Sessions in python flask

Like Cookie, Session data is stored on the client. Session is the time interval when a client logs into a server and logs out of it. The data, which is needed to be held across this session, is stored in the client browser.
A session with each client is assigned a Session ID. The Session data is stored on top of cookies and the server signs them cryptographically. For this encryption, a Flask application needs a defined SECRET_KEY.
Session object is also a dictionary object containing key-value pairs of session variables and associated values.
For example, to set a ‘username’ session variable use the statement −
Session[‘username’] = ’admin’

To release a session variable use pop() method.
session.pop('username', None)

The following code is a simple demonstration of session works in Flask. URL ‘/’ simply prompts the user to log in, as the session variable ‘username’ is not set.

```
@app.route('/')
def index():
  if 'username' in session:
    username = session['username']
      return 'Logged in as ' + username + '<br>' + \
      "<b><a href = '/logout'>click here to log out</a></b>"
  return "You are not logged in <br><a href = '/login'></b>" + \
    "click here to log in</b></a>"
```

As the user browses to ‘/login’ the login() view function, because it is called through the GET method, opens up a login form.
A Form is posted back to ‘/login’ and now the session variable is set. Application is redirected to ‘/’. This time session variable ‘username’ is found.

```
@app.route('/login', methods = ['GET', 'POST'])
def login():
  if request.method == 'POST':
    session['username'] = request.form['username']
    return redirect(url_for('index'))
  return '''

  <form action = "" method = "post">
    <p><input type = text name = username/></p>
    <p<<input type = submit value = Login/></p>
  </form>

  '''
```

The application also contains a logout() view function, which pops out ‘username’ session variable. Hence, ‘/’ URL again shows the opening page.

```
@app.route('/logout')
def logout():
   # remove the username from the session if it is there
   session.pop('username', None)
   return redirect(url_for('index'))
```

Run the application and visit the homepage. (Ensure to set secret_key of the application)

```
from flask import Flask, session, redirect, url_for, escape, request
app = Flask(__name__)
app.secret_key = 'any random string’
```

The output will be displayed as shown below. Click the link “click here to log in”.

Url_for in python:

The url_for() function is very useful for dynamically building a URL for a specific function. The function accepts the name of a function as first argument, and one or more keyword arguments, each corresponding to the variable part of URL.
The following script demonstrates use of url_for() function.

```
from flask import Flask, redirect, url_for
app = Flask(__name__)

@app.route('/admin')
def hello_admin():
   return 'Hello Admin'

@app.route('/guest/<guest>')
def hello_guest(guest):
   return 'Hello %s as Guest' % guest

@app.route('/user/<name>')
def hello_user(name):
   if name =='admin':
      return redirect(url_for('hello_admin'))
   else:
      return redirect(url_for('hello_guest',guest = name))

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

The above script has a function user(name) which accepts a value to its argument from the URL.
The User() function checks if an argument received matches ‘admin’ or not. If it matches, the application is redirected to the hello_admin() function using url_for(), otherwise to the hello_guest() function passing the received argument as guest parameter to it.
Save the above code and run from Python shell.
Open the browser and enter URL as − http://localhost:5000/user/admin
The application response in browser is −

```
Hello Admin
```

Enter the following URL in the browser − http://localhost:5000/user/mvl
The application response now changes to −

