School registration system management project further explanations with relevant screenshots

Sorin Hadarau, C00279836

Webserver additional code explanations

Modern Cryptography module

Table of Contents

School registration system management project further explanations with relevant screenshots

1. Code improvement overview

1.1 Before the time extension

Before this extension for code improvement, in the project due time was achieved the following:

- This repository at https://github.com/itcmartin/cyber-students was forked as requested (https://github.com/SorinHadarau/cyber-students) and cloned to my local machine
- The steps were taken to setup the project as requested by installing all the necessary packages and establishing a connection with the created server functionalities on port 4000 at http://localhost:4000/students/api that was accessed via curl commands
- After verifying the basic functionalities such user registration details, login, logout, display profile, the aim was to secure the data records with some encryption solutions
- Up to due date of the shared project (3rd May) and intermediary solution was achieved by hashing all sensitive information recorded in database and looking something like that:

```
_id: ObjectId('626fd8fd6fbfdd58ccf6247f')
email: "foo@bar.com"
password: "0x320ad3fc605ad3f3"
displayName: "-0x7117c0977a5b1829"
expiresIn: 1651508023
token: null
```

- These basic recordings were able to be accessed, viewed, etc but the service was not fully functional by the time the project written document was uploaded (see more details with relevant screenshots of server achieved functionalities on the above-mentioned document)

1.2 After the time extension

After the time extension "for code only" (3rd – 8th May 2022) the following achievements were done code-wise:

- Issues (SOLVED) with error "no modules named ……" due to improper previous installation → a common problem with Windows OS
- Proper encoding of passwords with a PBKDF2 method that is a one-way hashing algorithm. It's not possible to decrypt the generated hash (according to latest literature)
- Increasing the registration capabilities by diversifying entries (adding "address", "phoneNumber" and "disabilities" fields to the previously three already existent in the sample code) and showing that all fields for registration can be added as necessary
- Added functionality to encode/decode fields as requested and completed the aimed goal of having a fully GDPR compliant server where data is encrypted securely with modern cryptographic methods (sample of user fields encryption below)


```
_id: ObjectId('6276454ff6f4f0bba15683ef')
email: Binary('Z0FBQUFBQmlka1ZQdTBQZHZWcWx4NGlMM1p5c1dXWk1jMnpqblFVaV9pQ3JUNmtjVGd1UzdoSWs4WGRuMnItV1hJRUZSb1NHOWpt…', 0)
password: "$pbkdf2-sha256$29000$DGGM0RoDQIhR6l3rvXcuJQ$5x6mXz.wNJPKj6oLrTJSI38PGx…"
displayName: Binary('Z0FBQUFBQmlka1ZQTUh6VUo5V25adWNXRWhIUXFNQWJlTHNhX0l1OTQ2aHNhRGdod05QRkR4WWxHOHhTQm9WNC1nRi13eVVqZ1F1…', 0)
address: Binary('Z0FBQUFBQmlka1ZQenlqc2ozRGhWSHNFMU53ekgyRm1ZcXBjVTQwdW1yalQ4Tlg0b2s1UzVwd1dVUVZnYm9JWC11UlJlZHFGN1Vh…', 0)
phoneNumber: Binary('Z0FBQUFBQmlka1ZQLUpTU1h5X3huaXMxSnlGbHZYSDhGRTJFMGF2c0RmTzVjSGNNWGhPUmh0TWlnYW5uaXpaVzV4Zlg1TmIxU19n…', 0)
disabilities: Binary('Z0FBQUFBQmlka1ZQcVA4SUcxVFhjRkN6VHBTdi1HczRpVEhsOUVJSFVRM1JVY2hhVnktNFhBVFlNdkh0RjFmUkcxVmg2XzYxT1Y4…', 0)
```


- Details of solutions and how all ties together in a functional manner on this web server are explained in a relevant way within the chapter that follows.

2. Solutions details and code explanations

Three main python files were modified, one additional file for cypher method used and key storage was added and some other minor improvements to curl commands and other particular Command Line Interface (CLI) commands that are more adapted for particularities of a Windows machine that was used to do this work.

The subchapters are as follows:

- Minor packets installation issues
- Registration.py file code improvements
- Login.py file code solution
- User.py file code solution

— Additional relevant information

2.1 Minor packets installation issues

During various library installation that were needed to experiment with cryptographic solutions, the command **pip3 install …** was not working properly and gave errors like “tornado module not found” when running **python3 run_server.py**. Another example is the one illustrated below where the command pip3 install passlib installed ‘successfully’ but inefficiently the library:

```
PS C:\Users\Admin\Desktop\Sorin Hadarau Cyber-students\cyber-students> project-venv\Scripts\activate
(project-venv) PS C:\Users\Admin\Desktop\Sorin Hadarau Cyber-students\cyber-students> pip3 install passlib
Collecting passlib
  Downloading passlib-1.7.4-py2.py3-none-any.whl (525 kB)
     ---------------------------------------- 525.6/525.6 KB 4.1 MB/s eta 0:00:00
Installing collected packages: passlib
Successfully installed passlib-1.7.4
(project-venv) PS C:\Users\Admin\Desktop\Sorin Hadarau Cyber-students\cyber-students>
```

