SWPP Practice Session #7

Advanced Features in Django & Unit Test

2022 Oct 19

Announcement

- If you have not, please make an appointment with TA for Sprint Meeting #2.
- Sprint #2 Report:
  - Due: **10/29(Sat) 6pm**
  - Materials to submit: Design and Planning & Sprint Backlog
  - For requirement and spec doc, please update the doc in the GitHub wiki.
- If you have ***any*** suggestions about methods to enhance your understanding of principles learned from regular classes(professor's lectures), send us an email(swpp.22.ta@spl.snu.ac.kr).

What We Will Cover Today

1. Django Admin
2. Cookie, Session, and CSRF
3. Django Unit Test
4. Postman

Clone Repo

- **Please fork and clone this repository**
- https://github.com/swpp22fall-practice-sessions/swpp-p7-django-advanced-and-tests
- We have checkpoint branches ready. If you're in trouble and can't keep up, you can jump to the following branches with `$ git checkout {branch_name}`

Setup - python virtual environment

- create python virtual environment with virtualenv

```
# in the directory where you want
$ virtualenv --python=python3.9 django-env

# check if django-env dir has been created
$ ls

# activate virtual environment
$ source django-env/bin/activate

# deactivate virtual environment
$ deactivate
```

Setup - docker container

- run docker container

```
$ docker run --rm -it \
     --ipc=host \
     --name "django" \
     -p 0.0.0.0:8000:8000 \
     -v ${PWD}:/home \
     snuspl/swpp:django
```

- with docker, you have to modify *settings.py*, open server with 0.0.0.0:8000 (later)

```
# settings.py
...
ALLOWED_HOSTS = ['0.0.0.0']
```

Django Admin

Django Admin

- When you need to manage models by hand
  - Interactive Django shell
    - Too tedious work!
- Django provides *automated* creation of admin interface.
- Not for users, but for site administrators!

Creating admin user

- `$ python manage.py createsuperuser`
- Fill out your form

```
~/swpp2020-django-practice/toh / test
> python manage.py createsuperuser
Username (leave blank to use 'yunmokoo'):
Email address:
Password:
Password (again):
Superuser created successfully.
```

Register model

- To register your model to admin interface, modify `admin.py`

hero/admin.py

```
1 from django.contrib import admin
2 from .models import Hero
3
4
5 # this will register your Hero model to admin page interface
6 admin.site.register(Hero)
```

Check your admin site urlconf

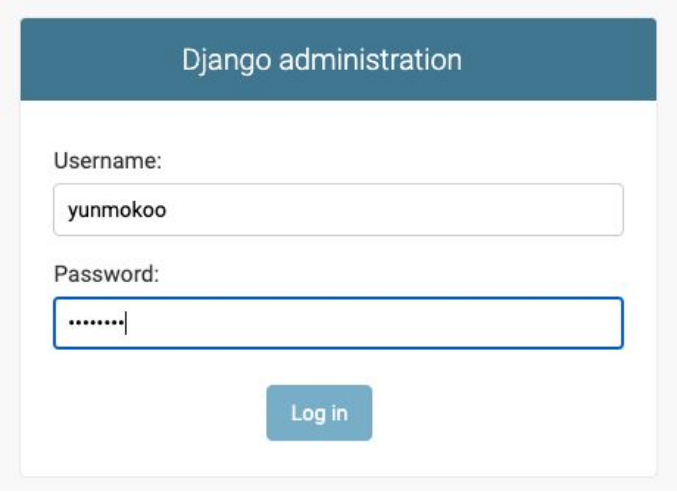
- Open toh/urls.py and check admin.site.urls path
  - Boilerplate code automatically generates this urlconf at the first time

toh/urls.py

```
1 from django.contrib import admin
2 from django.urls import include, path
3
4 urlpatterns = [
5     path('hero/', include('hero.urls')),
6     path('admin/', admin.site.urls),
7 ]
```

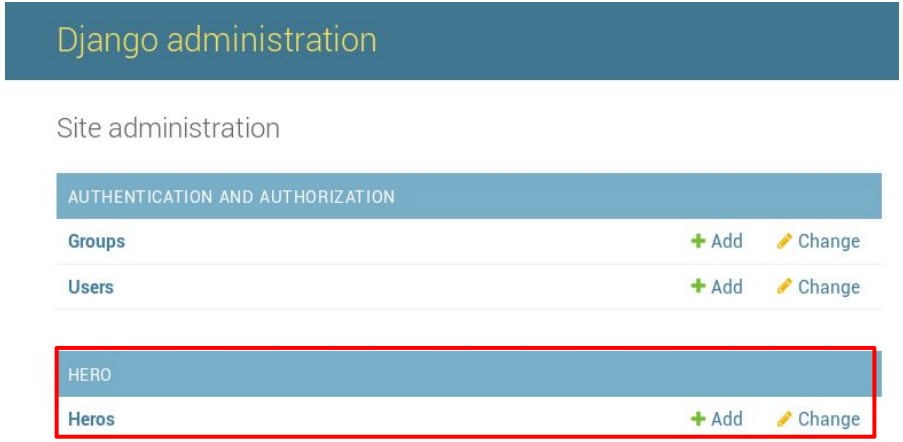
Test your admin page

- `$ python manage.py runserver`
- Open your web browser, and navigate to http://127.0.0.1:8000/admin/
- Sign in with your superuser credential



Check Hero model admin interface

- You can see the Hero model that you registered before
  - You can add, modify, and delete your Hero model easily



