(/)

Curriculum
Short Specializations
Average: **57.57%**

You just released the advanced tasks of this project. Have fun!

0x03. User authentication service

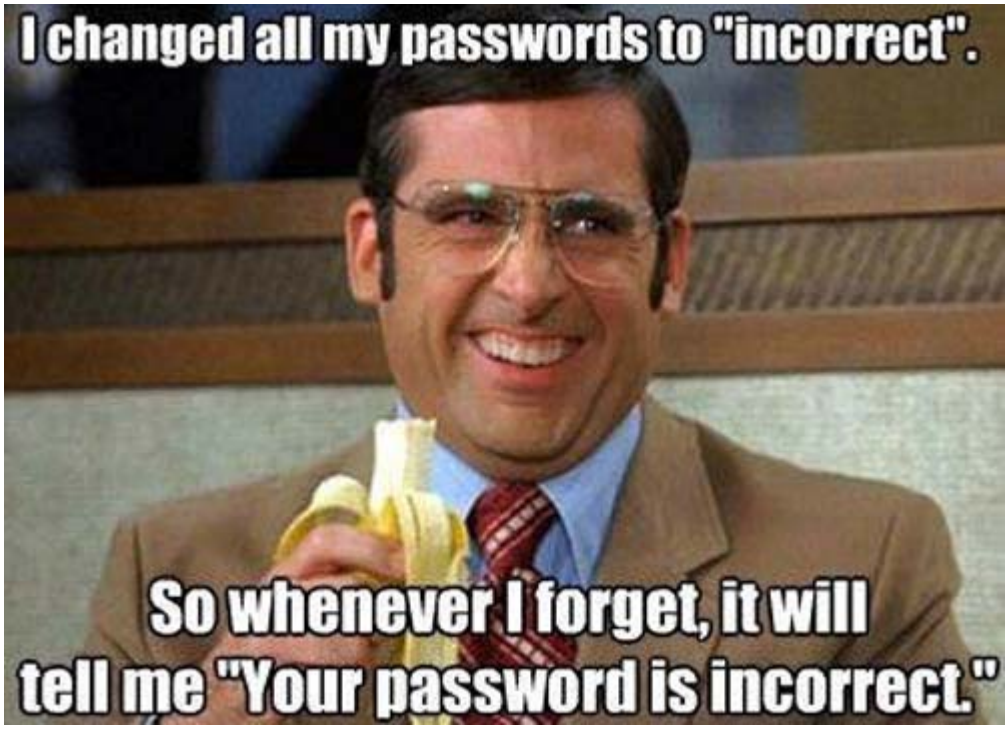
Back-end **Authentification**

Weight: 1

Project will start Aug 12, 2024 4:00 AM, must end by Aug 16, 2024 4:00 AM

Checker was released at Aug 13, 2024 4:00 AM

An auto review will be launched at the deadline



In the industry, you should **not** implement your own authentication system and use a module or framework that doing it for you (like in Python-Flask: Flask-User (/rltoken/9nVfotMI_1zpEzihMzBeTA)). Here, for the learning purpose, we will walk through each step of this mechanism to understand it by doing.

Resources

Read or watch:

- Flask documentation (/rltoken/lKExyvivrrW4eh0el8UV6A)
- Requests module (/rltoken/py7LuuD1u2MUwcaf8wnDzQ)
- HTTP status codes (/rltoken/cj-mc5ZHp_KyXn1yikHC0A)

Learning Objectives

At the end of this project, you are expected to be able to explain to anyone (/rltoken/oAqmZmipBdjCcfI5QqyFXA), **without the help of Google**:

- How to declare API routes in a Flask app
- How to get and set cookies
- How to retrieve request form data
- How to return various HTTP status codes

