Recitation 7

Flask Routes, Templates, Cookies, Sessions

Flask Routes

```
from flask import Flask

    app = Flask(__name__)

    @app.route(‘/hello')
    def home():
      return "<h1> Hello World </h1>"

    if __name__ =="__main__":
      app.run(port=8080)
```

- Decorator
- The **route()** decorator in Flask is used to bind URL to a function
- When client requests for the specific URL, the server call the corresponding function (home() in this case)

Flask Routes

- Can have multiple routes in a given program
- Like different routes for different functions associated with different pages – home page, login page, logout page, about page etc

- Can also bind a function using add_url_rule()

```
def hello_world():
   return 'hello world'
app.add_url_rule('/', 'hello', hello_world)
```

Flask – Variable Rules

- URL can be built dynamically
- The variable part is mark as **<variable-name>**
- In the given code snippet

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

@app.route('/hello/<name>')
def hello_name(name):
   return 'Hello %s!' % name

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

- The output on opening

- **http://localhost:5000/hello/CS1520**

- Hello CS1520

Flask Templates

- It is possible to return the output of a function bound to a certain URL in the form of HTML.

- Example:

```
from flask import Flask
app = Flask(__name__)
@app.route('/')
def index():
   return '<html><body><h1>'Hello World'</h1></body></html>'

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

Cumbersome approach.
Use templates instead!

Flask Templates

- Flask uses a Jinja2 template engine.
- HTML file can be rendered by the render_template() function

```
@app.route('/')
def index():
  return render_template(‘hello.html’)
```

hello.html has the same html code that we were returning in the previous slide

Basic Jinja Tags

- {{ ... }}
  - Expression tag, contents are evaluated and place in the text
- {% ... %}
  - Statement tag, used to define Jinja constructs and issue flow control statements
- {# ... #}
  - Comment

Give output of the following

- **hello.html**

```
<!doctype html>
<html>
  <body>

    <h1>Hello {{ name }}!</h1>

  </body>
</html>
```

- **user.py**

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

@app.route('/hello/<user>')
def hello_name(user):
  return render_template('hello.html', name
= user)

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

What do you see in page http://localhost:5000/hello/johnson after executing user.py?

Output

Output in **http://localhost:5000/hello/johnson** :

```
Hello Johnson!
```

Where to save your template .html files??

- Save the .html file in a folder named **templates** which is located in the same directory as the python file.

- For the two files in the previous example, you will save it in the following manner:
    - **Application_folder**
        - user.py
        - **templates**
            - hello.html

Another Example

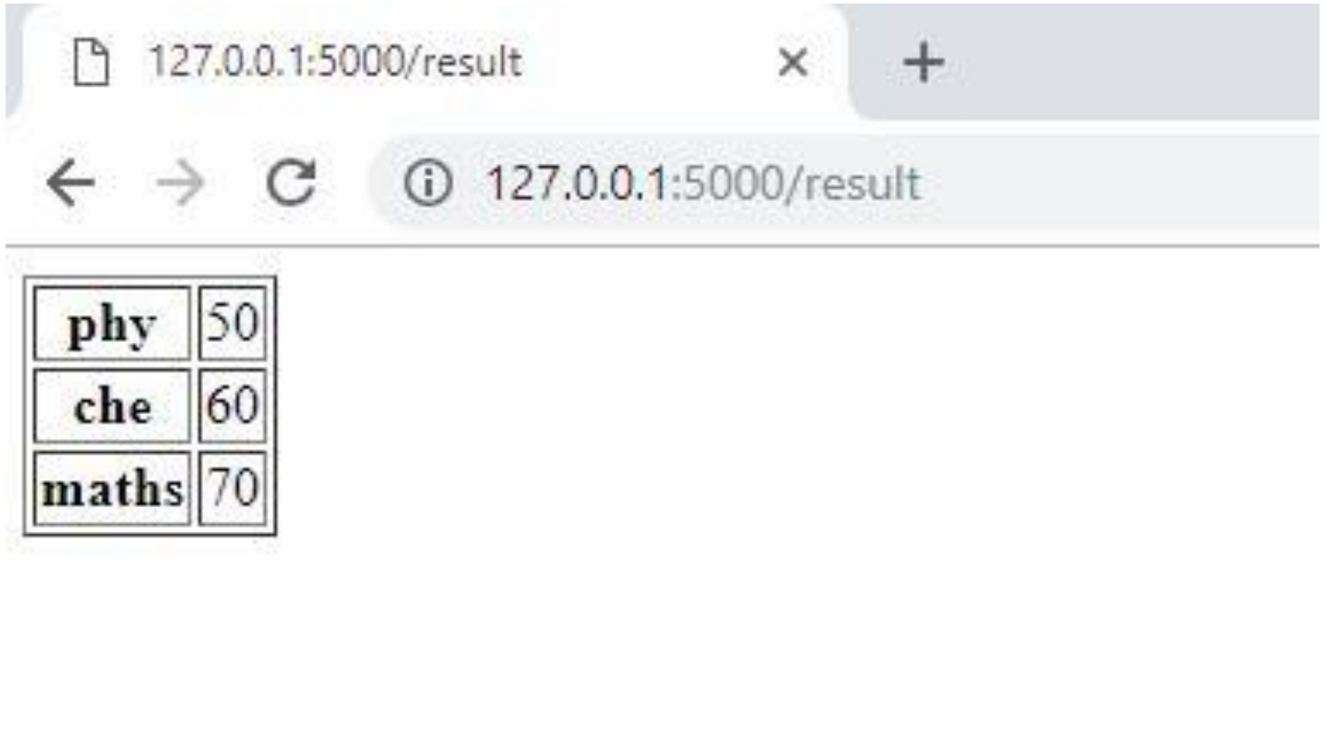
- **user.py**

