Assignment 2 - Development Team Project: Coding Output

Team: The Red Team

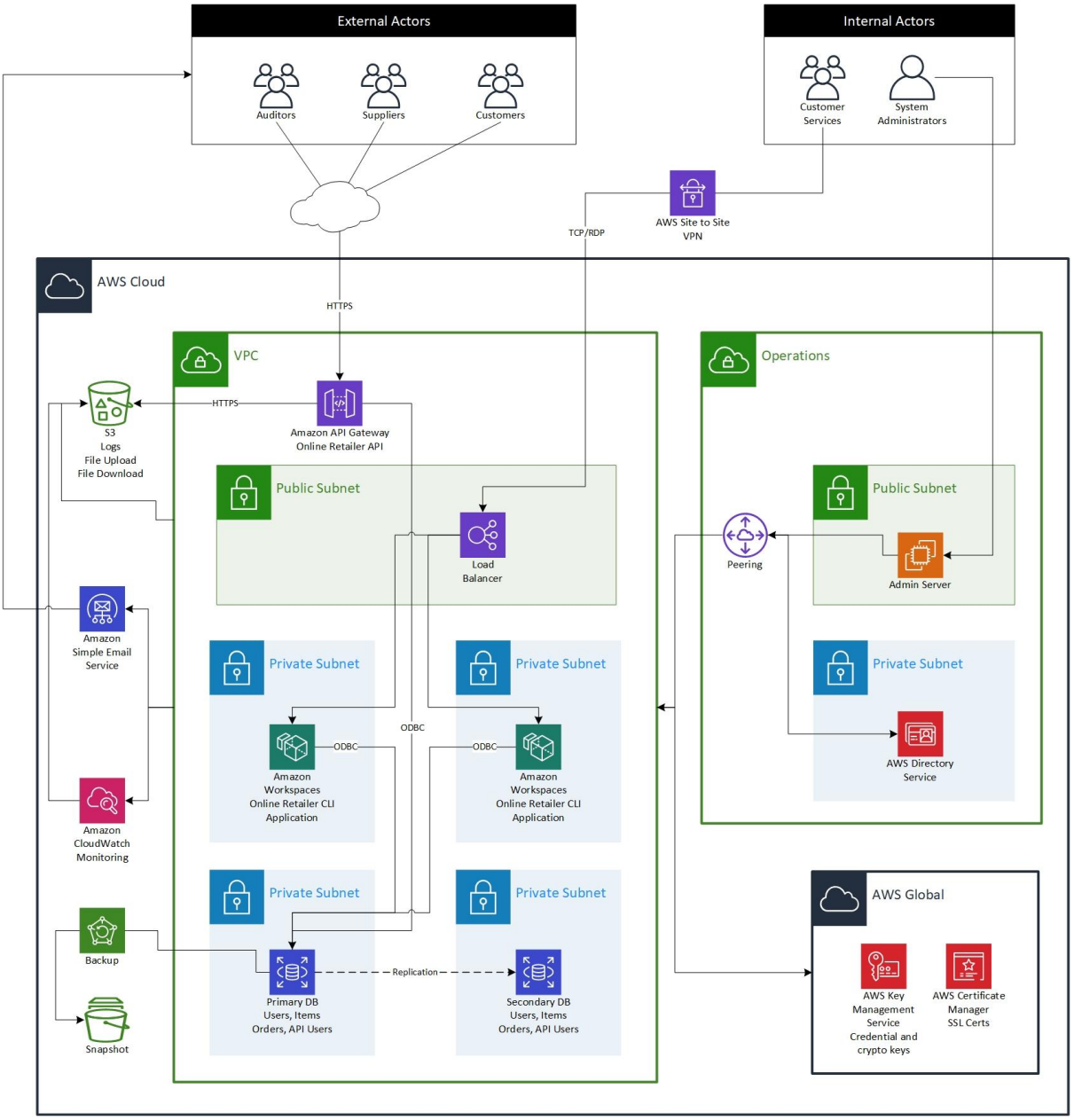
Members: Liam Willson, Simon Bolder, Fergus Nugent

Word Count: 1785

Application Description

The application build consists of Python scripts that define classes containing downstream functions which allow all application functionalities (including CRUD, encryption and API functionality), JSON files to store data, scripts to access and run the API and store information on it, files that store log records, and scripts which set and store login and security keys. Application usage or the UI occurs via a terminal Command Line Interface (CLI) and allows a public individual to register themselves as a user or for an administrator to login (both by running 'onlineretailer.py') after setting an administrator key in the backend via running the 'adminCreator.py' script. Both the user and administrator roles are permissions limited. The auditor role cannot access the CLI, but rather accesses the API to view logs of access and system events via running the 'server.py' script.

