CS437 ASSIGNMENT REPORT

TOPIC SELECTED: EXCESSIVE DATA EXPOSURE

NAMES:

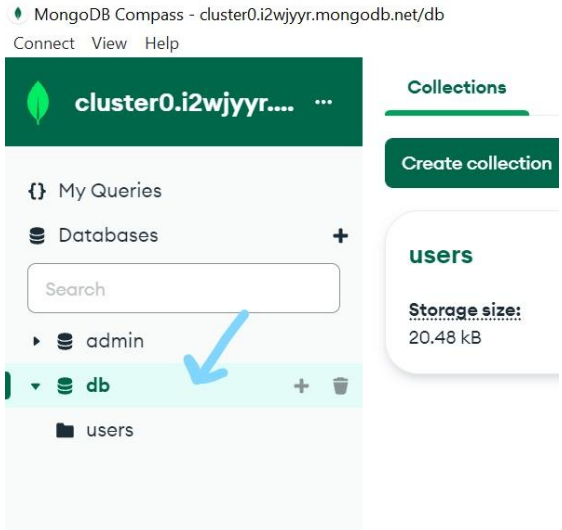
AHMET ENES SILACI - 22346
GÜRKAN TALHA SOYLU - 26883
OZAN YILDIRIM - 26579

GENERAL INFORMATION ABOUT EXCESSIVE DATA EXPOSURE VULNERABILITY

The main idea in excessive data exposure is when the API returns full data objects as they are stored in the backend database. Filtering can be done from responses in client application which and only shows the data that the users really need to see. However Attackers call the API directly and get also the sensitive data that the UI would filter out. For protecting from this vulnerability we should not rely on the client to filter data. Therefore, we need to review related API response and change them to match what the API consumers really need.

DATABASE CONNECTION

We used mongoDB for storing client data. MongoClient helps to connect database with a specific connection string. This string includes cluster information in the database such as database name which is 437_project and database password. Then client['db'] refers to the database folder where users informations are kept:



Then we can reach the database with db[“users”] and we named it as collection_name

And the code which establishes connection to database can be seen as below:

```
def get_database():

    CONNECTION_STRING = "mongodb+srv://437_project:sifre437sifre@cluster0.i2wjyyr.mongodb.net/test"

    client = MongoClient(CONNECTION_STRING)


    return client['db']

app.config["JWT_SECRET_KEY"] = 'secret'
app.config["SECRET_KEY"] = 'secret2'

db = get_database()

collection_name = db["users"]
```

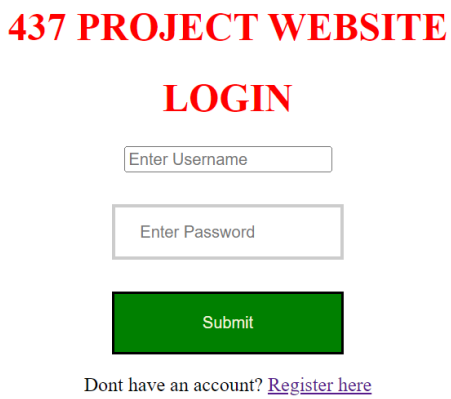
All data is randomized. Only names, surnames, passwords and usernames were taken from user with input during registration process. Other remaining informations were added randomly by putting extra field for each user such as credit number, cvc, address etc. And our database contains more than 5 users as it can be seen in the below (there is 6 user before submission the assignment):

PAGES

There are 4 pages in the website. These are signin, signup, profile and profile information page. Signin and Signup pages is for creating application which will have authentication mechanism as requested in the project requirements.

SIGNIN PAGE

This page will take two values from client. Username and Password, and server will respond this request if credentials are true, as username of client and access_token into the system. Otherwise, it will be rejected and will respond a message below of the Signin Page. The screenshot of the page as below:



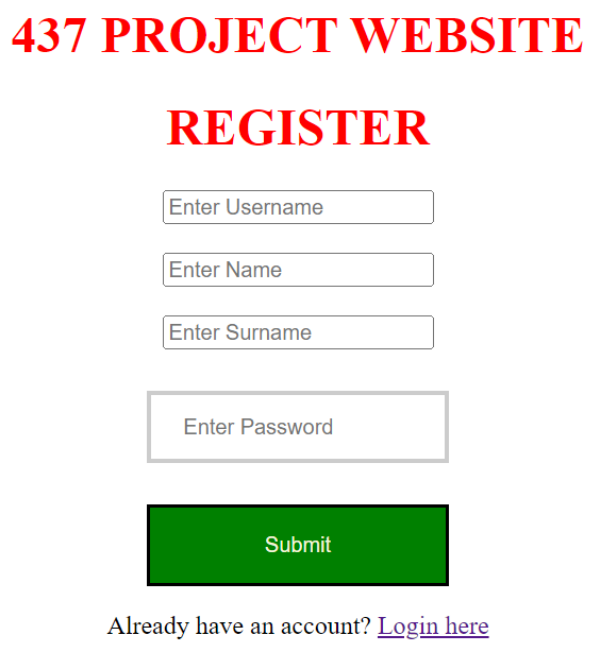
Then, if we look at server part of the request it is as follows:

```python
@app.route("/", methods=['GET', 'POST'])
def signin():
    if (request.method == 'GET'):        #This get request is for seeing the page when "/" endpoint triggered
        return render_template('signin.html')   #then we see the sigin page automatically

    else:
        #due to usage of form request from html request.form.get is used
        username = request.form.get('username')
        password =  request.form.get('password')

        #with getting username input collection in the database will be checking
        response = collection_name.find_one({'username':username})

        if response:     #if response true which means user is found


            if bcrypt.check_password_hash(response['password'], password):   #then password checking will be needed

                access_token = jwt.encode({     #after successfully entered correct credentials token will be created for client
                'username': username,
                'exp' : datetime.utcnow() + timedelta(seconds = 10)
                }, app.config["JWT_SECRET_KEY"])

                #session attribute of flask will be used during the client logins, token and its username stored in session
                session['set_token'] = access_token

                session['set_user'] = username


                return render_template("success.html", result = username)    #we do not pass token to the client because token inside the responding data is not a good idea

            else:    ##when username exists but wrong password

                message = "Wrong password or username"
                return render_template("signin.html", message = message)

        else:   #when username does not exist

            message = "Wrong password or username"
            return render_template("signin.html", message = message)
```

As we commented above in nearly each line, post request will be send when submit button is pressed. After getting request from client and successfully checking information which got from inputs, And Tokens are used to show fully authentication mechanism, for that token will be generated and saved with session. Session stores data as dictionary like object. So that server can easily do client based operations by calling specific session object. Then, success page will be shown. Else, if username is correct but password is wrong or username does not exist, it will return an error message below login button.

