All of the following are established on the front Flask port 5000 and the front and rear ends of the back django port 8000 are separated

The summary of my study is not completely correct. Welcome to comment and exchange.

Body---------------------------------------------------------------

Cross domain Restful

The django cors headers library needs to be installed

Configure CORS

·Add corsheaders to installed apps

·Add corsheaders.Middleware.middleware to the front as far as possible

·cors\_ origin\_ allow\_ All If it is true, the white list is not enabled

·cors\_ origin\_ Whitelist=[] white list, list of domain names allowed to access

·cors\_ allow\_ Methods Allowed Requests

·cors\_ allow\_ Headers=() allowed request headers, a tuple

·cors\_ preflight\_ max\_ Age defaults to 86400s

·cors\_ expose\_ Headers [] is empty by default

·cors\_ allow\_ The credentials boolean value is false by default

The above three defaults can be left unchanged (the above configuration items are all capitalized)

CORS: The back-end 8000 prepares an api for the front-end 5000 interface. The front-end uses ajax to send requests to the back-end. When transferring parameters to the back-end, it should also use the json string, enter the url of the back-end django, and then give the json data. The front-end uses json to obtain data for template editing

Restful design rule: industry mainstream api

When browsing online, we get resources. We get a specific resource according to each url

Each resource has its own form of expression -- the presentation layer

For example, if you want to see a web page, you can see html, xml, pictures, jpg

After we access a url, the status in the url will change. The client and server will interact by using post, put, get, patch, and delete

Design principles

1. Protocol, http and https

2. Domain name:

The domain name reflects the words api, such as http://api.example.com or https://example.org/api/ The second one is better. The first one should be set when buying a domain name

3. Version:

https://api.example.com/v1/

The meaning of version. When a website has different versions for three clients, these clients cannot use one version, so that each client can access the version corresponding to its own client

4. Path

Use nouns as much as possible https://api.example.com/v1/users , picture...

5. Meaning of HTTP verbs

Get: Take out resources

Post: Create a new resource -- create

Put: Update resources on the server -- update

Patch: Update resources in the server -- update

Delete: Delete resources from the server -- usually pseudo delete

Purpose --- reduce the consumption of front-end and back-end communication

·Example:

GET/zoos: List all zoos

POST/zoo: Create a new zoo

GET/zoo/ID: get the designated zoo information

PUT/zoo/ID: Update the information of a zoo (all)

PATCH/zoo/ID: Update the information of a zoo (partial)

DELETE/zoo/ID: Delete a specific zoo

GET zoos/ID/workers: obtain employee information of a specific zoo

------------------------

It is clear at a glance that the access of verb+noun can tell what the front end wants at a glance for the back end. It is OK to return json according to the API design view

------------------------

·Skillfully using query characters

? limit=100:

? orderby=age...

·Status code:

200 -- Ok GET returns data successfully

201 -- Created POST/PUT/PATCH created/updated data successfully

202 -- Accepted [\*]

Do as the romans do.

·Customize the internal code to correspond

More flexible

{'code':200, 'data':{}, 'error':}

·The backend's response to different apis should also comply with the rules

GET/users: object array

GET/users/666: Return a single resource object

POST/users: return the newly generated resource object

PUT/users/666: Return complete resource objects

PATCH/users/666: Return complete resource objects

DELETE/users/666: returns an empty document

Django backend

#FBV function-based-views

def view(request):

This is FBV

#CBV class

class CBV(View):

def get(self,request):

def post(self,request):

The route binding is different from that of fbv. The parent class of cbv, namely as in View (written by django himself), should be used\_ The view() method returns the whole function of a class, rather than simply binding a function in a file

------------This effect---------

All methods will be called according to the method name defined by the method in cbv

\*\*\*

For all ajax requests, their information will be stored in the request. body, including the request header and parameters. The request. body is used in the views instead of the original request. GET/request. POST. get [key]

Ajax request: including the request part, the parameter part, and the corresponding processing of the back end after the response is returned

Ajax is sent to the back-end api by the browser, not the front-end

Write tokens in login to replace cookies and sessions

Jwt -- Check the login session state

Previous session persistence, cookies and sessions

Cookie browser, session server database

Neither of them is a perfect solution. For cookies, although the server does not have so much storage pressure, the cookies will be modified (malicious); Session is a big problem for traversal and memory after a large number of users

The best solution: it is still best to put it on the browser. Then think about whether the stored things can be obtained by others --->use token to store data that only the server can understand and others cannot understand. So use jwt

