Curriculum
Short Specializations
Average: **133.67%**

0x01. Basic authentication

Back-end **Authentification**

Weight: 1

Ongoing second chance project - started Apr 15, 2024 6:00 AM, must end by Apr 20, 2024 6:00 AM

An auto review will be launched at the deadline

In a nutshell…

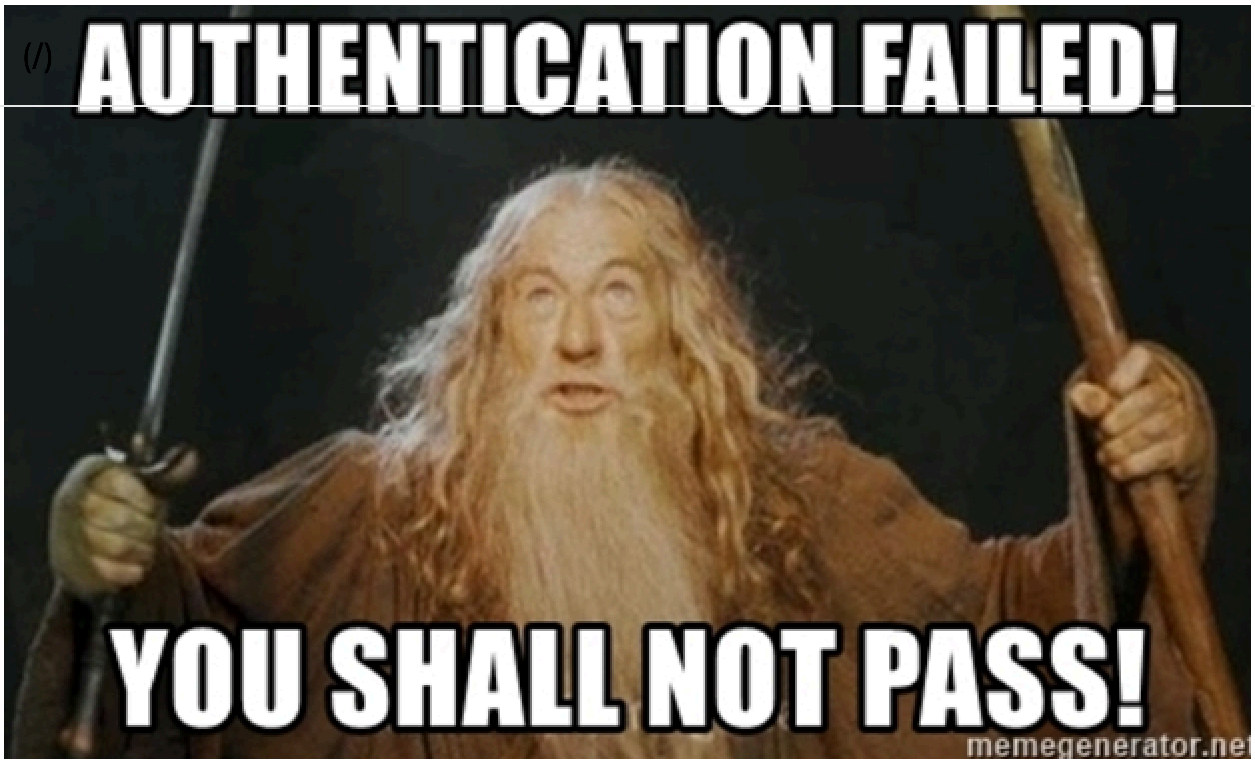
- **Auto QA review:** 169.0/169 mandatory & 27.0/27 optional
- **Altogether: 200.0%**
  - Mandatory: 100.0%
  - Optional: 100.0%
  - Calculation: 100.0% + (100.0% * 100.0%) == **200.0%**

Background Context

In this project, you will learn what the authentication process means and implement a **Basic Authentication** on a simple API.