I say ‘inefficiently’ because I had to rerun the installation with the alternative command **python3 -m pip install passlib** (because of error like ‘**no module named passlib**’):

```
PS C:\Users\Admin\Desktop\Sorin Hadarau Cyber-students\cyber-students> project-venv\Scripts\activate
(project-venv) PS C:\Users\Admin\Desktop\Sorin Hadarau Cyber-students\cyber-students> pip3 install passlib
Collecting passlib
  Downloading passlib-1.7.4-py2.py3-none-any.whl (525 kB)
     ---------------------------------------- 525.6/525.6 KB 4.1 MB/s eta 0:00:00
Installing collected packages: passlib
Successfully installed passlib-1.7.4
(project-venv) PS C:\Users\Admin\Desktop\Sorin Hadarau Cyber-students\cyber-students> python3 run_server.py
Traceback (most recent call last):
  File "C:\Users\Admin\Desktop\Sorin Hadarau Cyber-students\cyber-students\run_server.py", line 6, in <module>
    from api.app import Application
  File "C:\Users\Admin\Desktop\Sorin Hadarau Cyber-students\cyber-students\api\app.py", line 8, in <module>
    from .handlers.registration import RegistrationHandler
  File "C:\Users\Admin\Desktop\Sorin Hadarau Cyber-students\cyber-students\api\handlers\registration.py", line 1, in <module>
    from passlib.hash import pbkdf2_sha256
ModuleNotFoundError: No module named 'passlib'
(project-venv) PS C:\Users\Admin\Desktop\Sorin Hadarau Cyber-students\cyber-students> python3 -m pip install passlib
Collecting passlib
  Using cached passlib-1.7.4-py2.py3-none-any.whl (525 kB)
Installing collected packages: passlib
Successfully installed passlib-1.7.4
(project-venv) PS C:\Users\Admin\Desktop\Sorin Hadarau Cyber-students\cyber-students> python3 run_server.py
INFO:root:Starting server on port 4000...
```

All worked as expected after ‘proper’ reinstallation

2.2 Registration.py file code improvements

The first code solution needed was to improve the previously hashed password with a common hash function existent in python libraries and it gave me the same hash for the same input, which was not good enough. An IMPROVED solution was to use Passlib (PBKDF2) in Python that can offer a far better encrypted solution (according to various exert sources).

2.2.1 Password PBKDF2 type encryption

Short explanation for the chosen password encryption:

- PBKDF2 → Password-Based Key Derivation Function
- Not going too much into the gist of this method but I need to mention that the module used **pbkdf2_sha256.hash(password)** takes care by default about the 'salt' and the number of iterations (applies by default a 16 bytes salt and 29000 iterations to the hashed password)
- Every time we hash a password with this module, we get a different encoding, so it is made wildly resistant to brute-force and dictionary attacks

2.2.2 The password encryption code added to registration.py

The portion of code below shows the **pbkdf2_sha256.hash(password)** hashing encryption defined in code and also shows defined provisions for additional fields to be recorded in the database like 'address', 'phone_number', 'disabilities'

```
def post(self):
    try:
        body = json_decode(self.request.body)
        email = body['email'].lower().strip()
        if not isinstance(email, str):
            raise Exception()
        password = body['password']
        pbkdf2_sha256.hash(password)
        # hashed password
        hashed_pass = pbkdf2_sha256.hash(password)
        if not isinstance(password, str):
            raise Exception()
        display_name = body.get('displayName')
        if display_name is None:
            display_name = email
        if not isinstance(display_name, str):
            raise Exception()
        address = body['address']
        if not isinstance(address, str):
            raise Exception()
        phone_number = body['phoneNumber']
        if not isinstance(phone_number, str):
            raise Exception()
        disabilities = body['disabilities']
        if not isinstance(disabilities, str):
            raise Exception()
    except Exception as e:
```

Below is showed a portion of code where indications are five of how the curl inputs are written (and we see that next to ‘password’ is called the ‘hashed_pass’ function that encodes password as specified above.

```
yield self.db.users.insert_one({
    'email': email,
    'password': hashed_pass,
    'displayName': display_name,
    'address': address,
    'phoneNumber': phone_number,
    'disabilities': disabilities
```

— Curl registration command (modified accordingly as fields needed)

```
Admin@DESKTOP MINGW64 ~
$ curl -X POST http://localhost:4000/students/api/registration -d '{"email": "foo@bar.com", "password": "pass", "display
Name": "Foo Bar", "address": "address 123",  "phoneNumber": "12345678",  "disabilities": "disability 1, disability 2, di
sability 3"}'
{"email": "foo@bar.com", "displayName": "Foo Bar", "address": "address 123", "phoneNumber": "12345678", "disabilities":
"disability 1, disability 2, disability 3"}
```

— Database registration/recording as expected

```
_id: ObjectId('6274dfc0bb6f5e114e5cd32d')
email: "foo@bar.com"
password: "$pbkdf2-sha256$29000$uTdGaO0dg/D.XwvB2Juz1g$ijrtkhigALW.kG3oo7JpSewNT6…"
displayName: "Foo Bar"
address: "address 123"
phoneNumber: "12345678"
disabilities: "disability 1, disability 2, disability 3"
```

2.2.3 New file introduced named encrypt_decrypt.py

encrypt_decrypt.py

```
from cryptography.fernet import Fernet


key = Fernet.generate_key()
same_key = Fernet(key)

encrypting = same_key.encrypt(b'')

decrypting = str(same_key.decrypt(encrypting), 'UTF-8')
```

I stop a bit the further presentation of the regidtration.py file because it is necessary to explain a bit this new file introduced that serves as storage for my key and as foundation of

encrypting/decrypting code sample and source of Fernet Cryptography library that guarantees that a message encrypted using it cannot be manipulated or read without the key. Fernet is an implementation of symmetric (also known as “secret key”) authenticated cryptography.

2.2.3.1 Fernet encrypt/decrypt code solution explained

The variable **key = Fernet.generate_key()** generates a random key, and the problem with this key is that **every time** this function runs it generates a **different key**.