SIGNUP PAGE

This page is to populate the database when new client comes to the website. It will be taking username name, surname and password. And screenshot can be seen as below:



Then if we look at server side of application as below:

```python
@app.route("/signup", methods=['POST', 'GET'])
def signup():
    if (request.method == 'POST'):
        username = request.form.get('username')
        name = request.form.get('name')
        surname = request.form.get('surname')
        password = bcrypt.generate_password_hash(request.form.get('password')).decode('utf-8')


        user_found = collection_name.find_one({"username": username})

        if user_found == None:


            user_input = {'username': username, 'name': name, 'surname': surname, 'password': password}

            #according to collected data from client above user input will be inserted to database
            collection_name.insert_one(user_input)



            return redirect(url_for("signin"))
        #if user is found error message will be shown into register page
        else:


            message = "There is already a user with: " + username + "please try another username"
            return render_template("signup.html", message = message)
    else:

        return render_template('signup.html')
```

SUCCESS PAGE

This page will be shown after correctly entered in login page. It will display client's username and a button which is for the client to press to see profile information of herself/himself. Screenshot can be seen as below:

WELCOME

ahmet

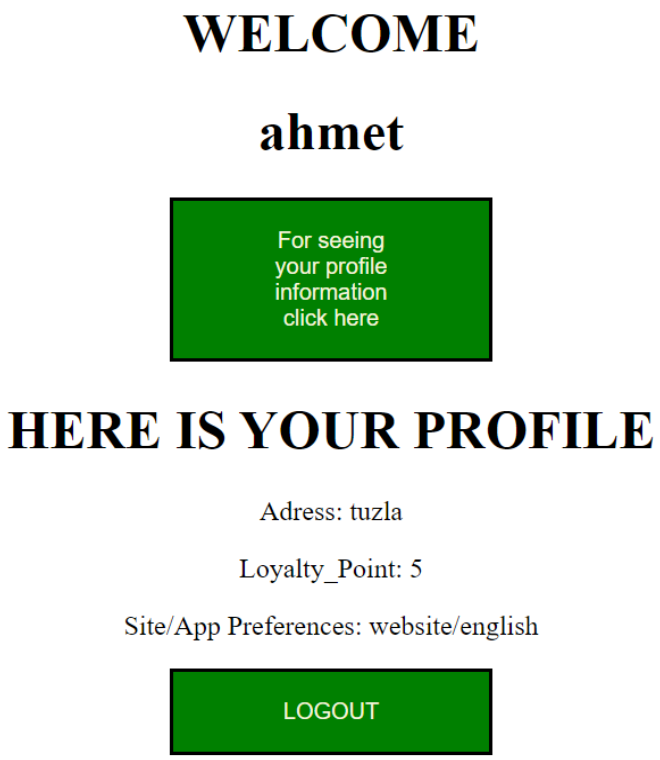
For seeing your profile information click here

PROFILE INFORMATION PAGE

After pressing above button as the informations that clients should see will be displayed. And there is a logout button which is for demonstrating of exitting from application. And when this button is clicked sessions in the server will be terminated which includes token and username information of the current client and a information text will be shown from server called “You are now logged out go to signin page”. In this point, after logout, making a get request to the “/api/v2/customer/profile” url, an error message will come from server called “token is missing” this means token is not available to use because current user logouts. Moreover, there is a expired time for a valid token after generating it. After this time exceeds, if the user making a get request to the “/api/v2/customer/profile” url again, he/she will be got a respond from server like “token is not valid”.

The mentioned information given above can be seen as a screenshot below:

After pressing information button:



After pressing information button after token expiration date:

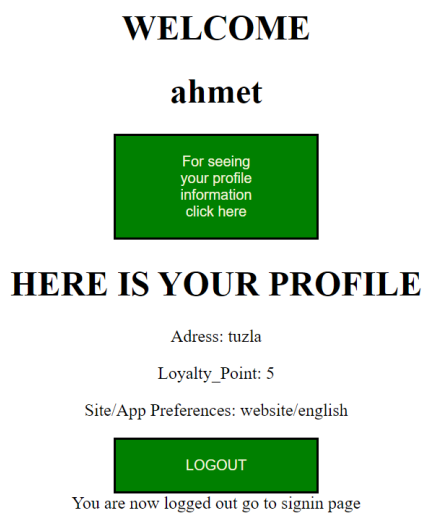
WELCOME

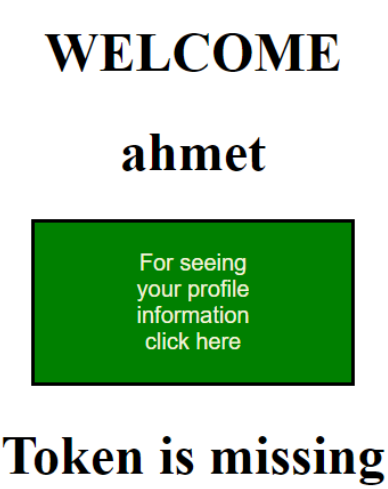
ahmet

For seeing your profile information click here

Token is invalid

After pressing logout button, then information button will give token is missing error:





Server side of this process as follows:

Below get request to the “/api/v2/customer/profile” route.

```
@app.route("/api/v2/customer/profile", methods=['GET'])

@token_required     #this decorator uses for catching token and session exceptions when user requests to above route

def profile():

        user_profile = collection_name.find_one({'username': session['set_user']}, projection={"exp": 0, "cc_num": 0, "cvc": 0, "payment": 0, "_id": 0}) #protection line of

        return make_response(render_template('profile.html', user_profile = user_profile))   # returns a html page and user information as json
```

@token_required decorated function is used when this request is made. Decorators are a useful way to add extra functionality to a function without changing the function itself.