In the industry, you should **not** implement your own Basic authentication system and use a module or framework that doing it for you (like in Python-Flask: Flask-HTTPAuth (/rltoken/rpsPy0M3_FJuCLGNPUbmvg)). Here, for the learning purpose, we will walk through each step of this mechanism to understand it by doing.

(/)



Resources

Read or watch:

- REST API Authentication Mechanisms (/rltoken/ssg5umgsMk5jKM8WRHk2Ug)
- Base64 in Python (/rltoken/RpaPRyKx1rdHgRSUyuPfeg)
- HTTP header Authorization (/rltoken/WIARq8tQPUGQq5VphLKM4w)
- Flask (/rltoken/HG5WXgSja5kMa29fbMd9Aw)
- Base64 - concept (/rltoken/br6Rp4iMaOce6EAC-JQnOw)

Learning Objectives

At the end of this project, you are expected to be able to explain to anyone (/rltoken/swilZazfz7mspY1vjuy_Zg), **without the help of Google**:

General

- What authentication means
- What Base64 is
- How to encode a string in Base64
- What Basic authentication means
- How to send the Authorization header

Requirements

Python Scripts

- 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)
- 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)

Tasks

0. Simple-basic-API

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Download and start your project from this archive.zip (/rltoken/2o4gAozNufil_KjoxKI5bA)

In this archive, you will find a simple API with one model: `User`. Storage of these users is done via a serialization/deserialization in files.

Setup and start server

```
bob@dylan:~$ pip3 install -r requirements.txt
...
bob@dylan:~$
bob@dylan:~$ API_HOST=0.0.0.0 API_PORT=5000 python3 -m api.v1.app
 * Serving Flask app "app" (lazy loading)
...
bob@dylan:~$
```

Use the API *(in another tab or in your browser)*

```
bob@dylan:~$ curl "http://0.0.0.0:5000/api/v1/status" -vvv
*   Trying 0.0.0.0...
* TCP_NODELAY set
* Connected to 0.0.0.0 (127.0.0.1) port 5000 (#0)
> GET /api/v1/status HTTP/1.1
> Host: 0.0.0.0:5000
> User-Agent: curl/7.54.0
> Accept: */*
>
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
< Content-Type: application/json
< Content-Length: 16
< Access-Control-Allow-Origin: *
< Server: Werkzeug/1.0.1 Python/3.7.5
< Date: Mon, 18 May 2020 20:29:21 GMT
<
{"status":"OK"}
* Closing connection 0
bob@dylan:~$
```

Repo:

- GitHub repository: `alx-backend-user-data`
- Directory: `0x01-Basic_authentication`

Done! Check your code Get a sandbox QA Review

1. Error handler: Unauthorized

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

What the HTTP status code for a request unauthorized? `401` of course!

Edit `api/v1/app.py`:

- Add a new error handler for this status code, the response must be:
  - a JSON: `{"error": "Unauthorized"}`
  - status code `401`
  - you must use `jsonify` from Flask

For testing this new error handler, add a new endpoint in `api/v1/views/index.py`:

- Route: `GET /api/v1/unauthorized`
- This endpoint must raise a 401 error by using `abort` - Custom Error Pages (/rltoken/RH0gY_XQuSB75Q-JbI-fdg)

By calling `abort(401)`, the error handler for 401 will be executed.

In the first terminal:

```
bob@dylan:~$ API_HOST=0.0.0.0 API_PORT=5000 python3 -m api.v1.app
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
....
```

In a second terminal:

```
bob@dylan:~$ curl "http://0.0.0.0:5000/api/v1/unauthorized"
{
  "error": "Unauthorized"
}
bob@dylan:~$
bob@dylan:~$ curl "http://0.0.0.0:5000/api/v1/unauthorized" -vvv
*   Trying 0.0.0.0...
* TCP_NODELAY set
* Connected to 0.0.0.0 (127.0.0.1) port 5000 (#0)
> GET /api/v1/unauthorized HTTP/1.1
> Host: 0.0.0.0:5000
> User-Agent: curl/7.54.0
> Accept: */*
>
* HTTP 1.0, assume close after body
< HTTP/1.0 401 UNAUTHORIZED
< Content-Type: application/json
< Content-Length: 30
< Server: Werkzeug/0.12.1 Python/3.4.3
< Date: Sun, 24 Sep 2017 22:50:40 GMT
<
{
  "error": "Unauthorized"
}
* Closing connection 0
bob@dylan:~$
```

