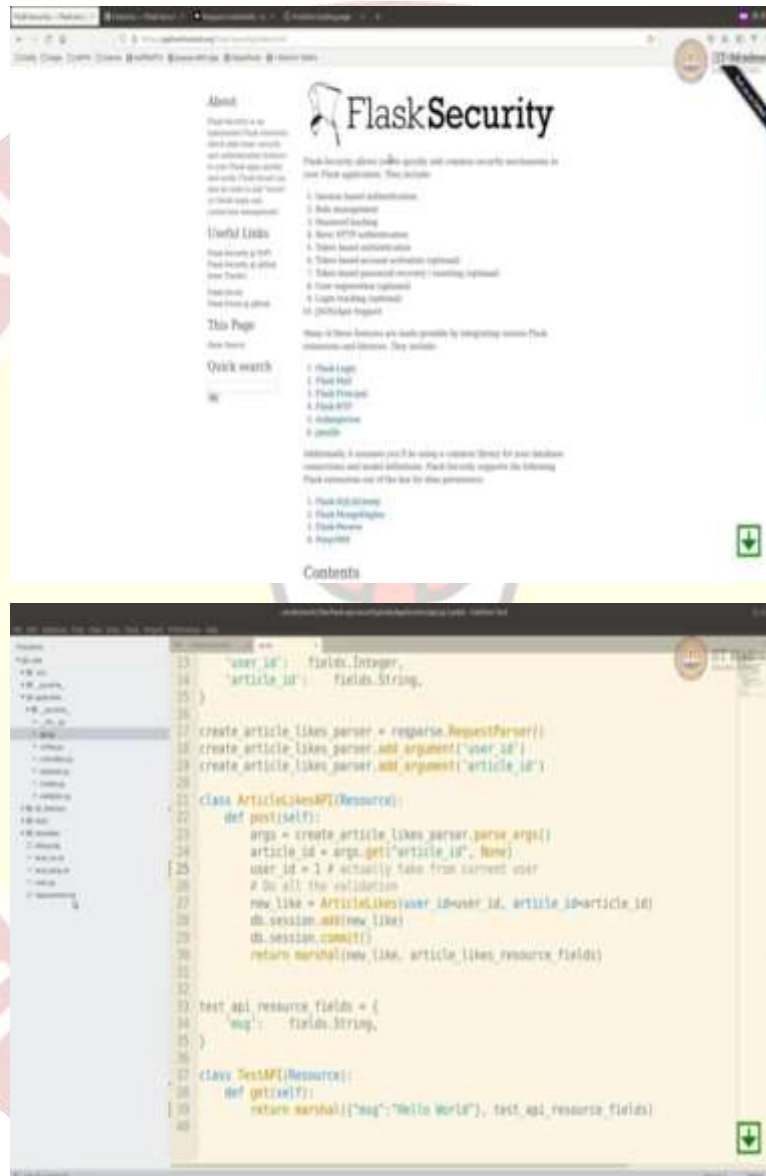




IIT Madras
ONLINE DEGREE

Modern Application Development – II
Professor Thejesh G N
Indian Institute of Technology, Madras
Flask API – Token Based Authentication

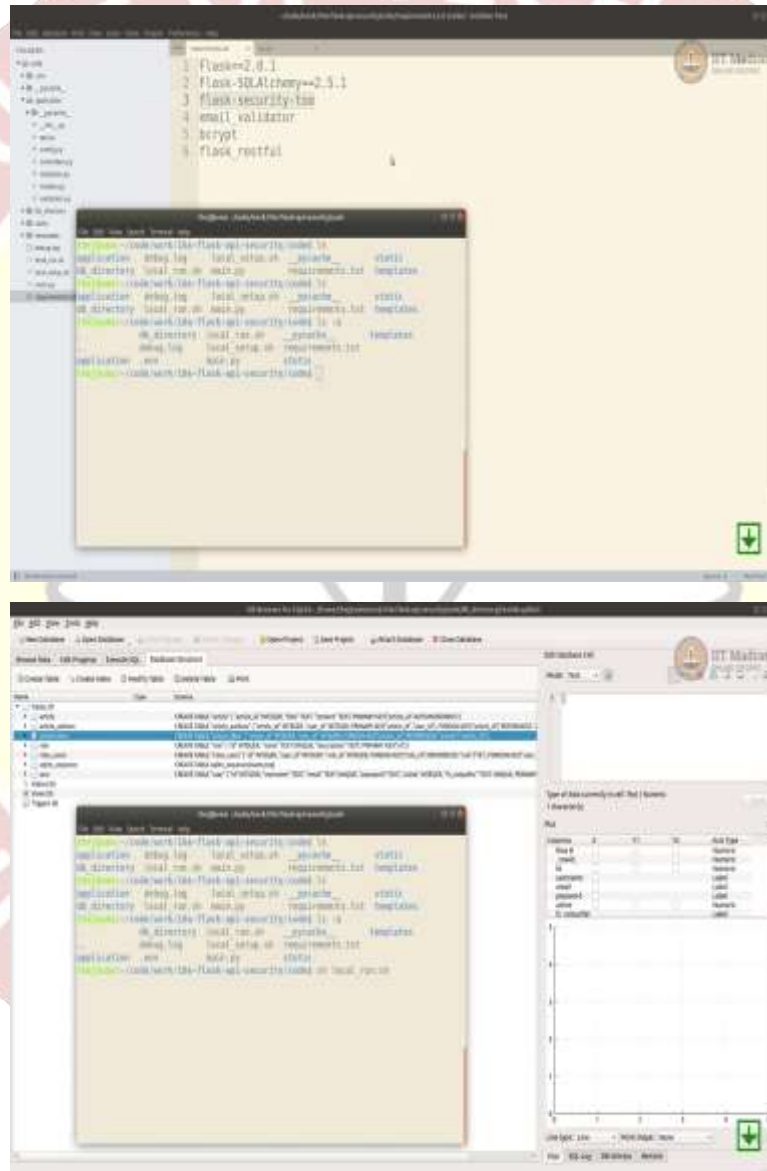
(Refer Slide Time: 00:15)



Welcome to the Modern Application Development two screencast. In this screencast will add token-based authentication to our APIs. So, it could be used in our AJAX applications or view based applications. For this, we need a terminal browser editor and Python installed. We will start with application that we had used before, I will do some changes to it.

But, that is where we will start with the same application API application along with the security. There is one change, we used to use a framework called or a package called flask security. Now, the flask security has an update that they are not maintaining it, and the same thing has been forked into flask security too. So, we are going to use this, just to be on the latest version and use the most updated version.

(Refer Slide Time: 01:18)



So, what I have done is in our requirements too, I have removed it removed the flask security and added flask security too here, the rest remains same. Virtually, the API is almost remain same you, so you do not have to make any changes in the code, it's a fork and improved version.

Unless you are using something that has breaking changes, you do not have to worry about it, you can continue to use it. So, I have already did the local install, I have already did pip install, flash security too, and remove the previous version as well, before installing the new version. You can do that by pip uninstall and give the packaging name here. I have already done that.

So, I am just going to start local run, your database is the same database that we are going to use a sqlite. I have added one table, I will explain why I have added that table later. But, you do not have to worry about it. So, let me start the application.

(Refer Slide Time: 02:40)

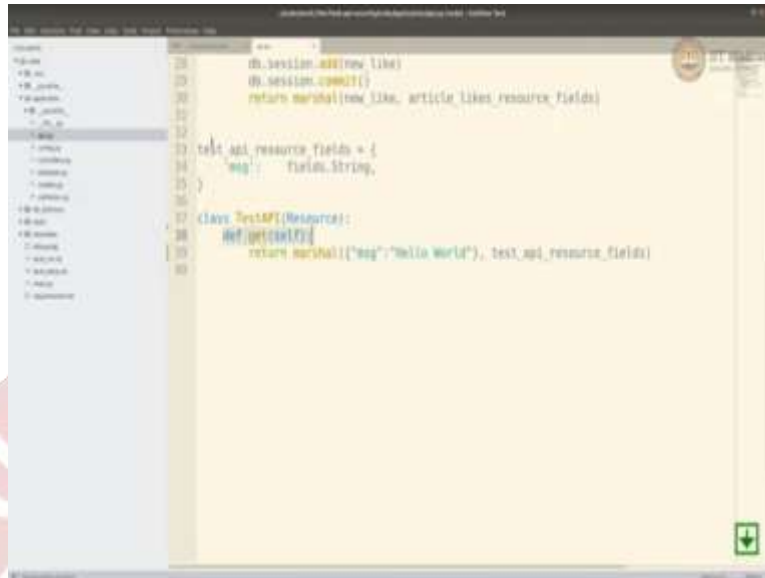


The image displays two screenshots of a code editor, likely Visual Studio Code, with a file explorer on the left. The top screenshot shows a shell script file named 'setup.sh' with the following content:

```
1 #!/bin/bash
2 echo "Welcome to the setup. This will setup the local virtual env."
3 echo "And then it will install all the required python libraries."
4 echo "You can rerun this without any issues."
5
6 echo "-----"
7 if [ -d ".env" ]; then
8     echo "Enabling virtual env"
9 else
10    echo "No Virtual env. Please run setup.sh first"
11    exit 0
12 fi
13
14 # Activate virtual env
15 . env/bin/activate
16 export ENV=development
17 python main.py
18 deactivate
```

The bottom screenshot shows a Python file named 'main.py' with the following content:

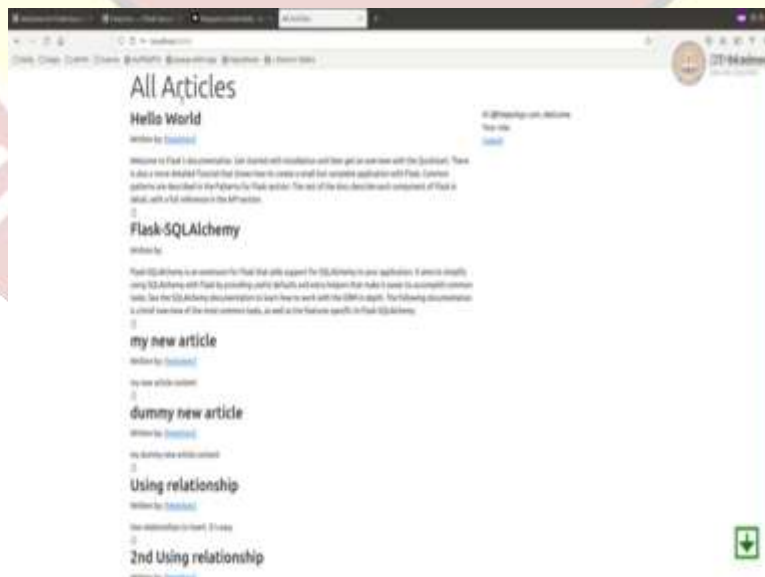
```
48 api.add_resource(ArticleResource, '/api/article likes', resource_class=ArticleResource)
49
50 from application.api import TestAPI
51 api.add_resource(TestAPI, '/api/test')
52
53 @app.errorhandler(404)
54 def page_not_found(e):
55     # note that we set the 404 status explicitly
56     return render_template("404.html"), 404
57
58 @app.errorhandler(403)
59 def not_authorized(e):
60     # note that we set the 403 status explicitly
61     return render_template("403.html"), 403
62
63 if __name__ == '__main__':
64     # Run the Flask app
65     app.run(host="0.0.0.0", port=8080)
```

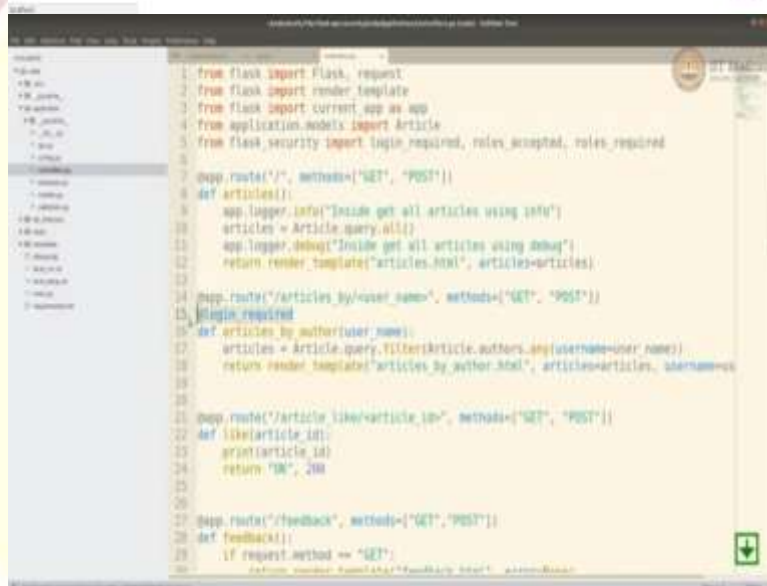


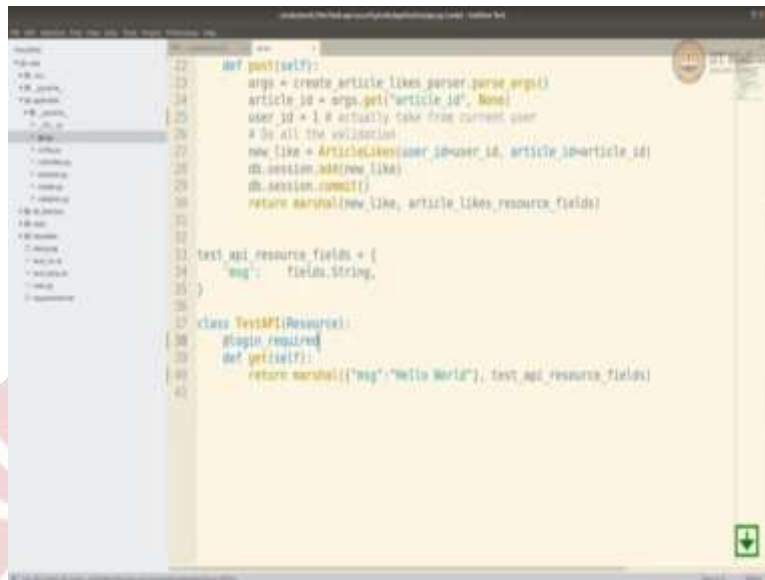
```
28 db.session.add(new_like)
29 db.session.commit()
30 return marshal(new_like, article_likes_resource_fields)
31
32
33 test_api_resource_fields = {
34     'msg': fields.String,
35 }
36
37 class TestAPI(Resource):
38     def get(self):
39         return marshal({'msg': 'Hello World'}, test_api_resource_fields)
40
```

So, it is the same script that we used to use before if you see, it is going to instance. Enable the virtual environment and then start running the main dot py. The main dot py, we are starting the server local device route 8080. And we will include the controllers in the main controllers, and also some APIs. I have one test API just for all our experimentation and testing. So, if you go to API and see test API, it does nothing, there is a get and it just marshals a message Hello World, and just written that that is it. So, now we have started the server, let us go to the browser and refresh this page.

(Refer Slide Time: 03:30)







```
22 def post(self):
23     args = create_article_likes_parser.parse_args()
24     article_id = args.get('article_id', None)
25     user_id = 1 # actually take from current user
26     # do all the validation
27     new_like = ArticleLike(user_id=user_id, article_id=article_id)
28     db.session.add(new_like)
29     db.session.commit()
30     return marshal(new_like, article_likes_resource_fields)
31
32 test_api_resource_fields = {
33     'msg': fields.String,
34 }
35
36 class TestAPI(Resource):
37     @login_required
38     def get(self):
39         return marshal({'msg': 'Hello World'}, test_api_resource_fields)
```

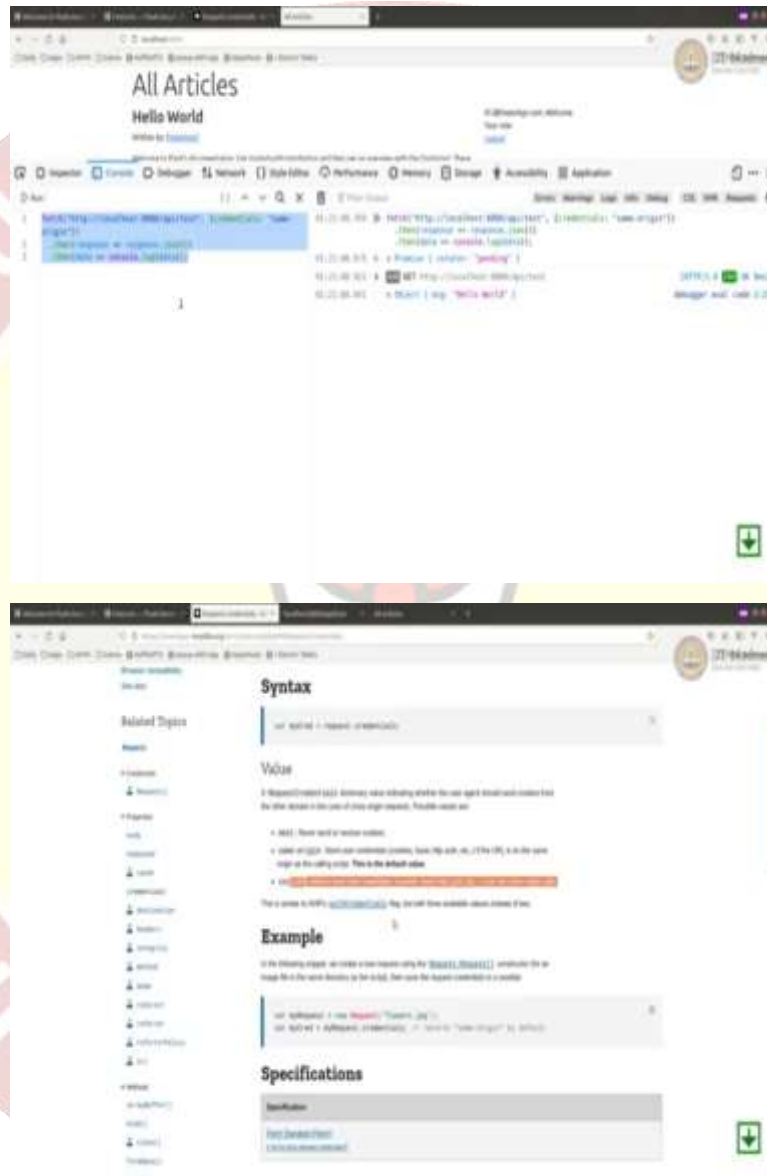
So, this is there and we used to use this right login page, and I could use login. So, you know, and then you could see all of the things that you could do with login. You used to achieve this, login enabled or disabled by adding this login required. Now, we can go ahead and add the same thing to our AJAX APIs too. Like for example, you can add it here, and let me say well it restarted, yes restarted. Now, let me log out first. And if I do slash API slash test, it will give me a login page.