They are often used to add additional features to an existing function, such as logging, caching, or authentication.

```python
def token_required(f):
    @wraps(f)
    def decorator(*args, **kwargs):


        try:


            token = session['set_token']

        except KeyError:  #gives exception when token == None


            return render_template('error.html', error = 'Token is missing') # Output: KeyError: 'set_token' when session popped


        try:

            data = jwt.decode(token, app.config["JWT_SECRET_KEY"], algorithms=['HS256'])


            #if datetime.fromtimestamp(data['exp']) < datetime.utcnow():    this line is an alternative to check token valid time and current time however it is better to

        except jwt.ExpiredSignatureError:    #this is a typical exception for expired dated token


            return render_template('error.html', error = 'Token is invalid')

        return f(*args, **kwargs)

    return decorator
```

When render_template from response of server sends, user_profile parameter can be taken in the client. And we are getting three information of user these are address, loyalty_point, site/app preferences. Whether vulnerable data is send from server or not client will see only these three information herself/himself. Another issue about requests htmx is used, this is a useful tool used in html which facilitates client-server interaction. hx-get, hx-post is used for request, hx-push-url is for changing url, and hx-target is used where responded data will be shown in the page.

```html
<!DOCTYPE html>
<html lang="en">
  <body id="reg" class="body1">

      <center>

        <h1> HERE IS YOUR PROFILE </h1>

        <p>Adress: {{user_profile['adress']}}</p>


        <p>Loyalty_Point: {{user_profile['loyalty_point']}}</p>


        <p>Site/App Preferences: {{user_profile['site/app_preferences']}}</p>

      </center>

      <form>
        <button class="submit" hx-post = "/logout" hx-target="#logout-message" hx-redirect="/"> LOGOUT </button>
      </form>

      <div id="logout-message">

      </div>
</body>
</html>
```

PROTECTION TO EXCESSIVE DATA EXPOSURE

As we mentioned above, After the request to “/api/v2/customer/profile” route, server is responding some information about client that stored in the database. However, without protection server is sending too much irrelevant data to the client. Here is the **vulnerable code** in the server:

```
@app.route("/api/v2/customer/profile", methods=['GET'])

@token_required     #this decorator uses for catching token and session exceptions when user requests to above route

def profile():

        user_profile = collection_name.find_one({'username': session['set_user']}) #protection line of excessive data exposure

        object_id = ObjectId(user_profile['_id'])    #114-115 lines is for getting rid of "Object of type ObjectId is not JSON serializable" error, database returns unique
        user_profile['_id'] = json.dumps(str(object_id))

        return user_profile   # returns a html page and user information as json
```

As we can see collection in the database specific to the client will return all the data in the database. And this information is sent to client with render_template in flask.

And the **protected code** can be seen as below:

```
@app.route("/api/v2/customer/profile", methods=['GET'])

@token_required     #this decorator uses for catching token and session exceptions when user requests to above route

def profile():

        user_profile = collection_name.find_one({'username': session['set_user']}, projection={"exp": 0, "cc_num": 0, "cvc": 0, "payment": 0, "_id": 0, "password": 0, "name

        return user_profile   # returns a html page and user information as json
```

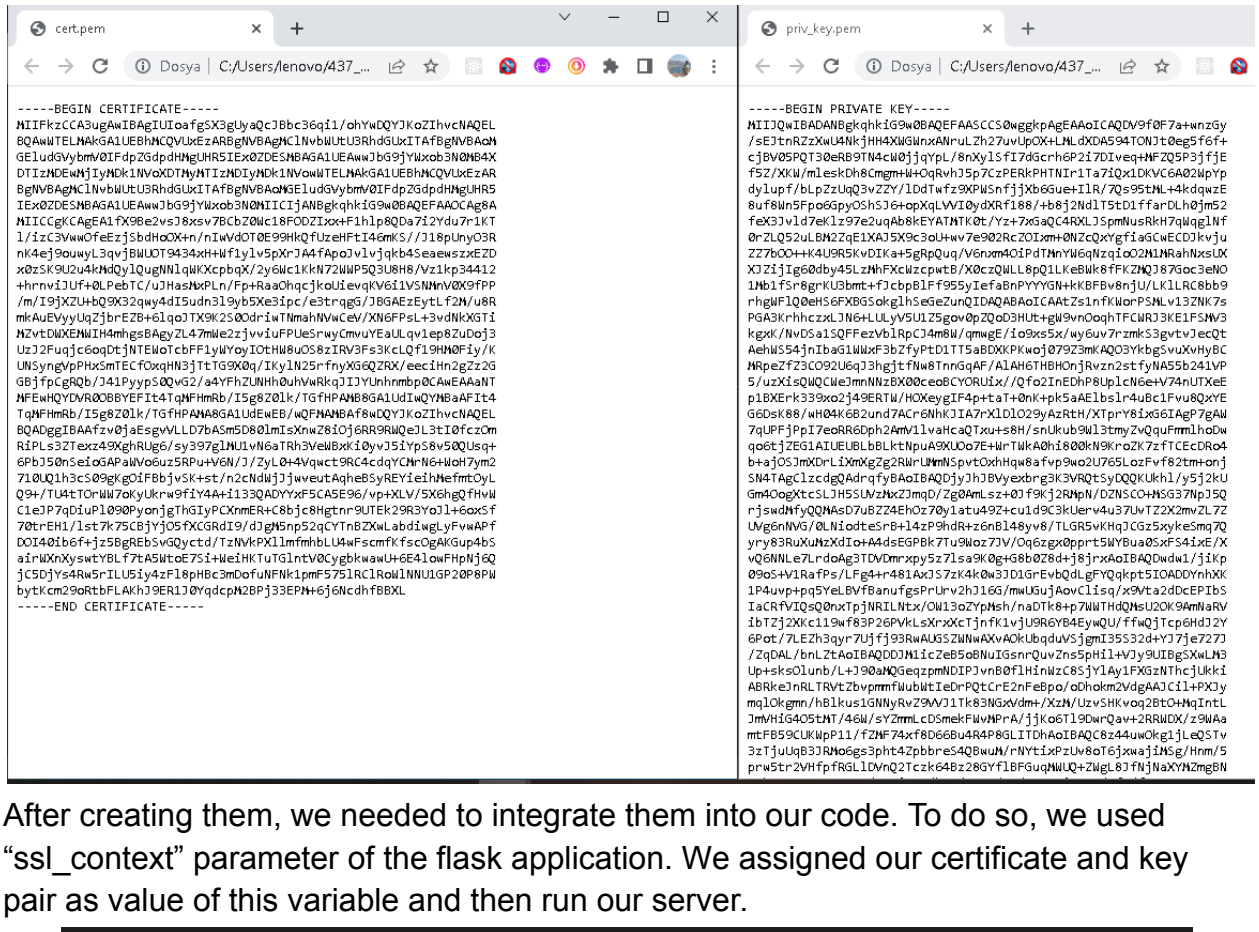
The difference between previous line, we added **projection** into find_one the projection parameter is used to specify which fields should be included in the returned document. It allows you to specify which fields you want to include or exclude in the returned document. The projection parameter is specified as a document that specifies the field and its corresponding value as either 1 (include) or 0 (exclude). This will prevent to send excessive data to client.