Repo:

- GitHub repository: `alx-backend-user-data`
- Directory: `0x01-Basic_authentication`
- File: `api/v1/app.py, api/v1/views/index.py`

Done! Check your code Get a sandbox QA Review

2. Error handler: Forbidden

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

What the HTTP status code for a request where the user is authenticate but not allowed to access to a resource? `403` of course!

Edit `api/v1/app.py`:
(/)
- Add a new error handler for this status code, the response must be:
  - a JSON: `{"error": "Forbidden"}`
  - status code `403`
  - you must use `jsonify` from Flask

For testing this new error handler, add a new endpoint in `api/v1/views/index.py`:

- Route: `GET /api/v1/forbidden`
- This endpoint must raise a 403 error by using `abort` - Custom Error Pages (/rltoken/RH0gY_XQuSB75Q-JbI-fdg)

By calling `abort(403)`, the error handler for 403 will be executed.

In the first terminal:

```
bob@dylan:~$ API_HOST=0.0.0.0 API_PORT=5000 python3 -m api.v1.app
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
....
```

In a second terminal:

```
bob@dylan:~$ curl "http://0.0.0.0:5000/api/v1/forbidden"
{
  "error": "Forbidden"
}
bob@dylan:~$
bob@dylan:~$ curl "http://0.0.0.0:5000/api/v1/forbidden" -vvv
*   Trying 0.0.0.0...
* TCP_NODELAY set
* Connected to 0.0.0.0 (127.0.0.1) port 5000 (#0)
> GET /api/v1/forbidden HTTP/1.1
> Host: 0.0.0.0:5000
> User-Agent: curl/7.54.0
> Accept: */*
>
* HTTP 1.0, assume close after body
< HTTP/1.0 403 FORBIDDEN
< Content-Type: application/json
< Content-Length: 27
< Server: Werkzeug/0.12.1 Python/3.4.3
< Date: Sun, 24 Sep 2017 22:54:22 GMT
<
{
  "error": "Forbidden"
}
* Closing connection 0
bob@dylan:~$
```

Repo:

- GitHub repository: `alx-backend-user-data`
- Directory: `0x01-Basic_authentication`
- File: `api/v1/app.py, api/v1/views/index.py`

Done! Check your code Get a sandbox QA Review

3. Auth class

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Now you will create a class to manage the API authentication.

- Create a folder `api/v1/auth`
- Create an empty file `api/v1/auth/__init__.py`
- Create the class `Auth` :
    - in the file `api/v1/auth/auth.py`
    - import `request` from `flask`
    - class name `Auth`
    - public method `def require_auth(self, path: str, excluded_paths: List[str]) -> bool:` that returns `False` - `path` and `excluded_paths` will be used later, now, you don't need to take care of them
    - public method `def authorization_header(self, request=None) -> str:` that returns `None` - `request` will be the Flask request object
    - public method `def current_user(self, request=None) -> TypeVar('User'):` that returns `None` - `request` will be the Flask request object

This class is the template for all authentication system you will implement.

```
bob@dylan:~$ cat main_0.py
#!/usr/bin/env python3
""" Main 0
"""
from api.v1.auth.auth import Auth

a = Auth()

print(a.require_auth("/api/v1/status/", ["/api/v1/status/"]))
print(a.authorization_header())
print(a.current_user())

bob@dylan:~$
bob@dylan:~$ API_HOST=0.0.0.0 API_PORT=5000 ./main_0.py
False
None
None
bob@dylan:~$
```

Repo:

- GitHub repository: `alx-backend-user-data`
- Directory: `0x01-Basic_authentication`
- File: `api/v1/auth`, `api/v1/auth/__init__.py`, `api/v1/auth/auth.py`

Done! Check your code Get a sandbox QA Review

4. Define which routes don't need authentication

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Update the method `def require_auth(self, path: str, excluded_paths: List[str]) -> bool:` in `Auth` that returns `True` if the `path` is not in the list of strings `excluded_paths`:

- Returns `True` if `path` is `None`
- Returns `True` if `excluded_paths` is `None` or empty
- Returns `False` if `path` is in `excluded_paths`
- You can assume `excluded_paths` contains string path always ending by a `/`
- This method must be slash tolerant: `path=/api/v1/status` and `path=/api/v1/status/` must be returned `False` if `excluded_paths` contains `/api/v1/status/`

```
bob@dylan:~$ cat main_1.py
#!/usr/bin/env python3
""" Main 1
"""
from api.v1.auth.auth import Auth

a = Auth()

print(a.require_auth(None, None))
print(a.require_auth(None, []))
print(a.require_auth("/api/v1/status/", []))
print(a.require_auth("/api/v1/status/", ["/api/v1/status/"]))
print(a.require_auth("/api/v1/status", ["/api/v1/status/"]))
print(a.require_auth("/api/v1/users", ["/api/v1/status/"]))
print(a.require_auth("/api/v1/users", ["/api/v1/status/", "/api/v1/stats"]))

bob@dylan:~$
bob@dylan:~$ API_HOST=0.0.0.0 API_PORT=5000 ./main_1.py
True
True
True
False
False
True
True
bob@dylan:~$
```

Repo:

(/)

- GitHub repository: `alx-backend-user-data`
- Directory: `0x01-Basic_authentication`
- File: `api/v1/auth/auth.py`

Done! Check your code Get a sandbox QA Review

5. Request validation!

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Now you will validate all requests to secure the API:

Update the method `def authorization_header(self, request=None) -> str:` in `api/v1/auth/auth.py` :

- If `request` is `None` , returns `None`
- If `request` doesn't contain the header key `Authorization` , returns `None`
- Otherwise, return the value of the header request `Authorization`

Update the file `api/v1/app.py` :

- Create a variable `auth` initialized to `None` after the `CORS` definition
- Based on the environment variable `AUTH_TYPE` , load and assign the right instance of authentication to `auth`
  - if `auth` :
    - import `Auth` from `api.v1.auth.auth`
    - create an instance of `Auth` and assign it to the variable `auth`

Now the biggest piece is the filtering of each request. For that you will use the Flask method before_request (/rltoken/kzBrJT9aaokbD6aWYyQzXg)

- Add a method in `api/v1/app.py` to handler `before_request`
  - if `auth` is `None` , do nothing
  - if `request.path` is not part of this list `['/api/v1/status/', '/api/v1/unauthorized/', '/api/v1/forbidden/']` , do nothing - you must use the method `require_auth` from the `auth` instance
  - if `auth.authorization_header(request)` returns `None` , raise the error `401` - you must use `abort`
  - if `auth.current_user(request)` returns `None` , raise the error `403` - you must use `abort`

In the first terminal:

```
bob@dylan:~$ API_HOST=0.0.0.0 API_PORT=5000 AUTH_TYPE=auth python3 -m api.v1.app
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
....
```

In a second terminal:

```
bob@dylan:~$ curl "http://0.0.0.0:5000/api/v1/status"
{
  "status": "OK"
}
bob@dylan:~$
bob@dylan:~$ curl "http://0.0.0.0:5000/api/v1/status/"
{
  "status": "OK"
}
bob@dylan:~$
bob@dylan:~$ curl "http://0.0.0.0:5000/api/v1/users"
{
  "error": "Unauthorized"
}
bob@dylan:~$
bob@dylan:~$ curl "http://0.0.0.0:5000/api/v1/users" -H "Authorization: Test"
{
  "error": "Forbidden"
}
bob@dylan:~$
```

Repo:

- GitHub repository: `alx-backend-user-data`
- Directory: `0x01-Basic_authentication`
- File: `api/v1/app.py, api/v1/auth/auth.py`

Done! Check your code Get a sandbox QA Review

6. Basic auth

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Create a class `BasicAuth` that inherits from `Auth`. For the moment this class will be empty.