```
from flask import Flask, render_template


app = Flask(__name__)
@app.route('/result')
def result():
   dict = {'phy':50,'che':60,'maths':70}
   return render_template('result.html',
result = dict)


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

- **result.html**

```
<!doctype html>
<html>
   <body>
         <table border = 1>
         {% for key, value in result.items() %}
           <tr>
               <th> {{ key }} </th>
               <td> {{ value }} </td>
            </tr>
         {% endfor %}
      </table>
         </body>
</html>
```

Output



phy	50
che	60
maths	70

Template Inheritance

- {% extends %} – key tag where a child template extends a base template
- {% block %} – define blocks that child templates can fill in
- {{ super() }} = To render the contents of a block defined in the parent template

- Please go through the example on http://flask.pocoo.org/docs/1.0/patterns/templateinheritance/

Flask Cookies

- A cookie is a piece of data which the server sets in the browser
    - Key:value pair

- Purpose
    - Login Information
    - For showing Ads related to the user

Creating Cookies in Flask

- set_cookie(key, value="", max_age=60*60)

- key
  - Name of the cookie
- value
  - Data we want to store in the cookie
  - Defaults to “”
- max_age
  - Expiration time of cookie in seconds
  - If not set, cookie expires when the browser is closed

Accessing Cookies

- The following code snippet accesses the cookie that was set

```
if not request.cookies.get('foo'):
    res = make_response("Setting a cookie")
    res.set_cookie('foo', 'bar', max_age=60*60*24*365*2)
else:
    res = make_response("Value of cookie foo is {}".format(request.cookies.get('foo')))
```