Let’s add some Heros again

Add hero

Name: Batman

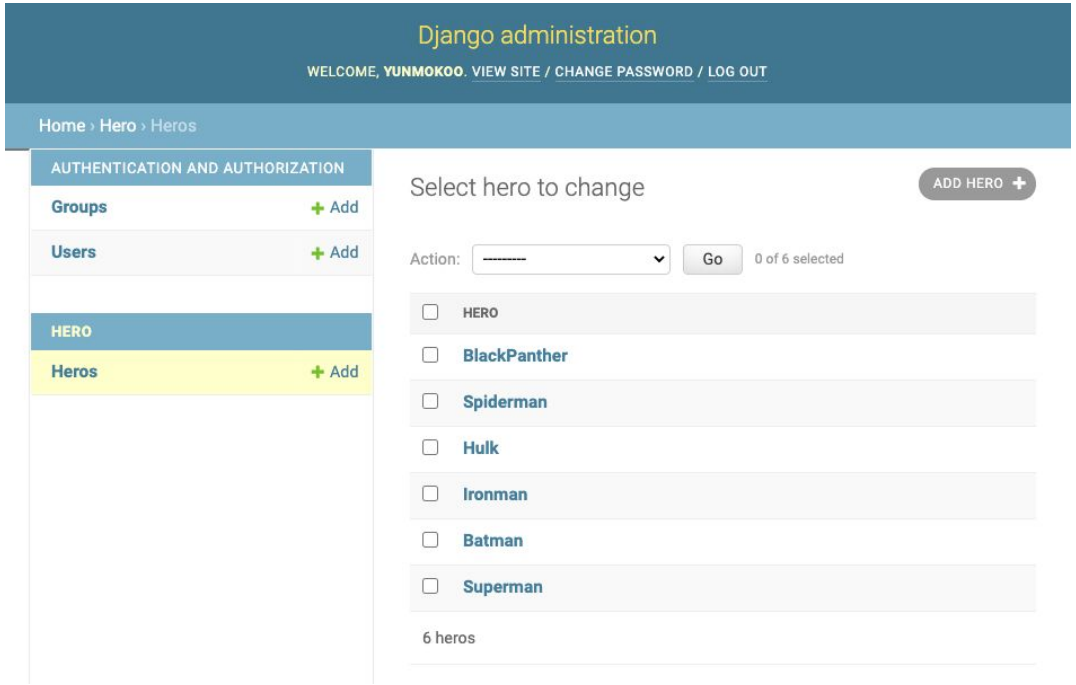
Age: 25

Score: 100

Save and add another | Save and continue editing | SAVE

Let’s add some Heros again

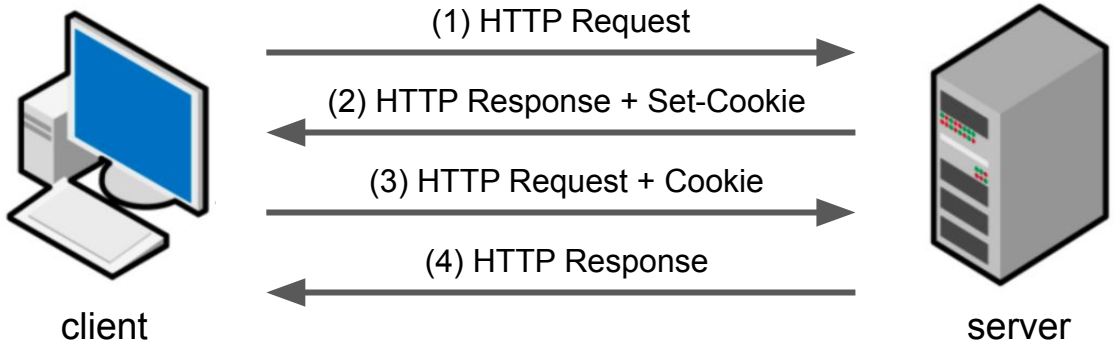
- You can also PUT, DELETE the record easily in the admin page.



Cookie, Session, and CSRF

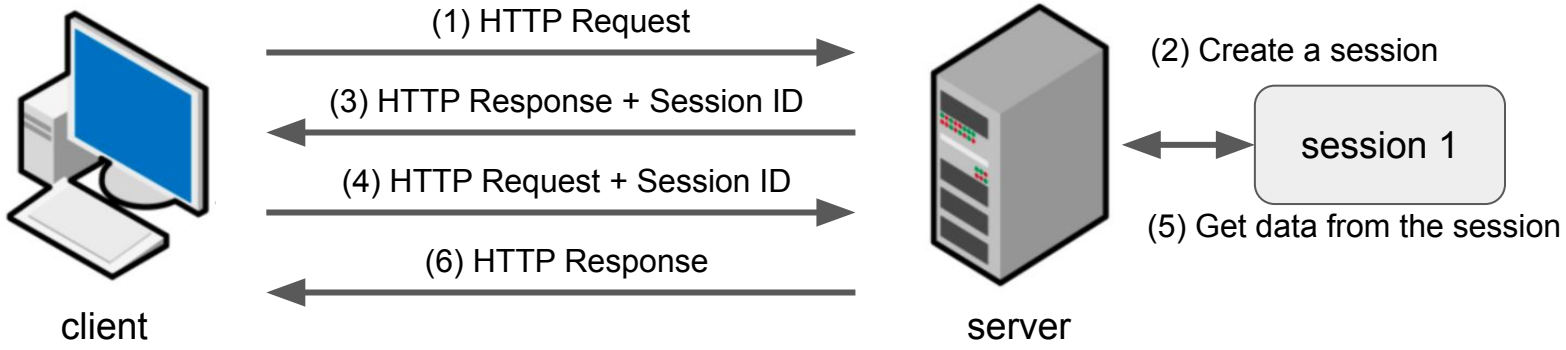
Cookie

- Problems of HTTP
  - Connectionless: The connection is closed after response.
  - Stateless: Once disconnected, states are not preserved.
- Cookie is a file stored in client local.
  - key-value pairs
  - name, value, expires, domain, secure, …



Session

- Problems of cookie
  - network traffic & security
- The data file is stored in server, not client local.
  - Client keeps HTTP session id as a cookie.



Session

- Associating arbitrary information with individual visitors
  - e.g. sign in status
- Distinguish users by setting different *Cookie*
- Use cookie as session key, not session value!
  - session value will be reside in Django side only, not user side
  - session id is generated based on host + client info(e.g. client device, software type), so even if session id is exposed, it is harder to use it for malicious attempts

Enable session in Django

- Make sure your `MIDDLEWARE` list contains:
  - `django.contrib.sessions.middleware.SessionMiddleware`
  - By default, your session store is your database

toh/settings.py

```
43 MIDDLEWARE = [
44     'django.middleware.security.SecurityMiddleware',
45     'django.contrib.sessions.middleware.SessionMiddleware',
46     'django.middleware.common.CommonMiddleware',
47     'django.middleware.csrf.CsrfViewMiddleware',
48     'django.contrib.auth.middleware.AuthenticationMiddleware',
49     'django.contrib.messages.middleware.MessageMiddleware',
50     'django.middleware.clickjacking.XFrameOptionsMiddleware',
51 ]
```

Using session in your view

- Just use `request.session` object to your key-value store
- Modify your `hero/views.py` to use session

hero/views.py

```
4  def index(request):
5      if 'visit_count' not in request.session:
6          request.session['visit_count'] = 1
7      else:
8          request.session['visit_count'] += 1
9      return HttpResponse('Hello, world! You visited {} times.\n'
10                         .format(request.session['visit_count']))
```

Test session

- `$ python manage.py runserver`
- Navigate to http://127.0.0.1:8000/hero/
  - Refresh the page, and observe the counter increments

More information

- Middleware
  - https://docs.djangoproject.com/en/4.1/ref/middleware/
- Using Session
  - https://docs.djangoproject.com/en/4.1/topics/http/sessions/
- Session Serializer
  - https://docs.djangoproject.com/en/4.1/ref/settings/#std:setting-SESSION_SERIALIZER
  - https://blog.scrt.ch/2018/08/24/remote-code-execution-on-a-facebook-server/

