(/)

Curriculum
Short Specializations
Average: **97.3%**

0x00. Personal data

Back-end **Authentification**

By: Emmanuel Turlay, Staff Software Engineer at Cruise

Weight: 1

Project over - took place from Jan 10, 2024 6:00 AM to Jan 12, 2024 6:00 AM

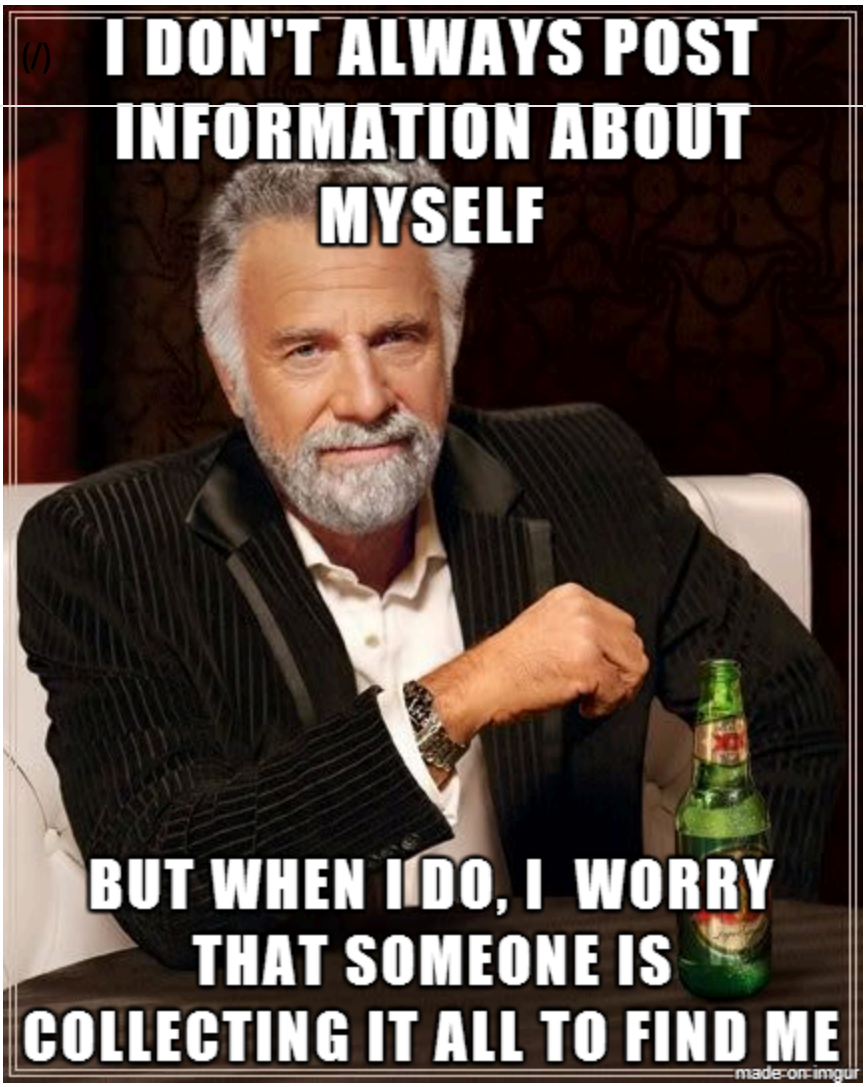
Manual QA review was done on Jan 21, 2024 11:15 AM

An auto review will be launched at the deadline

In a nutshell…

- **Manual QA review:** 0.0/3 mandatory
- **Auto QA review:** 4.0/32 mandatory
- **Altogether: 11.43%**
  - Mandatory: 11.43%
  - Optional: no optional tasks

(/)



Resources

Read or watch:

- What Is PII, non-PII, and Personal Data? (/rltoken/jf71oYqiETchcVhPzQVnyg)
- logging documentation (/rltoken/W2JiHD6cbJY1scJORyLqnw)
- bcrypt package (/rltoken/41oaQXfzwnF1i-wT8W0vHw)
- Logging to Files, Setting Levels, and Formatting (/rltoken/XCpI9uvguxlTCsAeRCW6SA)

Learning Objectives

At the end of this project, you are expected to be able to explain to anyone (/rltoken/yiowzem5NkzxawDmImXy8Q), **without the help of Google**:

- Examples of Personally Identifiable Information (PII)
- How to implement a log filter that will obfuscate PII fields
- How to encrypt a password and check the validity of an input password
- How to authenticate to a database using environment variables

Requirements

- 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)
- All your functions should be type annotated

Tasks

0. Regex-ing

mandatory

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

Write a function called `filter_datum` that returns the log message obfuscated:

- Arguments:
  - `fields` : a list of strings representing all fields to obfuscate
  - `redaction` : a string representing by what the field will be obfuscated
  - `message` : a string representing the log line
  - `separator` : a string representing by which character is separating all fields in the log line (`message`)
- The function should use a regex to replace occurrences of certain field values.
- `filter_datum` should be less than 5 lines long and use `re.sub` to perform the substitution with a single regex.