```
Hello mvl as Guest
```

Flask-WTF:

WTF stands for WT Forms which is intended to provide the interactive user interface for the user. The WTF is a built-in module of the flask which provides an alternative way of designing forms in the flask web applications.

Why WTF Useful?
WTF is useful due to the following factors.

The form elements are sent along with the request object from the client side to the server side. Server-Side script needs to recreate the form elements since there is no direct mapping between the client side form elements and the variables to be used at the server side.
There is no way to render the HTML form data at real time.
The WT Forms is a flexible, form rendering, and validation library used to provide the user interface.

The standard form fields are listed below.

SN	Form Field	Description
1	TextField	It is used to represent the text field HTML form element.
2	BooleanField	It is used to represent the checkbox HTML form element.
3	DecimalField	It is used to represent the text field to display the numbers with decimals.
4	IntegerField	It is used to represent the text field to display the integer values.
5	RadioField	It is used to represent the radio button HTML form element.
6	SelectField	It is used to represent the select form element.
7	TextAreaField	It is used to represent text area form elements.
8	PasswordField	It is used to take the password as the form input from the user.
9	SubmitField	It represents the <input type = 'submit' value = 'Submit'> html form element.

Object relational mapping is a technique of mapping object parameters to the underlying RDBMS table structure.
An ORM API provides methods to perform CRUD operations without having to write raw SQL statements.

Now create a Flask application object and set URI for the database to be used.

```
app = Flask(__name__) app.config['SQLALCHEMY_DATABASE_URI']
='sqlite:///students.sqlite3'
```

Step 4 − Then create an object of SQLAlchemy class with the application object as the parameter. This object contains helper functions for ORM operations. It also provides a parent Model class using which user defined models are declared. In the snippet below, a student model is created.

```
db = SQLAlchemy(app)
class students(db.Model):
  id = db.Column('student_id', db.Integer, primary_key = True)
  name = db.Column(db.String(100))
  city = db.Column(db.String(50))
  addr = db.Column(db.String(200))
  pin = db.Column(db.String(10))

def __init__(self, name, city, addr,pin):
  self.name = name
  self.city = city
  self.addr = addr
  self.pin = pin
```

Step 5 − To create / use the database mentioned in URI, run the create_all() method.

```
db.create_all()
```

The Session object of SQLAlchemy manages all persistence operations of ORM objects.
The following session methods perform CRUD operations −

- db.session.add(model object) − inserts a record into mapped table
- db.session.delete(model object) − deletes record from table
- model.query.all() − retrieves all records from table (corresponding to SELECT query).

Http Request methods – Python requests

Python requests module has several built-in methods to make Http requests to specific URL using GET, POST, PUT, PATCH or HEAD requests. A Http request is meant to either retrieve data from a specified URI or to push data to a server. It works as a request-response protocol between a client and server.

A web browser may be the client, and an application on a computer that hosts a web site may be the server. This article revolves around various methods that can be used to make a request to a specified URI.

Http Request methods:

METHOD	DESCRIPTION
GET	GET method is used to retrieve information from the given server using a given URI.
POST	POST request method requests that a web server accepts the data enclosed in the body of the request message, most likely for storing it
PUT	The PUT method requests that the enclosed entity be stored under the supplied URI. If the URI refers to an already existing resource, it is modified and if the URI does not point to an existing resource, then the server can create the resource with that URI.
DELETE	The DELETE method deletes the specified resource
HEAD	The HEAD method asks for a response identical to that of a GET request, but without the response body.
PATCH	It is used to modify capabilities. The PATCH request only needs to contain the changes to the resource, not the complete resource

GET:

GET method is used to retrieve information from the given server using a given URI. The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the ‘?’ character.

For example:

https://www.google.com/search?q=hello

How to make GET requests through Python Requests?

Python’s requests module provides an in-built method called get() for making a GET request to a specified URI.
requests.get(url, params={key: value}, args)

Example –

Let’s try making a request to github’s APIs for example purposes.