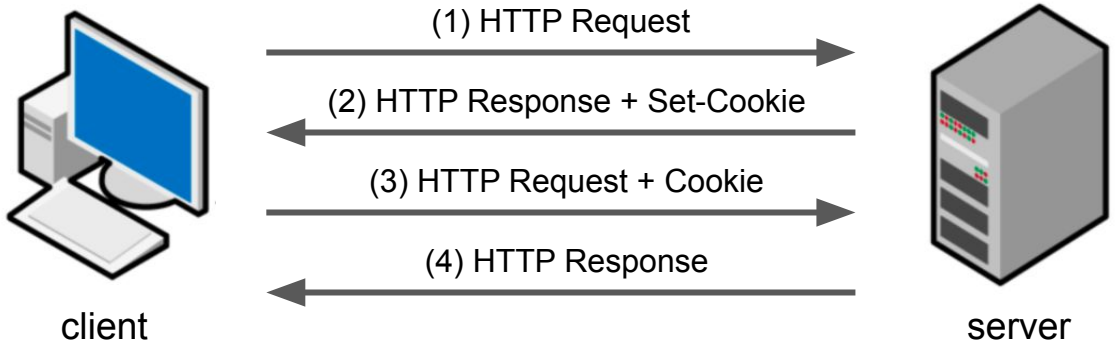
CSRF

CSRF

- Cross Site Request Forgery
- Also known as XSRF

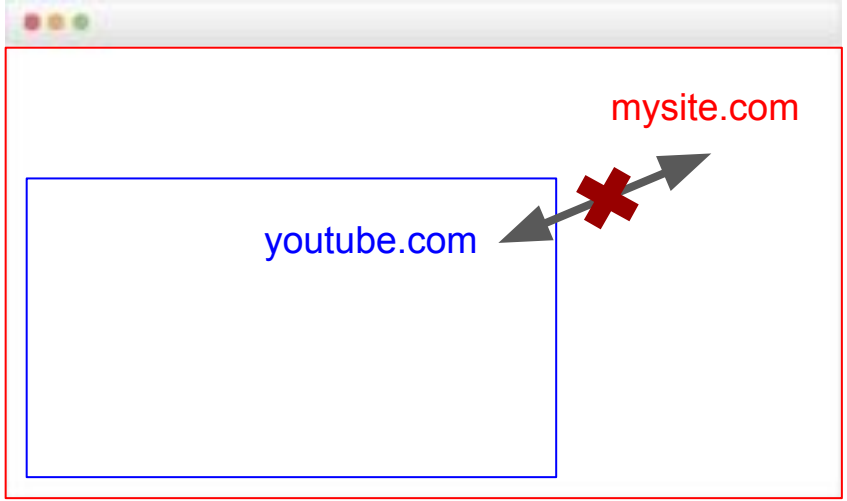
Remind the networking process

- When you login to a website, you get cookies which will be stored in your browser.
- When you send a request to the website again, the cookies in the browser are sent automatically.
  - → the website can verify your login.



Cross Domain Access Controls

- Let’s say you insert a Youtube video by using iframe.
  - `<iframe src=”..”> ... </iframe>`
- The cross iframe communication is impossible due to SOP
  - **S**ame **O**rigin **P**olicy



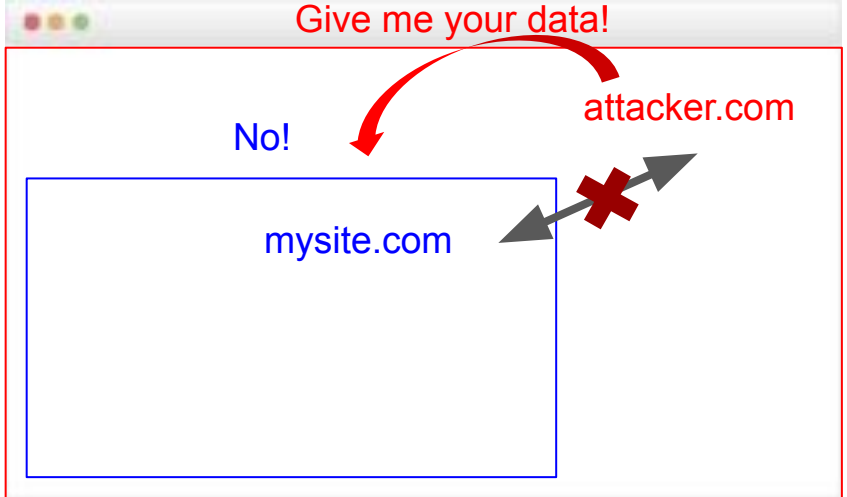
Same Origin Policy

- SOP is important because **cookies are sent automatically on every request.**
- The `attacker.com` cannot send malicious request to `mysite.com`.
- Any file from origin ‘A’ cannot handle any file from origin ‘B’.

protocol domain port

```
Website URL : http://mysite.com:80
iframed URL : http://mysite.com:80
```

Two origins are considered same if their protocol, domain, and port are identical. You can read cross domain data for same origins.



Same Origin Policy protection example

- Let's assume that the account is deleted by "`http://mysite.com/api/exit`" URL request.
- You logged in `http://mysite.com`.
- You access to `http://hacking.com`.
- `http://hacking.com` response HTML file with an Ajax request that try to delete your account.
  - `<script>$.get("http://mysite.com/api/exit");</script>`
- Thanks to SOP, the HTML file originated from `http://hacking.com`. cannot send request to other origin (`http://mystie.com/api/exit`)

Same Origin Policy exception

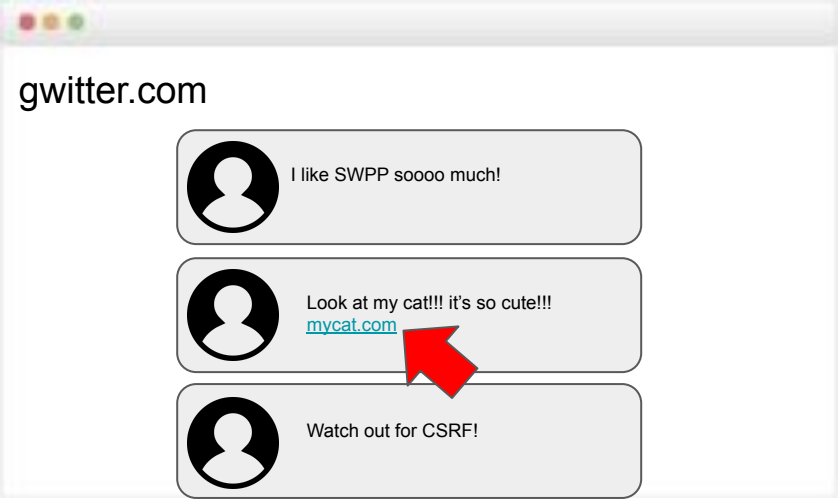
- If every resource from different origins is not accessible, it may be uncomfortable.
- Several exception of SOP
  - form submit (GET / ***POST***)
  - link click (GET request)
  - GET request for CSS, JS, iframe, image, media, etc.
- With GET request, side effects changing data are usually not allowed.
- Ajax request from the different origin is rejected by default.
  - Ajax: async Javascript communication method not to load the whole web page.

Loophole in SOP

- SOP allow form submit from the different origin.
- CSRF can occur!

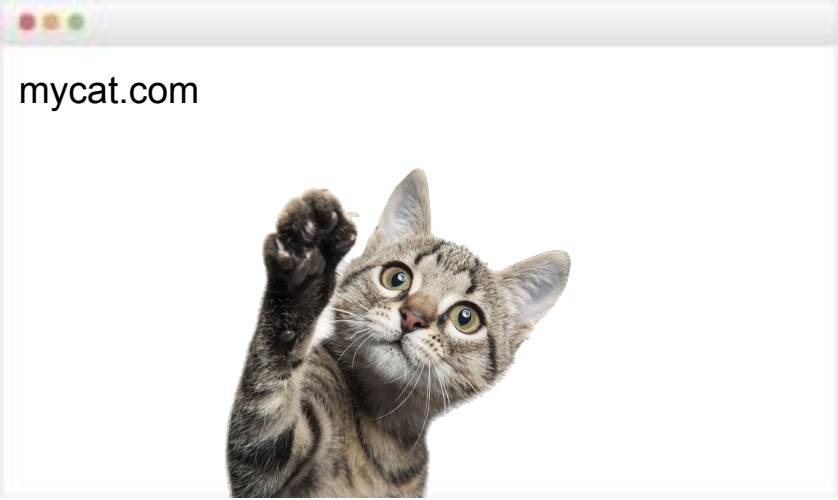
CSRF Scenario

- You sign in a normal website, which set your cookies
- You clicked a link which looks so cool.