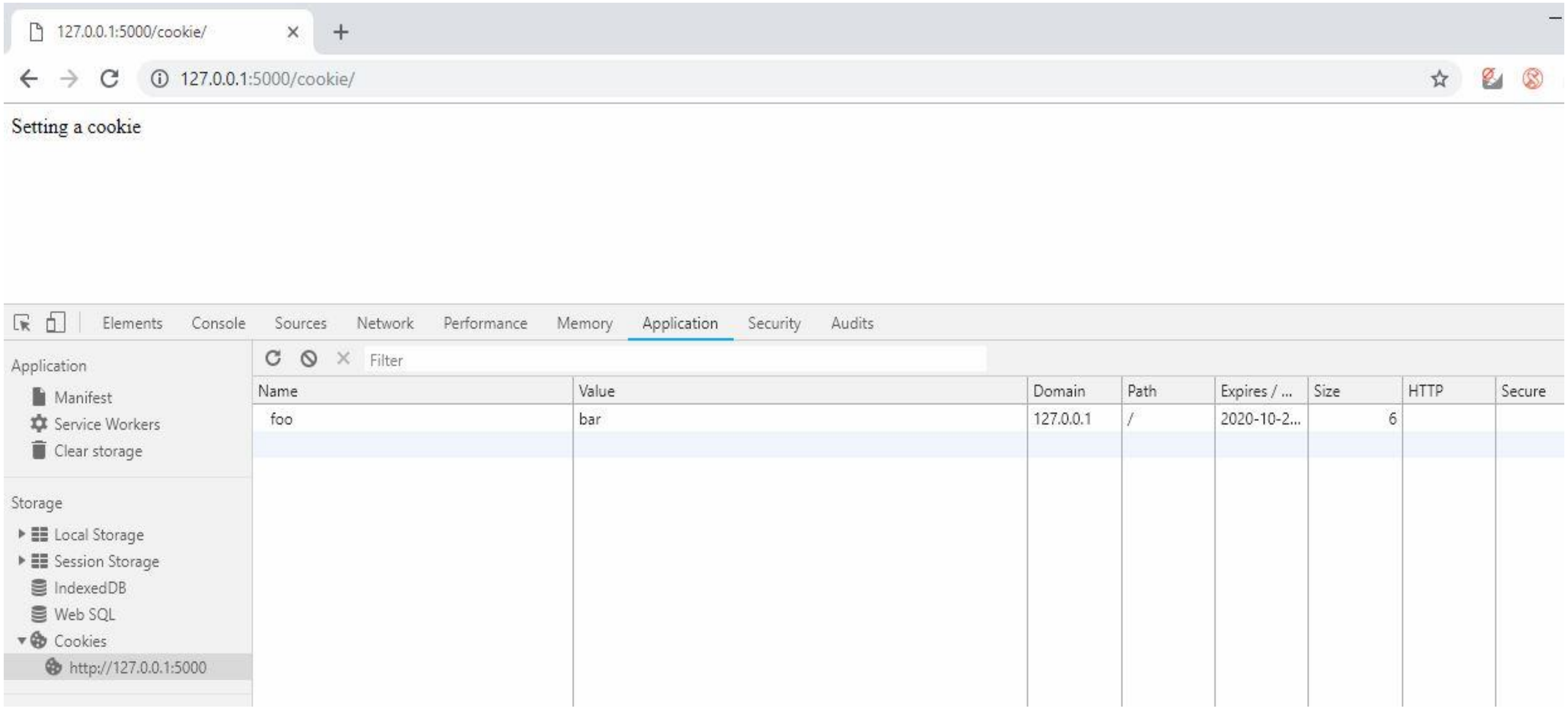
Deleting a cookie

- Call set_cookie() method with name and value of the cookie and set max_age = 0

```
@app.route('/delete-cookie/')
def delete_cookie():
     res = make_response("Cookie Removed")
    res.set_cookie('foo', 'bar', max_age=0)
    return res
```