Requirements

- Allowed editors: `vi`, `vim`, `emacs`
- All your files will be interpreted/compiled on Ubuntu 18.04 LTS using `python3` (version 3.7)
- All your files should end with a new line
- The first line of all your files should be exactly `#!/usr/bin/env python3`
- A `README.md` file, at the root of the folder of the project, is mandatory
- Your code should use the `pycodestyle` style (version 2.5)
- You should use `SQLAlchemy` 1.3.x
- All your files must be executable
- The length of your files will be tested using `wc`
- All your modules should have a documentation (`python3 -c 'print(__import__("my_module").__doc__)'`)
- All your classes should have a documentation (`python3 -c 'print(__import__("my_module").MyClass.__doc__)'`)
- All your functions (inside and outside a class) should have a documentation (`python3 -c 'print(__import__("my_module").my_function.__doc__)'` and `python3 -c 'print(__import__("my_module").MyClass.my_function.__doc__)'`)
- A documentation is not a simple word, it's a real sentence explaining what's the purpose of the module, class or method (the length of it will be verified)
- All your functions should be type annotated
- The flask app should only interact with `Auth` and never with `DB` directly.
- Only public methods of `Auth` and `DB` should be used outside these classes

Setup

You will need to install `bcrypt`

```
pip3 install bcrypt
```

Tasks

0. User model

mandatory

In this task you will create a SQLAlchemy model named `User` for a database table named `users` (by using the mapping declaration (/rltoken/-a69l-rGqoFdXnnu6qfKdA) of SQLAlchemy).

The model will have the following attributes:

- `id` , the integer primary key
- `email` , a non-nullable string
- `hashed_password` , a non-nullable string
- `session_id` , a nullable string
- `reset_token` , a nullable string

```
bob@dylan:~$ cat main.py
#!/usr/bin/env python3
"""
Main file
"""
from user import User

print(User.__tablename__)

for column in User.__table__.columns:
    print("{}: {}".format(column, column.type))

bob@dylan:~$ python3 main.py
users
users.id: INTEGER
users.email: VARCHAR(250)
users.hashed_password: VARCHAR(250)
users.session_id: VARCHAR(250)
users.reset_token: VARCHAR(250)
bob@dylan:~$
```

Repo:

- GitHub repository: `alx-backend-user-data`
- Directory: `0x03-user_authentication_service`
- File: `user.py`

Done? | Check your code | Get a sandbox

1. create user

mandatory

In this task, you will complete the `DB` class provided below to implement the `add_user` method.

```
"""DB module
"""
from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker
from sqlalchemy.orm.session import Session

from user import Base


class DB:
    """DB class
    """

    def __init__(self) -> None:
        """Initialize a new DB instance
        """
        self._engine = create_engine("sqlite:///a.db", echo=True)
        Base.metadata.drop_all(self._engine)
        Base.metadata.create_all(self._engine)
        self.__session = None

    @property
    def _session(self) -> Session:
        """Memoized session object
        """
        if self.__session is None:
            DBSession = sessionmaker(bind=self._engine)
            self.__session = DBSession()
        return self.__session
```

Note that `DB._session` is a private property and hence should NEVER be used from outside the `DB` class.

Implement the `add_user` method, which has two required string arguments: `email` and `hashed_password`, and returns a `User` object. The method should save the user to the database. No validations are required at this stage.

(7)

```
bob@dylan:~$ cat main.py
#!/usr/bin/env python3
"""
Main file
"""

from db import DB
from user import User

my_db = DB()

user_1 = my_db.add_user("test@test.com", "SuperHashedPwd")
print(user_1.id)

user_2 = my_db.add_user("test1@test.com", "SuperHashedPwd1")
print(user_2.id)

bob@dylan:~$ python3 main.py
1
2
bob@dylan:~$
```

Repo:

- GitHub repository: `alx-backend-user-data`
- Directory: `0x03-user_authentication_service`
- File: `db.py`

Done? | Check your code | Get a sandbox

2. Find user

mandatory

In this task you will implement the `DB.find_user_by` method. This method takes in arbitrary keyword arguments and returns the first row found in the `users` table as filtered by the method's input arguments. No validation of input arguments required at this point.

Make sure that SQLAlchemy's `NoResultFound` and `InvalidRequestError` are raised when no results are found, or when wrong query arguments are passed, respectively.