To encrypt/decrypt we need the same key. The solution to have the same key kept in this file was to store into the Fernet object **same_key = Fernet(key)**.

To encrypt I implemented the guide function **encryption = same_key.encrypt(b’text’)** where **same_key** is our stored key and **.encrypt()** is the encryption algorithm that takes in parenthesis the text/string to be encrypted. The **b** in **b’text’** transforms the text to be encrypted in bytes.

To decrypt I implemented a guide function (see below) which uses the encrypted text previously in bytes and decrypts it using the same key, and after decryption makes it a string UTF-8 encoded so it looks exactly as the input text (next chapter shows how the file it is used)

```
decrypting = str(same_key.decrypt(encrypting), 'UTF-8')
```

2.3 registration.py final modifications code comparison before and after

BEFORE

encrypt_decrypt.py | #registration - last version.py | #registration - initial version.py

```
1  from json import dumps
2  from logging import info
3  from tornado.escape import json_decode, utf8
4  from tornado.gen import coroutine
5
6  from .base import BaseHandler
7
8  class RegistrationHandler(BaseHandler):
```

AFTER

encrypt_decrypt.py | #registration - last version.py | #registration - initial version.py

```
1  from encrypt_decrypt import same_key
2  from passlib.hash import pbkdf2_sha256
3  from json import dumps
4  from logging import info
5  from tornado.escape import json_decode, utf8
6  from tornado.gen import coroutine
7
8  from .base import BaseHandler
9
10 class RegistrationHandler(BaseHandler):
```

BEFORE

encrypt_decrypt.py | #registration - last version.py | #registration - initial version.py

```python
    @coroutine
    def post(self):
        try:
            body = json_decode(self.request.body)
            email = body['email'].lower().strip()
            if not isinstance(email, str):
                raise Exception()
            password = body['password']
            if not isinstance(password, str):
                raise Exception()
            display_name = body.get('displayName')
            if display_name is None:
                display_name = email
            if not isinstance(display_name, str):
                raise Exception()
```

AFTER

encrypt_decrypt.py | #registration - last version.py | #registration - initial version.py

```python
    def post(self):
        try:
            body = json_decode(self.request.body)
            email = body['email'].lower().strip()
            encrypt_email = same_key.encrypt(email.encode())
            decrypt_name = str(same_key.decrypt(encrypt_email), 'UTF-8')
            if not isinstance(email, str):
                raise Exception()
            password = body['password']
            pbkdf2_sha256.hash(password)
            # hashed password
            hashed_pass = pbkdf2_sha256.hash(password)
            if not isinstance(password, str):
                raise Exception()
            display_name = body.get('displayName')
            encrypt_name = same_key.encrypt(display_name.encode())
            decrypt_name = str(same_key.decrypt(encrypt_name), 'UTF-8')
            if display_name is None:
                display_name = email
            if not isinstance(display_name, str):
                raise Exception()
            address = body['address']
            encrypt_address = same_key.encrypt(address.encode())
            decrypt_address = str(same_key.decrypt(encrypt_address), 'UTF-8')
            if not isinstance(address, str):
                raise Exception()
            phone_number = body['phoneNumber']
            encrypt_phone = same_key.encrypt(phone_number.encode())
            decrypt_phone = str(same_key.decrypt(encrypt_phone), 'UTF-8')
            if not isinstance(phone_number, str):
                raise Exception()
            disabilities = body['disabilities']
            encrypt_disabilities = same_key.encrypt(disabilities.encode())
            decrypt_disabilities = str(same_key.decrypt(encrypt_disabilities), 'UTF-8')
            if not isinstance(disabilities, str):
                raise Exception()
        except Exception as e:
```

We see that was introduced both encryption/decryption functions for each database entry definition. They can be called as desired (as we will see in further examples).

BEFORE

encrypt_decrypt.py | #registration - last version.py | #registration - initial version.py

```python
        except Exception as e:
            self.send_error(400, message='You must provide an email address, password and display name!')
            return

        if not email:
            self.send_error(400, message='The email address is invalid!')
            return

        if not password:
            self.send_error(400, message='The password is invalid!')
            return

        if not display_name:
            self.send_error(400, message='The display name is invalid!')
            return

        user = yield self.db.users.find_one({
          'email': email
        }, {})

        if user is not None:
            self.send_error(409, message='A user with the given email address already exists!')
            return

        yield self.db.users.insert_one({
            'email': email,
            'password': password,
            'displayName': display_name
        })

        self.set_status(200)
        self.response['email'] = email
        self.response['displayName'] = display_name

        self.write_json()
```

AFTER

encrypt_decrypt.py | #registration - last version.py | #registration - initial version.py

```python
        except Exception as e:
            self.send_error(400, message='You must provide an email address, password and display name!')
            return
        if not email:
            self.send_error(400, message='The email address is invalid!')
            return
        if not password:
            self.send_error(400, message='The password is invalid!')
            return
        if not display_name:
            self.send_error(400, message='The display name is invalid!')
            return
        if not address:
            self.send_error(400, message='The address is invalid!')
            return
        if not phone_number:
            self.send_error(400, message='The phone number is invalid!')
            return
        if not disabilities:
            self.send_error(400, message='Add N/A if no disabilities!')
            return

        user = yield self.db.users.find_one({
          'email': email
        }, {})

        if user is not None:
            self.send_error(409, message='A user with the given email address already exists!')
            return

        yield self.db.users.insert_one({
            'email': email,
            'password': hashed_pass,
            'displayName': encrypt_name,
            'address': encrypt_address,
            'phoneNumber': encrypt_phone,
            'disabilities': encrypt_disabilities

        })

        self.set_status(200)
        self.response['email'] = email
        self.response['displayName'] = display_name
        self.response['address'] = address
        self.response['phoneNumber'] = phone_number
        self.response['disabilities'] = disabilities

        self.write_json()
```