Jwt is not clear text. Use base64 to operate

(In fact, there is still a problem. People who know it can decode base64, but ordinary netizens really can't understand it)

Base64 is the same as hashlib.md5.update. The byte string b 'string' is required

import base64

Ciphertext: str=base64. b64encode (b'I am Fu ')

urlsafe\_ B64encode has the same effect as b64encode, but will change "+" ->'-', '/' ->'\_ '

Clear text: decoded\_ str = base64.b64decode(str)

urlsafe\_ B64decode is the same as b64decode

The urlsafe uses the urlsafe's decode for what encoding pair applies

HMAC-SHA256

h = hmac.new(key, str, digestmod='SHA256')

h. Digest () verifies the information source and whether it is tampered. The key is only known by the user, so the string written by base64 cannot be detected again

jwt json-web-token

jwt is used to pass authenticated user information between identity providers and servers for easy access to resources

Jwt consists of three layers (separated by two.)

·Part I Header

Is the dictionary converted to json string and base64 transcoding

{Algorithm 'alg': 'HS256', token type 'typ': 'JWT'}

·The second part is payload

Public statement: Jwt provides built-in keywords to describe common problems. This part is not mandatory. Users can add keys according to their own needs

Common public statements are as follows: (optional as required, but iat and aud will directly affect the decode results)

{'exp ': xxx # Expiration Time

'iss': xxx # Issuer indicates the Issuer of this token. It is not useful because it is often processed by multiple machines

'iat': xxx # Issued At The time when this token was created

'aud': xxx # Audience is useful for recording users' devices. After being stolen, if a token is sent from another device, it can be prevented

}

The above statement must be complied with

··Private statement: users can add their own keys if necessary

For example: {'username ':'RageFu'}

·Part III: Signature

The algorithm determined according to the alg in the header, and then transcoding

HS256 (user-defined key, header after base64+'.'+payload after base64)

Test jwt:

1. Parse the header and confirm the alg

2. Signature verification: sign according to the transmitted header and payload according to the algorithm specified by alg, and compare the signature result with the transmitted sign. If the comparison is consistent, the verification passes

3. Get payload custom content

Python comes with jwt pyjwt

pip install pyjwt

Pyjwt's method encode (payload, key, algorithm)

Payload is the payload in the jwt definition. Add public and private payloads according to the specification, and return Bytes

Key: user-defined encryption key, str

Algorithm, the encryption algorithm to be used, such as HS256, str

Use jwt:

s= jwt.encode({'username':'Fu'}, 'fufufu', algorithm='HS256')

Verification: use decode

Decode (token, key): token: token string, key: user-defined encryption key, which is the same as that in encode

Issuer: If the iss field is added to the encode payload, the field can be verified. If the iss verification error occurs, a jwt.InvalidIssuerError is thrown

Audience: the signed audience, the same as the issuer. If the verification fails, jwt.InvalidAudienceError will be thrown

---If all the above are passed, the parameters in payload will be returned. If they fail, an error will be reported directly

What is the whole process? Payload is the parameter uploaded to the cookie when the user logs in. It is protected when logging in. Encode HS256 uses the hash algorithm to create a signature through the header and payload. The signature is the most important verification part. The header, payload and special keys make the signature impossible to be cracked; When it is necessary to check the login status in the cookie, the requested signature will be compared with the saved signature code. If it passes, the user's payload will be returned. Otherwise, an error will be reported and the view function will handle it.

Cookies are insecure and may be attacked by csrf cross site forgery, so local storage in the browser (middleware cannot be used to prevent csrf after the front and rear ends are separated) will not be automatically done by the browser. It can only be operated with js, which is very secure and can be stored for a long time. It is suitable for storing tokens

Js directly transmits the token to the backend using ajax

The front-end sends an ajax request to the back-end user name+password, and the back-end verifies the database operation. If it succeeds, it makes a token and returns it to the front-end. The front-end stores it in window.localstorage after receiving it

Django test jwt

Use method\_ Decorator

@method\_ Decorators (their own decorators) can be used in functions and methods, which are common

Decode the token in the authorization in the request header sent from http in the decorator (using the defined key). If it succeeds, 200 will be returned to the front end, and then the payload in this token will be sent to the view function. If it fails, the user-defined error code and error will be sent to the front end

————————————————

Copyright notice: This is an original article of the CSDN blogger "Rage\_Fu", following CC