Warning:

- `NoResultFound` has been moved from `sqlalchemy.orm.exc` to `sqlalchemy.exc` between the version 1.3.x and 1.4.x of SQLAchemy - please make sure you are importing it from `sqlalchemy.orm.exc`

```
bob@dylan:~$ cat main.py
#!/usr/bin/env python3
"""
Main file
"""
from db import DB
from user import User

from sqlalchemy.exc import InvalidRequestError
from sqlalchemy.orm.exc import NoResultFound


my_db = DB()

user = my_db.add_user("test@test.com", "PwdHashed")
print(user.id)

find_user = my_db.find_user_by(email="test@test.com")
print(find_user.id)

try:
    find_user = my_db.find_user_by(email="test2@test.com")
    print(find_user.id)
except NoResultFound:
    print("Not found")

try:
    find_user = my_db.find_user_by(no_email="test@test.com")
    print(find_user.id)
except InvalidRequestError:
    print("Invalid")

bob@dylan:~$ python3 main.py
1
1
Not found
Invalid
bob@dylan:~$
```

Repo:

- GitHub repository: `alx-backend-user-data`
- Directory: `0x03-user_authentication_service`
- File: `db.py`

Done? | Check your code | Get a sandbox

3. update user

mandatory

In this task, you will implement the `DB.update_user` method that takes as argument a required `user_id` integer and arbitrary keyword arguments, and returns `None`.

The method will use `find_user_by()` to locate the user to update, then will update the user's attributes as passed in the method's arguments then commit changes to the database.

If an argument that does not correspond to a user attribute is passed, raise a `ValueError`.

```
bob@dylan:~$ cat main.py
#!/usr/bin/env python3
"""
Main file
"""
from db import DB
from user import User

from sqlalchemy.exc import InvalidRequestError
from sqlalchemy.orm.exc import NoResultFound


my_db = DB()

email = 'test@test.com'
hashed_password = "hashedPwd"

user = my_db.add_user(email, hashed_password)
print(user.id)

try:
    my_db.update_user(user.id, hashed_password='NewPwd')
    print("Password updated")
except ValueError:
    print("Error")

bob@dylan:~$ python3 main.py
1
Password updated
bob@dylan:~$
```

Repo:

- GitHub repository: `alx-backend-user-data`
- Directory: `0x03-user_authentication_service`
- File: `db.py`

Done? | Check your code | Get a sandbox

4. Hash password

mandatory

In this task you will define a `_hash_password` method that takes in a `password` string arguments and returns bytes.

The returned bytes is a salted hash of the input password, hashed with `bcrypt.hashpw`.

```
bob@dylan:~$ cat main.py
#!/usr/bin/env python3
"""
Main file
"""
from auth import _hash_password

print(_hash_password("Hello Holberton"))

bob@dylan:~$ python3 main.py
b'$2b$12$eUDdeuBtrD41c8dXvzh95ehsWYCCAi4VH1JbESzgbgZT.eMMzi.G2'
bob@dylan:~$
```

Repo:

- GitHub repository: `alx-backend-user-data`
- Directory: `0x03-user_authentication_service`
- File: `auth.py`

Done? | Check your code | Get a sandbox

5. Register user

mandatory

In this task, you will implement the `Auth.register_user` in the `Auth` class provided below:

```
from db import DB


class Auth:
    """Auth class to interact with the authentication database.
    """

    def __init__(self):
        self._db = DB()
```

Note that `Auth._db` is a private property and should NEVER be used from outside the class.

`Auth.register_user` should take mandatory `email` and `password` string arguments and return a `User` object.

If a user already exist with the passed email, raise a `ValueError` with the message `User <user's email> already exists`.

If not, hash the password with `_hash_password`, save the user to the database using `self._db` and return the `User` object.

```
bob@dylan:~$ cat main.py
#!/usr/bin/env python3
"""
Main file
"""
from auth import Auth

email = 'me@me.com'
password = 'mySecuredPwd'

auth = Auth()

try:
    user = auth.register_user(email, password)
    print("successfully created a new user!")
except ValueError as err:
    print("could not create a new user: {}".format(err))

try:
    user = auth.register_user(email, password)
    print("successfully created a new user!")
except ValueError as err:
    print("could not create a new user: {}".format(err))

bob@dylan:~$ python3 main.py
successfully created a new user!
could not create a new user: User me@me.com already exists
bob@dylan:~$
```

Repo:

- GitHub repository: `alx-backend-user-data`
- Directory: `0x03-user_authentication_service`
- File: `auth.py`

Done? | Check your code | Get a sandbox

6. Basic Flask app

mandatory

In this task, you will set up a basic Flask app.