SSL Certification and HTTPS

In the project description, we are asked to perform encryption during the transfer of data. A well-known way to do this is to establish HTTPS connection within the application. HTTPS is the secure version of HTTP which is HyperText Transfer Protocol and the backbone of the internet. Normally, HTTP transfers the user data in an unencrypted manner which causes data exposure. To overcome this, a secure version of HTTP has emerged. This is basically HTTP with SSL/TLS protocol. So, in our project we needed to establish SSL connection within our application. To do so, first I needed to create an SSL certificate and its private key. To do so, I used OpenSSL and created certificate named “cert.pem” and its corresponding private key “priv-key.pem”. To create

certificate and corresponding key, I used the below command and resulting certificate-key pair can be seen in next figure.

```
openssl req -x509 -newkey rsa:4096 -nodes -out cert.pem -keyout key.pem -days 365
```



After creating them, we needed to integrate them into our code. To do so, we used “ssl_context” parameter of the flask application. We assigned our certificate and key pair as value of this variable and then run our server.

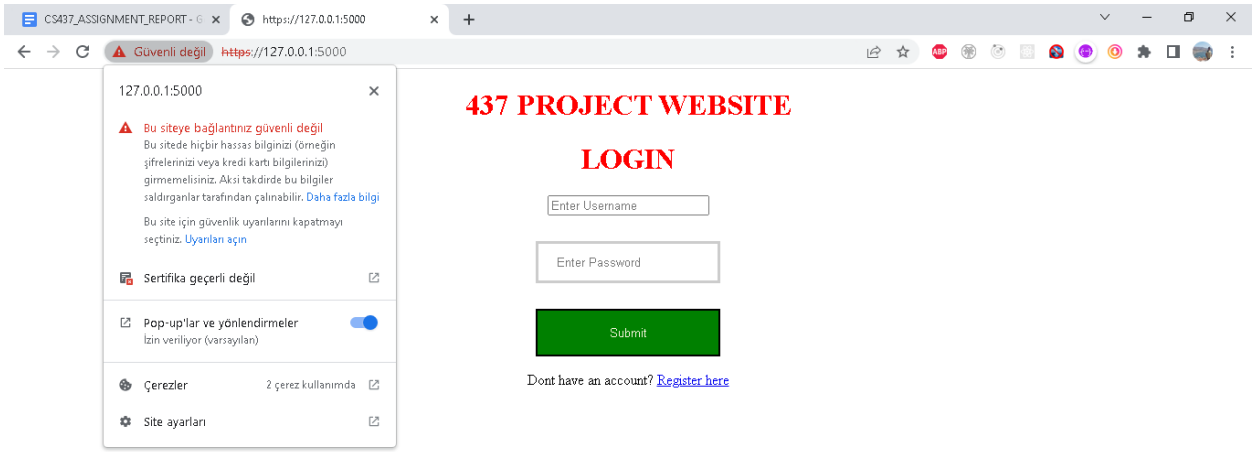
```
if __name__ == '__main__':

    app.run(debug=True, ssl_context=("cert.pem", "priv_key.pem"))
```

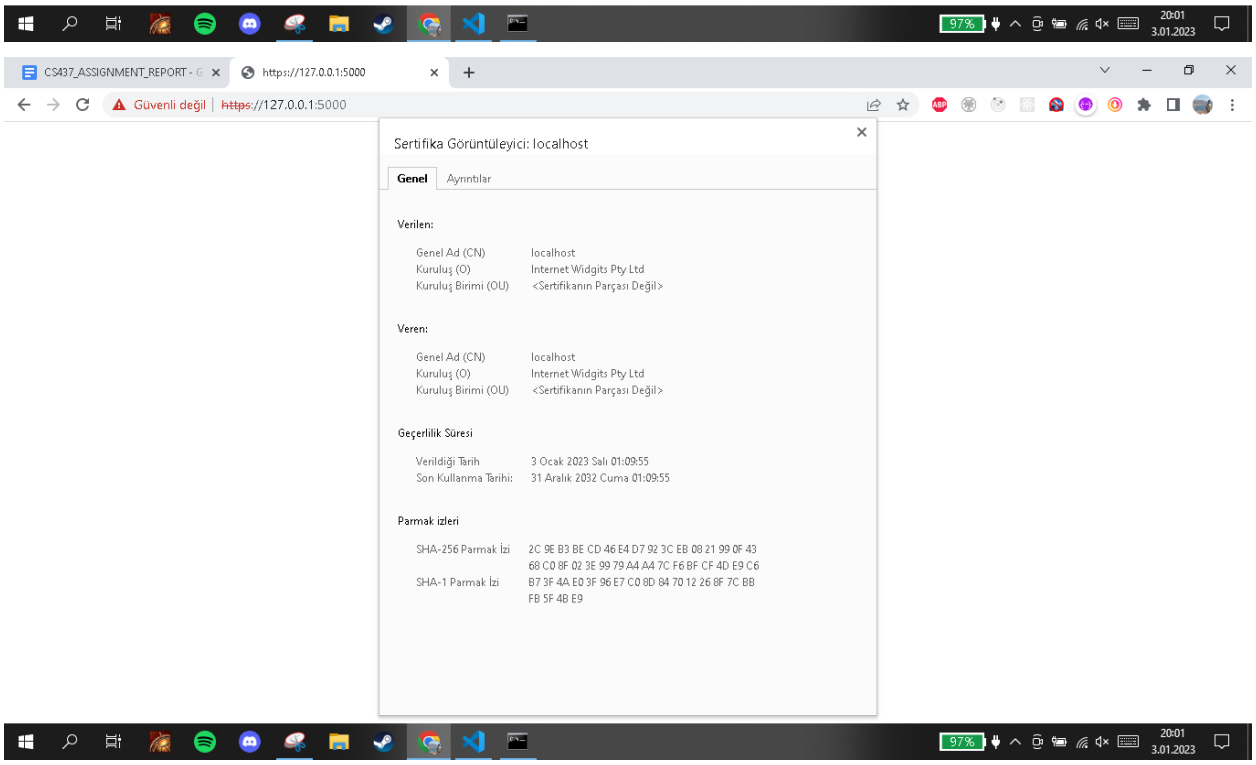
When we run our server, on command line, we can see that our server is establish on https and the link content https.

```
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on https://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 109-657-269
```

However, there is a problem. When we enter the website, what we see is that our connection is not considered as secure connection because of the certificate we use.