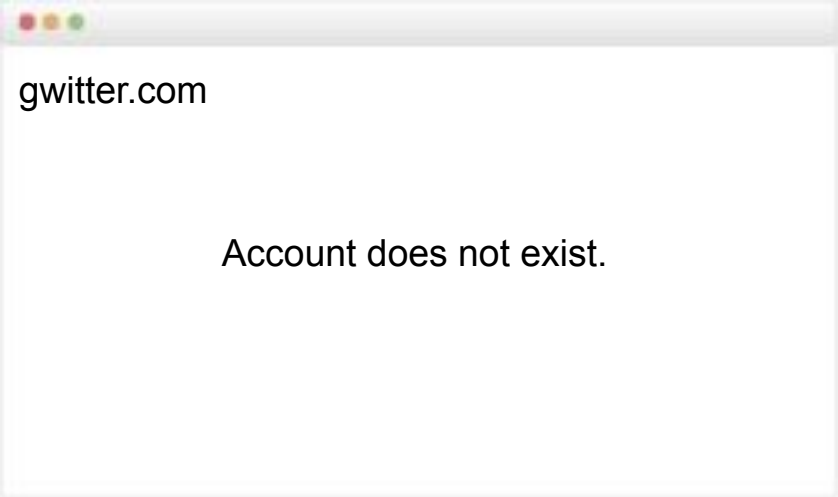
CSRF Scenario

- You may be satisfied with the linked site. But...



CSRF Scenario

- When you come back to `gwitter.com`, you will find out that your account is deleted.

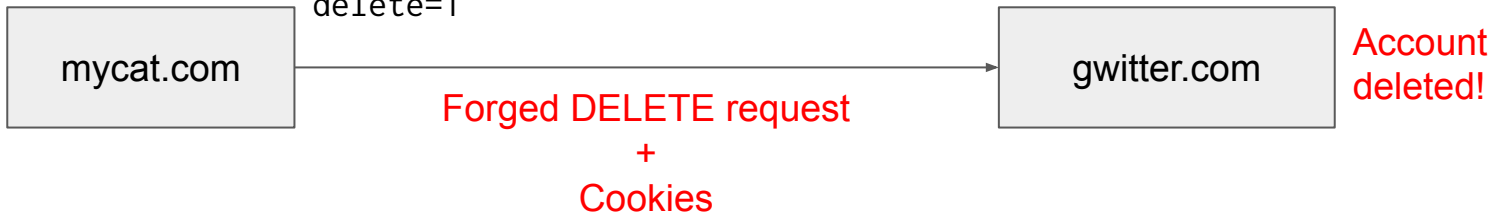


CSRF Scenario

- What happened?

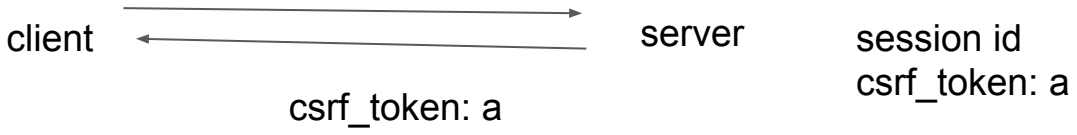
```
POST /delete_my_account HTTP/1.1
Host: gwitter.com
Content-Type:
application/x-www-form-urlencoded
Cookie: SessionID=d34dc0ssa

delete=1
```



How to prevent?

- Use CSRF token!
- Whenever server response to a request, it generates a new token.
  - Store it to session, and send it to client.
- Client send form(or request) with that CSRF token.
  - The token is embedded as a hidden property of the form.
- If server cannot find client’s CSRF token or the token from client does not match the token last issued by server, just reject the request.



Anti-CSRF token

- Anti-CSRF token (CSRF token) represents “I have an intention to submit this request”.
- Is it okay to send CSRF token via cookie?
  - YES! The attacker cannot read or change the cookies of other domain.

Django’s CSRF solution

- Django provides CSRF protection by default
  - Provide token by cookie, and check by request header
  - See this doc: https://docs.djangoproject.com/en/4.1/ref/csrf/
- The CSRF middleware is activated by default.
  - `django.middleware.csrf.CsrfViewMiddleware`
- If you use Django template, you can add the csrf_token inside the form.
  - `<form method="post">{% csrf_token %}`
  - The token is embedded as a hidden property of the form.

Ajax and CSRF token

- You can send Ajax POST request with setting X-CSRFToken header to the value of the CSRF token.

```
function getCookie(name) {
    let cookieValue = null;
    if (document.cookie && document.cookie !== '') {
        const cookies = document.cookie.split(';');
        for (let i = 0; i < cookies.length; i++) {
            const cookie = cookies[i].trim();
            if (cookie.substring(0, name.length + 1) ===
(name + '=')) {
                cookieValue =
decodeURIComponent(cookie.substring(name.length + 1));
                break;
            }
        }
    }
    return cookieValue;
}

const csrftoken = getCookie('csrftoken');
```

Ajax and CSRF token

- You can send Ajax POST request with setting X-CSRFToken header to the value of the CSRF token.

```
const request = new Request(
    ‘/api/something’, /* URL */
    {headers: {'X-CSRFToken': csrftoken}}
);

fetch(request, {
    method: 'POST',
    mode: 'same-origin'
}).then(function(response) {
    // ...

});
```

Don’t send csrftoken to another domain

CSRF exemption

- If you really want to disable it, use `csrf_exempt` decorator at view function (not recommended!)

```
from django.http import HttpResponse
from django.views.decorators.csrf import csrf_exempt

@csrf_exempt
def my_view(request):
    return HttpResponse('Hello world')
```

React side settings

- Set default CSRF cookie name and CSRF header name of axios.
- We will cover this in the next lecture.

```
// index.js
// ...
import { CSRF_TOKEN } from '../../../utils/csrf';
// …
axios.defaults.xsrfCookieName = "csrftoken";
axios.defaults.xsrfHeaderName = "X-CSRFTOKEN";
// …
```

More information

- CSRF
  - https://docs.djangoproject.com/en/4.1/ref/csrf/
- CORS
  - https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS
- Useful article
  - https://github.com/pillarjs/understanding-csrf

Unit Testing

Unit testing in Django

- Each apps has `tests.py` in their directory
- Based on Python 3 standard library: `unittest`
  - https://docs.python.org/3/library/unittest.html
- Use Django's `TestCase` class to get all advantages
  - e.g. Transaction isolation and database flushing

How to use

- Just subclass the `TestCase` class and write tests in tests.py

hero/tests.py

```
1  from django.test import TestCase
2  from .models import Hero
3
4
5  class HeroTestCase(TestCase):
6      def setUp(self):
7          Hero.objects.create(name='Superman')
8          Hero.objects.create(name='Batman')
9          Hero.objects.create(name='Ironman')
10
11     def test_hero_count(self):
12         self.assertEqual(Hero.objects.all().count(), 3)
```

TestCase

- `setUp()` and `tearDown()` method allows you to write instructions which will executed before and after *each* test.
- Write test case with method name starts with `‘test_’`
  - Django will run only these methods
- Django will create isolated database which only used by test cases and destroy it in the end of the test.

