.fields import RadioField, SubmitField
6
7  app = Flask(__name__) # application 'app' is object of class 'Flask'
8  app.config['SECRET_KEY'] = 'yourKeyHere!' # to avoid CSRF attack
9
10 subjects = ["English", "Dance", "Maths", "Geography"]
11 interests = ["Writing", "Dancing", "Logics", "Earth"]
12
13
14 class SelectChoiceForm(Form):
15     choice = RadioField('Select Yes/No', choices=[('yes','Yes'), ('no','No')])
16     submit = SubmitField('Submit')
17
18 # decorator 'app.route' binds the 'url' with 'function',
19 # i.e. url of 'home page (/)' will call function 'index' and
20 # 'return' value will be send back to the browser.
21 @app.route('/')  # root : main page
22 def index():
23     # by default, 'render_template' looks inside the folder 'template'
24     return render_template('index.html')
25
26 @app.route('/subject/<int:id>')
27 def subject(id):
28     return render_template('subject.html', sub=subjects[id])
29
30
31 @app.route('/interest/<int:id>', methods=['GET', 'POST'])
32 def interest(id):
33     form = SelectChoiceForm()
34     if form.validate_on_submit():
35     # if request.method == 'POST':
36         if request.form['choice'] == 'no':
37             return redirect(url_for('interest', id=id+1))
38         else:
39             return redirect(url_for('subject', id=id))
40     return render_template('interest.html', interest=interests[id], form=form)
41
42 if __name__ == '__main__':
43     # '0.0.0.0' = 127.0.0.1 i.e. localhost
44     # port = 5000 : we can modify it for localhost
45     app.run(host='0.0.0.0', port=5000, debug=True) # local webserver : app.run()
```

In the below code, the form is created in the html using the object 'form' (Lines 15-22) which is sent by 'app.py',

```
<!-- interest.html -->

<!DOCTYPE html>
<html>
<head>
    <title>Interest</title>
</head>
<body>
    <h1> Select your interest? </h1>

    <p> {{interest}} </p>

    <form method="post">

        {{ form.hidden_tag() }}
        <p>
            {% for ch in form.choice %}
                {{ ch }} {{ch.label}}<br>
            {% endfor %}
        </p>

        {{ form.submit }}

        <!-- <p>
            <input type="radio" name="choice" value="yes"> Yes <br>
            <input type="radio" name="choice" value="no"> No <br>
        </p>
        <input type="submit" value="Submit">  -->
    </form>

    <p><a href="{{ url_for('index')}}"> Begin again </a></p>

</body>
</html>
```

Lastly in 'subject.html', a link is added to go to 'index.html',

```
<!-- subject.html -->

<!DOCTYPE html>
<html>
<head>
    <title>Subject</title>
</head>
<body>
    <h1> Subject selection </h1>
    <p> Recommended subject : <b> {{ sub }} </b></p>

    <p>Thank you! Click <a href="{{ url_for('index')}}"> here </a> to try again</p>
</body>
</html>
```

Below are the outputs of different pages,

```
1.  http://0.0.0.0:5000/

        Tell your interest!

        Please select one option as 'yes' from the questions ...

        Start
```

(continues on next page)

(continued from previous page)

```
2. Click on 'start', the page will go to 'http://0.0.0.0:5000/interest/0'

        Select your interest?

        Writing

        Yes
        No

        Submit

        Begin again

3. Select 'No' and press submit, the page will go to 'http://0.0.0.0:5000/interest/1'

        Select your interest?

        Dancing

        Yes
        No

        Submit

        Begin again

4. Select 'Yes' and press submit. The page will go to 'http://0.0.0.0:5000/subject/1'

         Subject selection

        Recommended subject : Dance

        Thank you! Click here to try again

5. Click on 'here' to start again.
```

Warning: Since there are only 4 iterest-choices, therefore if we press 'No' four times, then following error will be generated,

```
IndexError: list index out of range
```

1.8 Sessions

In previous section, we vaildated the forms; and then 'interest' is asked from the user. If the choice is 'No', then next interest was shown. However, we can jump over to any interest by providing the correct 'id', i.e. in the below link, just provide the correct integer value i.e. 0, 1, 2 or 3 (as we have only 4 questions).

```
http://0.0.0.0:5000/interest/1'
```

Now, want that user should read all questions **one by one** i.e. we **do not** want that user can jump to any question just by changing the 'id'. This can be done using 'Sessions', as shown in below code,

- Line 23 creates a new session 'my_interest' with initial value 0.
- In Line 32, we removed the 'int:id' option from the URL, therefore user can not jump to any question by providing the 'id'.
- Next, the value of 'id' is provided by the session variable at Line 34.
- The value of 'id' is incremented at Line 39 (if the choice is 'No').

- Lastly, the 'id' is removed from the 'redirect' URL as well (see Line 40).