Example

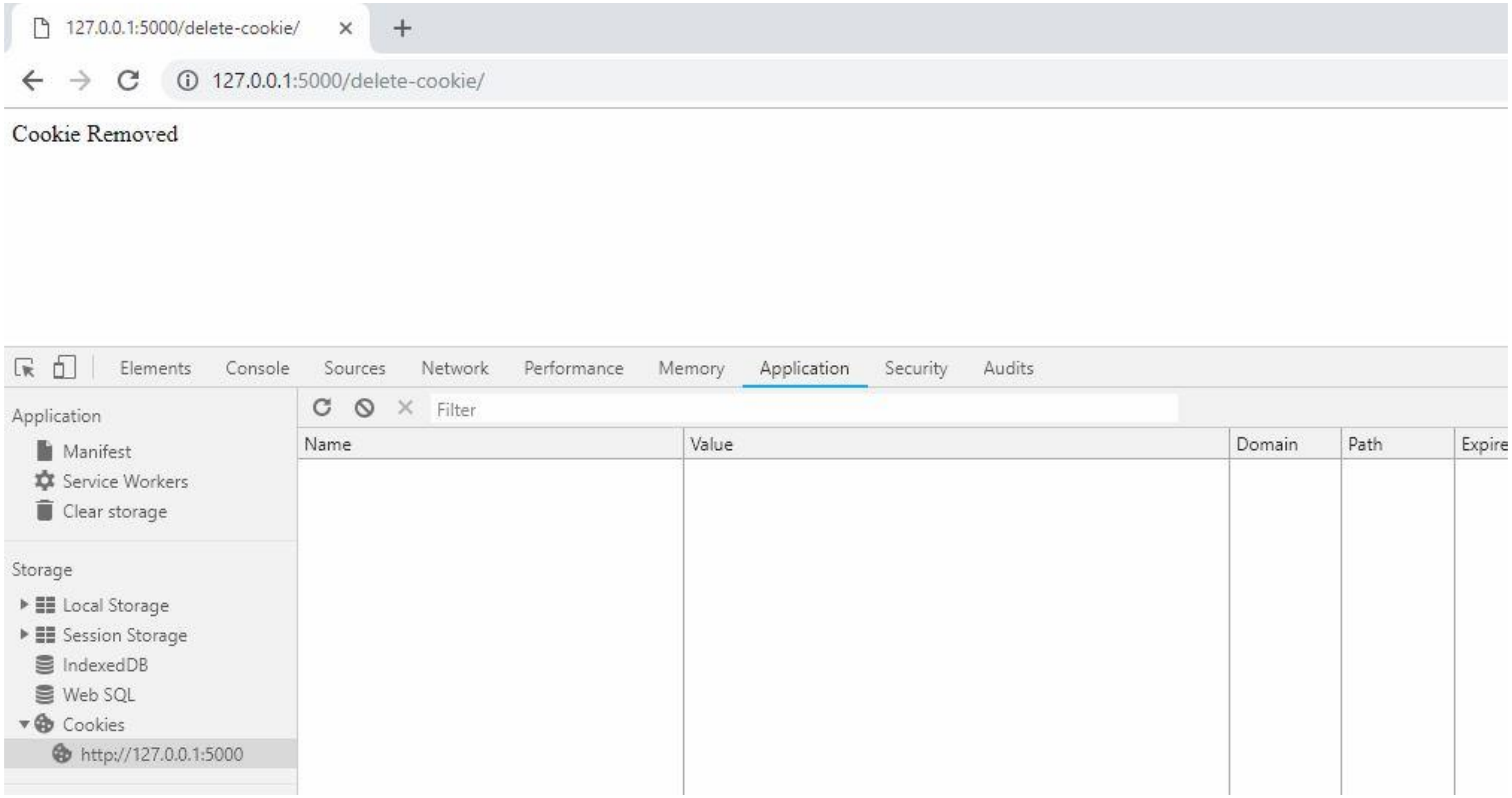
- Execute cookie.py
- See the cookie created in the browser



Setting a cookie

Name	Value	Domain	Path	Expires / ...	Size	HTTP	Secure
foo	bar	127.0.0.1	/	2020-10-2...	6		

After you execute *cookie.py*, you see a cookie named *foo* added to your browser when you visit the page http://127.0.0.1:5000/cookie


127.0.0.1:5000/delete-cookie/

127.0.0.1:5000/delete-cookie/

Cookie Removed

Name	Value	Domain	Path	Expire

The cookie named *foo* is removed when you visit http://127.0.0.1:5000/delete-cookie

Problems with cookies

- Not secure. Data stored in the cookie is visible to anyone
    - Shouldn’t use it to store sensitive information like password etc

- Cookies can be disabled. Most browsers give users option to disable cookies.

- Cookies are sent every time you request a page from the server. Causes additional payload

Flask Sessions

- Another way to store user specific data between requests

- Unlike an ordinary cookie, a session cookie encrypts its content such that anyone can view its content but cannot modify it.

- By default, sessions in Flask are client-based sessions. So, it suffers from similar vulnerabilities as cookies except that its content can’t be modified by a user

Creating a session

```
#!/usr/bin/python
from flask import Flask, session, request
import os

app = Flask(__name__)
app.secret_key = os.urandom(24)

@app.route('/')
def index():
	session['user'] = 'Mark'
	return 'Index'
```

Reading a session

```
@app.route('/getsession')
def getsession():
        if 'user' in session:
                return session['user']
        return 'Not logged in!’
```