Run test

- `$ python manage.py test`

```
~/swpp2020-django-practice/toh / test !5
❯ python manage.py test
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.
----------------------------------------------------------------------
Ran 1 test in 0.002s

OK
Destroying test database for alias 'default'...
```

To measure coverage

- `$ pip install coverage`
- `$ coverage run --source='.' manage.py test hero`
- `$ coverage report`
  (`coverage report -m`) to see uncovered lines

```
~/swpp2020-django-practice/toh  test !5
❯ coverage report
Name                                          Stmts   Miss  Cover
-----------------------------------------------------------------
hero/__init__.py                                  0      0   100%
hero/admin.py                                     3      0   100%
hero/apps.py                                      3      0   100%
hero/migrations/0001_initial.py                   5      0   100%
hero/migrations/0002_hero_age.py                  4      0   100%
hero/migrations/0003_auto_20201019_1419.py        5      0   100%
hero/migrations/__init__.py                       0      0   100%
hero/models.py                                   15      3    80%
hero/tests.py                                     9      0   100%
hero/urls.py                                      3      0   100%
hero/views.py                                    49     38    22%
manage.py                                        12      2    83%
toh/__init__.py                                   0      0   100%
toh/asgi.py                                       4      4     0%
toh/settings.py                                  18      0   100%
toh/urls.py                                       3      0   100%
toh/wsgi.py                                       4      4     0%
-----------------------------------------------------------------
TOTAL                                           137     51    63%
```

Testing HTTP

- Django provides `Client` to test HTTP.
  - Simulate HTTP GET and POST requests
  - Each `Client` has own context, also the session cookie
    - Yes, you can test with session context!

Use Client to test HTTP

- Import `Client` and write test

```
from django.test import TestCase, Client
```

```
def test_hero_id(self):
    client = Client()
    response = client.get('/hero/10/')

    self.assertEqual(response.status_code, 200)
    self.assertIn('10', response.content.decode())
```

More information

- Writing and running tests in Django
  - https://docs.djangoproject.com/en/4.1/topics/testing/overview/
- Testing tools
  - https://docs.djangoproject.com/en/4.1/topics/testing/tools/
- Python 3 unittest library
  - https://docs.python.org/3/library/unittest.html

Today’s Task

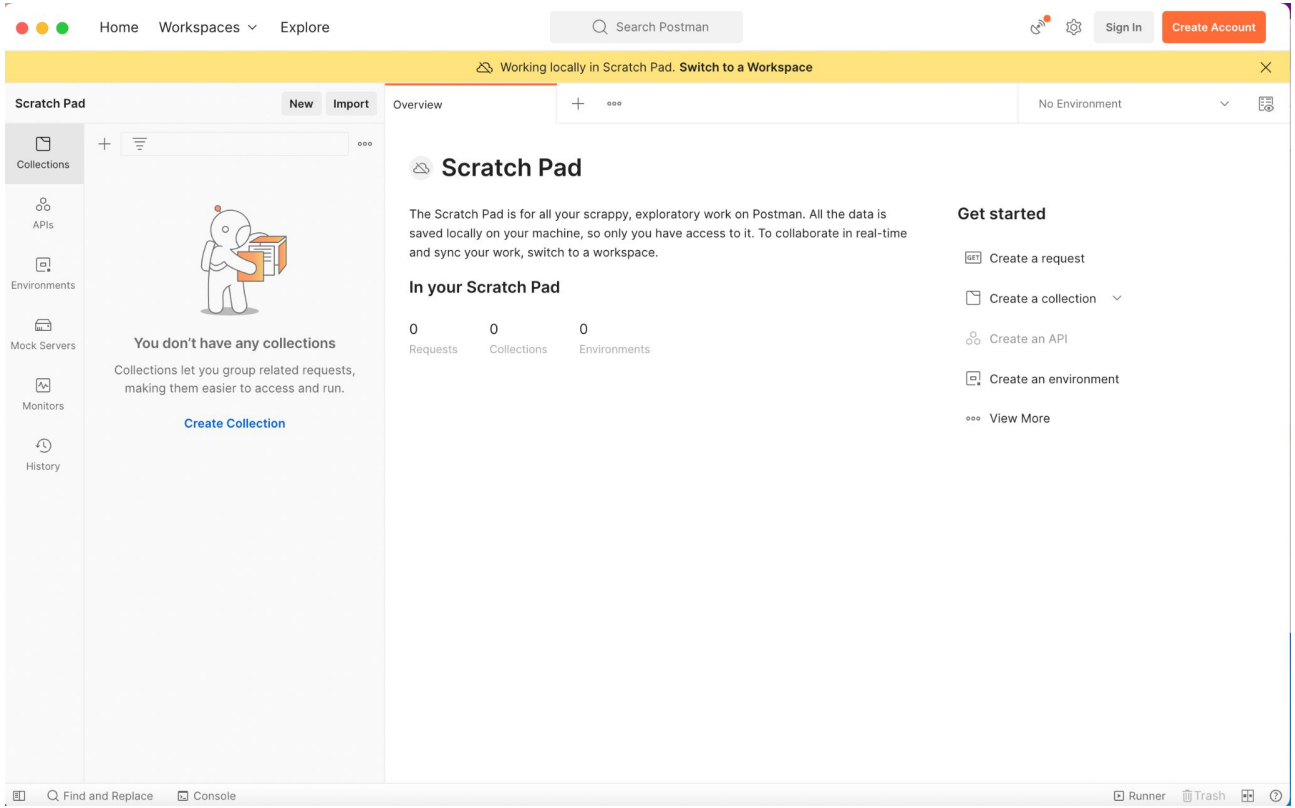
- Write a test for your `‘/hero/’` path
  - Let’s test the session-based visit counter
- You should test:
  - The response body of the first request contains ‘1’
  - The response body of the second request contains ‘2’
  - Hint: use `self.assertIn` to assert
- Send us an email with the following files.
  - screenshot of test result
    - the result of: `$ python manage.py test`
  - Python test code inside `hero/test.py`