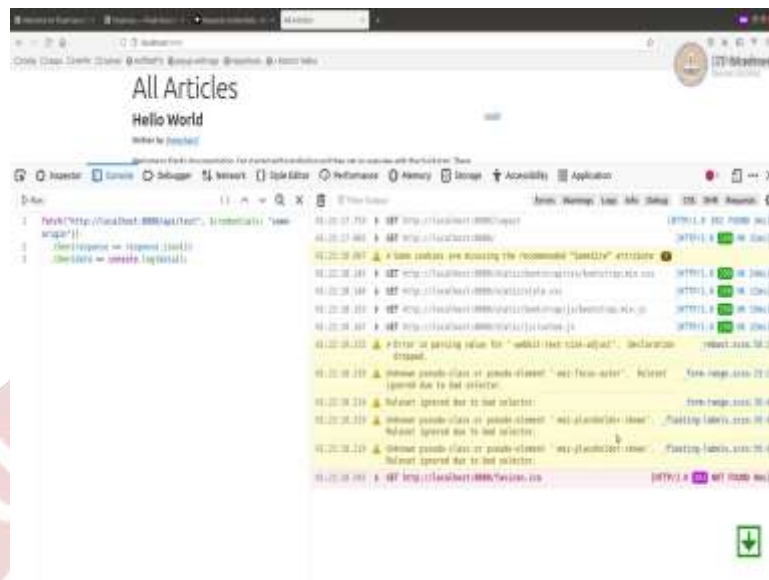
(Refer Slide Time: 04:23)



And dot com and enter the password. It is, you can see that once I login, I can make the calls to this path, without any problem, without any issues, because it uses the same cookie-based authorization or authentication.

(Refer Slide Time: 04:57)





Now, you can also do call a fetch request, you can make a fetch request, for example, using our fetch API. Just that you need to add the credential, same origin, this will send of the credentials to the same origin, same origin as in same domain, it will not to. If you are making a call to a different domain, it will not send any credential. But, if you are making a call to the same domain, from the same page and same domain, then it will send the credentials. We can see what does it mean exactly here, here. You can omit the credentials or never send or receive cookies when you are doing fetch.

But, if you do same origin, you will send cookies along with it, and then you can authorize or authenticate it. For you can also do cross origin, but we do not have to do cross origin here, because we are calling on the same same domain name, you know localhost 8080 here also it is local.

So, now if I run this, you can see that the call was made and the response was received. This is like a fetch call AJAX call. So, you could in all practical uses, you could use the same login for doing a good view application, making calls on F on, fetch. In all practical purposes, you could use the same cookie-based authentication to make fetch calls from your view application. You do not have to use anything new.

But that said, it is kind of weird, sometimes it goes to a login page and then it forwards it back, and then you make AJAX call. You have to make test calls to see whether you already have the cookie set. If not, you have to make a dummy call to make the cookie set all of that. You can avoid doing that by using token-based authentication, which means instead of cookies, we will use tokens in the header. Now, the same framework, flask security also supports token-based authentication here, token-based authentication.

So, we will use that instead of cookie-based authentication. So, what what it will do is you will login using username and password, and the system gives you a token. And you use this token to actually communicate with the server for the next few calls. Now, this has an advantage, because it does not have to be, it you you can set it in the header. You do not have to depend on that credentials, same origin kind of settings.

(Refer Slide Time: 08:05)

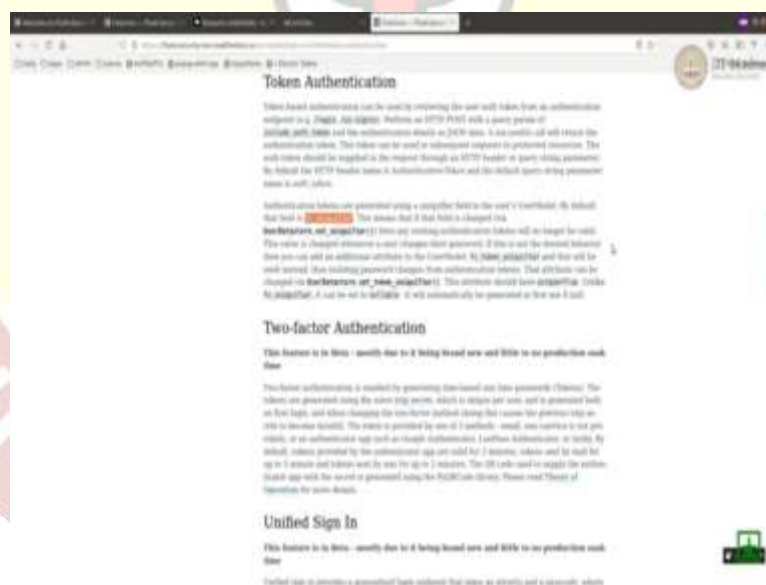
```
def post(self):
    args = create_article_likes_parser.parse_args()
    article_id = args.get('article_id', None)
    user_id = 1 # actually take from current user
    # do all the validation
    new_like = ArticleLikes(user_id=user_id, article_id=article_id)
    db.session.add(new_like)
    db.session.commit()
    return marshal(new_like, article_likes_resource_fields)

test_api_resource_fields = {
    'msg': fields.String,
}

class TestAPI(Resource):
    @login_required
    def get(self):
        return marshal({'msg': 'Hello World'}, test_api_resource_fields)
```

Let us do this. To do this, we will have to do a couple of changes in our code. Let me start with reading what it means to have a token-based authentication.

(Refer Slide Time: 08:19)

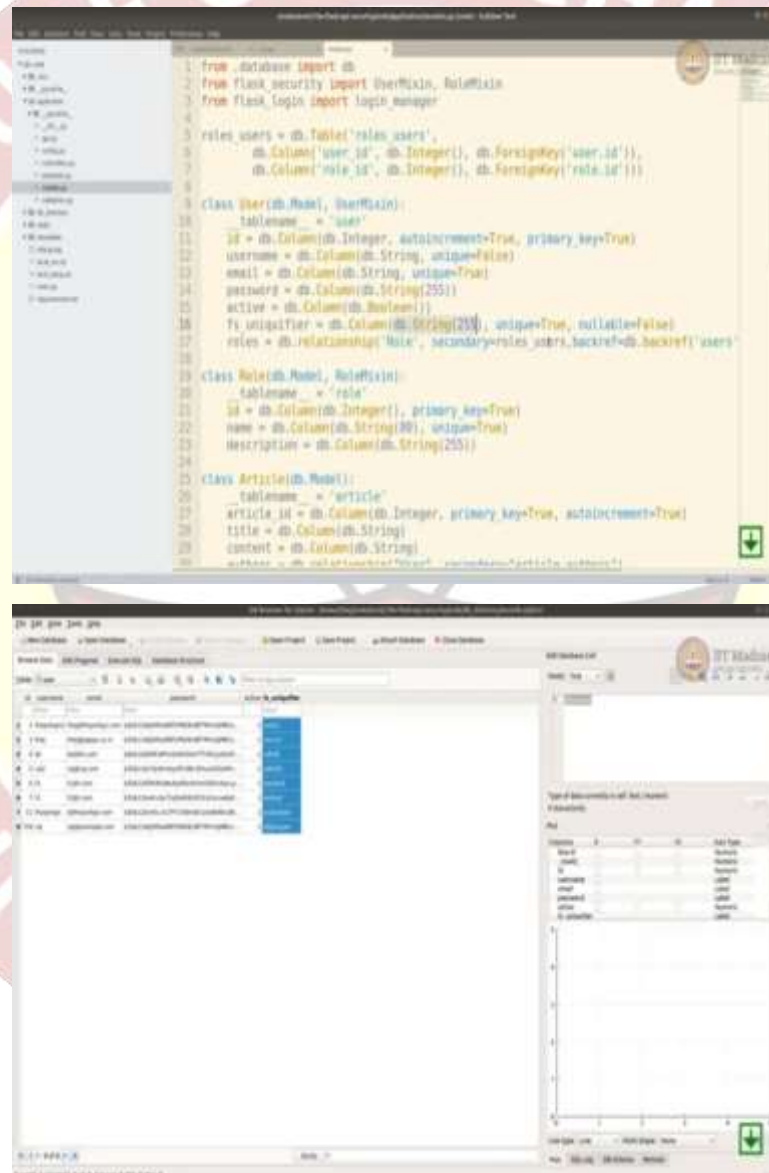


Here is the notes about it. Basically, you can make an AJAX call to the same end point, or JSON post to the same endpoint with the include auth token to get the token. And then you can use that token to send the authentication token in the header to actually authorize. Now, this requires one change to actually generate a token, the flask security uses a column called fs identifier in the

user table. This uses like us seed to generate the token. You can so in a way that if you change this, your token also changes.

So, anytime you want to reset the token, or get a new token, you can just reset this fs unique unquifier, and then you can change the token. That is one of the advantage, or you can call this you know, set token unquifier and then it will change the token.