Update `api/v1/app.py` for using `BasicAuth` class instead of `Auth` depending of the value of the environment variable `AUTH_TYPE`, If `AUTH_TYPE` is equal to `basic_auth`:

- import `BasicAuth` from `api.v1.auth.basic_auth`
- create an instance of `BasicAuth` and assign it to the variable `auth`

Otherwise, keep the previous mechanism with `auth` an instance of `Auth`.

In the first terminal:

```
bob@dylan:~$ API_HOST=0.0.0.0 API_PORT=5000 AUTH_TYPE=basic_auth python3 -m api.v1.app
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
....
```

In a second terminal:
(/)

```
bob@dylan:~$ curl "http://0.0.0.0:5000/api/v1/status"
{
  "status": "OK"
}
bob@dylan:~$
bob@dylan:~$ curl "http://0.0.0.0:5000/api/v1/status/"
{
  "status": "OK"
}
bob@dylan:~$
bob@dylan:~$ curl "http://0.0.0.0:5000/api/v1/users"
{
  "error": "Unauthorized"
}
bob@dylan:~$
bob@dylan:~$ curl "http://0.0.0.0:5000/api/v1/users" -H "Authorization: Test"
{
  "error": "Forbidden"
}
bob@dylan:~$
```

Repo:

- GitHub repository: `alx-backend-user-data`
- Directory: `0x01-Basic_authentication`
- File: `api/v1/app.py, api/v1/auth/basic_auth.py`

Done! Check your code Get a sandbox QA Review

7. Basic - Base64 part

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Add the method `def extract_base64_authorization_header(self, authorization_header: str) -> str:` in the class `BasicAuth` that returns the Base64 part of the `Authorization` header for a Basic Authentication:

- Return `None` if `authorization_header` is `None`
- Return `None` if `authorization_header` is not a string
- Return `None` if `authorization_header` doesn't start by `Basic` (with a space at the end)
- Otherwise, return the value after `Basic` (after the space)
- You can assume `authorization_header` contains only one `Basic`

```
bob@dylan:~$ cat main_2.py
#!/usr/bin/env python3
""" Main 2
"""
from api.v1.auth.basic_auth import BasicAuth

a = BasicAuth()

print(a.extract_base64_authorization_header(None))
print(a.extract_base64_authorization_header(89))
print(a.extract_base64_authorization_header("Holberton School"))
print(a.extract_base64_authorization_header("Basic Holberton"))
print(a.extract_base64_authorization_header("Basic SG9sYmVydG9u"))
print(a.extract_base64_authorization_header("Basic SG9sYmVydG9uIFNjaG9vbA=="))
print(a.extract_base64_authorization_header("Basic1234"))

bob@dylan:~$
bob@dylan:~$ API_HOST=0.0.0.0 API_PORT=5000 ./main_2.py
None
None
None
Holberton
SG9sYmVydG9u
SG9sYmVydG9uIFNjaG9vbA==
None
bob@dylan:~$
```

Repo:

- GitHub repository: `alx-backend-user-data`
- Directory: `0x01-Basic_authentication`
- File: `api/v1/auth/basic_auth.py`

Done! Check your code Get a sandbox QA Review

8. Basic - Base64 decode

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Add the method `def decode_base64_authorization_header(self, base64_authorization_header: str) -> str:` in the class `BasicAuth` that returns the decoded value of a Base64 string `base64_authorization_header`:

- Return `None` if `base64_authorization_header` is `None`
- Return `None` if `base64_authorization_header` is not a string
- Return `None` if `base64_authorization_header` is not a valid Base64 - you can use `try/except`
- Otherwise, return the decoded value as UTF8 string - you can use `decode('utf-8')`

```
bob@dylan:~$ cat main_3.py
#!/usr/bin/env python3
""" Main 3
"""
from api.v1.auth.basic_auth import BasicAuth

a = BasicAuth()

print(a.decode_base64_authorization_header(None))
print(a.decode_base64_authorization_header(89))
print(a.decode_base64_authorization_header("Holberton School"))
print(a.decode_base64_authorization_header("SG9sYmVydG9u"))
print(a.decode_base64_authorization_header("SG9sYmVydG9uIFNjaG9vbA=="))
print(a.decode_base64_authorization_header(a.extract_base64_authorization_header("Basic SG9s
YmVydG9uIFNjaG9vbA==")))

bob@dylan:~$
bob@dylan:~$ API_HOST=0.0.0.0 API_PORT=5000 ./main_3.py
None
None
None
Holberton
Holberton School
Holberton School
bob@dylan:~$
```

Repo:

- GitHub repository: `alx-backend-user-data`
- Directory: `0x01-Basic_authentication`
- File: `api/v1/auth/basic_auth.py`

Done! Check your code Get a sandbox QA Review

9. Basic - User credentials

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Add the method `def extract_user_credentials(self, decoded_base64_authorization_header: str) -> (str, str)` in the class `BasicAuth` that returns the user email and password from the Base64 decoded value.

- This method must return 2 values
- Return `None, None` if `decoded_base64_authorization_header` is `None`
- Return `None, None` if `decoded_base64_authorization_header` is not a string
- Return `None, None` if `decoded_base64_authorization_header` doesn't contain `:`
- Otherwise, return the user email and the user password - these 2 values must be separated by a `:`
- You can assume `decoded_base64_authorization_header` will contain only one `:`

```
bob@dylan:~$ cat main_4.py
#!/usr/bin/env python3
""" Main 4
"""
from api.v1.auth.basic_auth import BasicAuth

a = BasicAuth()

print(a.extract_user_credentials(None))
print(a.extract_user_credentials(89))
print(a.extract_user_credentials("Holberton School"))
print(a.extract_user_credentials("Holberton:School"))
print(a.extract_user_credentials("bob@gmail.com:toto1234"))

bob@dylan:~$
bob@dylan:~$ API_HOST=0.0.0.0 API_PORT=5000 ./main_4.py
(None, None)
(None, None)
(None, None)
('Holberton', 'School')
('bob@gmail.com', 'toto1234')
bob@dylan:~$
```

Repo:

- GitHub repository: `alx-backend-user-data`
- Directory: `0x01-Basic_authentication`
- File: `api/v1/auth/basic_auth.py`

Done! Check your code Get a sandbox QA Review

10. Basic - User object

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Add the method `def user_object_from_credentials(self, user_email: str, user_pwd: str) -> TypeVar('User'):` in the class `BasicAuth` that returns the `User` instance based on his email and password.

- Return `None` if `user_email` is `None` or not a string
- Return `None` if `user_pwd` is `None` or not a string
- Return `None` if your database (file) doesn't contain any `User` instance with email equal to `user_email` - you should use the class method `search` of the `User` to lookup the list of users based on their email. Don't forget to test all cases: "what if there is no user in DB?", etc.
- Return `None` if `user_pwd` is not the password of the `User` instance found - you must use the method `is_valid_password` of `User`
- Otherwise, return the `User` instance

```
bob@dylan:~$ cat main_5.py
#!/usr/bin/env python3
""" Main 5
"""
import uuid
from api.v1.auth.basic_auth import BasicAuth
from models.user import User

""" Create a user test """
user_email = str(uuid.uuid4())
user_clear_pwd = str(uuid.uuid4())
user = User()
user.email = user_email
user.first_name = "Bob"
user.last_name = "Dylan"
user.password = user_clear_pwd
print("New user: {}".format(user.display_name()))
user.save()

""" Retreive this user via the class BasicAuth """

a = BasicAuth()

u = a.user_object_from_credentials(None, None)
print(u.display_name() if u is not None else "None")

u = a.user_object_from_credentials(89, 98)
print(u.display_name() if u is not None else "None")

u = a.user_object_from_credentials("email@notfound.com", "pwd")
print(u.display_name() if u is not None else "None")

u = a.user_object_from_credentials(user_email, "pwd")
print(u.display_name() if u is not None else "None")

u = a.user_object_from_credentials(user_email, user_clear_pwd)
print(u.display_name() if u is not None else "None")

bob@dylan:~$
bob@dylan:~$ API_HOST=0.0.0.0 API_PORT=5000 ./main_5.py
New user: Bob Dylan
None
None
None
None
Bob Dylan
bob@dylan:~$
```