```
# app.py

from flask import Flask, render_template, request, redirect, url_for, session
from flask_wtf import Form
from wtforms.fields import RadioField, SubmitField

app = Flask(__name__) # application 'app' is object of class 'Flask'
app.config['SECRET_KEY'] = 'yourKeyHere!' # to avoid CSRF attack

subjects = ["English", "Dance", "Maths", "Geography"]
interests = ["Writing", "Dancing", "Logics", "Earth"]

class SelectChoiceForm(Form): # define elements of form e.g. RadioField
    # choices : (name for python, display on HTML)
    choice = RadioField('Select Yes/No', choices=[('yes','Yes'), ('no','No')])
    submit = SubmitField('Submit')

# decorator 'app.route' binds the 'url' with 'function',
# i.e. url of 'home page (/)' will call function 'index' and
# 'return' value will be send back to the browser.
@app.route('/')  # root : main page
def index():
    session['my_interest'] = 0
    # by default, 'render_template' looks inside the folder 'template'
    return render_template('index.html')

@app.route('/subject/<int:id>')
def subject(id):
    return render_template('subject.html', sub=subjects[id])


@app.route('/interest', methods=['GET', 'POST'])
def interest():
    id = session['my_interest']
    form = SelectChoiceForm()
    if form.validate_on_submit():
    # if request.method == 'POST':
        if request.form['choice'] == 'no':
            session['my_interest'] = id+1
            return redirect(url_for('interest'))
        else:
            return redirect(url_for('subject', id=id))
    return render_template('interest.html', interest=interests[id], form=form)

if __name__ == '__main__':
    # '0.0.0.0' = 127.0.0.1 i.e. localhost
    # port = 5000 : we can modify it for localhost
    app.run(host='0.0.0.0', port=5000, debug=True) # local webserver : app.run()
```

Now run the code again and observe following,

- First go to 'Main Page' i.e. 'http://0.0.0.0:5000/' and click on 'Start'.
  - And we will reach to URL 'http://0.0.0.0:5000/interest?id=0'. If we change the value of 'id' in this URL, nothing will happen as 'interest' method does not take argument any more.
  - If we try 'http://0.0.0.0:5000/interest/1', the 'Page not found' error will be displayed.
- Now, select 'No' from the option list, and next page question (i.e. **Dancing**) will be displayed. But, URL will not have any id i.e. 'http://0.0.0.0:5000/interest'
- Now, open a new tab in the browser (**do not close the browser**), and close the previous tab.
- Type the 'interest' URL in new tab i.e. 'http://0.0.0.0:5000/interest'. This will start the page from where we left i.e. second question **Dancing**, which is read by the session from the cookies.
- Next, close the browser and type type the interest link ('http://0.0.0.0:5000/interest') again. The error

KeyError: 'my_interest' will be displayed as 'session' is lost after closing the browser.

- Finally, go to main page (http://0.0.0.0:5000/) to start again.

Chapter 2

Flask with database

2.1 Introduction

In this chapter, we will use the Flask with database. Please understand the basic concepts of Flask in Chapter 1.

In this chapter, we will store some questions in the database; and then ask those questions to the user.

2.2 Basic setup

2.2.1 Files

- The file 'my_app.py' is the top level application.

```
# my_app.py

from flask import Flask

app = Flask(__name__) # application 'app' is object of class 'Flask'

# import files
from routes import *


if __name__ == '__main__':
    # '0.0.0.0' = 127.0.0.1 i.e. localhost
    # port = 5000 : we can modify it for localhost
    app.run(host='0.0.0.0', port=5000, debug=True) # local webserver : app.run()
```

- The file 'routes.py' stores all the 'routing-related' codes,

---

Note: This is complete code but data is not saved in the database. All the elements of this code are discussed in Chapter 1. The remaining database-related codes are as follows,

- Store 'questions' in the database (Lines 23-24).
- Read question from the database (Lines 34-35).
- Read answer from the database to compare with submitted-answer (Lines 42-43).