```
import requests

# Making a GET request
r = requests.get('https://www.google.com/')

# check status code for response received
# success code - 200
print(r)

# print content of request
print(r.content)
```

Output –
```
<Response [200]>
b'<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" so on
```

POST

POST is a request method supported by HTTP used by the World Wide Web. By design, the POST request method requests that a web server accepts the data enclosed in the body of the request message, most likely for storing it. It is often used when uploading a file or when submitting a completed web form.

How to make POST requests through Python Requests?

Python’s requests module provides an in-built method called post() for making a POST request to a specified URI.

Syntax –
requests.post(url, params={key: value}, args)

Example –

```
import requests

# Making a POST request
r = requests.post('https://httpbin.org / post', data ={'key':'value'})

# check status code for response recieved
# success code - 200
print(r)

# print content of request
print(r.json())
```

Output –

PUT

PUT is a request method supported by HTTP used by the World Wide Web. The PUT method requests that the enclosed entity be stored under the supplied URI. If the URI refers to an already existing resource, it is modified and if the URI does not point to an existing resource, then the server can create the resource with that URI.

How to make PUT request through Python Requests?

Python's requests module provides an in-built method called put() for making a PUT request to a specified URI.

Syntax –
requests.put(url, params={key: value}, args)

Example –
Let's try making a request to httpbin's APIs for example purposes.

```
import requests

# Making a PUT request
r = requests.put('https://httpbin.org / put', data ={'key':'value'})

# check status code for response recieved
# success code - 200
print(r)

# print content of request
print(r.content)
```

Output –

DELETE

DELETE is a request method supported by HTTP used by the World Wide Web.
The DELETE method deletes the specified resource.
As with a PUT request, you need to specify a particular resource for this operation.
A successful response SHOULD be 200 (OK) if the response includes an entity describing the status, 202 (Accepted) if the action has not yet been enacted, or 204 (No Content) if the action has been enacted but the response does not include an entity.

An example URI looks like for delete operation
http://www.example.com/articles/12345

How to make a DELETE request through Python Requests?

Python's requests module provides an in-built method called delete() for making a DELETE request to a specified URI.

Syntax –
requests.delete(url, params={key: value}, args)

Example –

Let's try making a request to httpbin's APIs for example purposes.
filter_none
brightness_4

```
import requests

# Making a DELETE request
r = requests.delete('https://httpbin.org / delete', data ={'key':'value'})

# check status code for response received
# success code - 200
print(r)

# print content of request
print(r.json())
```

Output –

HEAD

HEAD is a request method supported by HTTP used by the World Wide Web.
The HEAD method asks for a response identical to that of a GET request, but without the response body.
This is useful for retrieving meta-information written in response headers, without having to transport the entire content.
How to make HEAD request through Python Requests
Python's requests module provides an in-built method called head() for making a HEAD request to a specified URI.

Syntax –
requests.head(url, params={key: value}, args)

Example –
Let’s try making a request to httpbin’s APIs for example purposes.

```
 import requests

 # Making a HEAD request
 r = requests.head('https://httpbin.org/', data ={'key':'value'})

 # check status code for response received
 # success code - 200
 print(r)

 # print headers of request
 print(r.headers)

 # checking if request contains any content
 print(r.content)
```

Output –

PATCH

PATCH is a request method supported by HTTP used by the World Wide Web.
It is used to modify capabilities.
The PATCH request only needs to contain the changes to the resource, not the complete resource.
This resembles PUT, but the body contains a set of instructions describing how a resource currently residing on the server should be modified to produce a new version.
This means that the PATCH body should not just be a modified part of the resource, but in some kind of patch language like JSON Patch or XML Patch. PATCH is neither safe nor idempotent.

How to make Patch request through Python Requests?

Python’s requests module provides an in-built method called patch() for making a PATCH request to a specified URI.

Syntax –
requests.patch(url, params={key: value}, args)

Example –
Let’s try making a request to httpbin’s APIs for example purposes.

```
import requests

# Making a PATCH request
r = requests.patch('https://httpbin.org / patch', data ={'key':'value'})

# check status code for response received
# success code - 200
print(r)

# print content of request
print(r.content)
```

save this file as request.py and through terminal run,
python request.py
Output –

CSRF token:

Cross-site request forgery, A CSRF token is a random, hard-to-guess string. On a page with a form you want to protect, the server would generate a random string, the CSRF token, add it to the form as a hidden field and also remember it somehow, either by storing it in the session or by setting a cookie containing the value.