Repo:

- GitHub repository: `alx-backend-user-data`

- Directory: `0x01-Basic_authentication`
- File: `api/v1/auth/basic_auth.py`

Done! Check your code Get a sandbox QA Review

11. Basic - Overload current_user - and BOOM!

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Now, you have all pieces for having a complete Basic authentication.

Add the method `def current_user(self, request=None) -> TypeVar('User')` in the class `BasicAuth` that overloads `Auth` and retrieves the `User` instance for a request:

- You must use `authorization_header`
- You must use `extract_base64_authorization_header`
- You must use `decode_base64_authorization_header`
- You must use `extract_user_credentials`
- You must use `user_object_from_credentials`

With this update, now your API is fully protected by a Basic Authentication. Enjoy!

In the first terminal:

```
bob@dylan:~$ cat main_6.py
#!/usr/bin/env python3
""" Main 6
"""
import base64
from api.v1.auth.basic_auth import BasicAuth
from models.user import User

""" Create a user test """
user_email = "bob@hbtn.io"
user_clear_pwd = "H0lbertonSchool98!"
user = User()
user.email = user_email
user.password = user_clear_pwd
print("New user: {} / {}".format(user.id, user.display_name()))
user.save()

basic_clear = "{}:{}".format(user_email, user_clear_pwd)
print("Basic Base64: {}".format(base64.b64encode(basic_clear.encode('utf-8')).decode("utf-
8")))

bob@dylan:~$
bob@dylan:~$ API_HOST=0.0.0.0 API_PORT=5000 ./main_6.py
New user: 9375973a-68c7-46aa-b135-29f79e837495 / bob@hbtn.io
Basic Base64: Ym9iQGhidG4uaW86SDBsYmVydG9uU2Nob29sOTgh
bob@dylan:~$
bob@dylan:~$ API_HOST=0.0.0.0 API_PORT=5000 AUTH_TYPE=basic_auth python3 -m api.v1.app
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
....
```

In a second terminal:

```
bob@dylan:~$ curl "http://0.0.0.0:5000/api/v1/status"
{
  "status": "OK"
}
bob@dylan:~$
bob@dylan:~$ curl "http://0.0.0.0:5000/api/v1/users"
{
  "error": "Unauthorized"
}
bob@dylan:~$
bob@dylan:~$ curl "http://0.0.0.0:5000/api/v1/users" -H "Authorization: Test"
{
  "error": "Forbidden"
}
bob@dylan:~$
bob@dylan:~$ curl "http://0.0.0.0:5000/api/v1/users" -H "Authorization: Basic test"
{
  "error": "Forbidden"
}
bob@dylan:~$
bob@dylan:~$ curl "http://0.0.0.0:5000/api/v1/users" -H "Authorization: Basic Ym9iQGhidG4uaW
86SDBsYmVydG9uU2Nob29sOTgh"
[
  {
    "created_at": "2017-09-25 01:55:17",
    "email": "bob@hbtn.io",
    "first_name": null,
    "id": "9375973a-68c7-46aa-b135-29f79e837495",
    "last_name": null,
    "updated_at": "2017-09-25 01:55:17"
  }
]
bob@dylan:~$
```

Repo:

- GitHub repository: `alx-backend-user-data`
- Directory: `0x01-Basic_authentication`
- File: `api/v1/auth/basic_auth.py`

Done! Check your code Get a sandbox QA Review

12. Basic - Allow password with ":"

#advanced

Score: 100.0% (*Checks completed: 100.0%*)

Improve the method `def extract_user_credentials(self, decoded_base64_authorization_header)` to allow password with `:`.