```
1 # routes.py
2
3 from my_app import app
```

(continues on next page)

(continued from previous page)

```
from flask import render_template, request

# home page
@app.route('/')  # root : main page
def index():
    # by default, 'render_template' looks inside the folder 'template'
    return render_template('index.html')

# Create question
@app.route('/create', methods=['GET', 'POST'])
def create():
    if request.method == 'GET':
        # send the form
        return render_template('create.html')
    else: # request.method == 'POST':
        # read data from the form and save in variable
        question = request.form['question']
        answer = request.form['answer']

        # store in database
        # add code here

        return render_template('createThanks.html', question=question)


# Display question
@app.route('/question/<int:id>', methods=['GET', 'POST'])
def question(id):
    if request.method == 'GET':
        # send the form
        # add code here to read the question from database
        question = "Not added yet"

        return render_template('question.html', question=question)
    else: # request.method == 'POST':
        # read and check answers
        submitted_answer = request.form['answer']

        # add code here to read the answer from database
        correct_answer = "Not added"

        if submitted_answer == correct_answer:
            return render_template('correct.html');
        else:
            return render_template('sorry.html',
                answer = correct_answer,
                yourAnswer = submitted_answer
            )
```

- The 'index.html' files is the main page. Note that 'id=1 (instead of id=0)' is used here as the SQLite-database index starts from '1'.

```
<!-- index.html -->

<!DOCTYPE html>
<html>
<head>
    <title>Home</title>
</head>
<body>
    <h1> Home Page </h1>
```

(continues on next page)

(continued from previous page)

```
    <p><a href="{{url_for('create')}}">Create Question</a></p>
    <p><a href="{{url_for('question', id=1)}}">Answer Question</a></p>
</body>
</html>
```

- The 'create.html' will store the form to 'create' and 'save' questions in the database.

```
<!-- create.html -->

<!DOCTYPE html>
<html>
<head>
    <title>Create</title>
</head>
<body>
    <h1> Create question </h1>

    <form method='post'>

            <b>Question</b> <input type="text" name="question"><br>
            <b>Answer</b> <input type="text" name="answer"><br>

            <input type="submit" name="submit" value="Submit">
            <input type="reset" name="reset" value="Reset"><br>
    </form>

    <p>Go to <a href="{{url_for('index')}}">main page</a></p>
</body>
</html>
```

- The 'question.html' file will present the question to the users,

```
<!-- question.html -->

<!DOCTYPE html>
<html>
<head>
    <title>Question</title>
</head>
<body>
    <h1> {{ question }} </h1>

    <form method='post'>
        <b>Answer</b> <input type="text" name="answer"><br>
        <input type="submit" name="submit" value="Submit">
        <input type="reset" name="reset" value="Reset"><br>
    </form>

    <p>Go to <a href="{{url_for('index')}}">main page</a></p>
</body>
</html>
```

- The 'correct.html' page will be displayed if the answer is correct,

```
<!-- correct.html  -->

<!DOCTYPE html>
<html>
<head>
    <title>Congratulation</title>
```

(continues on next page)

(continued from previous page)

```
</head>
<body>
    <h2> Congratulation! Your answer is correct. </h2>

    <p>Go to <a href="{{url_for('index')}}">main page</a></p>
</body>
</html>
```

- The 'sorry.html' will be displayed for incorrect answer,

```
<!-- sorry.html  -->

<!DOCTYPE html>
<html>
<head>
    <title>Sorry</title>
</head>
<body>
    <h2> Sorry! Your answer is incorrect.</h2>
    <p> The correct answer is : <b> {{ answer }} </b> </p>
    <p> You submitted : <b> {{ yourAnswer }} </b> </p>

    <p>Go to <a href="{{url_for('index')}}">main page</a></p>
</body>
</html>
```

- The 'createThanks.html' will be displayed for after creating the question.

```
<!-- createThanks.html  -->

<!DOCTYPE html>
<html>
<head>
    <title>Thanks</title>
</head>
<body>
    <h1> Thank You! Your question is submitted successfully. </h1>
    <p> Your question was : <b> {{ question }} </b> </p>

    <p>Go to <a href="{{url_for('index')}}">main page</a></p>
</body>
</html>
```

2.2.2 Check setup

Now run the project using following command,

```
$ python my_app.py
```

Go to following links and check the output,

- Home page http://0.0.0.0:5000/ with below output,

```
 Home Page

Create Question

Answer Question
```

- Create page http://0.0.0.0:5000/create with below output,

```
Create question

    Question
    Answer

    Submit  Reset

    Go to main page
```

- Question page http://0.0.0.0:5000/question/0 with below output,

```
Not added yet

Answer .........

Go to main page
```

- Type 'Not added' in the answer field, press submit,
- Correct page http://0.0.0.0:5000/correct with below output

```
Congratulation! Your answer is correct.