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

filter_datum = __import__('filtered_logger').filter_datum

fields = ["password", "date_of_birth"]
messages = ["name=egg;email=eggmin@eggsample.com;password=eggcellent;date_of_birth=12/12/1986;", "name=bob;email=bob@dylan.com;password=bobbycool;date_of_birth=03/04/1993;"]

for message in messages:
    print(filter_datum(fields, 'xxx', message, ';'))

bob@dylan:~$
bob@dylan:~$ ./main.py
name=egg;email=eggmin@eggsample.com;password=xxx;date_of_birth=xxx;
name=bob;email=bob@dylan.com;password=xxx;date_of_birth=xxx;
bob@dylan:~$
```

Repo:

- GitHub repository: `alx-backend-user-data`
- Directory: `0x00-personal_data`
- File: `filtered_logger.py`

Done! | Help | Check your code | Get a sandbox | QA Review

1. Log formatter

mandatory

Score: 0.0% (*Checks completed: 0.0%*)

Copy the following code into `filtered_logger.py` .

(7)

```
import logging


class RedactingFormatter(logging.Formatter):
    """ Redacting Formatter class
        """

    REDACTION = "***"
    FORMAT = "[HOLBERTON] %(name)s %(levelname)s %(asctime)-15s: %(message)s"
    SEPARATOR = ";"

    def __init__(self):
        super(RedactingFormatter, self).__init__(self.FORMAT)

    def format(self, record: logging.LogRecord) -> str:
        NotImplementedError
```

Update the class to accept a list of strings `fields` constructor argument.

Implement the `format` method to filter values in incoming log records using `filter_datum` . Values for fields in `fields` should be filtered.

DO NOT extrapolate `FORMAT` manually. The `format` method should be less than 5 lines long.

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

import logging
import re

RedactingFormatter = __import__('filtered_logger').RedactingFormatter

message = "name=Bob;email=bob@dylan.com;ssn=000-123-0000;password=bobby2019;"
log_record = logging.LogRecord("my_logger", logging.INFO, None, None, message, None,
None)
formatter = RedactingFormatter(fields=("email", "ssn", "password"))
print(formatter.format(log_record))

bob@dylan:~$
bob@dylan:~$ ./main.py
[HOLBERTON] my_logger INFO 2019-11-19 18:24:25,105: name=Bob; email=***; ssn=***; pa
ssword=***;
bob@dylan:~$
```

Repo:

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

- Directory: `0x00-personal_data`
- File: `filtered_logger.py`

(/)

Done? | Help | Check your code | Ask for a new correction | Get a sandbox | QA Review

2. Create logger

mandatory

Score: 0.0% (*Checks completed: 0.0%*)

Use user_data.csv (/rltoken/cVQXXtttuAobcFjYFKZTow) for this task

Implement a `get_logger` function that takes no arguments and returns a `logging.Logger` object.

The logger should be named `"user_data"` and only log up to `logging.INFO` level. It should not propagate messages to other loggers. It should have a `StreamHandler` with `RedactingFormatter` as formatter.

Create a tuple `PII_FIELDS` constant at the root of the module containing the fields from `user_data.csv` that are considered PII. `PII_FIELDS` can contain only 5 fields - choose the right list of fields that can are considered as “important” PIIs or information that you **must hide** in your logs. Use it to parameterize the formatter.