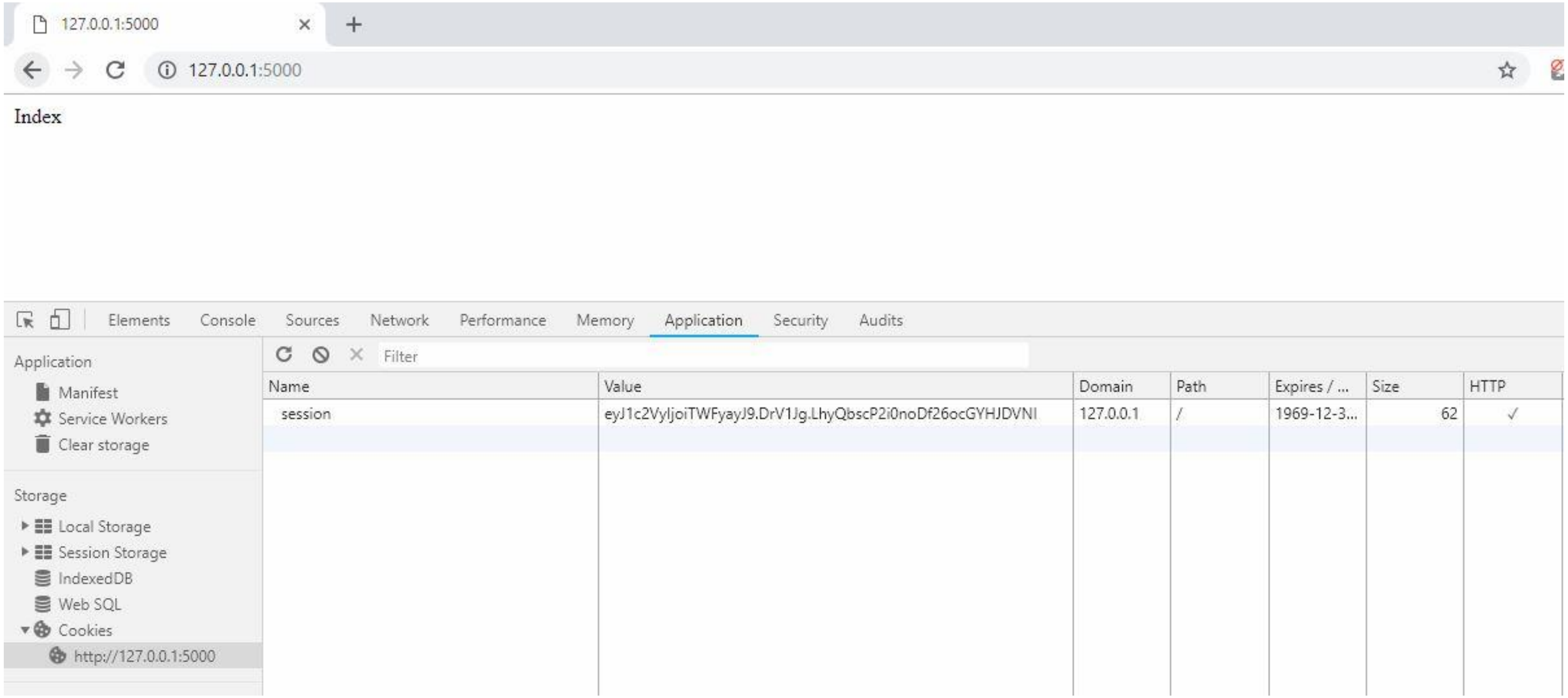
- Instead, we can also use session.get(‘user’)

Deleting a session

```
@app.route('/dropsession')
def dropsession():
            session.pop('user', None)
            return 'Session dropped'
if __name__ == "__main__":
    app.run()
```