2.3.1 Confirming that the registration.py code works as expected

— Database encrypting all entries


```
_id: ObjectId('6276454ff6f4f0bba15683ef')
email: Binary('Z0FBQUFBQmlka1ZQdTBQZHZWcWx4NGlMM1p5c1dXWk1jMnpqblFVaV9pQ3JUNmtjVGd1UzdoSWs4WGRuMnItV1hJRUZSb1NHOWpt…', 0)
password: "$pbkdf2-sha256$29000$DGGM0RoDQIhR6l3rvXcuJQ$5x6mXz.wNJPKj6oLrTJSI38PGx…"
displayName: Binary('Z0FBQUFBQmlka1ZQTUh6VUo5V25adWNXRWhIUXFNQWJlTHNhX0l1OTQ2aHNhRGdod05QRkR4WWxHOHhTQm9WNC1nRil3eVVqZ1F1…', 0)
address: Binary('Z0FBQUFBQmlka1ZQenlqc2ozRGhWSHNFMU53ekgyRm1ZcXBjVTQwdW1yalQ4Tlg0b2slUzVwd1dVUVZnYm9JWC11UlJlZHFGN1Vh…', 0)
phoneNumber: Binary('Z0FBQUFBQmlka1ZQLUpTU1h5X3huaXMxSnlGbHZYSDhGRTJFMGF2c0RmTzVjSGNNWGhPUmh0TWlnYW5uaXpaVzV4Zlg1TmIxU19n…', 0)
disabilities: Binary('Z0FBQUFBQmlka1ZQcVA4SUcxVFhjRkN6VHBTdilHczRpVEhsOUVJSFVRM1JVY2hhVnktNFhBVFlNdkh0RjFmUkcxVmg2XzYxT1Y4…', 0)
```


Observation: I needed to let the email field un-encrypted afterwards, because I could not find a solution to login via curl if the email is encrypted.

```
yield self.db.users.insert_one({
    'email': email,
    'password': hashed_pass,
    'displayName': encrypt_name,
    'address': encrypt_address,
    'phoneNumber': encrypt_phone,
    'disabilities': encrypt_disabilities
```

— Database entries encrypted


```
_id: ObjectId('627793a976f38e2fd158552b')
email: "foo@bar.com"
password: "$pbkdf2-sha256$29000$whhjTAmh9F5LiVGqFQLA2A$EHGfN2Fo1eFHf.y76WD3sI939s…"
displayName: Binary('Z0FBQUFBQmlkNU9wRmZfUUl0TDBlYUtIYXQxblgyc3RDTnhIZE11SEJuMXhsUEVQTkxkemVoQ2ZtUWVEbGR3TGVrZDYyV0tjRXUt…', 0)
address: Binary('Z0FBQUFBQmlkNU9wNS1SQXJtTlAtNmcwcEluY3F5TGhHSmU1R2NNRElaa3NlczVuZzFONERINVQzR2VodVlyNTF4WU5kbC1HV0RG…', 0)
phoneNumber: Binary('Z0FBQUFBQmlkNU9wTzBLSjVYeTRtUVJwMDdkMG9rbjFzZ3NfQU5panYwQzg3R1RlUkdmTW1HR2pjT05VQXBTRklobklyVFhRR2NL…', 0)
disabilities: Binary('Z0FBQUFBQmlkNU9wTDdKRkVXYnpmVVE3Wm13T3RRZzdXaXNJNEthSnRkMi00V0J6SDNMelV3dk1OcnVxRVhLakNJZXBVb3FOX00x…', 0)
expiresIn: 1652014568
token: null
```


CURL commands shown the posting as above, and the successful login, show/access data and logout even though **the password is always encrypted, and the name and other personal details are always encrypted in the database**.

```
Admin@DESKTOP MINGW64 ~
$ curl -X POST http://localhost:4000/students/api/registration -d '{"email": "foo@bar.com", "password": "pass", "display
Name": "Foo Bar", "address": "address 123",  "phoneNumber": "12345678",  "disabilities": "disability 1, disability 2, di
sability 3"}'
{"email": "foo@bar.com", "displayName": "Foo Bar", "address": "address 123", "phoneNumber": "12345678", "disabilities":
"disability 1, disability 2, disability 3"}
Admin@DESKTOP MINGW64 ~
$ curl -X POST http://localhost:4000/students/api/login -d '{"email": "foo@bar.com", "password": "pass"}'
{"token": "e72ccc61286440868863198ca8940a2f", "expiresIn": 1652014568.0}
Admin@DESKTOP MINGW64 ~
$ curl -H "X-TOKEN: e72ccc61286440868863198ca8940a2f" http://localhost:4000/students/api/user
{"email": "foo@bar.com", "displayName": "Foo Bar"}
Admin@DESKTOP MINGW64 ~
$ curl -X POST -H "X-TOKEN: e72ccc61286440868863198ca8940a2f" http://localhost:4000/students/api/logout
{}
Admin@DESKTOP MINGW64 ~
$
```

2.4 Login.py file code solution

Let's see what change was done on login.py file to achieve the login when the password is encrypted with **pbkdf2_sha256.hash(password)** →Login.py initial and final in comparison:

BRFORE

#login - initial version.py | #login - last version.py | #user - initial version.py

```
1  from datetime import datetime, timedelta
2  from time import mktime
3  from tornado.escape import json_decode, utf8
4  from tornado.gen import coroutine
5  from uuid import uuid4
6
7  from .base import BaseHandler
8
9  class LoginHandler(BaseHandler):
```