Postman

Postman

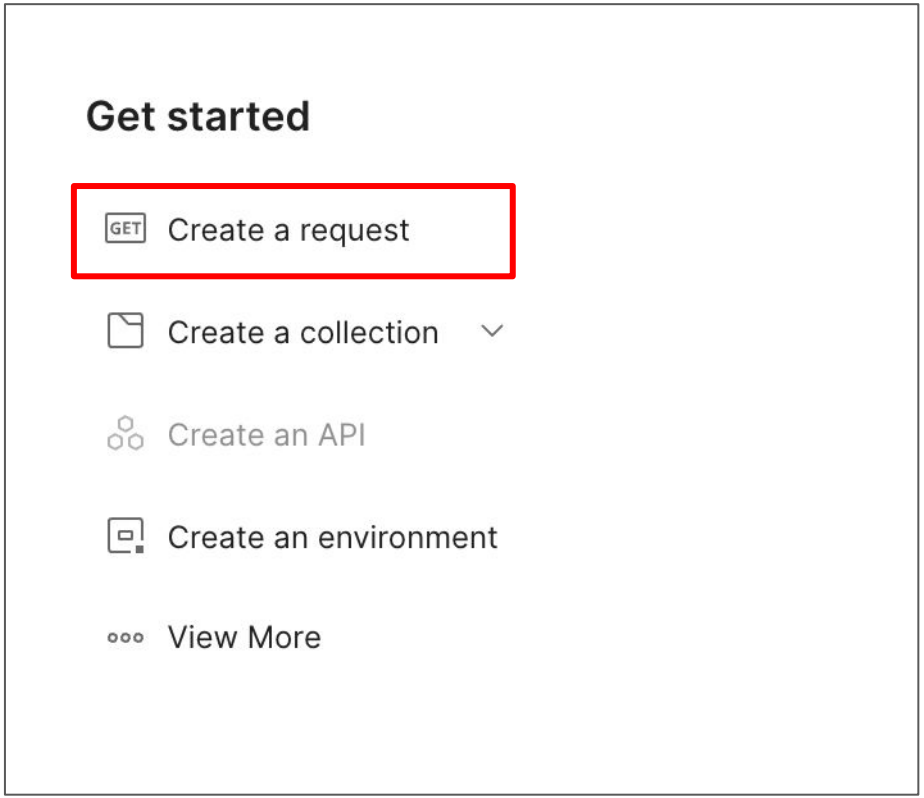
- API test GUI platform
- Download link
  - https://www.postman.com/downloads/
- Or you can use it as a Chrome extension.
  - https://chrome.google.com/webstore/detail/postman/fhbjgbiflinjbdggehcddcbncdddomop?hl=ko
- This tool will be useful for HW4 and your team project.

Postman


Home
Workspaces
Explore
Search Postman
Sign In
Create Account
Working locally in Scratch Pad. Switch to a Workspace
Scratch Pad
New
Import
Overview
No Environment
Collections
APIs
Environments
Mock Servers
Monitors
History
You don't have any collections
Collections let you group related requests, making them easier to access and run.
Create Collection
Scratch Pad
The Scratch Pad is for all your scrappy, exploratory work on Postman. All the data is saved locally on your machine, so only you have access to it. To collaborate in real-time and sync your work, switch to a workspace.
In your Scratch Pad
0
Requests
0
Collections
0
Environments
Get started
Create a request
Create a collection
Create an API
Create an environment
View More
Find and Replace
Console
Runner
Trash

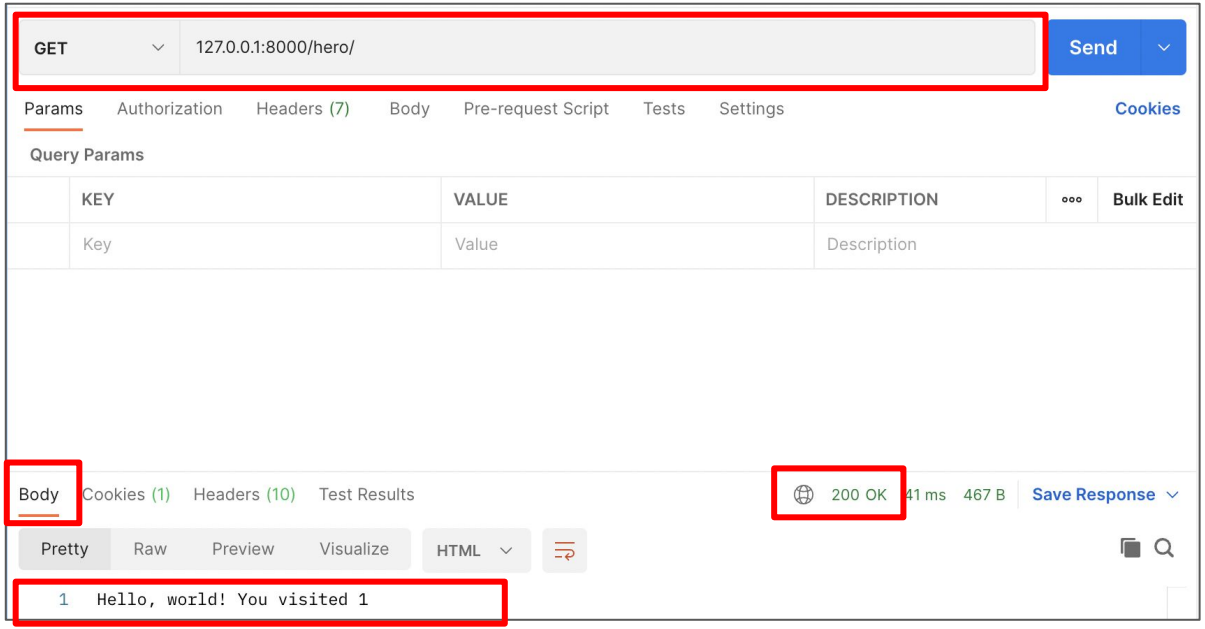
Simple test

- Create a new tab to test a single HTTP request.



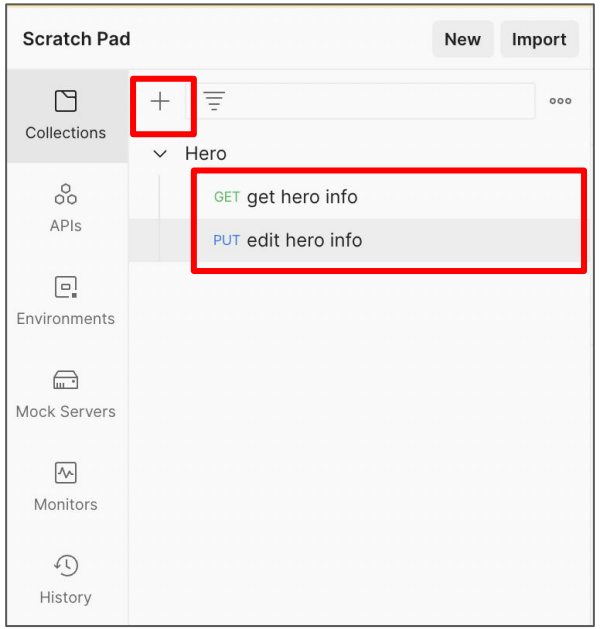
Simple test

- Send a GET request to `http://127.0.0.1:8000/hero/`
- You can check response body, header, and cookies.