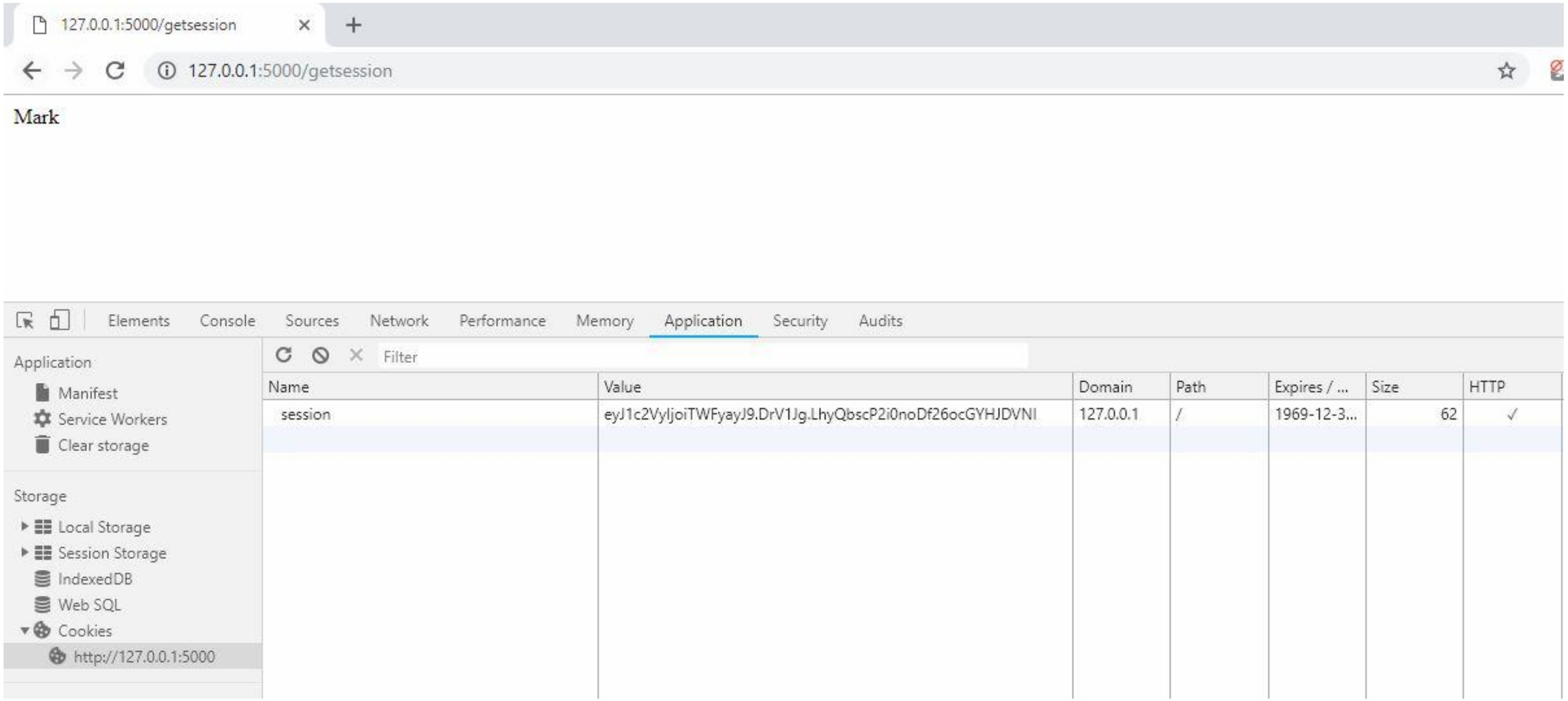
Example

- Run *session.py* and notice the cookie content after visiting various pages



Name	Value	Domain	Path	Expires / ...	Size	HTTP
session	eyJ1c2VyljoiTWFyayJ9.DrV1Jg.LhyQbscP2i0noDf26ocGYHJDVNI	127.0.0.1	/	1969-12-3...	62	✓