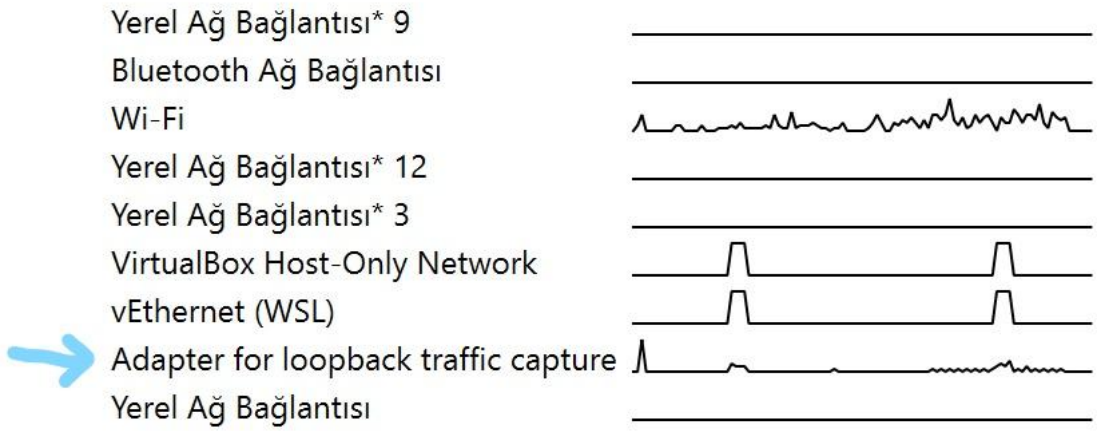
We created this certificate by self-signing, so basically we are not considered as a trusted certificate authority by the web browsers. However, when we click on the warning, we are able to see the certificate and when we try to reach our server with http://127.0.0.1:5000, we are not able to see any website on http.





WIRESHARK RESULTS

For comparing vulnerable and without vulnerable result taken from server in wireshark, we are using “Adapter for loopback traffic capture”. In Wireshark, the loopback interface refers to the network interface that is used to communicate with the local host (the computer that is running Wireshark). It is used to capture traffic that is sent and received by the local host, such as HTTP requests and responses. For seeing actual response from server we are returning only the json data from server not the html page which is “profile.html”



After clicking above choice we are ready to capture packets and profile information button clicked below second screenshot shows the detail:

No.	Time	Source	Destination	Protocol	Length	Info
19	7.670103	127.0.0.1	127.0.0.1	TCP	44	5000 → 61216 [ACK] Seq=1 Ack=905 Win=2619648 Len=
20	7.906483	127.0.0.1	127.0.0.1	TCP	224	5000 → 61216 [PSH, ACK] Seq=1 Ack=905 Win=2619648
21	7.906574	127.0.0.1	127.0.0.1	TCP	44	61216 → 5000 [ACK] Seq=905 Ack=181 Win=2619392 Le
22	7.906672	127.0.0.1	127.0.0.1	HTTP/J…	377	HTTP/1.1 200 OK , JavaScript Object Notation (app
23	7.906723	127.0.0.1	127.0.0.1	TCP	44	61216 → 5000 [ACK] Seq=905 Ack=514 Win=2619136 Le
24	7.907053	127.0.0.1	127.0.0.1	TCP	44	5000 → 61216 [FIN, ACK] Seq=514 Ack=905 Win=26196
25	7.907148	127.0.0.1	127.0.0.1	TCP	44	61216 → 5000 [ACK] Seq=905 Ack=515 Win=2619136 Le
26	7.910963	127.0.0.1	127.0.0.1	TCP	44	61216 → 5000 [FIN, ACK] Seq=905 Ack=515 Win=26191
27	7.911036	127.0.0.1	127.0.0.1	TCP	44	5000 → 61216 [ACK] Seq=515 Ack=906 Win=2619648 Le

Wireshark · TCP Akışı izle (tcp.stream eq 0) · Adapter for loopback traffic capture

```
GET /api/v2/customer/profile HTTP/1.1
Host: 127.0.0.1:5000
Connection: keep-alive
sec-ch-ua: "Not?A_Brand";v="8", "Chromium";v="108", "Google Chrome";v="108"
HX-Request: true
HX-Target: some-info
HX-Current-URL: http://127.0.0.1:5000/
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.0.0 Safari/537.36
sec-ch-ua-platform: "Windows"
Accept: */*
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://127.0.0.1:5000/
Accept-Encoding: gzip, deflate, br
Accept-Language: tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: session=.eJwdzkETQkAYgOH_sucyViHdpNnmY6gkyaXBbGNXi2Ebqem_Zzq_7-
H5oJ7Km2wqWqM1oqNb5ruC7ZkL5zfggEEPdagXDhhQtUnsuJYyTbjQ4jEXRKanKQjS5JdYBTawNCkH4M0r4MUY8ErzuT_cj0rv1gchOsvTneQa5fZmoS9N0yORGrZErrI5PlmPEBNj
a6PZX_TsaTeBslJQib4_WeQ3AA.Y7NRnA.YApE0iMwDe6tER9MQuArsCPNT_A

HTTP/1.1 200 OK
Server: Werkzeug/2.2.2 Python/3.10.4
Date: Mon, 02 Jan 2023 21:50:23 GMT
Content-Type: application/json
Content-Length: 333
Vary: Cookie
Connection: close

{"_id":"\"63a0e57f6ab2c7a5cd49eb83\"","adress":"tuzla","cc_num":"454545454545454545","cvc":"454","exp":"03/25","loyalty_point":"5","name":"ahmet","password":"$2b
$12$PP4rMY3AppPiUx.GOEtLQ.FqwYHobyyv1XEg8um7LCYaXuNe/SWJ.","payment":"mastercard","site/app_preferences":"website/
english","surname":"s\u0131lac\u0131","username":"ahmet"}
```

It can be seen above json data is returning excessive json data from server response and wireshark captures this vulnerable plaintext data. When we apply protection as we can see only the necessary information is provided as below:

Wireshark · TCP Akışı izle (tcp.stream eq 0) · Adapter for loopback traffic capture

```
GET /api/v2/customer/profile HTTP/1.1
Host: 127.0.0.1:5000
Connection: keep-alive
sec-ch-ua: "Not?A_Brand";v="8", "Chromium";v="108", "Google Chrome";v="108"
HX-Request: true
HX-Target: some-info
HX-Current-URL: http://127.0.0.1:5000/
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.0.0 Safari/537.36
sec-ch-ua-platform: "Windows"
Accept: */*
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://127.0.0.1:5000/
Accept-Encoding: gzip, deflate, br
Accept-Language: tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: session=.eJwdzk0LgjAcgPHvsnNJSxTsJiPlb2loaOZF3Fi4yVRy5Uv03ZPOz3P4fdDAdam7hrfogPgc1NRn4iICSBfAkYAB2sRiBGxo-
jwjgWOsE2b7bKbK08V1Dcrr6C3bgRhFkdcjyG6KJJsj2ZjhAtYjNs4xPb7NOD4RfA-
ZVtSrBrL1se2Efpm4adIrS06FWyUN2vxFr4E_V1BVK67R9wdpeTf1.Y7NV5w.F18csMpflLCLVzxO4eN44kScBVU

HTTP/1.1 200 OK
Server: Werkzeug/2.2.2 Python/3.10.4
Date: Mon, 02 Jan 2023 22:08:43 GMT
Content-Type: application/json
Content-Length: 99
Vary: Cookie
Connection: close

{"adress":"tuzla","loyalty_point":"5","site/app_preferences":"website/english","username":"ahmet"}
```

POSTMAN RESULTS

Since we used form actions both register and login page, x-www-form-urlencoded body will be used as a choice. Then below output (which is success page) is obtained with true credentials:

POST http://127.0.0.1:5000/ Send

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

none form-data x-www-form-urlencoded raw binary GraphQL

☑	username	ahmet
☑	password	ahmet

Body Cookies (1) Headers (7) Test Results 200 OK 2.14 s 1.26 KB Save Response

Pretty Raw Preview Visualize HTML

```
13    <h1> WELCOME </h1>
14
15    <h1></h1>
16
17    <form>
18        <button class="submit" hx-get="/api/v2/customer/profile" hx-target="#some-info"
              hx-push-url="true"> For seeing your profile information click here </button>
```

Then when we try vulnerable code we can get below output in postman, as we can see server is sending a json data which includes too much data:

GET http://127.0.0.1:5000/api/v2/customer/profile Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL

This request does not have a body

Body Cookies (1) Headers (6) Test Results 200 OK 229 ms 513 B Save Response

Pretty Raw Preview Visualize JSON


```
{
    "_id": "\"63a0e57f6ab2c7a5cd49eb83\"",
    "adress": "tuzla",
    "cc_num": "454545454545454545",
    "cvc": "454",
    "exp": "03/25",
    "loyalty_point": "5",
    "name": "ahmet",
    "password": "$2b$12$PP4rMY3AppPiUx.GOEtLQ.FqwYHobyyv1XEg8um7LCYaXuNe/SWJ.",
    "payment": "mastercard"
```


But when we try with protected data postman shows only desired data:

GET http://127.0.0.1:5000/api/v2/customer/profile Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL

This request does not have a body

Body Cookies (1) Headers (6) Test Results 200 OK 249 ms 278 B Save Response

Pretty Raw Preview Visualize JSON


```
{
    "adress": "tuzla",
    "loyalty_point": "5",
    "site/app_preferences": "website/english",
    "username": "ahmet"
}
```


Testing with Static Code Analysers

PROSPECTOR RESULTS

Used commands: pip install prospector, prospector + where python file is located

```
PS C:\Users\ASUS\437_project> prospector C:\Users\ASUS\437_project\app.py
Messages
========

app.py
  Line: 1
    pylint: unused-import / Unused jsonify imported from flask
  Line: 7
    pylint: unused-import / Unused ExpiredSignatureError imported from jwt.exceptions
  Line: 8
    pylint: unused-import / Unused ObjectId imported from bson.objectid
  Line: 9
    pylint: unused-import / Unused import json
  Line: 22
    pycodestyle: E305 / expected 2 blank lines after class or function definition, found 1 (col 1)
  Line: 52
    pylint: unused-variable / Unused variable 'data' (col 12)
  Line: 55
    pycodestyle: E501 / line too long (198 > 159 characters) (col 160)
  Line: 70
    pylint: inconsistent-return-statements / Either all return statements in a function should return an expression, or none of them should.
  Line: 71
    pylint: no-else-return / Unnecessary "else" after "return", remove the "else" and de-indent the code inside it (col 4)
  Line: 85
    pylint: no-else-return / Unnecessary "else" after "return", remove the "else" and de-indent the code inside it (col 12)
  Line: 98
    pycodestyle: E501 / line too long (172 > 159 characters) (col 160)
  Line: 112
    pycodestyle: E117 / over-indented (col 9)
    pycodestyle: E501 / line too long (236 > 159 characters) (col 160)
    pylint: bad-indentation / Bad indentation. Found 8 spaces, expected 4
  Line: 114
    pylint: pointless-string-statement / String statement has no effect (col 8)
    pylint: bad-indentation / Bad indentation. Found 8 spaces, expected 4
  Line: 118
```

```
  Line: 118
    pycodestyle: E501 / line too long (199 > 159 characters) (col 160)
  Line: 123
    pylint: bad-indentation / Bad indentation. Found 8 spaces, expected 4
  Line: 138
    pylint: no-else-return / Unnecessary "else" after "return", remove the "else" and de-indent the code inside it (col 8)
    pylint: singleton-comparison / Comparison 'user_found == None' should be 'user_found is None' (col 11)
```

Prospector catches some unused variables in the code such as imports, non-returning in else part etc. And it suggested “is None” instead “== None”.

RATS (Rough-Auditing-Tool-for-Security) RESULTS

Used Commands:

wget https://rough-auditing-tool-for-security.googlecode.com/files/rats-2.4.tgz tar -xzvf rats-2.4.tgz,
cd rats-2.4,
./configure,
&& make && sudo make install
./rats

Due to some unexpected issues such as installation through file rats-2.4, i tried rats in kali linux. And @ character is warned in this code analyzer. Screenshot can be seen as below:

```
┌──(kali㉿kali)-[~/Desktop]
└─$ rats app.py
Entries in perl database: 33
Entries in ruby database: 46
Entries in python database: 62
Entries in c database: 334
Entries in php database: 55
Analyzing app.py
app.py:35: warning: bad token `@'
app.py:69: warning: bad token `@'
app.py:106: warning: bad token `@'
app.py:108: warning: bad token `@'
app.py:127: warning: bad token `@'
app.py:161: warning: bad token `@'
Total lines analyzed: 175
Total time 0.000795 seconds
220125 lines per second
```