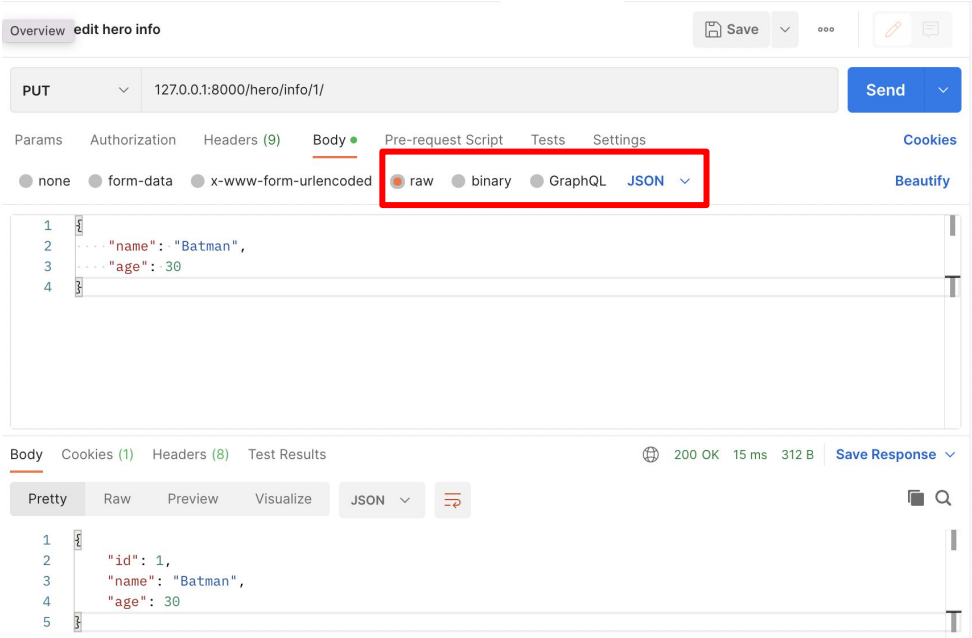
Test collection

- Click “New Collection”
- Under a collection folder, you can add multiple request tests.



Test collection

- Test hero info edit.
- PUT request to `127.0.0.1:8000/hero/info/1`
- You can select data format.



POST request with CSRF token

- Delete `@csrf_exempt` from `hero_list` function.

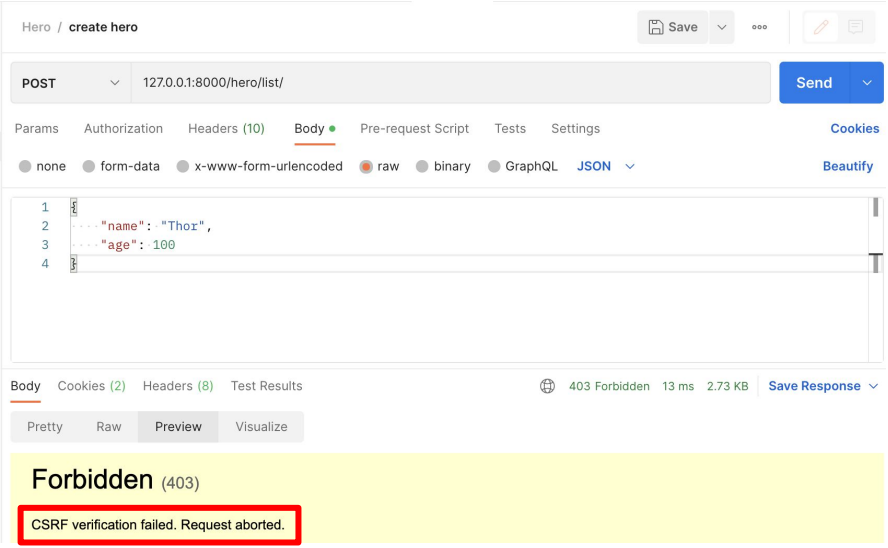
Delete this!

```
@csrf_exempt
def hero_list(request):
    if request.method == 'GET':
        hero_all_list = [hero for hero in Hero.objects.all().values()]
        return JsonResponse(hero_all_list, safe=False)

    elif request.method == 'POST':
```

POST request with CSRF token

- CSRF verification error occurs when you send a POST request.
  - `127.0.0.1:8000/hero/list/`



POST request with CSRF token

- Import `ensure_csrf_cookie`
  - `from django.views.decorators.csrf import ensure_csrf_cookie`
- Add a new view function to get a CSRF token.

```
@ensure_csrf_cookie
def token(request):
    if request.method == 'GET':
        return HttpResponse(status=204)
    else:
        return HttpResponseNotAllowed(['GET'])
```

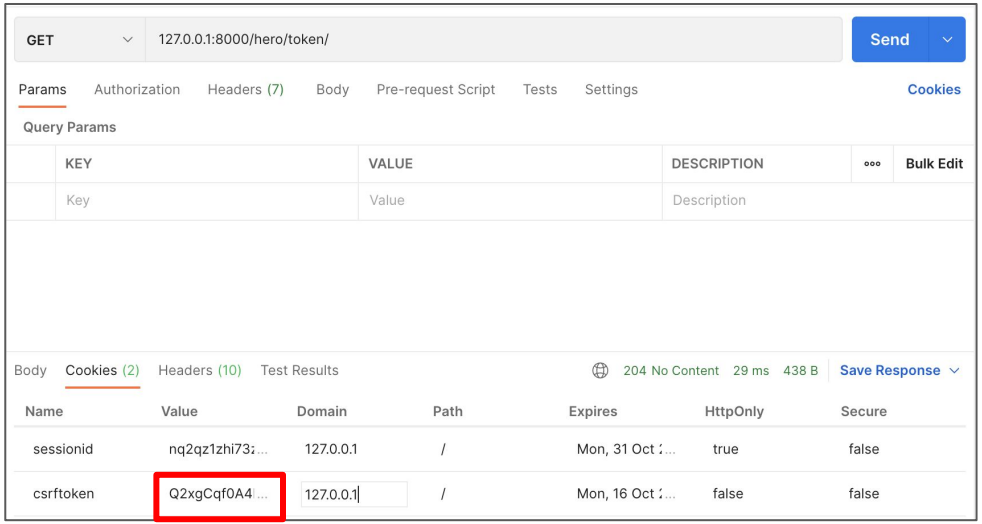
POST request with CSRF token

- Add a new url to get a CSRF token.

```
urlpatterns = [
    path('<int:id>/', views.hero_id, name='hero_id'),
    path('info/', views.hero_list, name='hero_list'),
    path('info/<int:id>/', views.hero_info, name='hero_info'),
    path('token/', views.token, name='token'),
]
```

POST request with CSRF token

- New send a GET request to `127.0.0.1:8000/hero/token/`
- Then, you will find two cookies at “Cookies” tab.
  - ‘sessionid’ and ‘csrftoken’
- Copy the value of ‘csrftoken’.



POST request with CSRF token

- Add a new header for CSRF token.
  - key: X-CSRFTOKEN
  - value: your CSRF token

Hero / create hero

Save

POST 127.0.0.1:8000/hero/list/ Send

Params Authorization Headers (10) Body Pre-request Script Tests Settings Cookies

Headers 9 hidden

KEY	VALUE	DESCRIPTIO
X-CSRFTOKEN	Q2xgCqf0A4Fj81cFPs0dzxSGH6JoWuvU	
Key	Value	Description