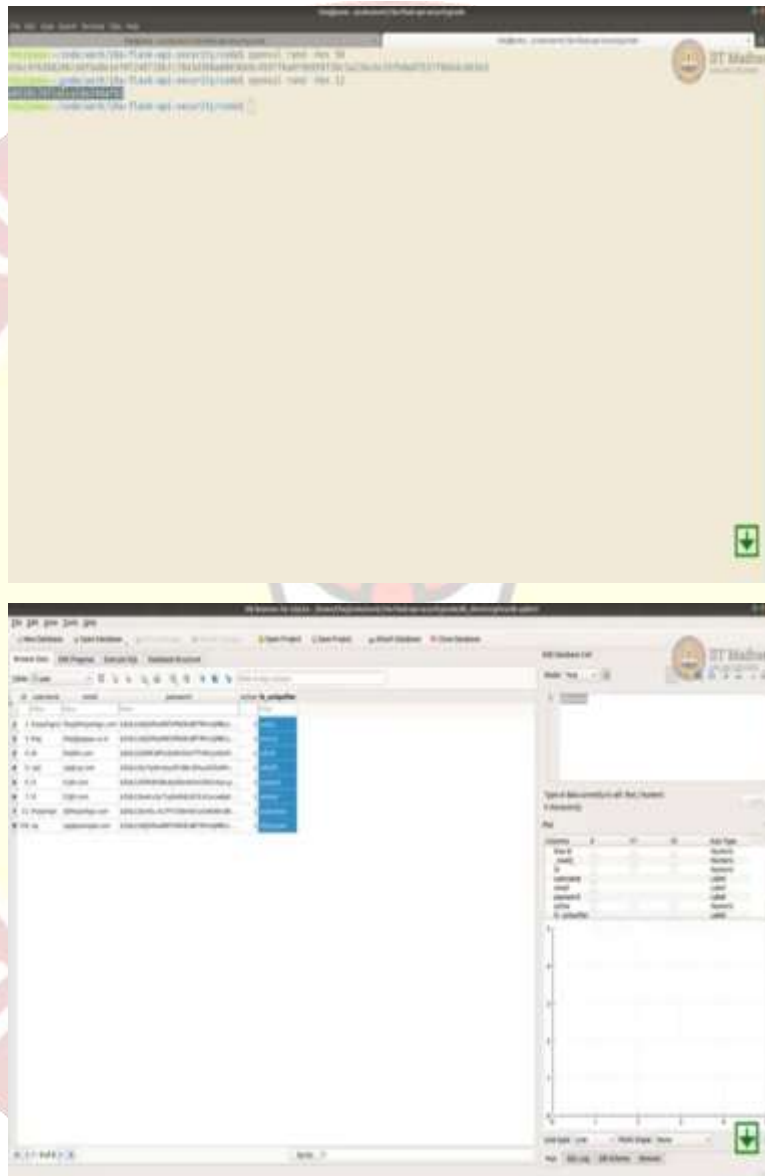
(Refer Slide Time: 09:27)

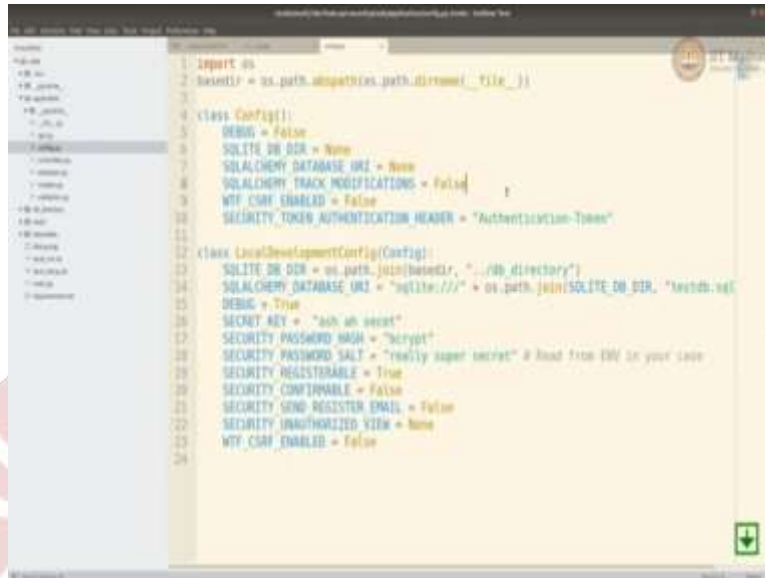


So to do this, will do go back to our models and add a new column called fs unquifier. I have already added it called db dot column. String of 255, it could be less or more, but I have just chosen pretty standard-length string. If you want to use a longer unique code and it has to be

unique across the all the users, so those if you see your table I have added some values, simple values here. But actually, in real world if I really want to set it up, I would use something like a random generator, like a good random generator to generate code.

(Refer Slide Time: 10:17)

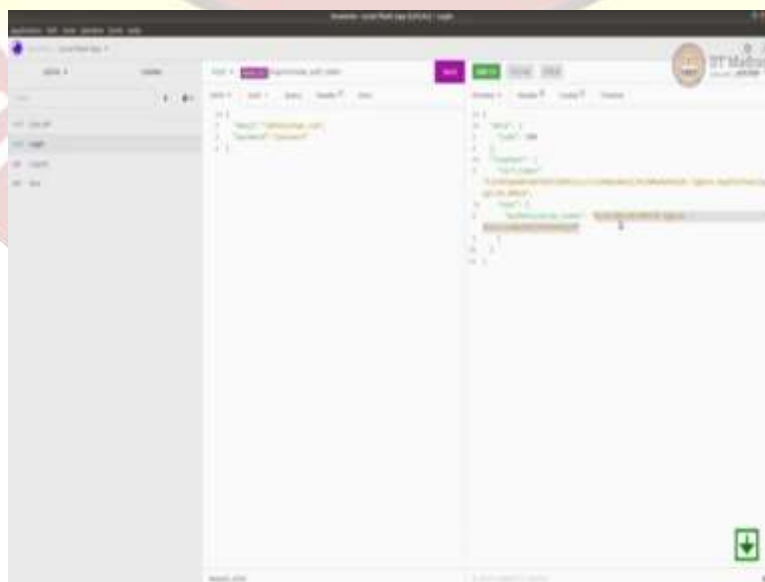




```
1 import os
2 base_dir = os.path.abspath(os.path.dirname(__file__))
3
4 class Config:
5     DEBUG = False
6     SOLITE_DB_DIR = None
7     SQLALCHEMY_DATABASE_URI = None
8     SQLALCHEMY_TRACK_MODIFICATIONS = False
9     WTF_CSRF_ENABLED = False
10     SECURITY_TOKEN_AUTHENTICATION_HEADER = "Authentication-Token"
11
12 class LocalDevelopmentConfig(Config):
13     SOLITE_DB_DIR = os.path.join(base_dir, "..", "db_directory")
14     SQLALCHEMY_DATABASE_URI = "sqlite:///{}".format(os.path.join(SOLITE_DB_DIR, "testdb.sql"))
15     DEBUG = True
16     SECRET_KEY = "ach an secret"
17     SECURITY_PASSWORD_HASH = "bcrypt"
18     SECURITY_PASSWORD_SALT = "really super secret" # Read from ENV in your code
19     SECURITY_REGISTERABLE = True
20     SECURITY_CONFIRMABLE = False
21     SECURITY_SEND_REGISTER_EMAIL = False
22     SECURITY_UNAUTHORIZED_VIEW = None
23     WTF_CSRF_ENABLED = False
24
```

Like, let us say 50 characters one, or hex 50, and then this is what I would use, or maybe it is just smaller, maybe around 12. I will use this rather than actually what I am using here, which are very simple, but anyway, I am using it here for testing, so it is fine. So, once you do that change this fs uniquifier, then you can go to your config. And then just check security token authentication header, by default it will be authentication token. If you want to change it to anything else, you can you can make. Like for example, instead of authentication token, you want to use just contiguous auth token or something else, then you can set that here. I have currently disabled the CSRF, but you can enable it.

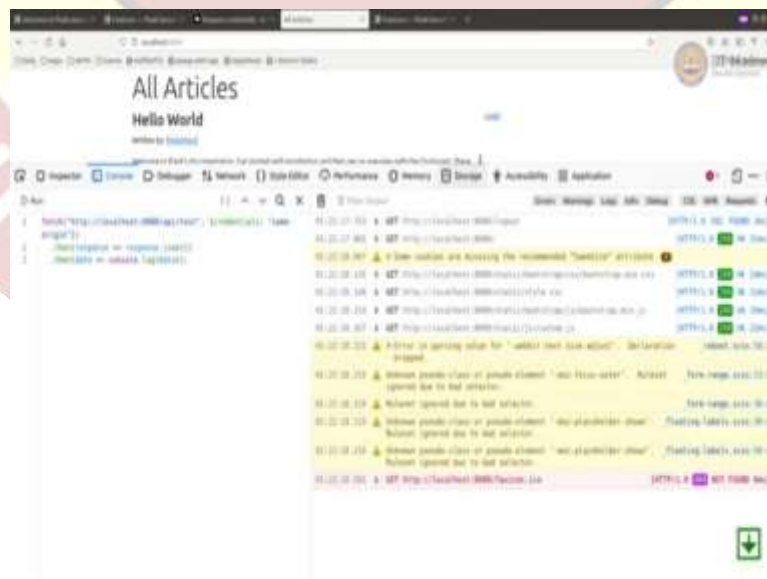
(Refer Slide Time: 11:20)

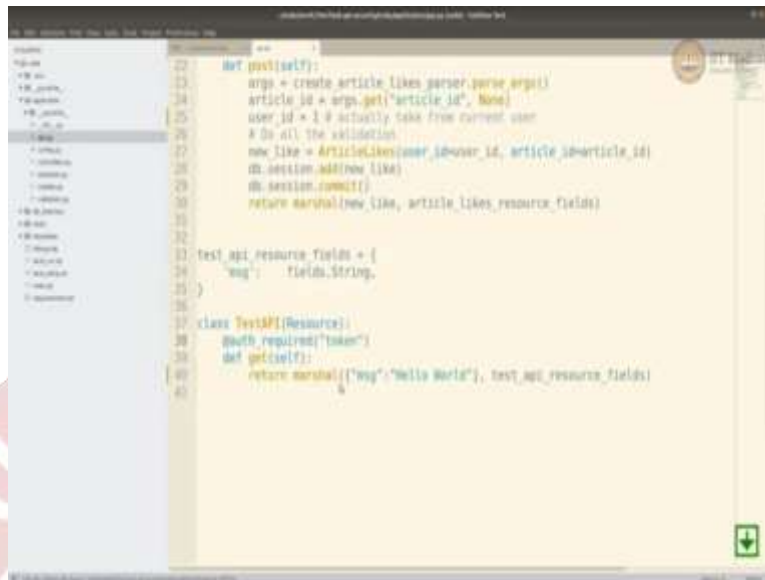




Once you do that, you can actually post to the same endpoint, here you can see it is a base URL is localhost 8080, slash login and question mark include auth token, that is what the documentation says. If you see here it says, performance HTTP POST with a query param of include auth token, and the authentication details as JSON data. Authentication details are email and password, might dummy password is also password, and then it has to be JSON body. Then if you send it, it locks in and gives you a key. Now, you have a authentication token key.