BANDIT CODE ANALYZER RESULTS

Used Commands: virtualenv bandit-env,
python3 -m venv bandit-env,
pip install bandit
.\bandit-env\Scripts\activate.ps1
bandit -r + path of the code located

In this code analyzer, we can see that it captures hardcoded secret key or passwords. Screenshot can be seen as below:

```
PS C:\Users\ASUS\437_project> .\bandit-env\Scripts\activate.ps1
(bandit-env) PS C:\Users\ASUS\437_project> bandit -r app.py
[main]  INFO    profile include tests: None
[main]  INFO    profile exclude tests: None
[main]  INFO    cli include tests: None
[main]  INFO    cli exclude tests: None
[main]  INFO    running on Python 3.10.4
[node_visitor]  WARNING Unable to find qualified name for module: app.py
Run started:2023-01-02 22:59:05.262220

Test results:
>> Issue: [B105:hardcoded_password_string] Possible hardcoded password: 'secret'
   Severity: Low   Confidence: Medium
   CWE: CWE-259 (https://cwe.mitre.org/data/definitions/259.html)
   Location: app.py:22:11
   More Info: https://bandit.readthedocs.io/en/1.7.4/plugins/b105_hardcoded_password_string.html
21
22      app.config["JWT_SECRET_KEY"] = 'secret'
23      app.config["SECRET_KEY"] = 'secret2'

--------------------------------------------------
>> Issue: [B105:hardcoded_password_string] Possible hardcoded password: 'secret2'
   Severity: Low   Confidence: Medium
   CWE: CWE-259 (https://cwe.mitre.org/data/definitions/259.html)
   Location: app.py:23:11
   More Info: https://bandit.readthedocs.io/en/1.7.4/plugins/b105_hardcoded_password_string.html
22      app.config["JWT_SECRET_KEY"] = 'secret'
23      app.config["SECRET_KEY"] = 'secret2'
24

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

Code scanned:
        Total lines of code: 87
        Total lines skipped (#nosec): 0

Run metrics:
        Total issues (by severity):
                Undefined: 0
                Low: 2
                Medium: 0
                High: 0
        Total issues (by confidence):
                Undefined: 0
                Low: 0
                Medium: 2
                High: 0
Files skipped (0):
(bandit-env) PS C:\Users\ASUS\437_project>
```

PYT (Python Taint) CODE ANALYZER RESULTS

Used Commands: pip install python-taint, python -m pyt + location of python file,

In this analyzer, we got error called **“AttributeError: 'str' object has no attribute '_fields'”.** It seems that some string value is equal a json data field value, however, after researching in the code we tried all possible places by deleting all parts except where json data implementation is done then running the code analyzer, Interestingly, It gives “no vulnerabilities was found”. Screenshot can be seen as below:

```
PS C:\Users\ASUS\437_project> py -m pyt app.py
Traceback (most recent call last):
  File "C:\Users\ASUS\AppData\Local\Programs\Python\Python310\lib\runpy.py", line 196, in _run_module_as_main
    return _run_code(code, main_globals, None,
  File "C:\Users\ASUS\AppData\Local\Programs\Python\Python310\lib\runpy.py", line 86, in _run_code
    exec(code, run_globals)
  File "C:\Users\ASUS\AppData\Local\Programs\Python\Python310\lib\site-packages\pyt\__main__.py", line 156, in <module>
    main()
  File "C:\Users\ASUS\AppData\Local\Programs\Python\Python310\lib\site-packages\pyt\__main__.py", line 101, in main
    cfg = make_cfg(
  File "C:\Users\ASUS\AppData\Local\Programs\Python\Python310\lib\site-packages\pyt\cfg\make_cfg.py", line 36, in make_cfg
    visitor = ExprVisitor(
  File "C:\Users\ASUS\AppData\Local\Programs\Python\Python310\lib\site-packages\pyt\cfg\expr_visitor.py", line 69, in __init__
    self.init_cfg(node)
  File "C:\Users\ASUS\AppData\Local\Programs\Python\Python310\lib\site-packages\pyt\cfg\expr_visitor.py", line 76, in init_cfg
    module_statements = self.visit(node)
  File "C:\Users\ASUS\AppData\Local\Programs\Python\Python310\lib\ast.py", line 410, in visit
    return visitor(node)
  File "C:\Users\ASUS\AppData\Local\Programs\Python\Python310\lib\site-packages\pyt\cfg\stmt_visitor.py", line 67, in visit_Module
    return self.stmt_star_handler(node.body)
  File "C:\Users\ASUS\AppData\Local\Programs\Python\Python310\lib\site-packages\pyt\cfg\stmt_visitor.py", line 88, in stmt_star_handler
    node = self.visit(stmt)
  File "C:\Users\ASUS\AppData\Local\Programs\Python\Python310\lib\ast.py", line 410, in visit
    return visitor(node)
  File "C:\Users\ASUS\AppData\Local\Programs\Python\Python310\lib\site-packages\pyt\cfg\stmt_visitor.py", line 460, in visit_Assign
    label.visit(node)
  File "C:\Users\ASUS\AppData\Local\Programs\Python\Python310\lib\ast.py", line 410, in visit
    return visitor(node)
  File "C:\Users\ASUS\AppData\Local\Programs\Python\Python310\lib\site-packages\pyt\helper_visitors\label_visitor.py", line 52, in visit_Assign
    self.visit(target)
  File "C:\Users\ASUS\AppData\Local\Programs\Python\Python310\lib\ast.py", line 410, in visit
    return visitor(node)
  File "C:\Users\ASUS\AppData\Local\Programs\Python\Python310\lib\site-packages\pyt\helper_visitors\label_visitor.py", line 173, in visit_Subscript
    self.slicev(node.slice)
  File "C:\Users\ASUS\AppData\Local\Programs\Python\Python310\lib\site-packages\pyt\helper_visitors\label_visitor.py", line 190, in slicev
    self.visit(node.value)
  File "C:\Users\ASUS\AppData\Local\Programs\Python\Python310\lib\ast.py", line 410, in visit
    return visitor(node)
  File "C:\Users\ASUS\AppData\Local\Programs\Python\Python310\lib\ast.py", line 414, in generic_visit
    for field, value in iter_fields(node):
  File "C:\Users\ASUS\AppData\Local\Programs\Python\Python310\lib\ast.py", line 252, in iter_fields
    for field in node._fields:
AttributeError: 'str' object has no attribute '_fields'
PS C:\Users\ASUS\437_project>
```

When we tried only the some parts of the code where str object is equivalent to some fields we got below output:

```
PS C:\Users\ASUS\437_project> py -m pyt app.py
No vulnerabilities found.
PS C:\Users\ASUS\437_project>
```

Lastly, Above four code analyzer was not able to catch excessive data exposure vulnerability. They can catch some other vulnerabilities in the code such as hardcoded passwords, indentation, bad tokens, no returning inside else etc.