Create a Flask app that has a single `GET` route (`"/"`) and use `flask.jsonify` to return a JSON payload of the form:

```
{"message": "Bienvenue"}
```

Add the following code at the end of the module:

```
if __name__ == "__main__":
    app.run(host="0.0.0.0", port="5000")
```

Repo:

- GitHub repository: `alx-backend-user-data`
- Directory: `0x03-user_authentication_service`
- File: `app.py`

Done? | Check your code | Get a sandbox

7. Register user

mandatory

In this task, you will implement the end-point to register a user. Define a `users` function that implements the `POST /users` route.

Import the `Auth` object and instantiate it at the root of the module as such:

```
from auth import Auth


AUTH = Auth()
```

The end-point should expect two form data fields: `"email"` and `"password"`. If the user does not exist, the end-point should register it and respond with the following JSON payload:

```
{"email": "<registered email>", "message": "user created"}
```

If the user is already registered, catch the exception and return a JSON payload of the form

```
{"message": "email already registered"}
```

and return a 400 status code

Remember that you should only use `AUTH` in this app. `DB` is a lower abstraction that is proxied by `Auth`.

Terminal 1:

```
bob@dylan:~$ python3 app.py
* Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

Terminal 2:

7)

```
bob@dylan:~$ curl -XPOST localhost:5000/users -d 'email=bob@me.com' -d 'password=mySuperPwd' -v
Note: Unnecessary use of -X or --request, POST is already inferred.
*   Trying 127.0.0.1...
* TCP_NODELAY set
* Connected to localhost (127.0.0.1) port 5000 (#0)
> POST /users HTTP/1.1
> Host: localhost:5000
> User-Agent: curl/7.58.0
> Accept: */*
> Content-Length: 40
> Content-Type: application/x-www-form-urlencoded
>
* upload completely sent off: 40 out of 40 bytes
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
< Content-Type: application/json
< Content-Length: 52
< Server: Werkzeug/1.0.1 Python/3.7.3
< Date: Wed, 19 Aug 2020 00:03:18 GMT
<
{"email":"bob@me.com","message":"user created"}

bob@dylan:~$
bob@dylan:~$ curl -XPOST localhost:5000/users -d 'email=bob@me.com' -d 'password=mySuperPwd' -v
Note: Unnecessary use of -X or --request, POST is already inferred.
*   Trying 127.0.0.1...
* TCP_NODELAY set
* Connected to localhost (127.0.0.1) port 5000 (#0)
> POST /users HTTP/1.1
> Host: localhost:5000
> User-Agent: curl/7.58.0
> Accept: */*
> Content-Length: 40
> Content-Type: application/x-www-form-urlencoded
>
* upload completely sent off: 40 out of 40 bytes
* HTTP 1.0, assume close after body
< HTTP/1.0 400 BAD REQUEST
< Content-Type: application/json
< Content-Length: 39
< Server: Werkzeug/1.0.1 Python/3.7.3
< Date: Wed, 19 Aug 2020 00:03:33 GMT
<
{"message":"email already registered"}
bob@dylan:~$
```

Repo:

- GitHub repository: alx-backend-user-data
- Directory: 0x03-user_authentication_service
- File: app.py

Done? Check your code Get a sandbox

8. Credentials validation

mandatory

In this task, you will implement the `Auth.valid_login` method. It should expect `email` and `password` required arguments and return a boolean.

Try locating the user by email. If it exists, check the password with `bcrypt.checkpw`. If it matches return `True`. In any other case, return `False`.