Go to main page
```

2.3 Create database and save data

We will use the SQLite database in this tutorial. Please see the MySQL tutorials to use the MySQL database. Further, the process of connecting to database using Python is discussed there.

Modify the 'routes.py' as below,

- Lines 8-11 create a database 'qa_database.db (Line 9)' and adds a table 'tbl_QA (Line 10)' to it. Note that, a new database will be creade if it does not exist; otherwise it **will not** create the new database.
- Lines 30-44 saves the form-data (Lines 27-28) to the database (Lines 35-36). Read comments for more details.
- The 'database_error.html' is used at Line 42 which shows the database-connection-error, if exists.

Note:

- In SQLite, the primary key (i.e. 'ID' at Line 9) starts with '1' (not with 0).
- Database queries are case-insensitive i.e 'id' and 'ID' are same thing.
- We can use the firefox-addon 'SQLite Manager', which is quite handy tool to see the data in SQLite database.

```
1  # routes.py
2  import sqlite3 as sql
3
4  from my_app import app
5  from flask import render_template, request
6
7  # connect to qa_database.sq (database will be created, if not exist)
8  con = sql.connect('qa_database.db')
9  con.execute('CREATE TABLE IF NOT EXISTS tbl_QA (ID INTEGER PRIMARY KEY AUTOINCREMENT,'
10             + 'question TEXT, answer TEXT)')
11 con.close
12
13 # home page
14 @app.route('/')  # root : main page
15 def index():
16     # by default, 'render_template' looks inside the folder 'template'
```

(continues on next page)

(continued from previous page)

```
    return render_template('index.html')

# Create question
@app.route('/create', methods=['GET', 'POST'])
def create():
    if request.method == 'GET':
        # send the form
        return render_template('create.html')
    else: # request.method == 'POST':
        # read data from the form and save in variable
        question = request.form['question']
        answer = request.form['answer']

        # store in database
        try:
            con = sql.connect('qa_database.db')
            c =  con.cursor() # cursor
            # insert data
            c.execute("INSERT INTO tbl_QA (question, answer) VALUES (?,?)",
                (question, answer))
            con.commit() # apply changes
            # go to thanks page
            return render_template('createThanks.html', question=question)
        except con.Error as err: # if error
            # then display the error in 'database_error.html' page
            return render_template('database_error.html', error=err)
        finally:
            con.close() # close the connection


# Display question
@app.route('/question/<id>', methods=['GET', 'POST'])
def question(id):
    if request.method == 'GET':
        # send the form
        # add code here to read the question from database
        question = "Not added yet"

        return render_template('question.html', question=question)
    else: # request.method == 'POST':
        # read and check answers
        submitted_answer = request.form['answer']

        # add code here to read the answer from database
        correct_answer = "Not added"

        if submitted_answer == correct_answer:
            return render_template('correct.html');
        else:
            return render_template('sorry.html',
                answer = correct_answer,
                yourAnswer = submitted_answer
            )
```

- Below is the content of file 'database_error.html' to display the database connection errors,

```
<!-- database_error.html  -->

<!DOCTYPE html>
<html>
<head>
```

(continues on next page)

(continued from previous page)

```
    <title>Database</title>
</head>
<body>
    <h2> Error in database connection </h2>

    <p> Error : <b>{{ error }}</b> </p>
    <p>Go to <a href="{{url_for('index')}}">main page</a></p>
</body>
</html>
```

- Now, go to the 'create page (http://0.0.0.0:5000/create)' and save some data in the database as shown below,

```
Question : 2+2
Answer = 4

Add some more question like this.
```

2.4 Read the data from database

Now, we will add the code display the question (based on ID) by reading the database; and reading the 'answer' from the database to compare it with submitted answer.

- Lines 54-68 reads the question from the database.
- Lines 75-88 reads the answer from the database.
- Read comments for further details.

```
1  # routes.py
2
3  import sqlite3 as sql
4
5  from my_app import app
6  from flask import render_template, request
7
8  # connect to qa_database.sq (database will be created, if not exist)
9  con = sql.connect('qa_database.db')
10 con.execute('CREATE TABLE IF NOT EXISTS tbl_QA (ID INTEGER PRIMARY KEY AUTOINCREMENT,'
11             + 'question TEXT, answer TEXT)')
12 con.close
13
14 # home page
15 @app.route('/')  # root : main page
16 def index():
17     # by default, 'render_template' looks inside the folder 'template'
18     return render_template('index.html')
19
20 # Create question
21 @app.route('/create', methods=['GET', 'POST'])
22 def create():
23     if request.method == 'GET':
24         # send the form
25         return render_template('create.html')
26     else: # request.method == 'POST':
27         # read data from the form and save in variable
28         question = request.form['question']
29         answer = request.form['answer']
30
31         # store in database
32         try:
```

(continues on next page)

(continued from previous page)