The application is built with a microservice architecture. This means that each of the application's code sets are encapsulated in their own methods and interact only with the necessary modules. Improving scalability, modifiability, testability and efficiency. The current version of the application is not network supported, it is locally accessed for the locally registered user. An administrator can access the application locally or via http or https if the optional SSL certification functionality is turned on.



^^^ Illustrated above is the structure of a proposed network supported version of the application running in AWS. By exposing backend applications through API gateways and load balancers to manage client based traffic and implementing various AWS components it would allow a secure and adaptable infrastructure and application stack.

Application Security Features

Security measures implemented include, input sanitisation, data encryption, recurring authentication requirements, system action and event logging, password complexity requirements, data normalisation, role based access control, tagging inputs with user ID, manipulating store and user data via an API, review of the coding after linting and optional SSL certification functionality which is turn-'onable' or -'offable'.

Password encryption involves encrypting a password (which must meet specified complexity requirements) with a key that is then used downstream to decrypt the password when matching it to user input when it is retrieved. Encrypted login data is used to separate the access of differing types of users because the permissions accessible to them are associated with their specific type of account and authentication is required at key stages of use, not only the login stage. Additionally, data input sanitisation is built-in to the application functionality for when the user is prompted to provide input. All data retrieved or generated is formatted and deposited into JSON or .key files and this is even more so true for the logging data accessible to an auditor via the API. Furthermore, not just any data is retrieved, there are complexity requirements filtering user input. And, in more detail, the API built provides not only logs and records of system actions for an auditor to analyse, also it allows administrators to edit the item catalogue, order record and details for users accounts via http or https (if SSL certification is enabled).

Overall, the application's low attack surface is allowed because the application is accessible locally by a single terminal in which a user can only view their account and related data. Thus, the application is built based on an enclosed deployment and a limited usership model.

In accordance with the principles of agile and lean software development, the team perceived no need at this time, considering the style of the application usage and this being a first-time deployment, that the entire database should be encrypted. Furthermore, SSL

certification as it relates to accessing the API can be enabled or disabled as desired and this is ideal because it would be more likely for appropriate use of this program if the state of it was network deployable, which at this stage it is not.

Design vs. Actual Coding Output - Analysis

The key differences between the design documentation and the actual coding output developed is as follows…

A. Flask was replaced with FastAPI as the API technology or web framework utilised. This is because, although Flask is a lightweight and flexible microframework, FastAPI has a lower ‘ease of first time use’ barrier and more importantly, it provides the user with a user interface via http or https if SSL certification is turned on via the application back-end.
B. The Mccabe linter was not utilised, instead Pylint and Flake8 were used

Application Data Structures

Relational databases implemented in the application build…

1. JSON file containing the items available in the online retailer catalogue
2. JSON file containing the orders made by users
3. JSON file containing the locally registered secure user account
4. API database store containing the authorised auditor accounts
5. API database store containing the system actions and events log

Application Run & Setting User Credentials

***Visit the README.md file in the application source folder for more insight, especially regarding installation of the correct modules and tools.

for administrators…

Run ‘admincreator.py’.

Set the administrator key.

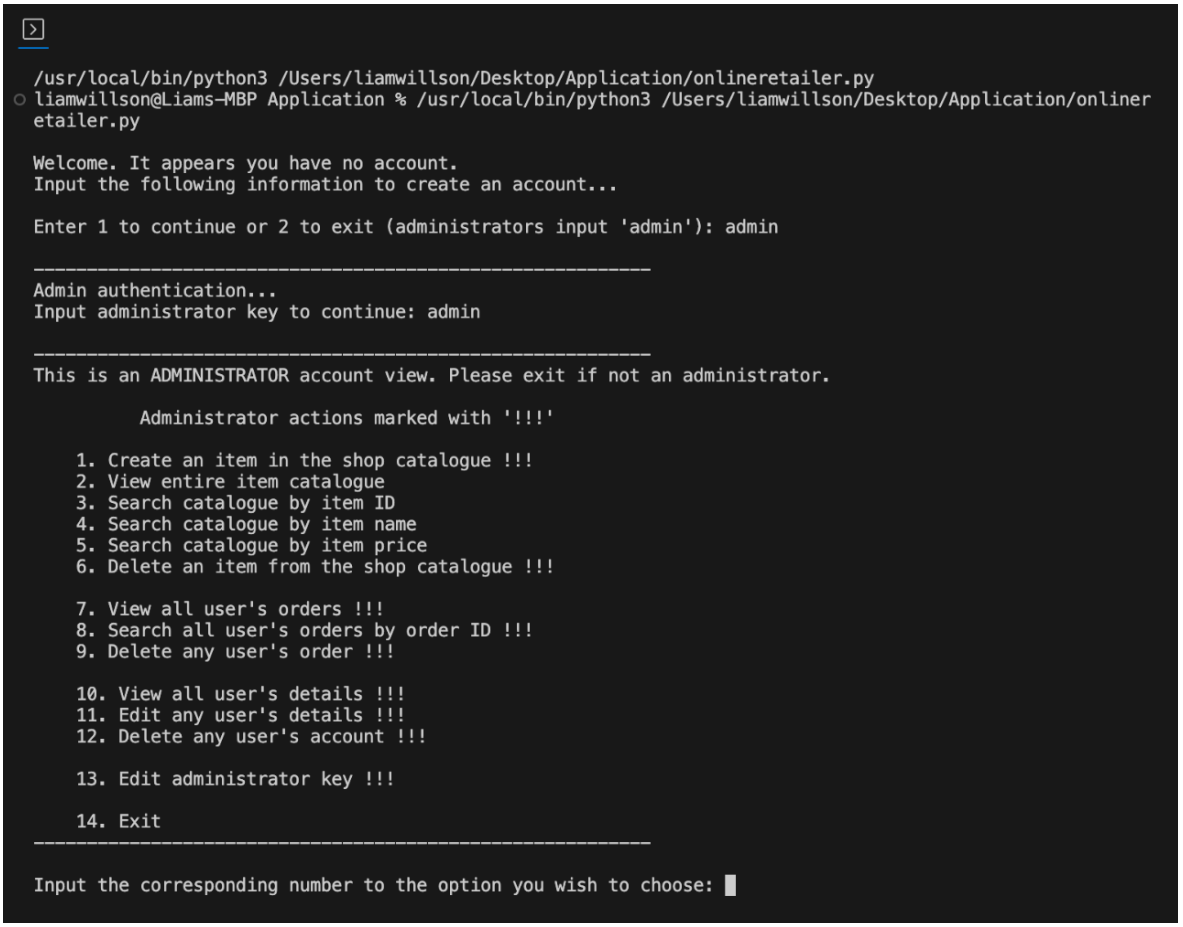
Run ‘onlineretailer.py’.