```
bob@dylan:~$ cat main.py
#!/usr/bin/env python3
"""
Main file
"""
from auth import Auth

email = 'bob@bob.com'
password = 'MyPwdOfBob'
auth = Auth()

auth.register_user(email, password)

print(auth.valid_login(email, password))

print(auth.valid_login(email, "WrongPwd"))

print(auth.valid_login("unknown@email", password))

bob@dylan:~$ python3 main.py
True
False
False
bob@dylan:~$
```

Repo:

- GitHub repository: `alx-backend-user-data`
- Directory: `0x03-user_authentication_service`
- File: `auth.py`

Done? Check your code Get a sandbox

9. Generate UUIDs

mandatory

In this task you will implement a `_generate_uuid` function in the `auth` module. The function should return a string representation of a new UUID. Use the `uuid` module.

Note that the method is private to the `auth` module and should **NOT** be used outside of it.

(/)

Repo:

- GitHub repository: `alx-backend-user-data`
- Directory: `0x03-user_authentication_service`
- File: `auth.py`

Done? Check your code Get a sandbox

10. Get session ID

mandatory

In this task, you will implement the `Auth.create_session` method. It takes an `email` string argument and returns the session ID as a string.

The method should find the user corresponding to the email, generate a new UUID and store it in the database as the user's `session_id`, then return the session ID.

Remember that only public methods of `self._db` can be used.

```
bob@dylan:~$ cat main.py
#!/usr/bin/env python3
"""
Main file
"""
from auth import Auth

email = 'bob@bob.com'
password = 'MyPwdOfBob'
auth = Auth()

auth.register_user(email, password)

print(auth.create_session(email))
print(auth.create_session("unknown@email.com"))

bob@dylan:~$ python3 main.py
5a006849-343e-4a48-ba4e-bbd523fcca58
None
bob@dylan:~$
```

Repo:

- GitHub repository: `alx-backend-user-data`
- Directory: `0x03-user_authentication_service`
- File: `auth.py`

Done? Check your code Get a sandbox

11) Log in

mandatory

In this task, you will implement a `login` function to respond to the `POST /sessions` route.

The request is expected to contain form data with `"email"` and a `"password"` fields.

If the login information is incorrect, use `flask.abort` to respond with a 401 HTTP status.

Otherwise, create a new session for the user, store it the session ID as a cookie with key `"session_id"` on the response and return a JSON payload of the form

```
{"email": "<user email>", "message": "logged in"}
```

(7)

```
bob@dylan:~$ curl -XPOST localhost:5000/users -d 'email=bob@bob.com' -d 'password=mySuperPwd'
{"email":"bob@bob.com","message":"user created"}
bob@dylan:~$
bob@dylan:~$  curl -XPOST localhost:5000/sessions -d 'email=bob@bob.com' -d 'password=mySuperPw
d' -v
Note: Unnecessary use of -X or --request, POST is already inferred.
*   Trying 127.0.0.1...
* TCP_NODELAY set
* Connected to localhost (127.0.0.1) port 5000 (#0)
> POST /sessions HTTP/1.1
> Host: localhost:5000
> User-Agent: curl/7.58.0
> Accept: */*
> Content-Length: 37
> Content-Type: application/x-www-form-urlencoded
>
* upload completely sent off: 37 out of 37 bytes
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
< Content-Type: application/json
< Content-Length: 46
< Set-Cookie: session_id=163fe508-19a2-48ed-a7c8-d9c6e56fabd1; Path=/
< Server: Werkzeug/1.0.1 Python/3.7.3
< Date: Wed, 19 Aug 2020 00:12:34 GMT
<
{"email":"bob@bob.com","message":"logged in"}
* Closing connection 0
bob@dylan:~$
bob@dylan:~$ curl -XPOST localhost:5000/sessions -d 'email=bob@bob.com' -d 'password=BlaBla' -v
Note: Unnecessary use of -X or --request, POST is already inferred.
*   Trying 127.0.0.1...
* TCP_NODELAY set
* Connected to localhost (127.0.0.1) port 5000 (#0)
> POST /sessions HTTP/1.1
> Host: localhost:5000
> User-Agent: curl/7.58.0
> Accept: */*
> Content-Length: 34
> Content-Type: application/x-www-form-urlencoded
>
* upload completely sent off: 34 out of 34 bytes
* HTTP 1.0, assume close after body
< HTTP/1.0 401 UNAUTHORIZED
< Content-Type: text/html; charset=utf-8
< Content-Length: 338
< Server: Werkzeug/1.0.1 Python/3.7.3
< Date: Wed, 19 Aug 2020 00:12:45 GMT
<
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<title>401 Unauthorized</title>
<h1>Unauthorized</h1>
<p>The server could not verify that you are authorized to access the URL requested. You either supplied th
e wrong credentials (e.g. a bad password), or your browser doesn't understand how to supply the credentia
ls required.</p>
```