(Refer Slide Time: 12:11)

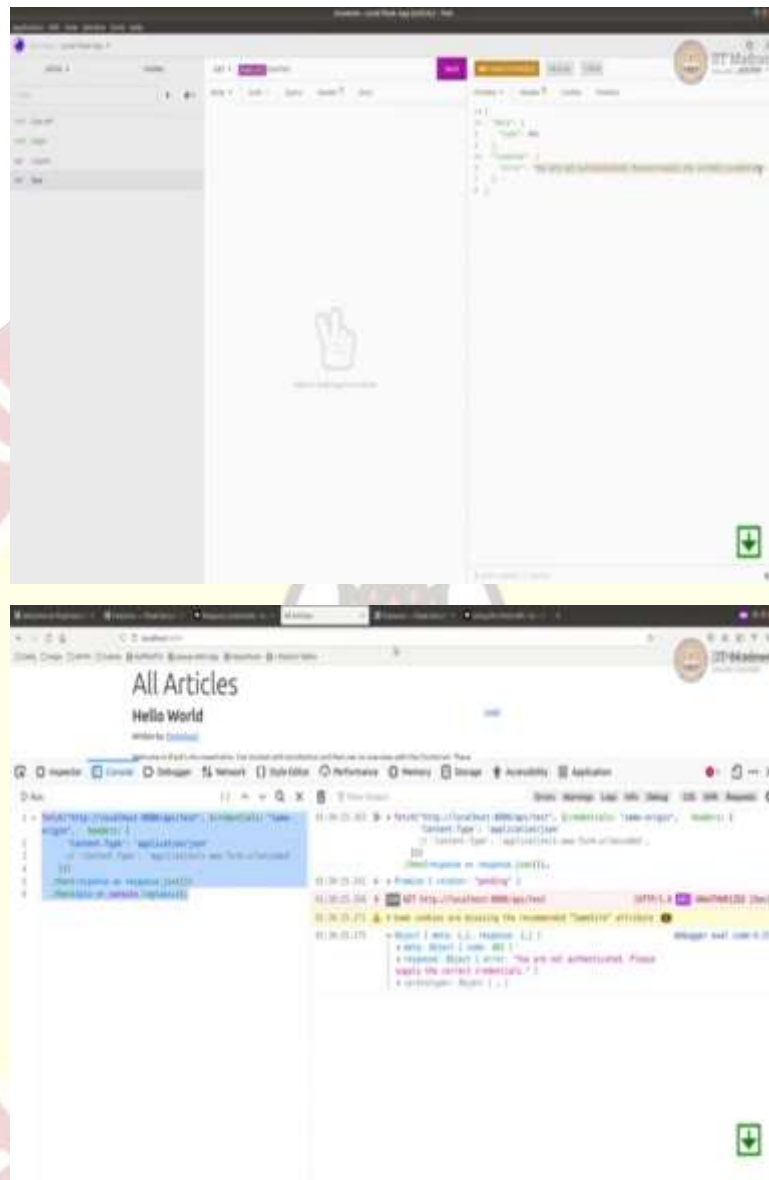




```
22 def post_save(sender, instance, created, **kwargs):
23     args = create_article_likes_parser.parse_args()
24     article_id = args.get('article_id', None)
25     user_id = 1 # actually take from current user
26     # Do all the validation
27     new_like = ArticleLikes(user_id=user_id, article_id=article_id)
28     db.session.add(new_like)
29     db.session.commit()
30     return marshal(new_like, article_likes_resource_fields)
31
32
33 test_api_resource_fields = {
34     'msg': fields.String,
35 }
36
37 class TestAPI(Resource):
38     @auth.requires('token')
39     def get(self):
40         return marshal({'msg': 'Hello World'}, test_api_resource_fields)
41
42
```

Now, let us go back here where we were, and then let us delete the cookie here, if there are no application storage cookies, just going to delete it. Now, if I go to console, clear this, and run this. You can see that it is trying to forward it to the next page, we do not want this to happen. Now, you want it to throw an error saying login not possible, or login error, or an unauthorized. So, that is where we have to change in the API. So, instead of login required, we need to put auth token required, in this case we want auth token specifically. And do not want to use login required, because login required also does forwarding to a login page if it not logged in already. Let us add that auth token required or auth required, type token not session cookie. I am going to save this. See whether it is restarted.

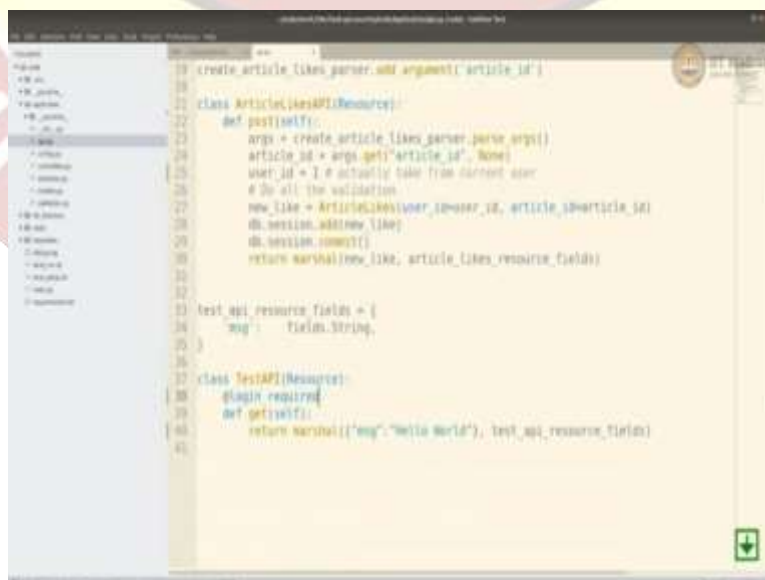
(Refer Slide Time: 13:41)





Because I want to set the header as JSON, the body is JSON like. For example, if I do this here, it shows that I am trying to get out JSON and chose you are not authenticated, please supply the correct credentials, and the header is content type JSON. So, I just want to send the fetch header type here. See fetch, set headers MDN just want to set with that when it our flask security things written. We do not want this actually, but anyway you can keep it, it does not make any difference. Now, we can see here you have got a response, saying you are not authenticated, please supply current credential. You want to put same origin. That is because there is no there is no cookie, we deleted the cookie. If we had the cookie, I think we would have got it.

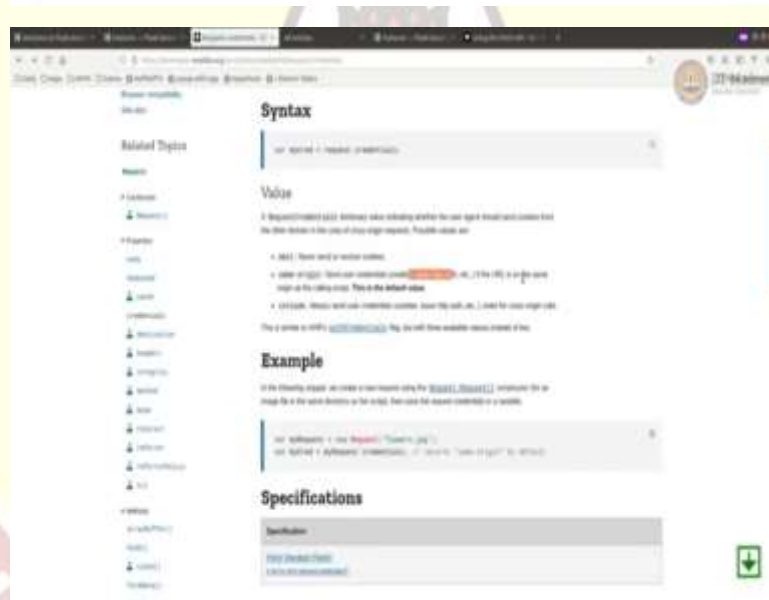
(Refer Slide Time: 15:14)





Let us say once more, let us try I@thejeshgn.com, click login and then delete this. And if I run, say in order to authenticate it. Now at this point, it is expecting not a cookie, but the token right? If I had put login required, I am sure it would have worked, let see. There you go, it worked. But we do not want login required, first thing we want to use token. And now even though we are sending our cookie, it is not using it. But anyway, even if you do not send cookie it will be the same, because they are the same effect. Now, we will do one change. I will delete the cookie just to be clear. I will go back to console, I will just clear this, it should throw an error now. It is throwing an error 401.