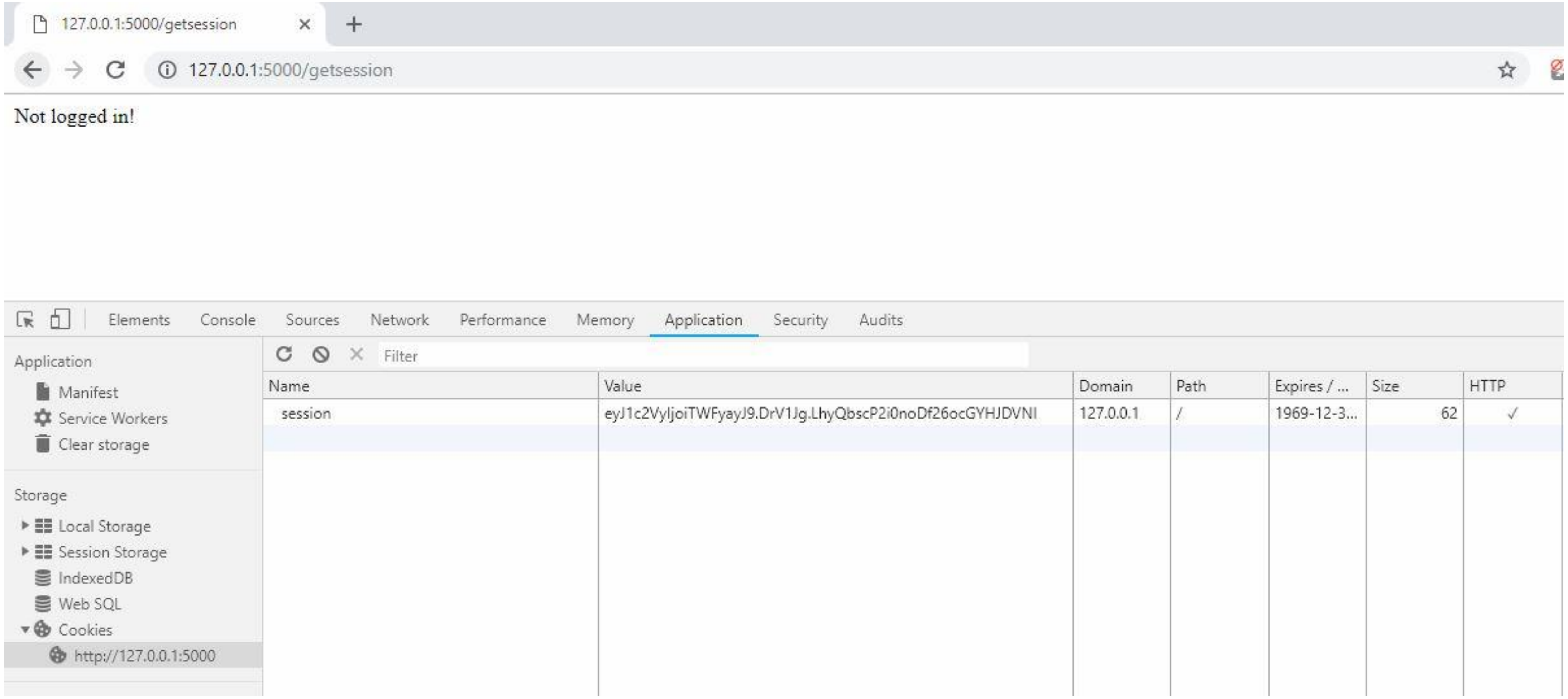
After executing session.py, a session cookie is created when you visit http://127.0.0.1:5000/. Notice that the value of the session cookie is encrypted by the secret key that we set in app.secret_key



Mark

Name	Value	Domain	Path	Expires / ...	Size	HTTP
session	eyJ1c2VyIjoiTWFyayJ9.DrV1Jg.LhyQbscP2i0noDf26ocGYHJDVNI	127.0.0.1	/	1969-12-3...	62	✓