```
* Closing connection 0
bob@dylan:~$
```

Repo:

- GitHub repository: `alx-backend-user-data`
- Directory: `0x03-user_authentication_service`
- File: `app.py`

Done? Check your code Get a sandbox

12. Find user by session ID

mandatory

In this task, you will implement the `Auth.get_user_from_session_id` method. It takes a single `session_id` string argument and returns the corresponding `User` or `None`.

If the session ID is `None` or no user is found, return `None`. Otherwise return the corresponding user.

Remember to only use public methods of `self._db`.

Repo:

- GitHub repository: `alx-backend-user-data`
- Directory: `0x03-user_authentication_service`
- File: `auth.py`

Done? Check your code Get a sandbox

13. Destroy session

mandatory

In this task, you will implement `Auth.destroy_session`. The method takes a single `user_id` integer argument and returns `None`.

The method updates the corresponding user's session ID to `None`.

Remember to only use public methods of `self._db`.

Repo:

- GitHub repository: `alx-backend-user-data`
- Directory: `0x03-user_authentication_service`
- File: `auth.py`

Done? Check your code Get a sandbox

14. Log out

mandatory

In this task, you will implement a `logout` function to respond to the `DELETE /sessions` route.

The request is expected to contain the session ID as a cookie with key `"session_id"` .

Find the user with the requested session ID. If the user exists destroy the session and redirect the user to `GET /` . If the user does not exist, respond with a 403 HTTP status.

Repo:

- GitHub repository: `alx-backend-user-data`
- Directory: `0x03-user_authentication_service`
- File: `app.py`

Done? | Check your code | Get a sandbox

15. User profile

mandatory

In this task, you will implement a `profile` function to respond to the `GET /profile` route.

The request is expected to contain a `session_id` cookie. Use it to find the user. If the user exist, respond with a 200 HTTP status and the following JSON payload:

```
{"email": "<user email>"}
```

If the session ID is invalid or the user does not exist, respond with a 403 HTTP status.

(7)

```
bob@dylan:~$ curl -XPOST localhost:5000/sessions -d 'email=bob@bob.com' -d 'password=mySuperPwd' -v
Note: Unnecessary use of -X or --request, POST is already inferred.
*   Trying 127.0.0.1...
* TCP_NODELAY set
* Connected to localhost (127.0.0.1) port 5000 (#0)
> POST /sessions HTTP/1.1
> Host: localhost:5000
> User-Agent: curl/7.58.0
> Accept: */*
> Content-Length: 37
> Content-Type: application/x-www-form-urlencoded
>
* upload completely sent off: 37 out of 37 bytes
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
< Content-Type: application/json
< Content-Length: 46
< Set-Cookie: session_id=75c89af8-1729-44d9-a592-41b5e59de9a1; Path=/
< Server: Werkzeug/1.0.1 Python/3.7.3
< Date: Wed, 19 Aug 2020 00:15:57 GMT
<
{"email":"bob@bob.com","message":"logged in"}
* Closing connection 0
bob@dylan:~$
bob@dylan:~$ curl -XGET localhost:5000/profile -b "session_id=75c89af8-1729-44d9-a592-41b5e59de9a1"
{"email": "bob@bob.com"}
bob@dylan:~$
bob@dylan:~$ curl -XGET localhost:5000/profile -b "session_id=nope" -v
Note: Unnecessary use of -X or --request, GET is already inferred.
*   Trying 127.0.0.1...
* TCP_NODELAY set
* Connected to localhost (127.0.0.1) port 5000 (#0)
> GET /profile HTTP/1.1
> Host: localhost:5000
> User-Agent: curl/7.58.0
> Accept: */*
> Cookie: session_id=75c89af8-1729-44d9-a592-41b5e59de9a
>
* HTTP 1.0, assume close after body
< HTTP/1.0 403 FORBIDDEN
< Content-Type: text/html; charset=utf-8
< Content-Length: 234
< Server: Werkzeug/1.0.1 Python/3.7.3
< Date: Wed, 19 Aug 2020 00:16:43 GMT
<
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<title>403 Forbidden</title>
<h1>Forbidden</h1>
<p>You don't have the permission to access the requested resource. It is either read-protected or not readable by the server.</p>
* Closing connection 0