AFTER

#login - initial version.py | #login - last version.py | #user - initial version.py

```
1  from passlib.hash import pbkdf2_sha256
2  from datetime import datetime, timedelta
3  from time import mktime
4  from tornado.escape import json_decode, utf8
5  from tornado.gen import coroutine
6  from uuid import uuid4
7
8  from .base import BaseHandler
9
10 class LoginHandler(BaseHandler):
```

BEFORE

#login - initial version.py | #login - last version.py | #user - initial version.py | #user - last version.py

```
40         except:
41             self.send_error(400, message='You must provide an email address and password!')
42             return
43
44         if not email:
45             self.send_error(400, message='The email address is invalid!')
46             return
47
48         if not password:
49             self.send_error(400, message='The password is invalid!')
50             return
51
52         user = yield self.db.users.find_one({
53             'email': email
54         }, {
55             'password': 1
56         })
57
58         if user is None:
59             self.send_error(403, message='The email address and password are invalid!')
60             return
61
62         if user['password'] != password:
63             self.send_error(403, message='The email address and password are invalid!')
64             return
65
66         token = yield self.generate_token(email)
67
68         self.set_status(200)
69         self.response['token'] = token['token']
70         self.response['expiresIn'] = token['expiresIn']
71
72         self.write_json()
```

AFTER

#login - initial version.py | #login - last version.py | #user - initial version.py | #user - last version.py

```
40                  raise Exception()
41          except:
42              self.send_error(400, message='You must provide an email address and password!')
43              return
44
45          if not email:
46              self.send_error(400, message='The email address is invalid!')
47              return
48
49          if not password:
50              self.send_error(400, message='The password is invalid!')
51              return
52
53          user = yield self.db.users.find_one({
54              'email': email
55          }, {
56              'password': 1
57          })
58
59          if user is None:
60              self.send_error(403, message='The email address and password are invalid!')
61              return
62
63          if not pbkdf2_sha256.verify(password, user['password']):
64              self.send_error(403, message='The email address and password are invalid!')
65              return
66
67          token = yield self.generate_token(email)
68
69          self.set_status(200)
70          self.response['token'] = token['token']
71          self.response['expiresIn'] = token['expiresIn']
72
73          self.write_json()
74
```

```
if not pbkdf2_sha256.verify(password, user['password']):
    self.send_error(403, message='The email address and password are invalid!')
    return
```

Having a closer look at the sample code above it is seen that PBKDF2 one-way encoding has a **.verify** function that works her magic into identifying that the password entered by user in unencoded text is the same as the encoded one in the database. This video https://www.youtube.com/watch?v=U7FaYdxZLA4 helped me craft a solution for my own problem. I mention it because there are very few knowing about this particular solution online.

The mentioned video above on YouTube is the author understanding of the coding principles explained at https://passlib.readthedocs.io/en/stable/lib/passlib.hash.pbkdf2_digest.html (PassLib, 2020)

2.5 User.py file code solution

Showing below only the modified part in comparison with the earlier version

BEFORE

```
#login - initial version.py | #login - last version.py | #user - initial version.py | #user - last version.py
1   from tornado.escape import json_decode, utf8
2   from tornado.gen import coroutine
3   from tornado.web import authenticated
4
5   from .auth import AuthHandler
6
7   class UserHandler(AuthHandler):
8
9       @authenticated
10      def get(self):
11          self.set_status(200)
12          self.response['email'] = self.current_user['email']
13          self.response['displayName'] = self.current_user['display_name']
14          self.write_json()
15
16      @coroutine
```

AFTER

```
#login - initial version.py | #login - last version.py | #user - initial version.py | #user - last version.py
1   from encrypt_decrypt import same_key
2   from tornado.escape import json_decode, utf8
3   from tornado.gen import coroutine
4   from tornado.web import authenticated
5
6   from .auth import AuthHandler
7
8   class UserHandler(AuthHandler):
9
10      @authenticated
11      def get(self):
12          self.set_status(200)
13          self.response['email'] = self.current_user['email']
14          self.response['displayName'] = str(same_key.decrypt(self.current_user['display_name']), 'UTF-8')
15
16          self.write_json()
17
18      @coroutine
```

Having a closer look to before/after situation is clear that portion of code from encrypt_decrypt.py file was used to successfully decrypt the name stored in the database.

```
self.response['displayName'] = self.current_user['display_name']
```

```
self.response['displayName'] = str(same_key.decrypt(self.current_user['display_name']), 'UTF-8')
```

This example code above can be easily replicated as needed for decoding as many encoded details as we want.

For example ,if we want to visualise the encrypted stored address, we can add in the appropriate position on **user.py** file:

self.response['address'] = str(same_key.decrypt(self.current_user['address']), 'UTF-8')

2.6 Additional relevant information

2.6.1 Working code demonstration

curl commands (da capo al fine)