Bulk Edit Presets

Body Cookies (2) Headers (8) Test Results 201 Created 8 ms 304 B Save Response

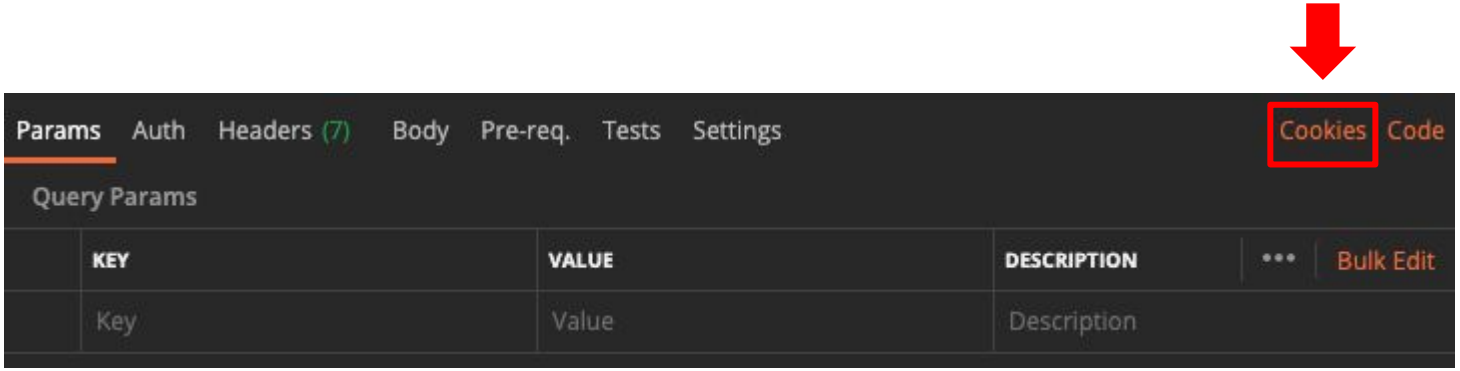
Pretty Raw Preview Visualize

{"id": 8, "name": "Thor"}

Now it works!

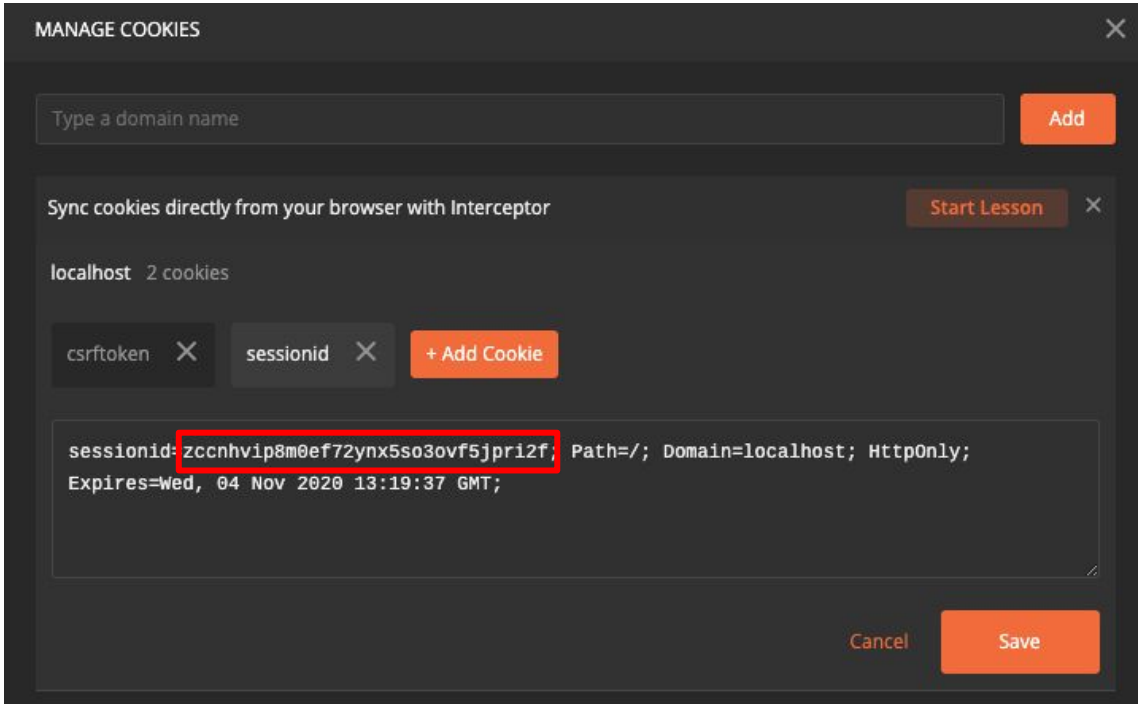
Session ID

- You can also send a request with a session id.
- Click “Cookies” button.



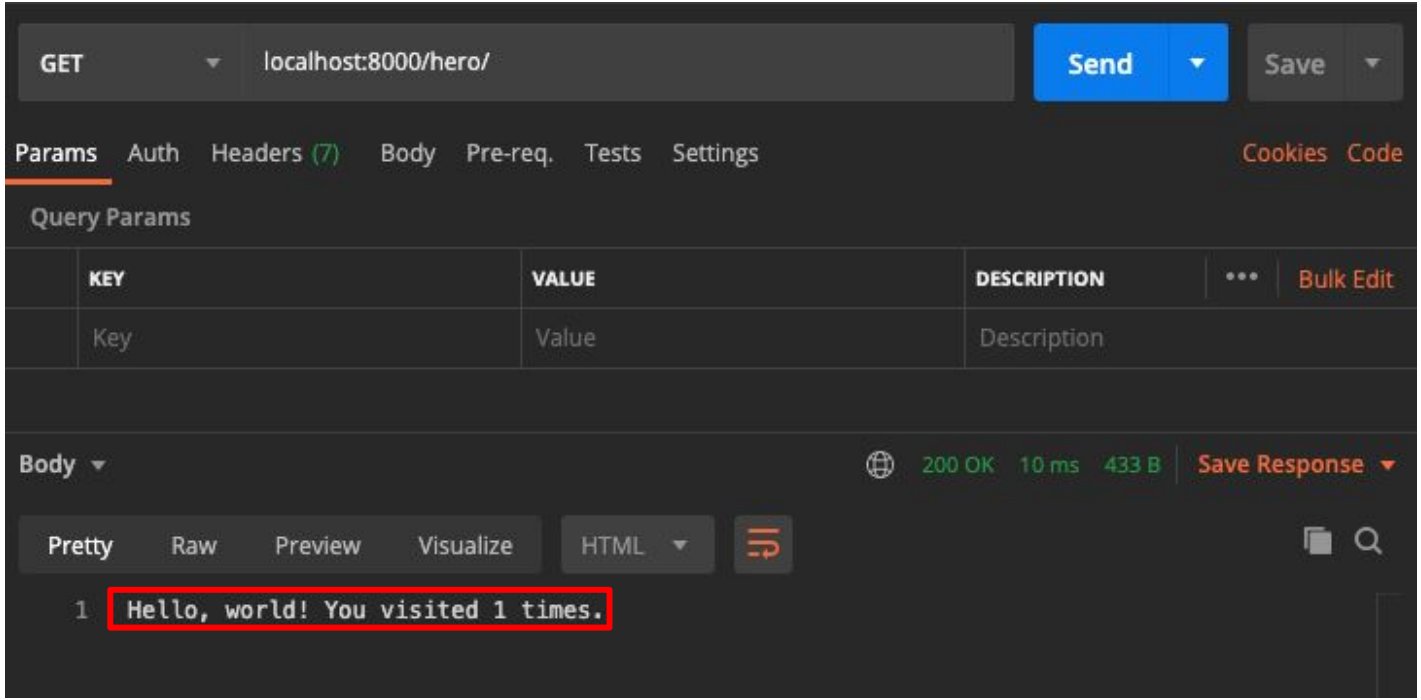
Session ID

- Click “sessionid” and change the value of it.



Session ID

- Now the visit count starts from 1.



Any Question?