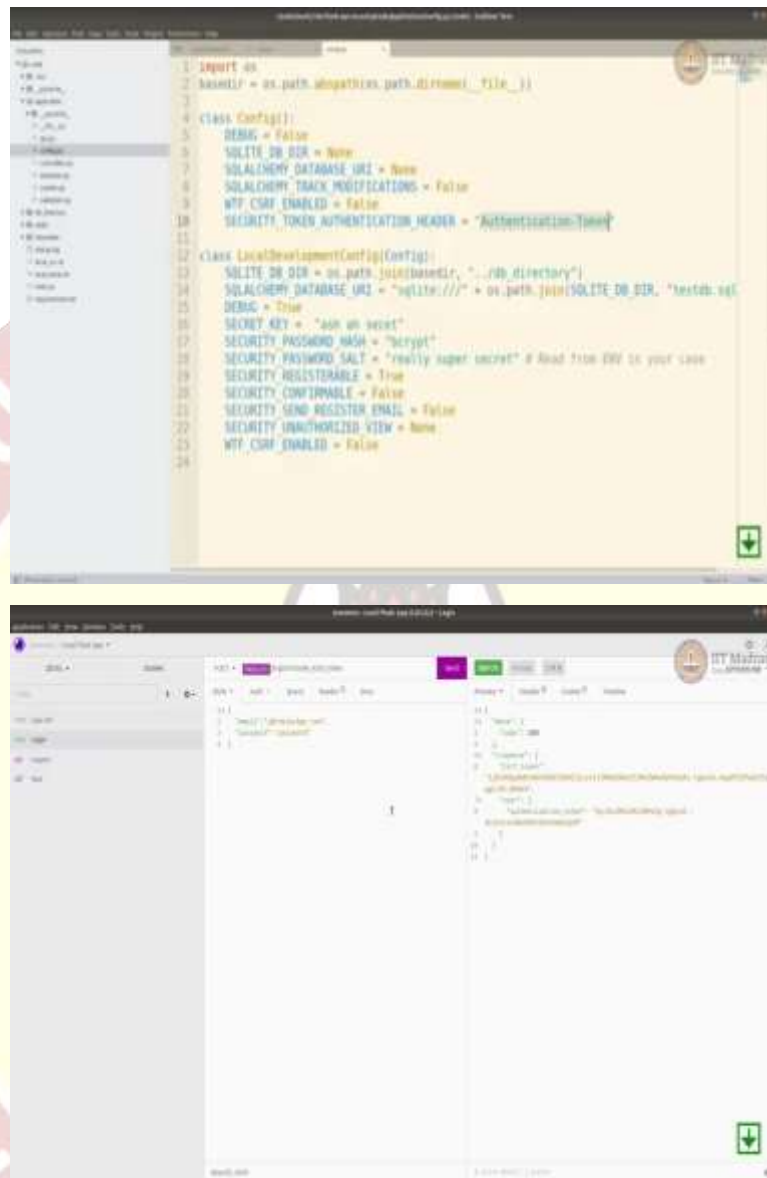
(Refer Slide Time: 16:42)





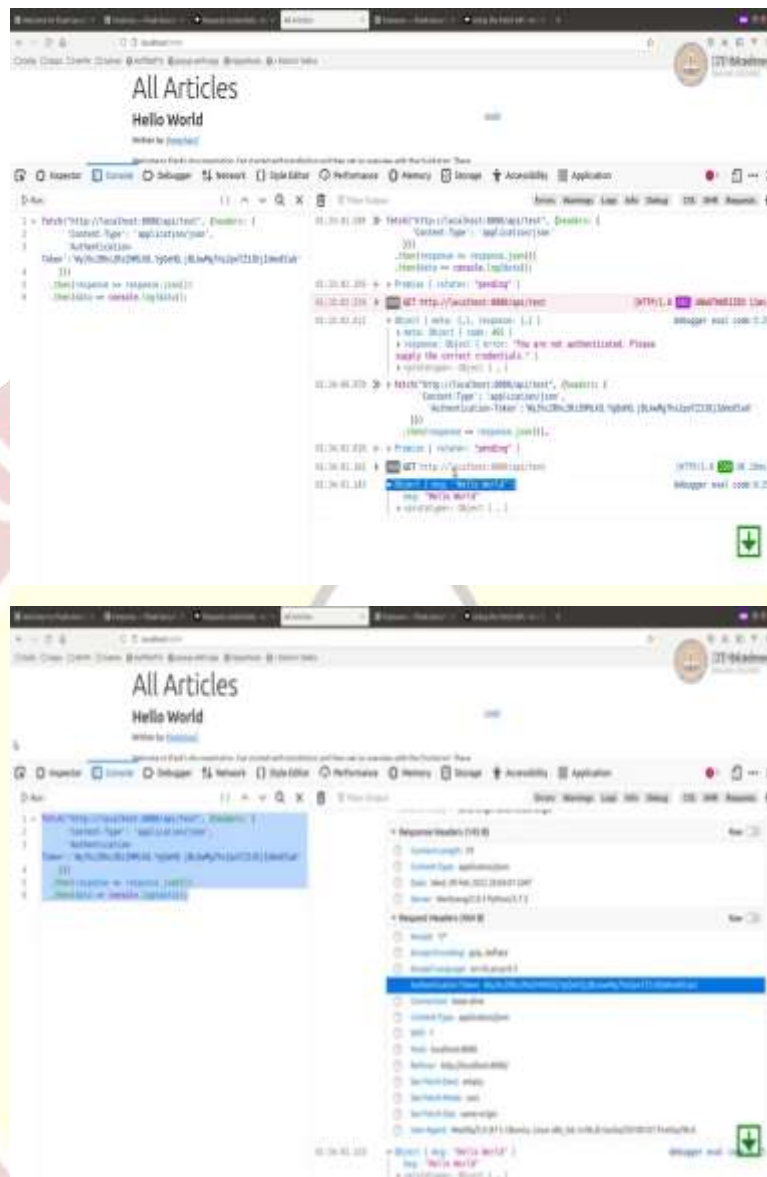
Now, I will remove these credentials should not matter, let us see what it means. Basic auth, so it does not include headers, so headers will go anyway. So, we do not need this, so we will add another header thing. I will just log out here also, just to make it clear for you. If right, so there is no way this is going to work now, just to make sure that you understand. There is no cookie anyway, I have deleted the cookie, so it would not have worked. So, it is still throwing an error. Yep. So, now I will add another header component. what is the header component? You said this. We are going to send the token as authentication token. What will be the value?

(Refer Slide Time: 17:52)



```
1 import os
2 base_dir = os.path.abspath(os.path.dirname(__file__))
3
4 class Config:
5     DEBUG = False
6     SQLALCHEMY_DATABASE_URI = None
7     SQLALCHEMY_TRACK_MODIFICATIONS = False
8     WTF_CSRF_ENABLED = False
9     SECURITY_TOKEN_AUTHENTICATION_HEADER = 'Authentication-Token'
10
11 class LocalDevelopmentConfig(Config):
12     SQLALCHEMY_DATABASE_URI = os.path.join(base_dir, '..', 'db_directory')
13     SQLALCHEMY_DATABASE_URI = "sqlite:///" + os.path.join(SQLALCHEMY_DATABASE_URI, "testdb.sqlite")
14     DEBUG = True
15     SECRET_KEY = "ash is secret"
16     SECURITY_PASSWORD_HASH = "bcrypt"
17     SECURITY_PASSWORD_SALT = "really super secret" # Read from ENV in your case
18     SECURITY_REGISTERABLE = True
19     SECURITY_CONFIRMABLE = False
20     SECURITY_SEND_REGISTER_EMAIL = False
21     SECURITY_UNAUTHORIZED_VIEW = None
22     WTF_CSRF_ENABLED = False
23
24
```

The bottom screenshot shows a web browser displaying a Flask application. The application has a sidebar on the left with a logo and navigation links. The main content area shows a login form with fields for email and password, and a 'Login' button. The application is running on a local server at 127.0.0.1:5000.



The value will be this we what we have sent to the login thing, and what the key we got like? If you login, you are already logged in. Send logout here in this specific thing, but let me just to log in again. You get a token here authentication token, this is a token that we need to send. And you can save this token. If you are using browser, you can use in or save it in local storage, then keep sending it every time.

Now, let us run this, there you go, you got. If you see the header request header, you can see that we are sending the authentication token, and no cookies or anything. So that way, we are able to use authentication. Now, what you could do? In the same logic, we can send both login and then fetch the thing, let us do that.