In the first terminal:

(/)

```
bob@dylan:~$ cat main_100.py
#!/usr/bin/env python3
""" Main 100
"""
import base64
from api.v1.auth.basic_auth import BasicAuth
from models.user import User

""" Create a user test """
user_email = "bob100@hbtn.io"
user_clear_pwd = "H0lberton:School:98!"

user = User()
user.email = user_email
user.password = user_clear_pwd
print("New user: {}".format(user.id))
user.save()

basic_clear = "{}:{}".format(user_email, user_clear_pwd)
print("Basic Base64: {}".format(base64.b64encode(basic_clear.encode('utf-8')).decode("utf-
8")))

bob@dylan:~$
bob@dylan:~$ API_HOST=0.0.0.0 API_PORT=5000 ./main_100.py
New user: 5891469b-d2d5-4d33-b05d-02617d665368
Basic Base64: Ym9iMTAwQGhidG4uaW86SDBsYmVydG9uOlNjaG9vbDo5OCE=
bob@dylan:~$
bob@dylan:~$ API_HOST=0.0.0.0 API_PORT=5000 AUTH_TYPE=basic_auth python3 -m api.v1.app
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
....
```

In a second terminal:

```
bob@dylan:~$ curl "http://0.0.0.0:5000/api/v1/status"
{
  "status": "OK"
}
bob@dylan:~$
bob@dylan:~$ curl "http://0.0.0.0:5000/api/v1/users"
{
  "error": "Unauthorized"
}
bob@dylan:~$
bob@dylan:~$ curl "http://0.0.0.0:5000/api/v1/users" -H "Authorization: Test"
{
  "error": "Forbidden"
}
bob@dylan:~$
bob@dylan:~$ curl "http://0.0.0.0:5000/api/v1/users" -H "Authorization: Basic test"
{
  "error": "Forbidden"
}
bob@dylan:~$
bob@dylan:~$ curl "http://0.0.0.0:5000/api/v1/users" -H "Authorization: Basic Ym9iMTAwQGhidG
4uaW86SDBsYmVydG9uOlNjaG9vbDo5OCE="
[
  {
    "created_at": "2017-09-25 01:55:17",
    "email": "bob@hbtn.io",
    "first_name": null,
    "id": "9375973a-68c7-46aa-b135-29f79e837495",
    "last_name": null,
    "updated_at": "2017-09-25 01:55:17"
  },
  {
    "created_at": "2017-09-25 01:59:42",
    "email": "bob100@hbtn.io",
    "first_name": null,
    "id": "5891469b-d2d5-4d33-b05d-02617d665368",
    "last_name": null,
    "updated_at": "2017-09-25 01:59:42"
  }
]
bob@dylan:~$
```

Repo:

- GitHub repository: `alx-backend-user-data`
- Directory: `0x01-Basic_authentication`
- File: `api/v1/auth/basic_auth.py`

Done! Check your code Get a sandbox QA Review

13. Require auth with stars (7)

#advanced

Score: 100.0% (*Checks completed: 100.0%*)

Improve `def require_auth(self, path, excluded_paths)` by allowing `*` at the end of excluded paths.

Example for `excluded_paths = ["/api/v1/stat*"]`:

- `/api/v1/users` will return `True`
- `/api/v1/status` will return `False`
- `/api/v1/stats` will return `False`

Repo:

- GitHub repository: `alx-backend-user-data`
- Directory: `0x01-Basic_authentication`
- File: `api/v1/auth/auth.py`

Done! Check your code Get a sandbox QA Review