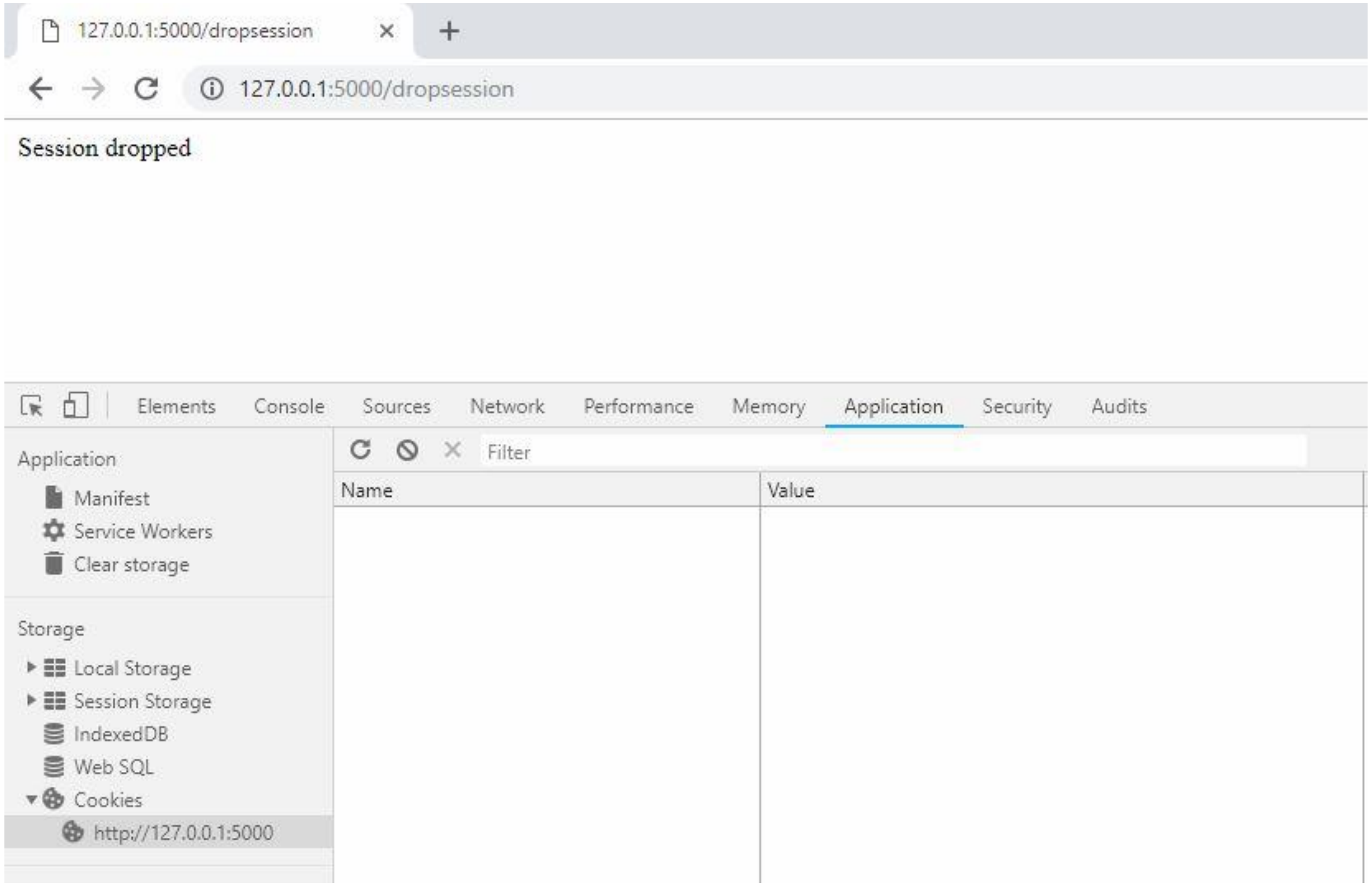
When you visit http://127.0.0.1:5000/getsession, we see that the value of the session cookie, Mark, is shown



Not logged in!

Name	Value	Domain	Path	Expires / ...	Size	HTTP
session	eyJ1c2VyljoiTWFyayJ9.DrV1Jg.LhyQbscP2i0noDf26ocGYHJDVNI	127.0.0.1	/	1969-12-3...	62	✓

- When you quit the Flask web server (Ctrl+C in your terminal) and restart it again (by executing **python session.py** in your terminal), and then refresh the page http://127.0.0.1:5000/getsession, you will see that the session has expired and it shows **Not Logged in!** even though the session cookie still remains in the browser.
  You need to visit http://127.0.0.1:5000 again for the session to be active again



Session dropped

When you visit http://127.0.0.1:5000/dropsession, the session cookie is deleted

References

- https://www.tutorialspoint.com/flask/
- https://overiq.com/flask/0.12/cookies-in-flask/
- https://overiq.com/flask/0.12/sessions-in-flask/
- https://w8s-class.github.io/CS1520-Class-Information/flask-templates.html
- https://www.youtube.com/watch?v=0SQdkDpMzKE&list=PLXmMXHVSvS-CMpHUeyIeqzs3kl-tIG-8R
- https://www.youtube.com/watch?v=T1ZVyY1LWOg&list=PLXmMXHVSvS-CMpHUeyIeqzs3kl-tIG-8R&index=2
- https://www.youtube.com/watch?v=QnDWIZuWYW0

Additional Reference

- A good video I found that discusses about how you create a Flask login page using sessions:
    - https://www.youtube.com/watch?v=eBwhBrNbrNI&index=3&list=PLXmMXHVSvS-CMpHUeyIeqzs3kl-tIG-8R