$ curl -X POST http://localhost:4000/students/api/registration -d '{"email": "foo@bar.com", "password": "pass", "displayName": "Foo Bar", "address": "address 123", "phoneNumber": "12345678", "disabilities": "disability 1, disability 2, disability 3"}'

```
Admin@DESKTOP MINGW64 ~
$ curl -X POST http://localhost:4000/students/api/registration -d '{"email": "foo@bar.com", "password": "pass", "display
Name": "Foo Bar", "address": "address 123", "phoneNumber": "12345678", "disabilities": "disability 1, disability 2, di
sability 3"}'
{"email": "foo@bar.com", "displayName": "Foo Bar", "address": "address 123", "phoneNumber": "12345678", "disabilities":
"disability 1, disability 2, disability 3"}
Admin@DESKTOP MINGW64 ~
$ curl -X POST http://localhost:4000/students/api/login -d '{"email": "foo@bar.com", "password": "pass"}'
{"token": "ecbf6385b7a6430980e6a518a8d76456", "expiresIn": 1652050203.0}
Admin@DESKTOP MINGW64 ~
$ curl -H "X-TOKEN: ecbf6385b7a6430980e6a518a8d76456" http://localhost:4000/students/api/user
{"email": "foo@bar.com", "displayName": "Foo Bar"}
Admin@DESKTOP MINGW64 ~
$ curl -X POST -H "X-TOKEN: ecbf6385b7a6430980e6a518a8d76456" http://localhost:4000/students/api/logout
{}
Admin@DESKTOP MINGW64 ~
$
```

Corresponding database recording


```
_id: ObjectId('62781eddc6b14c2241c45464')
email: "foo@bar.com"
password: "$pbkdf2-sha256$29000$Solxbk3JeW8tJQSAcG4tZQ$Uh9rGvljBR4TKMOLRcwAGyeDzf…"
displayName: Binary('Z0FBQUFBQmllQjdkalVQcGF6a1ZvVDZHSXR6MHdNS3BfakRtOHpxWEdqOW1JcmFYdWxON1ZQVnFvTnJIV1RJZThWVkppT3RmUzNl…', 0)
address: Binary('Z0FBQUFBQmllQjdkc19KZERiZXFDRlhBWkJmYWZlOUhnLV9xZE5wUGY2SHBPNmNqUlpERDBxN3ZtN2JVLUNPbjMyMzRod2NmZUtl…', 0)
phoneNumber: Binary('Z0FBQUFBQmllQjdkc0V1X00wMS1IdWhlWWdVYXE0VHlaUnFuNW1aSEZIWGttajY2OUJiQjlFc1I4YnpUTExBdkdhcGw5OE9RSjVa…', 0)
disabilities: Binary('Z0FBQUFBQmllQjdkZXVYVUdRV2VZZGxaTERQSE5nM0twamlob2JUS0RGNTJCajlNYWJUTTFDcnF2cDVUM0ZEZmxfbVdwTXRTTHB6…', 0)
expiresIn: 1652050203
token: null
```


Visualisation on CLI

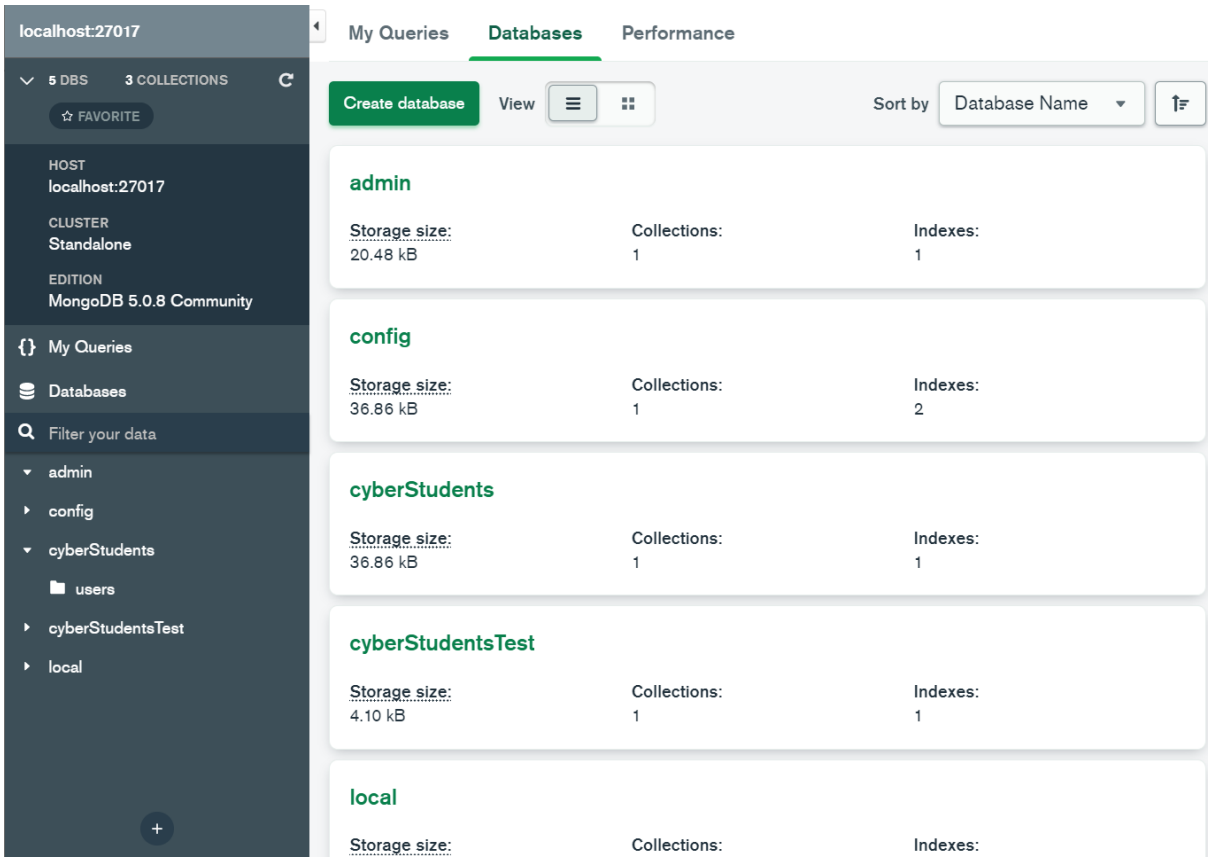
```
PS C:\Users\Admin\Desktop\Sorin Hadarau Cyber-students\cyber-students> project-venv\Scripts\activate
(project-venv) PS C:\Users\Admin\Desktop\Sorin Hadarau Cyber-students\cyber-students> python3 run_server.py
INFO:root:Starting server on port 4000...
INFO:tornado.access:200 POST /students/api/registration (127.0.0.1) 54.37ms
INFO:tornado.access:200 POST /students/api/login (127.0.0.1) 30.56ms
INFO:tornado.access:200 GET /students/api/user (127.0.0.1) 3.54ms
INFO:tornado.access:200 POST /students/api/logout (127.0.0.1) 3.02ms
```