Logging in/registering a secure account >>

1. Administrators log in to their account by entering the unique administrator key that was set prior.

Accessing the main administrator menu >>

2. Administrators access the application menu where they can enter any prompted number to access the associated functionality.



for users…

Run ‘onlineretailer.py’.

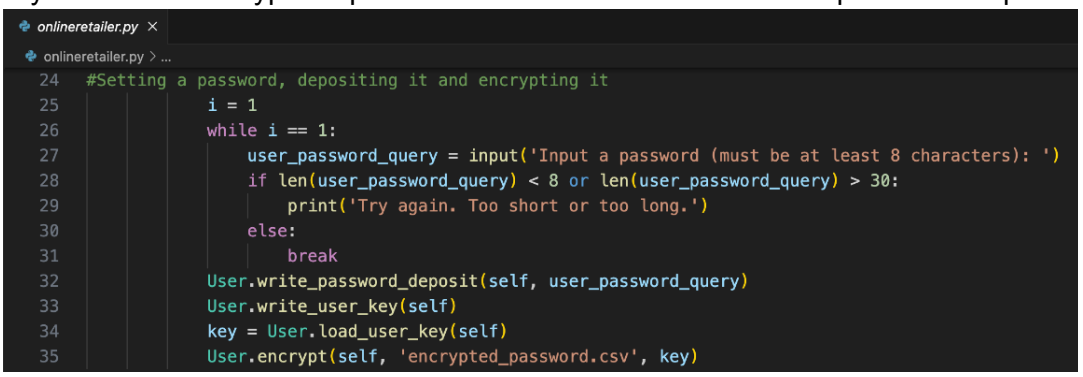
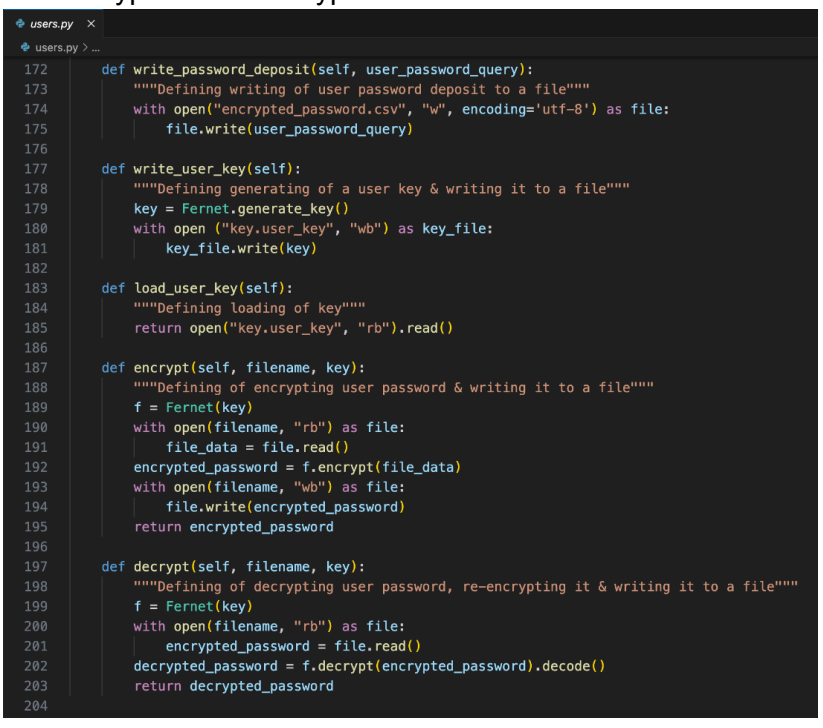
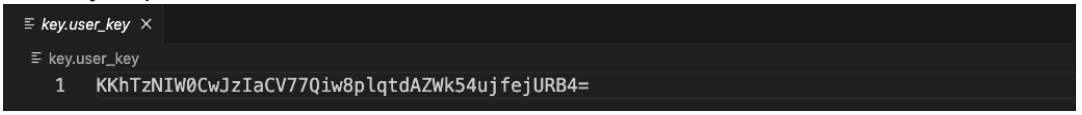
Logging in/registering a secure account >>