Tips:

- What Is PII, non-PII, and personal data? (/rltoken/jf71oYqiETchcVhPzQVnyg)
- Uncovering Password Habits (/rltoken/HznI8kpvBxdnRM92BRoUmQ)

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

import logging

get_logger = __import__('filtered_logger').get_logger
PII_FIELDS = __import__('filtered_logger').PII_FIELDS

print(get_logger.__annotations__.get('return'))
print("PII_FIELDS: {}".format(len(PII_FIELDS)))

bob@dylan:~$
bob@dylan:~$ ./main.py
<class 'logging.Logger'>
PII_FIELDS: 5
bob@dylan:~$
```

Repo:

- GitHub repository: `alx-backend-user-data`
- Directory: `0x00-personal_data`
- File: `filtered_logger.py`

(/)

Done? | Help | Check your code | Ask for a new correction | Get a sandbox | QA Review

3. Connect to secure database

mandatory

Score: 0.0% (*Checks completed: 0.0%*)

Database credentials should NEVER be stored in code or checked into version control. One secure option is to store them as environment variable on the application server.

In this task, you will connect to a secure `holberton` database to read a `users` table. The database is protected by a username and password that are set as environment variables on the server named `PERSONAL_DATA_DB_USERNAME` (set the default as "root"), `PERSONAL_DATA_DB_PASSWORD` (set the default as an empty string) and `PERSONAL_DATA_DB_HOST` (set the default as "localhost").

The database name is stored in `PERSONAL_DATA_DB_NAME`.

Implement a `get_db` function that returns a connector to the database (`mysql.connector.connection.MySQLConnection` object).

- Use the `os` module to obtain credentials from the environment
- Use the module `mysql-connector-python` to connect to the MySQL database (`pip3 install mysql-connector-python`)

(/)

```
bob@dylan:~$ cat main.sql
-- setup mysql server
-- configure permissions
CREATE DATABASE IF NOT EXISTS my_db;
CREATE USER IF NOT EXISTS root@localhost IDENTIFIED BY 'root';
GRANT ALL PRIVILEGES ON my_db.* TO 'root'@'localhost';

USE my_db;

DROP TABLE IF EXISTS users;
CREATE TABLE users (
    email VARCHAR(256)
);

INSERT INTO users(email) VALUES ("bob@dylan.com");
INSERT INTO users(email) VALUES ("bib@dylan.com");

bob@dylan:~$
bob@dylan:~$ cat main.sql | mysql -uroot -p
Enter password:
bob@dylan:~$
bob@dylan:~$ echo "SELECT COUNT(*) FROM users;" | mysql -uroot -p my_db
Enter password:
2
bob@dylan:~$
bob@dylan:~$ cat main.py
#!/usr/bin/env python3
"""
Main file
"""

get_db = __import__('filtered_logger').get_db

db = get_db()
cursor = db.cursor()
cursor.execute("SELECT COUNT(*) FROM users;")
for row in cursor:
    print(row[0])
cursor.close()
db.close()

bob@dylan:~$
bob@dylan:~$ PERSONAL_DATA_DB_USERNAME=root PERSONAL_DATA_DB_PASSWORD=root PERSONAL_
DATA_DB_HOST=localhost PERSONAL_DATA_DB_NAME=my_db ./main.py
2
bob@dylan:~$
```

Repo:

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

- Directory: `0x00-personal_data`
- File: `filtered_logger.py`

(/)

Done? | Help | Check your code | Ask for a new correction | Get a sandbox | QA Review

4. Read and filter data

mandatory

Score: 0.0% (*Checks completed: 0.0%*)

Implement a `main` function that takes no arguments and returns nothing.

The function will obtain a database connection using `get_db` and retrieve all rows in the `users` table and display each row under a filtered format like this:

```
[HOLBERTON] user_data INFO 2019-11-19 18:37:59,596: name=***; email=***; phone=***;
ssn=***; password=***; ip=e848:e856:4e0b:a056:54ad:1e98:8110:ce1b; last_login=2019-1
1-14T06:16:24; user_agent=Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64;
Trident/5.0; KTXN);
```

Filtered fields:

- name
- email
- phone
- ssn
- password

Only your `main` function should run when the module is executed.

(/)

```
bob@dylan:~$ cat main.sql
-- setup mysql server
-- configure permissions
CREATE DATABASE IF NOT EXISTS my_db;
CREATE USER IF NOT EXISTS root@localhost IDENTIFIED BY 'root';
GRANT ALL PRIVILEGES ON my_db.* TO root@localhost;

USE my_db;

DROP TABLE IF EXISTS users;
CREATE TABLE users (
    name VARCHAR(256),
        email VARCHAR(256),
        phone VARCHAR(16),
    ssn VARCHAR(16),
        password VARCHAR(256),
    ip VARCHAR(64),
        last_login TIMESTAMP,
    user_agent VARCHAR(512)
);