```
            con = sql.connect('qa_database.db')
            c =  con.cursor() # cursor
            # insert data
            c.execute("INSERT INTO tbl_QA (question, answer) VALUES (?,?)",
                (question, answer))
            con.commit() # apply changes
            # go to thanks page
            return render_template('createThanks.html', question=question)
        except con.Error as err: # if error
            # then display the error in 'database_error.html' page
            return render_template('database_error.html', error=err)
        finally:
            con.close() # close the connection


# Display question
@app.route('/question/<int:id>', methods=['GET', 'POST'])
def question(id):
    if request.method == 'GET':
        # send the form
        # code to read the question from database
        try:
            con = sql.connect('qa_database.db')
            c =  con.cursor() # cursor
            # read question : SQLite index start from 1 (see index.html)
            query = "Select question FROM tbl_QA where id = {0}".format(id)
            c.execute(query)
            question = c.fetchone() # fetch the data from cursor
            con.commit() # apply changes
            # go to thanks page : pass the value of tuple using question[0]
            return render_template('question.html', question=question[0])
        except con.Error as err: # if error
            # then display the error in 'database_error.html' page
            return render_template('database_error.html', error=err)
        finally:
            con.close() # close the connection

        return render_template('question.html', question=question)
    else: # request.method == 'POST':
        # read and check answers
        submitted_answer = request.form['answer']

        # code to read the answer from database
        try:
            con = sql.connect('qa_database.db')
            c =  con.cursor() # cursor
            # read answer : SQLite index start from 1 (see index.html)
            query = "Select answer FROM tbl_QA where id = {0}".format(id)
            c.execute(query)
            correct_answer = c.fetchone()[0] # fetch and store tuple-value (see [0])
            con.commit() # apply changes
        except con.Error as err: # if error
            # then display the error in 'database_error.html' page
            return render_template('database_error.html', error=err)
        finally:
            con.close() # close the connection

        if submitted_answer == correct_answer:
            return render_template('correct.html');
        else:
            return render_template('sorry.html',
```

(continues on next page)

(continued from previous page)

```
                answer = correct_answer,
                yourAnswer = submitted_answer
            )
```

- Now, go to question link e.g. http://0.0.0.0:5001/question/0 and answer the question.
- If answer is correct then we will get message from 'correct.html' as below,

```
Congratulation! Your answer is correct.

Go to main page
```

- For incorrect answer, following message will be displayed through 'sorry.html'

```
Sorry! Your answer is incorrect.

The correct answer is : 4

You submitted : 5

Go to main page
```

Note: We did not add the app.route for 'correct.html' and 'sorry.html', therefore the URL will not change after submitting the answers as shown in Fig. 2.1

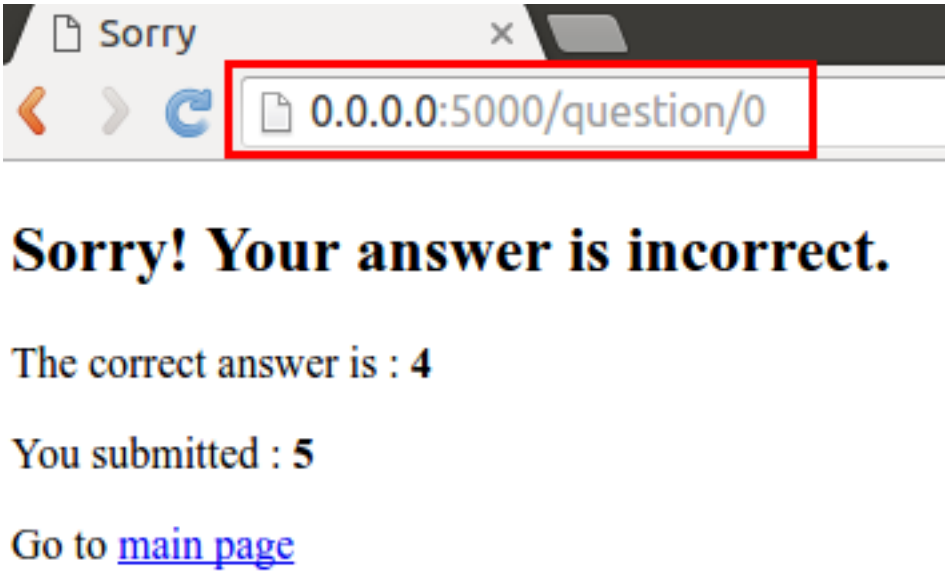

Fig. 2.1: URL after submitting the answer