1. Users register or log in to their account
    - If registering, the user enters a username and password, in addition to their personal details related to their account (personal and payment details).
    - If logging in, the user enters their unique username and password.

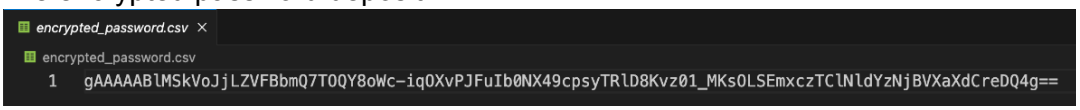
Accessing the main user menu >>

2. User accesses the application menu where they can enter any prompted number to access the associated functionality.

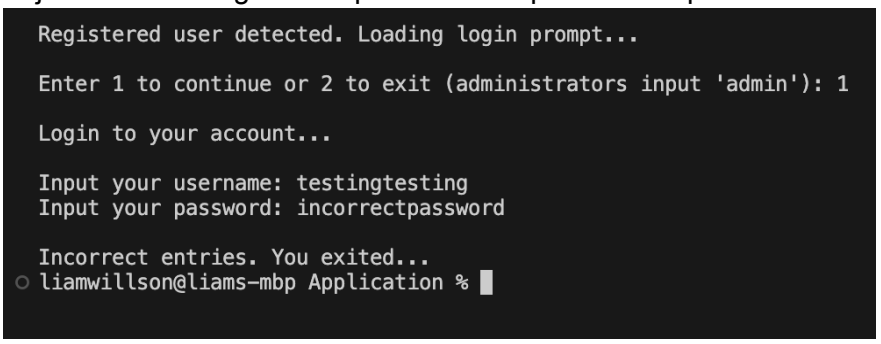
Application Functional Testing

Security Measure	***Functional Testing***
Fernet Cryptography Encryption for users & for administrators	Fernet cryptography was used to encrypt a user's password when they set it during registering of their secure account and then it was used to decrypt it and compare it to the user's input when they entered their password to login to their secure account. Retrieving input from the user to set a password, creating a deposit for the password, generating a Fernet encryption key and storing the key, and loading the key to use it to encrypt the password which is then written to the password deposit:  The encryption and decryption functions defined:  The key deposit: 

The encrypted password deposit:



Rejection of the login attempt if incorrect password inputted:



Acceptance of the login attempt if correct password inputted:

Note: The combination of a correct username and password is required to login.

Backend Script To Set Administrator Key

An administrator is required to run in the backend the ‘adminCreator.py’ script to set the administrator key which is required to login as an administrator. The key is encrypted with Fernet cryptography and as described above, a key is generated which can decrypt the key when logging in.

The ‘adminCreator.py’ script:

```python
"""Importing necessary modules"""
from admins import Admin

def main(self):
    """Defining main program"""
#Defining administrator authentication, depositing it and encrypting it
    admin_authenticator = input('Set the administrator key: ')
    Admin.write_administrator_deposit(self, admin_authenticator)
    Admin.write_admin_key(self)
    key = Admin.load_admin_key(self)
    Admin.encrypt(self, 'admin_key.csv', key)

#Run program
if __name__ == '__main__':
    main(main)

```

If an administrator key does not yet exist and login is attempted:

```
Registered user detected. Loading login prompt...

Enter 1 to continue or 2 to exit (administrators input 'admin'): admin

Administrator access unavailable. You exited...
liamwillson@liams-mbp Application %
```

Setting the administrator key:

```
esktop/SSD_Team_Project-main 2/Application/adminCreator.py"
Set the administrator key: admin
liamwillson@liams-mbp Application %
```