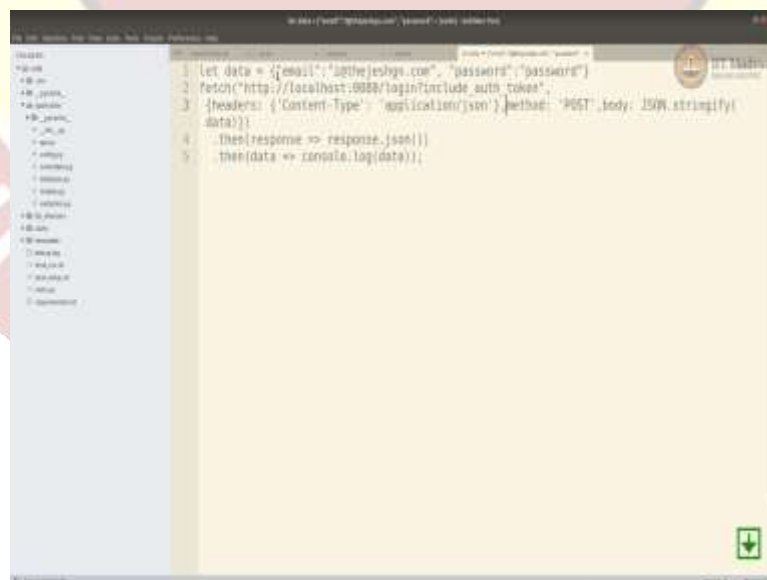
(Refer Slide Time: 19:13)



So, I am just going to cut this here, I am just going to create two JavaScript calls, one is this anyway. Another one is a login request, just for our understanding. That it is working properly login and question mark. If we do not include this, it will try to set the HTTP cookie, that is why that is important. Then we want send this, you have to send the content body. How do we send the content body in the view APIs? It will be a post method. And then body will be JSON stringify data, and I will just do let data equal to user email I@thejeshgn.com, password, my dummy password. Then I am going to send the body JSON stringify, we do not need this, because we are trying to login.

So, we are going to run this first, get the response, based on that we are going to call that, so, let us see. So, let us refresh this page just to show that we are not already logged in. If we go to our application or storage, cookies, there is no cookies, so here other call. Sorry, I think there is while editing I missed something, you do not need this. Basically, this is the login JavaScript and this is the testing one, we will replace this with the actual thing that we get. We do not have this, let us run this, let us run this first something some value. Basically, throws an error 401.

(Refer Slide Time: 21:52)

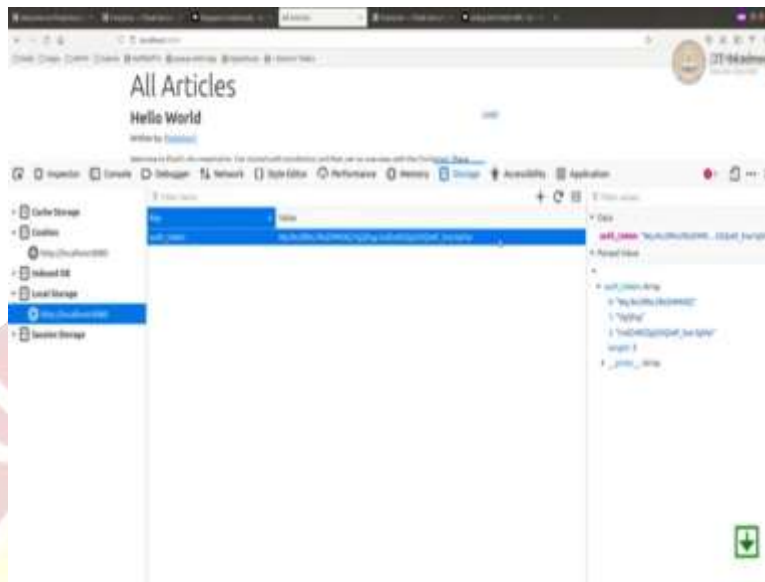




Now, we will not do that, then we will do this, where we are creating the data, username and password. Let us say you are in a view component, we are taking username and password. Then you are submitting to this URL endpoint with post, body, comma here, JSON dot Stringify data, then send headers also, yes formatting is little out of things. So, let me just do formatting well, just making sure everything is closed, this to this, this to this, this to this. So, now we are sending the headers, sending the method as post body as body. Now, if I run this, it can give me a response. Now, in the response, the login has happened.

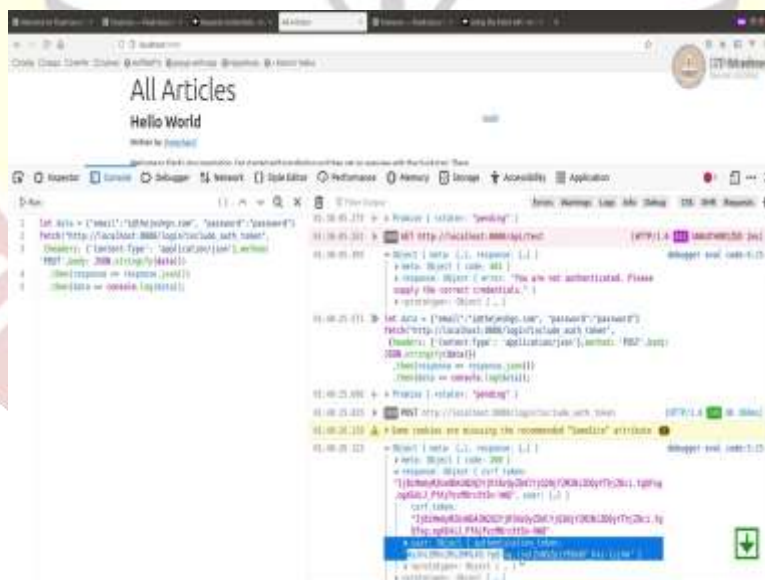
And it is given me response object to here csrf token. And within the user object there is a authentication token you can see that. Then you can copy this authentication token and into JavaScript you can store it in the local storage. You can go to storage, you can store in the local storage, and then you can pull it from the local storage and use it.

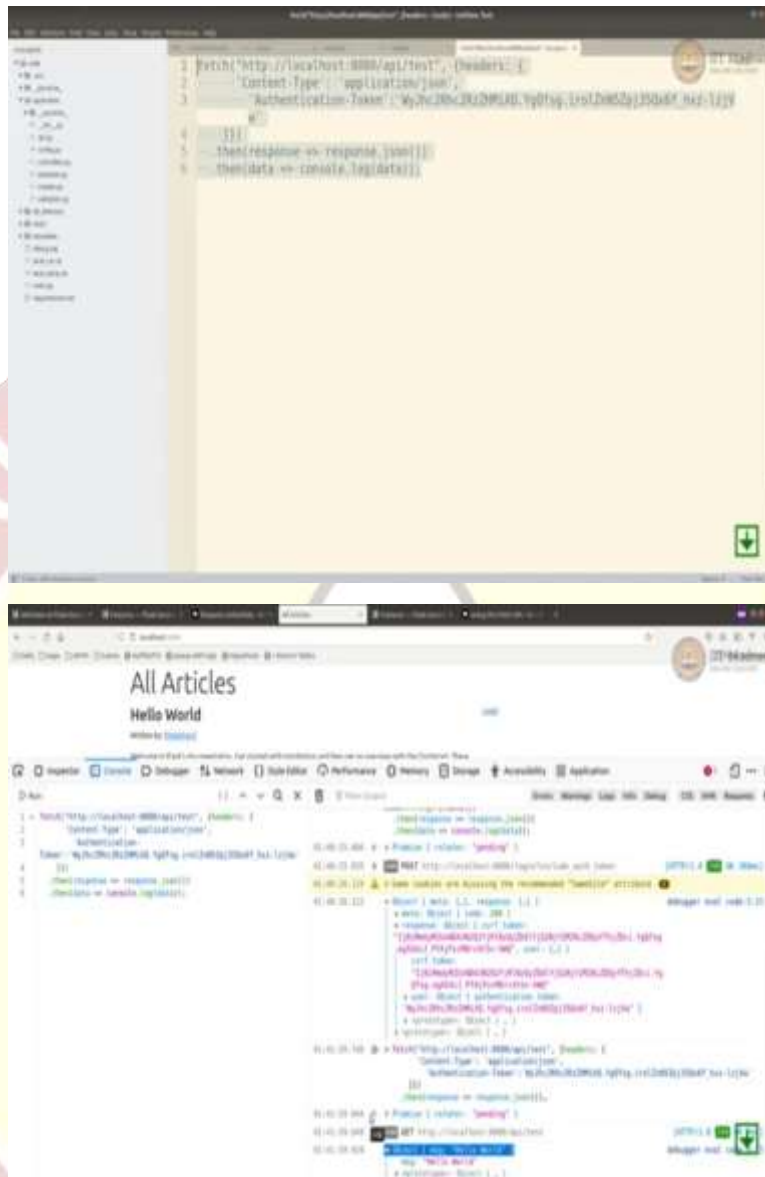
(Refer Slide Time: 23:20)



For example, you can do it here give some maybe like key as auth token, and you store it like that as long as you want. And then when you do not want, you can just remove that, and use it to read it from this key value from the local storage to make the calls. Now, I am not going to do that, but, I am just giving an example.

(Refer Slide Time: 23:47)

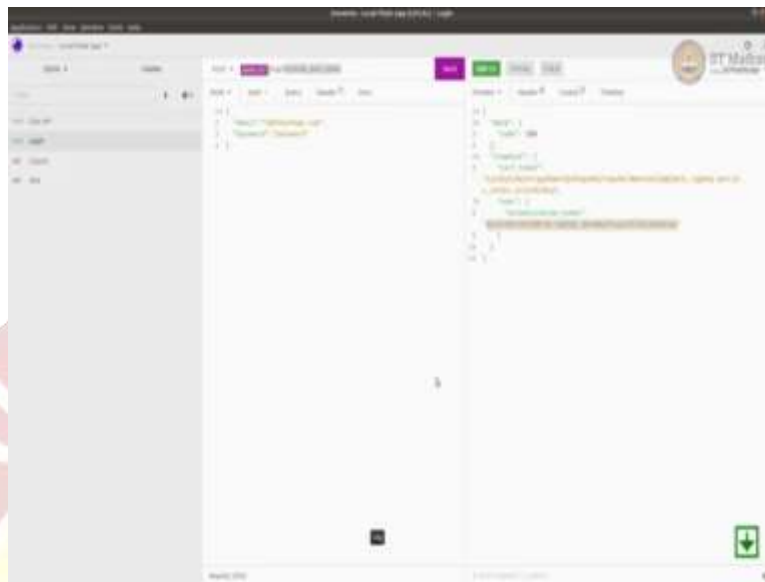




Now, once you get this, you can copy this. I am just going to make another call just to show that it works that would look like this, just the same auth token VW, and then we can run it again. So, you can see here it ran and give me an answer. So, that is how we are going to use auth token. So, we saw two methods here, one using cookies. Use the standard way, you do not have to do play around, if you as long as you use the same domain.