PYLINT CODE ANALYZER RESULTS (EXTRA CODE ANALYZER)

Used Commands: pip install pylint, pylint + path to the python file

In this analyzer some unused import is warned. Line too long, bad indentation comments are given. Moreover, some imports should be placed on another imports as lines 6 and 9 gives that warning. Also some snack-case naming in the code is warned in CONNECTION_STRING constant value and parameter inside token_required function. Snack case naming means that method name which equals to the variable should be shortened.

```
PS C:\Users\ASUS\437_project> pylint app.py
************* Module app
app.py:1:0: C0301: Line too long (101/100) (line-too-long)
app.py:3:31: C0303: Trailing whitespace (trailing-whitespace)
app.py:47:0: C0301: Line too long (128/100) (line-too-long)
app.py:55:0: C0301: Line too long (198/100) (line-too-long)
app.py:71:0: C0301: Line too long (109/100) (line-too-long)
app.py:71:0: C0325: Unnecessary parens after 'if' keyword (superfluous-parens)
app.py:85:0: C0301: Line too long (115/100) (line-too-long)
app.py:87:0: C0301: Line too long (128/100) (line-too-long)
app.py:92:0: C0301: Line too long (123/100) (line-too-long)
app.py:93:51: C0303: Trailing whitespace (trailing-whitespace)
app.py:98:0: C0301: Line too long (172/100) (line-too-long)
app.py:108:0: C0301: Line too long (116/100) (line-too-long)
app.py:112:0: C0301: Line too long (236/100) (line-too-long)
app.py:112:0: W0311: Bad indentation. Found 8 spaces, expected 4 (bad-indentation)
app.py:118:0: C0301: Line too long (199/100) (line-too-long)
app.py:114:0: W0311: Bad indentation. Found 8 spaces, expected 4 (bad-indentation)
app.py:123:0: C0301: Line too long (146/100) (line-too-long)
app.py:123:0: W0311: Bad indentation. Found 8 spaces, expected 4 (bad-indentation)
app.py:129:0: C0325: Unnecessary parens after 'if' keyword (superfluous-parens)
app.py:141:0: C0301: Line too long (103/100) (line-too-long)
app.py:174:0: C0303: Trailing whitespace (trailing-whitespace)
app.py:1:0: C0114: Missing module docstring (missing-module-docstring)
app.py:13:0: C0116: Missing function or method docstring (missing-function-docstring)
app.py:15:4: C0103: Variable name "CONNECTION_STRING" doesn't conform to snake_case naming style (invalid-name)
app.py:34:0: C0116: Missing function or method docstring (missing-function-docstring)
app.py:34:19: C0103: Argument name "f" doesn't conform to snake_case naming style (invalid-name)
app.py:52:12: W0612: Unused variable 'data' (unused-variable)
app.py:70:0: C0116: Missing function or method docstring (missing-function-docstring)
app.py:71:4: R1705: Unnecessary "else" after "return", remove the "else" and de-indent the code inside it (no-else-return)
app.py:85:12: R1705: Unnecessary "else" after "return", remove the "else" and de-indent the code inside it (no-else-return)
app.py:70:0: R1710: Either all return statements in a function should return an expression, or none of them should. (inconsistent-return-statements)
app.py:110:0: C0116: Missing function or method docstring (missing-function-docstring)
app.py:114:8: W0105: String statement has no effect (pointless-string-statement)
app.py:128:0: C0116: Missing function or method docstring (missing-function-docstring)
app.py:138:8: R1705: Unnecessary "else" after "return", remove the "else" and de-indent the code inside it (no-else-return)
app.py:138:11: C0121: Comparison 'user_found == None' should be 'user_found is None' (singleton-comparison)
app.py:162:0: C0116: Missing function or method docstring (missing-function-docstring)
app.py:5:0: C0411: standard import "from datetime import datetime, timedelta" should be placed before "from flask import Flask, render_template, request, red
irect, url_for, jsonify, make_response, session" (wrong-import-order)
app.py:6:0: C0411: standard import "from functools import wraps" should be placed before "from flask import Flask, render_template, request, redirect, url_fo
r, jsonify, make_response, session" (wrong-import-order)
app.py:9:0: C0411: standard import "import json" should be placed before "from flask import Flask, render_template, request, redirect, url_for, jsonify, make
_response, session" (wrong-import-order)
app.py:1:0: W0611: Unused jsonify imported from flask (unused-import)
app.py:7:0: W0611: Unused ExpiredSignatureError imported from jwt.exceptions (unused-import)
app.py:8:0: W0611: Unused ObjectId imported from bson.objectid (unused-import)
app.py:9:0: W0611: Unused import json (unused-import)

------------------------------------
Your code has been rated at 3.71/10
```

DEMO VIDEOS
https://drive.google.com/drive/folders/1n-K2cNrp-VvCX0-Gm4-Kl4j37VqcW1e7?usp=share_link

SOURCES THAT USED DURING IMPLEMENTATION

http://man.hubwiz.com/docset/MongoDB.docset/Contents/Resources/Documents/docs.mongodb.org/manual/tutorial/project-fields-from-query-results/index.html

https://stackoverflow.com/questions/34495632/how-to-implement-login-required-decorator-in-flask

https://htmx.org/docs/

https://stackabuse.com/integrating-mongodb-with-flask-using-flask-pymongo/

https://blog.miguelgrinberg.com/post/running-your-flask-application-over-https

https://www.educba.com/flask-https/

https://betterprogramming.pub/from-http-to-https-easily-secure-flask-web-apps-with-talisman-3359692d3eac

RESPONSIBILITY TABLE

AHMET ENES SILACI	I did flask implementation including database connection, server functions get-post requests, except data transmission encryption related issue, html-htmx usage, page related issues such as design etc, testing rats, bandit, python-taint, prospector and pylint code analyzers, finding protection to excessive data exposure and applying to the code, Trying vulnerable-protected code inside wireshark and postman, writing the report which i explained my responsibilities as above.
GÜRKAN TALHA SOYLU	Creating and utilizing SSL certificate for encryption of transmission.
OZAN YILDIRIM	explanation of HTTPS, SSL in the report part, and how tools could be used.