Input Sanitisation

When input is retrieved from the user, not just any input will be accepted.
Inputs that require strings will accept strings and inputs that require integers will accept integers:

```python
#Setting payment information for user account
            print('')
            i = 1
            while i == 1:
                print('Finally, set your payment information...')
                user_bank_name_query = input('Input your bank name: ')
                user_bank_account_name_query = input('Input your bank account name: ')
                user_bank_account_bsb_query = input('Input your bank account BSB: ')
                user_bank_account_number_query = input('Input your bank account number: ')
                print('')
                user_card_name_query = input('Input your card name: ')
                user_card_number_query = input('Input your card number: ')
                user_card_expiry_query = input('Input your card expiry: ')
                user_card_cvc_query = input('Input your card cvc: ')
                if (user_bank_account_bsb_query.isdigit() is False or user_bank_account_number_query.isdigit() is False or
                        user_card_number_query.isdigit() is False or user_card_expiry_query.isdigit() is False or
                        user_card_cvc_query.isdigit() is False):
                    print('')
                    print('You entered characters in an input field that required digits only. Try again.')
                    print('')
                else:
                    break
```

README.md

Setting a password requires a minimum and maximum character limit:

```python
#Setting a password, depositing it and encrypting it
            i = 1
            while i == 1:
                user_password_query = input('Input a password (must be at least 8 characters): ')
                if len(user_password_query) < 8 or len(user_password_query) > 30:
                    print('Try again. Too short or too long.')
                else:
                    break
```

Note: These are examples of some of the input sanitisation measures implemented in the application.

Data normalisation

When input is retrieved from users, in some cases it is normalised before it's deposited in storage or a database. For example, data inputted by the user is transformed to lowercase:

```python
#Requesting order information input from user
        order_name_query = str(input('Input the name for the order: ')).lower()
        order_item_query = str(input('Input the item you want to order: ')).lower()
        order_quantity_query = str(input('Input the number of the item you want to order: ')).lower()
        order_delivery_type_query = str(input('Input P for pickup OR M for mail delivery: ')).lower()
        print('')
        abort_query = str(input("To execute order input '1' or to cancel '2': "))
        if abort_query == str(1):
#Calling create order function which commits inputted information to orders.json
            Order.create_order(self, order_id_query=order_id_query,
                               order_username_query=order_username_query,
                               order_name_query=order_name_query, order_item_query=order_item_query,
                               order_quantity_query=order_quantity_query,
                               order_delivery_type_query=order_delivery_type_query)
            user_menu(self)
```

API: Authentication & SSL Certification

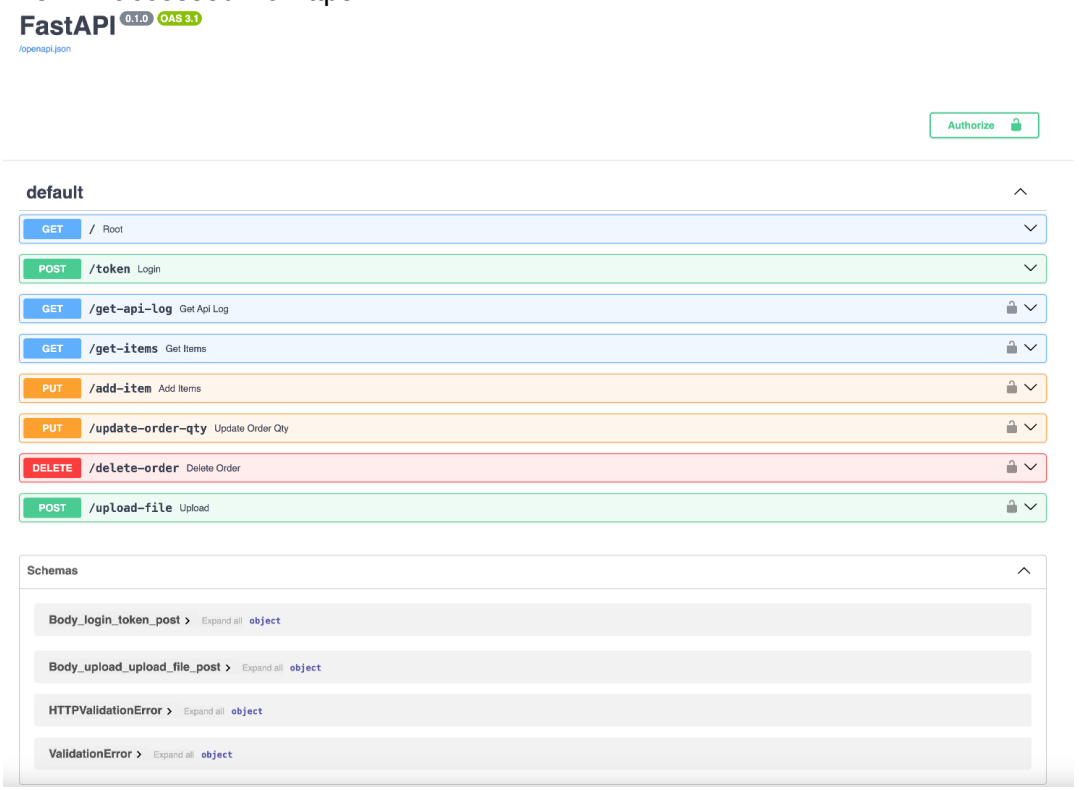
This application is built with an API that allows an auditor or administrator to retrieve a log of system actions and events, retrieve the item catalogue, add an time to the item catalogue, update the quantity of an order submitted, delete an order submitted or upload a data file (e.g. a comprehensive item catalogue) to the application.