bob@dylan:~$
```

(/)

Repo:

- GitHub repository: `alx-backend-user-data`
- Directory: `0x03-user_authentication_service`
- File: `app.py`

Done? Check your code Get a sandbox

16. Generate reset password token

mandatory

In this task, you will implement the `Auth.get_reset_password_token` method. It take an `email` string argument and returns a string.

Find the user corresponding to the email. If the user does not exist, raise a `ValueError` exception. If it exists, generate a UUID and update the user's `reset_token` database field. Return the token.

Repo:

- GitHub repository: `alx-backend-user-data`
- Directory: `0x03-user_authentication_service`
- File: `auth.py`

Done? Check your code Get a sandbox

17. Get reset password token

mandatory

In this task, you will implement a `get_reset_password_token` function to respond to the `POST /reset_password` route.

The request is expected to contain form data with the `"email"` field.

If the email is not registered, respond with a 403 status code. Otherwise, generate a token and respond with a 200 HTTP status and the following JSON payload:

```
{"email": "<user email>", "reset_token": "<reset token>"}
```

Repo:

- GitHub repository: `alx-backend-user-data`
- Directory: `0x03-user_authentication_service`
- File: `app.py`

Done? Check your code Get a sandbox

18. Update password

mandatory

In this task, you will implement the `Auth.update_password` method. It takes `reset_token` string argument and a `password` string argument and returns `None`.

Use the `reset_token` to find the corresponding user. If it does not exist, raise a `ValueError` exception.

Otherwise, hash the password and update the user's `hashed_password` field with the new hashed password and the `reset_token` field to `None`.

Repo:

- GitHub repository: `alx-backend-user-data`
- Directory: `0x03-user_authentication_service`
- File: `auth.py`

Done? Check your code Get a sandbox

19. Update password end-point

mandatory

In this task you will implement the `update_password` function in the `app` module to respond to the `PUT /reset_password` route.

The request is expected to contain form data with fields `"email"`, `"reset_token"` and `"new_password"`.

Update the password. If the token is invalid, catch the exception and respond with a 403 HTTP code.

If the token is valid, respond with a 200 HTTP code and the following JSON payload:

```
{"email": "<user email>", "message": "Password updated"}
```

Repo:

- GitHub repository: `alx-backend-user-data`
- Directory: `0x03-user_authentication_service`
- File: `app.py`

Done? Check your code Get a sandbox

20. End-to-end integration test

#advanced

Start your app. Open a new terminal window.

Create a new module called `main.py`. Create one function for each of the following tasks. Use the `requests` module to query your web server for the corresponding end-point. Use `assert` to validate the response's expected status code and payload (if any) for each task.

- register_user(email: str, password: str) -> None
- log_in_wrong_password(email: str, password: str) -> None
- log_in(email: str, password: str) -> str
- profile_unlogged() -> None
- profile_logged(session_id: str) -> None
- log_out(session_id: str) -> None
- reset_password_token(email: str) -> str
- update_password(email: str, reset_token: str, new_password: str) -> None

Then copy the following code at the end of the `main` module:

```
EMAIL = "guillaume@holberton.io"
PASSWD = "b4l0u"
NEW_PASSWD = "t4rt1fl3tt3"


if __name__ == "__main__":

    register_user(EMAIL, PASSWD)
    log_in_wrong_password(EMAIL, NEW_PASSWD)
    profile_unlogged()
    session_id = log_in(EMAIL, PASSWD)
    profile_logged(session_id)
    log_out(session_id)
    reset_token = reset_password_token(EMAIL)
    update_password(EMAIL, reset_token, NEW_PASSWD)
    log_in(EMAIL, NEW_PASSWD)
```

Run `python main.py`. If everything is correct, you should see no output.

Repo:

- GitHub repository: `alx-backend-user-data`
- Directory: `0x03-user_authentication_service`
- File: `main.py`

Done? | Check your code | Get a sandbox