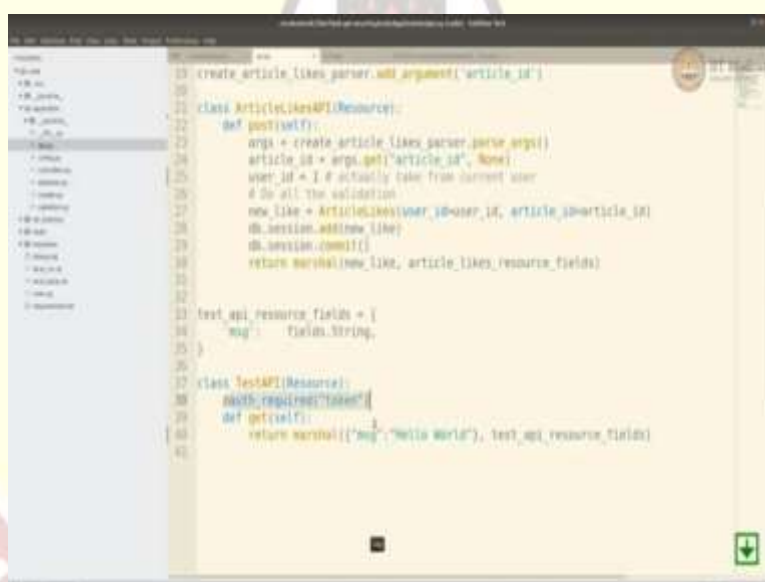
And you have a way to log show the login page and log in. And then the rest of your calls will go through easily, because it uses cookie authentication, now we wanted it to be much more clearer. You do not want to show in your own login page, but you want to make your own login page in some AJAX way using view component or something.

(Refer Slide Time: 25:11)

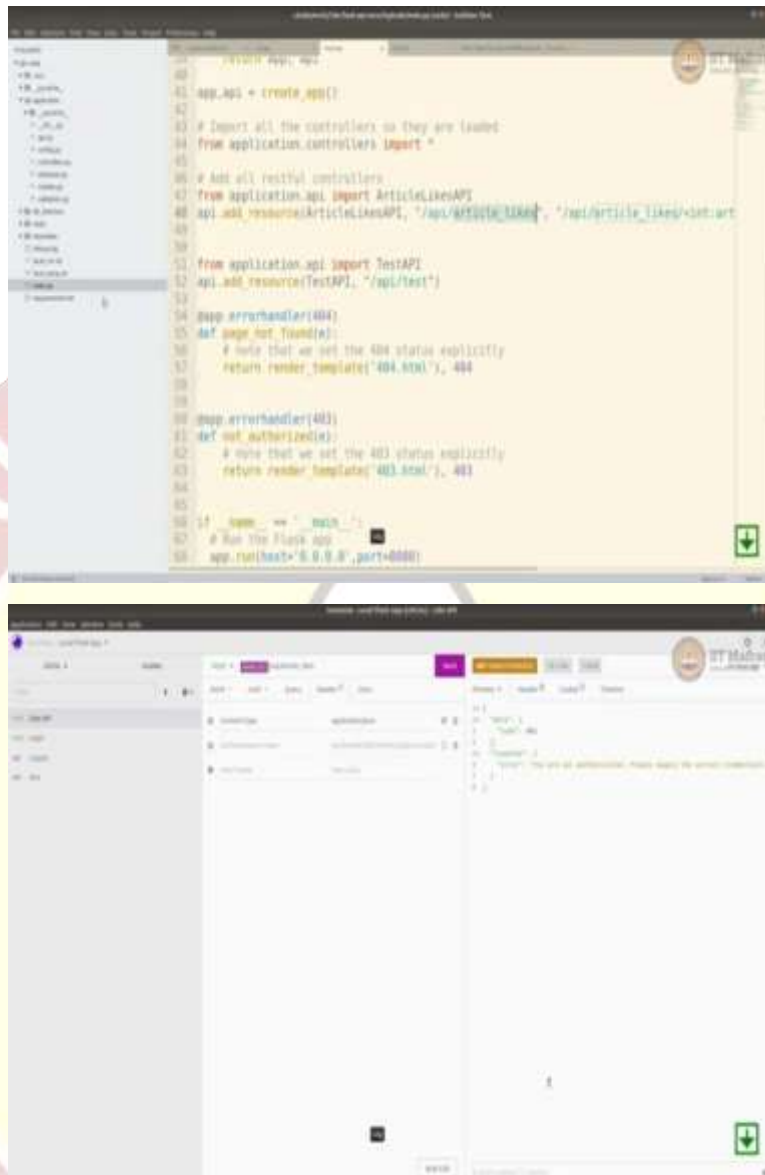


```
POST /api/articles/1/likes
{
  "user_id": 1,
  "article_id": 1
}
```

```
201
Location: /api/articles/1/likes/1
```



```
13 create_article_likes_parser.add_argument('article_id')
14
15 class ArticleLikesAPIResource:
16     def post(self):
17         args = create_article_likes_parser.parse_args()
18         article_id = args.get('article_id', None)
19         user_id = 1 # I actually take from current user
20         # Do all the validation
21         new_like = ArticleLikes(user_id=user_id, article_id=article_id)
22         db.session.add(new_like)
23         db.session.commit()
24         return marshal(new_like, article_likes_resource_fields)
25
26 test_api_resource_fields = {
27     'slug': fields.String,
28 }
29
30 class TestAPI(Resource):
31     @auth.requires('token')
32     def get(self):
33         return marshal(['slug', 'Hello World'], test_api_resource_fields)
```

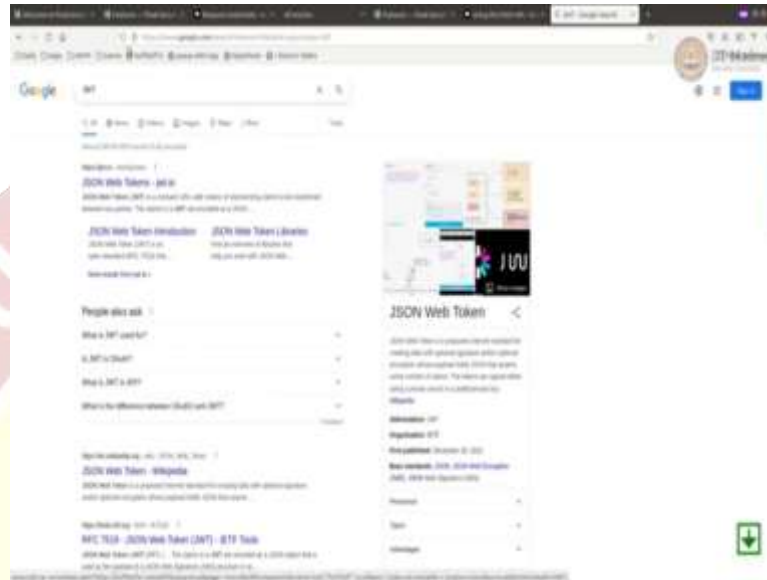


Then, you could use this this way to log in or post the logged email and password, get the authentication token, store these authentication token, and then may call your calls, from then on using authentication token. They should be possible for all of them that we have tagged in our API as auth required. And we can mark like another article LikesAPI, I wanted to show articles, LikesAPI. For example, if you want to do, then you can have an API URL that looks like, that looks like this article likes. And then, a user can like an article by sending the article id and along with authentication token that once enabled authentication token.

I should be able to like the article, if I do not send it, then now it gives you 401 error. Then you would need you will force the user to log in. That is it actually. So, I hope this will be useful

when you are developing your view applications. You can use any of the ways, I think the second way of using token is much more useful.

(Refer Slide Time: 26:45)



Now, if you want to go like beyond and do some more experiments at your own time or learn more, you can try JWT also called JSON web tokens. There is a package called Python JWT, and the one for flask security also, you can use that to use JWT. But I think now our case, the specific case that we have, it just says same system that generates the token, and the same system that validates the token. So, I think a regular token should be more than enough. I think JSON web token or JWT will become very useful, if you have distributed systems.

For example, a system is generating tokens for you, and a very different on another system wants to use that token to authenticate you that when JWTs are very useful. In our case, I think we are going to have a single system that will generate the token for you, which means that will give you auth endpoint, to generate the token for you. And you will also consume the same token, so we are going to use simple tokens, that is it.

(Refer Slide Time: 28:00)



I hope you will find this useful. Go ahead, and explore by yourself what else you can do and implement them. Some of the interesting stuff are like two-factor authentication, it is still an experimental thing. But it is an interesting thing, where you after the user logs in, you ask for one more level of authentication, like either TOTP Time Based Authentication, or SMS based second factor authentication, things like that to add more security. I think you should learn about it. But it is not must at this point. But, I think if you spend some time and learn about it, you can always implement it, it will be a very useful security feature. Thank you so much. Have a good day.