Hacker.py running

```
PS C:\Users\Admin\Desktop\Sorin Hadarau Cyber-students\cyber-students> python3 run_hacker.py list
There are 1 registered users:
{'_id': ObjectId('62781eddc6b14c2241c45464'), 'email': 'foo@bar.com', 'password': '$pbkdf2-sha256$29000$Solxbk3JeW8tJQSAcG4tZQ$Uh9rGvljBR4TKMOLRcwAGye
Dzfa5fJiHxtWCTtTXZi4', 'displayName': b'gAAAAABieB7djUPpazkVoT6GItz0wMKp_jDm8zqXGj9mIraXulN7VPVqoNrHWTIe8VVJiOtfS3eluOH5IE__F3e34QR9HX5LvA==', 'addres
s': b'gAAAAABieB7ds_JdDbeqCFXAZBfafe9Hg-_qdNpPf6HpO6cjRZDD0q7vm7bU-COn3234hwcfeKeX-fJUTm-WNzDdNF2QcCYV1g==', 'phoneNumber': b'gAAAAABieB7dsEu_M01-Huhe
YgUaq4TyZRqn5mZHFHXkmj669BbB9EsR8bzTLLAvGapl98OQJ5Zdnj1uH8tgJnqH1ziCym328g==', 'disabilities': b'gAAAAABieB7deuXUGQWeYdlZLDPHNg3KpjihobTKDF52Bj9MabTM1
Crqvp5T3FDfl_mWpMtSlpzNr47rY8mBUvWibYcw9iQUp_N8Y6tQWPUUyt4HIx2r8IF-jryxe97XYAPGu1psoVJb'}
PS C:\Users\Admin\Desktop\Sorin Hadarau Cyber-students\cyber-students>
```

```
(project-venv) PS C:\Users\Admin\Desktop\Sorin Hadarau Cyber-students\cyber-students> python3 run_hacker.py list
There are 1 registered users:
{'_id': ObjectId('62781eddc6b14c2241c45464'), 'email': 'foo@bar.com', 'password': '$pbkdf2-sha256$29000$Solxbk3JeW8tJQSAcG4tZQ$Uh9rGvljBR4TKMOLRcwAGye
Dzfa5fJiHxtWCTtTXZi4', 'displayName': b'gAAAAABieB7djUPpazkVoT6GItz0wMKp_jDm8zqXGj9mIraXulN7VPVqoNrHWTIe8VVJiOtfS3eluOH5IE__F3e34QR9HX5LvA==', 'addres
s': b'gAAAAABieB7ds_JdDbeqCFXAZBfafe9Hg-_qdNpPf6HpO6cjRZDD0q7vm7bU-COn3234hwcfeKeX-fJUTm-WNzDdNF2QcCYV1g==', 'phoneNumber': b'gAAAAABieB7dsEu_M01-Huhe
YgUaq4TyZRqn5mZHFHXkmj669BbB9EsR8bzTLLAvGapl98OQJ5ZdnjluH8tgJnqH1ziCym328g==', 'disabilities': b'gAAAAABieB7deuXUGQWeYdlZLDPHNg3KpjihobTKDF52Bj9MabTM1
Crqvp5T3FDfl_mWpMtSLpzNr47rY8mBUvWibYcw9iQUp_N8Y6tQWPUUyt4HIx2r8IF-jryxe97XYAPGu1psoVJb'}
(project-venv) PS C:\Users\Admin\Desktop\Sorin Hadarau Cyber-students\cyber-students>
```

2.6.2 MongoDB Compass database overview



2.6.3 Short reflection

I found this exercise very difficult especially because I don't have a background in coding, but the successful set-up of the basic functionalities of the server motivated me to keep going and use logic and try/error steps (hundreds of them) until I fully understood how the code works, how it should work and how to adapt/implement what I've learned.

Overall, this was a very meaningful experience (at least from the cryptographic aspect of it) because I could use my brain to learn, observe, try, retry, and finally implement solutions that, even though not perfect, at least they are a success for my level and I am proud I did not give up. I hope my work can be understood the way I do and it can be reproduced with the same results I had when running my code adventure on the realm of cryptography.