INSERT INTO users(name, email, phone, ssn, password, ip, last_login, user_agent) VAL
UES ("Marlene Wood","hwestiii@att.net","(473) 401-4253","261-72-6780","K5?BMNv","60e
d:c396:2ff:244:bbd0:9208:26f2:93ea","2019-11-14 06:14:24","Mozilla/5.0 (Windows NT 1
0.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3729.157 Safari/
537.36");
INSERT INTO users(name, email, phone, ssn, password, ip, last_login, user_agent) VAL
UES ("Belen Bailey","bcevc@yahoo.com","(539) 233-4942","203-38-5395","^3EZ~TkX","f72
4:c5d1:a14d:c4c5:bae2:9457:3769:1969","2019-11-14 06:16:19","Mozilla/5.0 (Linux; U;
Android 4.1.2; de-de; GT-I9100 Build/JZO54K) AppleWebKit/534.30 (KHTML, like Gecko)
Version/4.0 Mobile Safari/534.30");

bob@dylan:~$
bob@dylan:~$ cat main.sql | mysql -uroot -p
Enter password:
bob@dylan:~$
bob@dylan:~$ echo "SELECT COUNT(*) FROM users;" | mysql -uroot -p my_db
Enter password:
2
bob@dylan:~$
bob@dylan:~$ PERSONAL_DATA_DB_USERNAME=root PERSONAL_DATA_DB_PASSWORD=root PERSONAL_
DATA_DB_HOST=localhost PERSONAL_DATA_DB_NAME=my_db ./filtered_logger.py
[HOLBERTON] user_data INFO 2019-11-19 18:37:59,596: name=***; email=***; phone=***;
ssn=***; password=***; ip=60ed:c396:2ff:244:bbd0:9208:26f2:93ea; last_login=2019-11-
14 06:14:24; user_agent=Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/74.0.3729.157 Safari/537.36;
[HOLBERTON] user_data INFO 2019-11-19 18:37:59,621: name=***; email=***; phone=***;
ssn=***; password=***; ip=f724:c5d1:a14d:c4c5:bae2:9457:3769:1969; last_login=2019-1
1-14 06:16:19; user_agent=Mozilla/5.0 (Linux; U; Android 4.1.2; de-de; GT-I9100 Buil
```

```
d/JZO54K) AppleWebKit/534.30 (KHTML, like Gecko) Version/4.0 Mobile Safari/534.30;
bob@dylan:~$
```

Repo:

- GitHub repository: `alx-backend-user-data`
- Directory: `0x00-personal_data`
- File: `filtered_logger.py`

Done? | Help | Get a sandbox | QA Review

5. Encrypting passwords

mandatory

Score: 0.0% (*Checks completed: 0.0%*)

User passwords should NEVER be stored in plain text in a database.

Implement a `hash_password` function that expects one string argument name `password` and returns a salted, hashed password, which is a byte string.

Use the `bcrypt` package to perform the hashing (with `hashpw`).

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

hash_password = __import__('encrypt_password').hash_password

password = "MyAmazingPassw0rd"
print(hash_password(password))
print(hash_password(password))

bob@dylan:~$
bob@dylan:~$ ./main.py
b'$2b$12$Fnjf6ew.oPZtVksngJjh1.vYCnxRjPm2yt18kw6AuprMRpmhJVxJO'
b'$2b$12$xSAw.bxfSTAlIBglPMXeL.SJnzme3Gm0E7eOEKOVV2OhqOakyUN5m'
bob@dylan:~$
```

Repo:

- GitHub repository: `alx-backend-user-data`
- Directory: `0x00-personal_data`
- File: `encrypt_password.py`

(7) Done? | Help | Check your code | Ask for a new correction | Get a sandbox | QA Review

6. Check valid password

mandatory

Score: 0.0% (*Checks completed: 0.0%*)

Implement an `is_valid` function that expects 2 arguments and returns a boolean.

Arguments:

- `hashed_password` : `bytes` type
- `password` : string type

Use `bcrypt` to validate that the provided password matches the hashed password.

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

hash_password = __import__('encrypt_password').hash_password
is_valid = __import__('encrypt_password').is_valid

password = "MyAmazingPassw0rd"
encrypted_password = hash_password(password)
print(encrypted_password)
print(is_valid(encrypted_password, password))

bob@dylan:~$
bob@dylan:~$ ./main.py
b'$2b$12$Fnjf6ew.oPZtVksngJjh1.vYCnxRjPm2yt18kw6AuprMRpmhJVxJO'
True
bob@dylan:~$
```

Repo:

- GitHub repository: `alx-backend-user-data`
- Directory: `0x00-personal_data`
- File: `encrypt_password.py`

Done? | Help | Check your code | Ask for a new correction | Get a sandbox | QA Review

Ready for a new manual review

(/)