SSL certification can be enabled or disabled in the current version. If enabled, the API is accessed via https and SSL certification is required and if disabled, via http and SSL certification is not required.

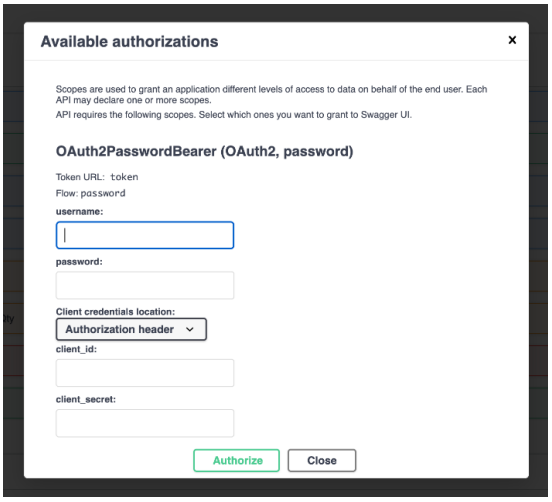
SSL certification is enabled if 'SSL = 'TRUE'' & 'SECURE = 'TRUE'' or disabled if 'SSL = 'FALSE'' & 'SECURE = 'FALSE'':

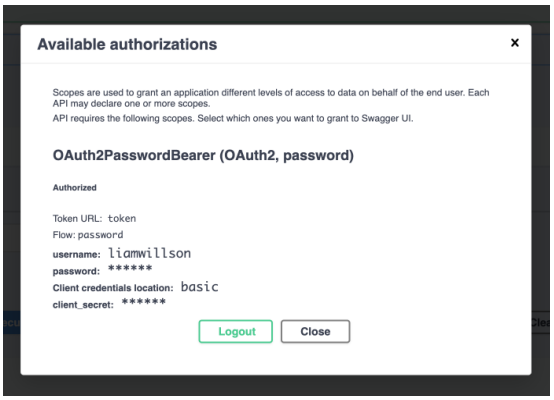
```python
"""Importing necessary modules"""
import uvicorn

#Variables SSL
SSL = 'TRUE'
SECURE = 'TRUE'
HOST = 'localhost'
PORT = 8432
RELOAD = True
SSL_KEY_FILE = 'localhost+2-key.pem'
SSL_CERT_FILE = 'localhost+2.pem'
```

The API accessed via https:



SSL certification enabled requires the auditor or administrator accessing the API to authenticate themselves:

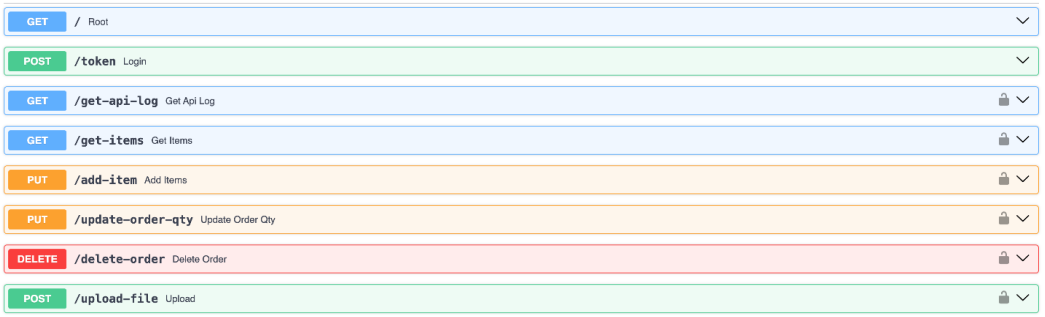




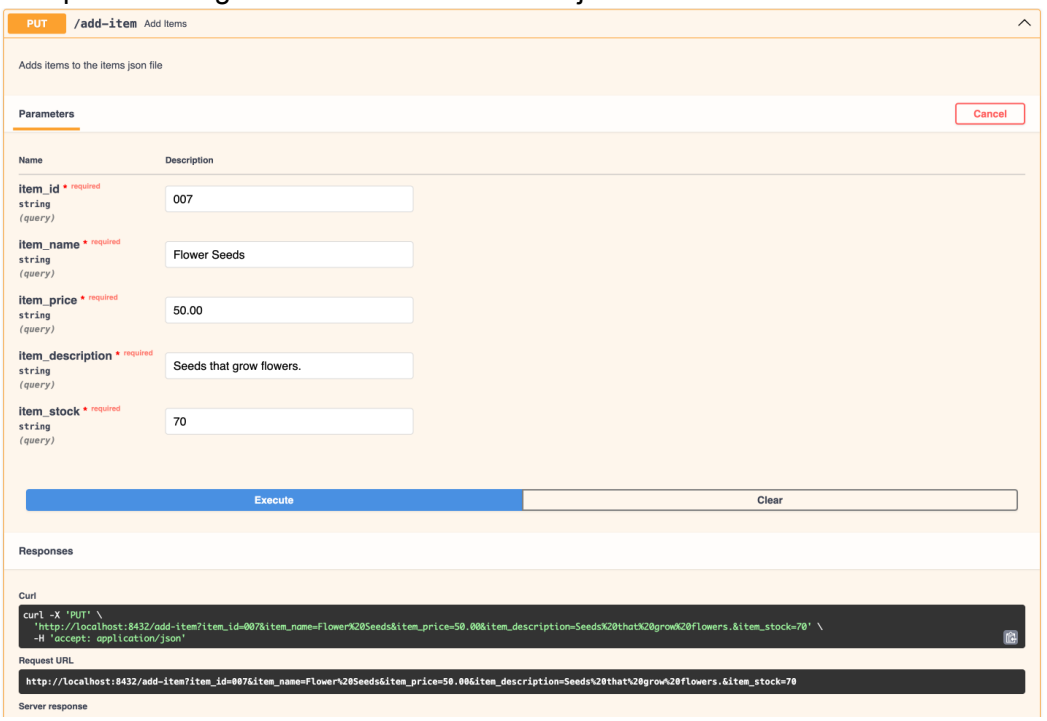
README.md

API: Application Actions

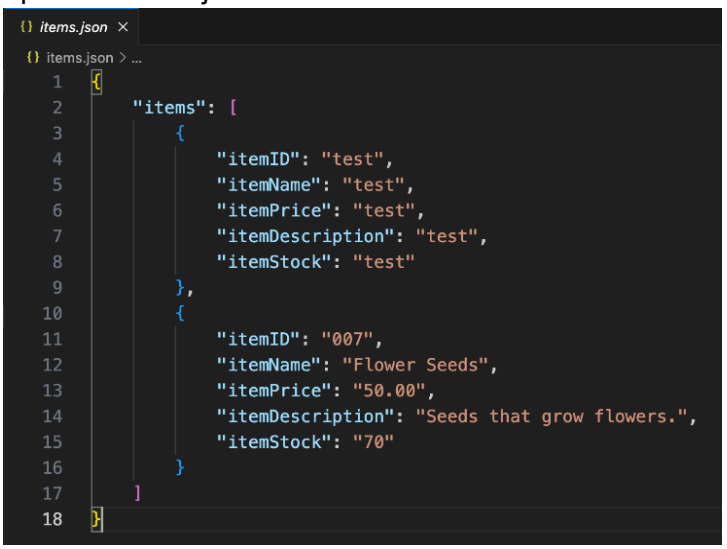
Actions allowed by the API:



Example of adding an item to the local ‘items.json’ via the API:



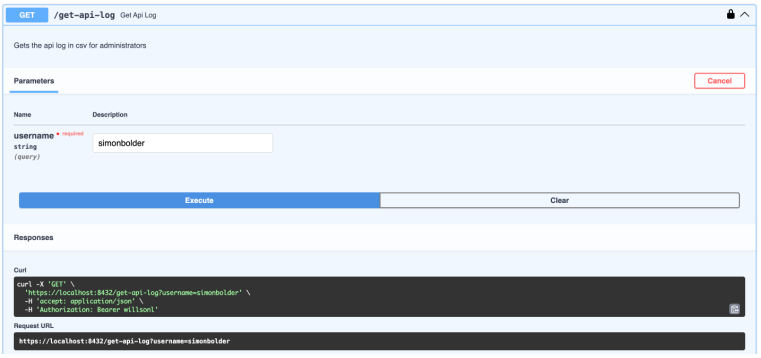
Updated ‘items.json’:



README.md

API: System Action & Event Logging

The API also allows auditors and administrators to view logs of system actions and events:



Password Complexity Requirements

When it is required for the user to set a password, they must meet password complexity requirements. The following illustrates a basic implementation of this:

```python
#Setting a password, depositing it and encrypting it
            i = 1
            while i == 1:
                user_password_query = input('Input a password (must be at least 8 characters): ')
                if len(user_password_query) < 8 or len(user_password_query) > 30:
                    print('Try again. Too short or too long.')
                else:
                    break
```

Role Based Access Control

The roles defined by the application build are: public, user, administrator and auditor. The public can open the application and register to become a user. When the application is launched, there is a user menu and an administrator menu available depending on which user type has logged in. Either menu presents different options characterised by different levels of permission to the appropriate user type. Thus, there is role based access control available.