2.6.4 Latest versions of Python files modified with this exercise

#registration - last version.py

```python
from encrypt_decrypt import same_key
from passlib.hash import pbkdf2_sha256
from json import dumps
from logging import info
from tornado.escape import json_decode, utf8
from tornado.gen import coroutine

from .base import BaseHandler

class RegistrationHandler(BaseHandler):

    @coroutine
    def post(self):
        try:
            body = json_decode(self.request.body)
            email = body['email'].lower().strip()
            encrypt_email = same_key.encrypt(email.encode())
            decrypt_name = str(same_key.decrypt(encrypt_email), 'UTF-8')
            if not isinstance(email, str):
                raise Exception()
            password = body['password']
            pbkdf2_sha256.hash(password)
            # hashed password
            hashed_pass = pbkdf2_sha256.hash(password)
            if not isinstance(password, str):
                raise Exception()
            display_name = body.get('displayName')
            encrypt_name = same_key.encrypt(display_name.encode())
            decrypt_name = str(same_key.decrypt(encrypt_name), 'UTF-8')
            if display_name is None:
                display_name = email
            if not isinstance(display_name, str):
                raise Exception()
            address = body['address']
            encrypt_address = same_key.encrypt(address.encode())
            decrypt_address = str(same_key.decrypt(encrypt_address), 'UTF-8')
            if not isinstance(address, str):
                raise Exception()
            phone_number = body['phoneNumber']
            encrypt_phone = same_key.encrypt(phone_number.encode())
            decrypt_phone = str(same_key.decrypt(encrypt_phone), 'UTF-8')
            if not isinstance(phone_number, str):
                raise Exception()
            disabilities = body['disabilities']
            encrypt_disabilities = same_key.encrypt(disabilities.encode())
            decrypt_disabilities = str(same_key.decrypt(encrypt_disabilities), 'UTF-8')
            if not isinstance(disabilities, str):
                raise Exception()
        except Exception as e:
            self.send_error(400, message='You must provide an email address, password and display name!')
            return
        if not email:
            self.send_error(400, message='The email address is invalid!')
            return
        if not password:
            self.send_error(400, message='The password is invalid!')
            return
        if not display_name:
            self.send_error(400, message='The display name is invalid!')
            return
        if not address:
            self.send_error(400, message='The address is invalid!')
            return
        if not phone_number:
            self.send_error(400, message='The phone number is invalid!')
            return
        if not disabilities:
            self.send_error(400, message='Add N/A if no disabilities!')
            return

        user = yield self.db.users.find_one({
          'email': email
        }, {})

        if user is not None:
            self.send_error(409, message='A user with the given email address already exists!')
            return

        yield self.db.users.insert_one({
            'email': email,
            'password': hashed_pass,
            'displayName': encrypt_name,
            'address': encrypt_address,
            'phoneNumber': encrypt_phone,
            'disabilities': encrypt_disabilities

        })

        self.set_status(200)
        self.response['email'] = email
        self.response['displayName'] = display_name
        self.response['address'] = address
        self.response['phoneNumber'] = phone_number
        self.response['disabilities'] = disabilities

        self.write_json()
```

#login - last version.py

```python
from passlib.hash import pbkdf2_sha256
from datetime import datetime, timedelta
from time import mktime
from tornado.escape import json_decode, utf8
from tornado.gen import coroutine
from uuid import uuid4

from .base import BaseHandler

class LoginHandler(BaseHandler):

    @coroutine
    def generate_token(self, email):
        token_uuid = uuid4().hex
        expires_in = datetime.now() + timedelta(hours=2)
        expires_in = mktime(expires_in.utctimetuple())

        token = {
            'token': token_uuid,
            'expiresIn': expires_in,
        }

        yield self.db.users.update_one({
            'email': email
        }, {
            '$set': token
        })

        return token

    @coroutine
    def post(self):
        try:
            body = json_decode(self.request.body)
            email = body['email'].lower().strip()
            if not isinstance(email, str):
                raise Exception()
            password = body['password']
            if not isinstance(password, str):
                raise Exception()
        except:
            self.send_error(400, message='You must provide an email address and password!')
            return

        if not email:
            self.send_error(400, message='The email address is invalid!')
            return

        if not password:
            self.send_error(400, message='The password is invalid!')
            return

        user = yield self.db.users.find_one({
            'email': email
        }, {
            'password': 1
        })

        if user is None:
            self.send_error(403, message='The email address and password are invalid!')
            return

        if not pbkdf2_sha256.verify(password, user['password']):
            self.send_error(403, message='The email address and password are invalid!')
            return

        token = yield self.generate_token(email)

        self.set_status(200)
        self.response['token'] = token['token']
        self.response['expiresIn'] = token['expiresIn']

        self.write_json()
```

#user - initial version.py

```python
from tornado.escape import json_decode, utf8
from tornado.gen import coroutine
from tornado.web import authenticated

from .auth import AuthHandler

class UserHandler(AuthHandler):

    @authenticated
    def get(self):
        self.set_status(200)
        self.response['email'] = self.current_user['email']
        self.response['displayName'] = self.current_user['display_name']
        self.write_json()

    @coroutine
    @authenticated
    def put(self):
        try:
            body = json_decode(self.request.body)
            display_name = body['displayName']
            if not isinstance(display_name, str):
                raise Exception()
        except:
            self.send_error(400, message='You must provide a display name!')
            return

        if not display_name:
            self.send_error(400, message='The display name is invalid!')
            return

        yield self.db.users.update_one({
            'email': self.current_user['email'],
        }, {
            '$set': {
                'displayName': display_name
            }
        })

        self.current_user['display_name'] = display_name

        self.set_status(200)
        self.response['email'] = self.current_user['email']
        self.response['displayName'] = self.current_user['display_name']
        self.write_json()
```

References

Harrigan, M., (2022.) *Shared Project for Modern Cryptography and Security Management & Compliance.* [Online]
Available at: https://github.com/itcmartin/cyber-students
[Accessed 26 April 2022].

PassLib, (2020). *passlib.hash.pbkdf2_digest - Generic PBKDF2 Hashes*¶. [Online]
Available at: https://passlib.readthedocs.io/en/stable/lib/passlib.hash.pbkdf2_digest.html
[Accessed 5 May 2022].