The user menu:

```python
#Defining the user menu
def user_menu(self):
    """Defining the user ID and username to present it in the user menu"""
    for i in User.user_data['users']:
        user_id_query = i['userID']
        user_username_query = i['userUsername']
#Defining the user menu display
    print(f'''
------------------------------------------------------------
Welcome to the shop. Please explore the following options.

          Your user ID is: {user_id_query}
            Your username is: {user_username_query}

    1. View entire item catalogue
    2. Search catalogue by item ID
    3. Search catalogue by item name
    4. Search catalogue by item price

    5. Order an item
    6. View your orders
    7. Search your orders by order ID
    8. Edit an order of yours
    9. Delete an order of yours

    10. View your user details
    11. Edit your user details
    12. Delete your user account

    13. Exit
------------------------------------------------------------
    ''')
#Offering the user menu choices
    user_choice = str(input('Input the corresponding number to the option you wish to choose: '))
```

The administrator menu:

onlineretailer.py 2 ×

onlineretailer.py > ...

```
491  def administrator_menu(self):
492      """Defining the administrator menu"""
493      print('')
494      print('------------------------------------------------------------')
495  #Defining administrator authentication
496      print('Admin authentication...')
497      admin_authenticator = str(input('Input administrator key to continue: '))
498      key = Admin.load_admin_key(self)
499      admin_verify = User.decrypt(self, "admin_key.csv", key)
500      if admin_authenticator == str(admin_verify):
501          pass
502      else:
503          print('')
504          print('Incorrect entries. You exited...')
505          sys.exit()
506  #Defining the administrator menu display
507      print('''
508  ------------------------------------------------------------
509  This is an ADMINISTRATOR account view. Please exit if not an administrator.
510
511          Administrator actions marked with '!!!'
512
513      1. Create an item in the shop catalogue !!!
514      2. View entire item catalogue
515      3. Search catalogue by item ID
516      4. Search catalogue by item name
517      5. Search catalogue by item price
518      6. Delete an item from the shop catalogue !!!
519
520      7. View all user's orders !!!
521      8. Search all user's orders by order ID !!!
522      9. Delete any user's order !!!
523
524      10. View all user's details !!!
525      11. Edit any user's details !!!
526      12. Delete any user's account !!!
527
528      13. Edit administrator key !!!
529
530      14. Exit
531  ------------------------------------------------------------
532      ''')
533  #Offering the user menu choices
534      user_choice = str(input('Input the corresponding number to the option you wish to choose: '))
```

ID Tagging

Upon registering a secure account, a random user ID is allocated to the user. When the user creates and submits an order, their user ID is tagged to their order for tracking and identification purposes. The user ID is unchangeable neither by the user themselves or administrators and is a combination of the username and a random string.

Generating the random user ID:

onlineretailer.py 2 ×

onlineretailer.py > ...

```
39  #Generating a user ID
40              user_id_source = string.ascii_letters
41              user_id_suffix = ''.join(random.choice(user_id_source) for i in range(10))
42              user_id_query = str(user_username_query) + '.' + str(user_id_suffix)
```

Tagging the user ID to an order when the user creates an order:

```python
#Defining the order ID and username to equal the user's
        for i in User.user_data['users']:
            order_id_query = i['userID']
            order_username_query = i['userUsername']
#Requesting order information input from user
        order_name_query = str(input('Input the name for the order: ')).lower()
        order_item_query = str(input('Input the item you want to order: ')).lower()
        order_quantity_query = str(input('Input the number of the item you want to order: ')).lower()
        order_delivery_type_query = str(input('Input P for pickup OR M for mail delivery: ')).lower()
        print('')
        abort_query = str(input("To execute order input '1' or to cancel '2': "))
        if abort_query == str(1):
#Calling create order function which commits inputted information to orders.json
            Order.create_order(self, order_id_query=order_id_query,
                               order_username_query=order_username_query,
                               order_name_query=order_name_query, order_item_query=order_item_query,
                               order_quantity_query=order_quantity_query,
                               order_delivery_type_query=order_delivery_type_query)
            user_menu(self)
```

Implementation of only necessary and known libraries

Only necessary and known libraries have been imported to ensure that no unnecessary dependencies are created. This is the same consideration for importing classes from scripts to other scripts.

'onlineretailer.py' modules:

```python
"""Importing necessary modules"""
import sys
import os
import random
import string
from inputimeout import inputimeout
from users import User
from admins import Admin
from items import Item
from orders import Order

```

'system.py' modules:

```python
"""https://www.geeksforgeeks.org/encrypt-and-decrypt-files-using-python/
https://regex-generator.olafneumann.org"""

#Importing necessary modules
import json
from datetime import datetime
import re
from cryptography.fernet import Fernet
import filelogging

```

'

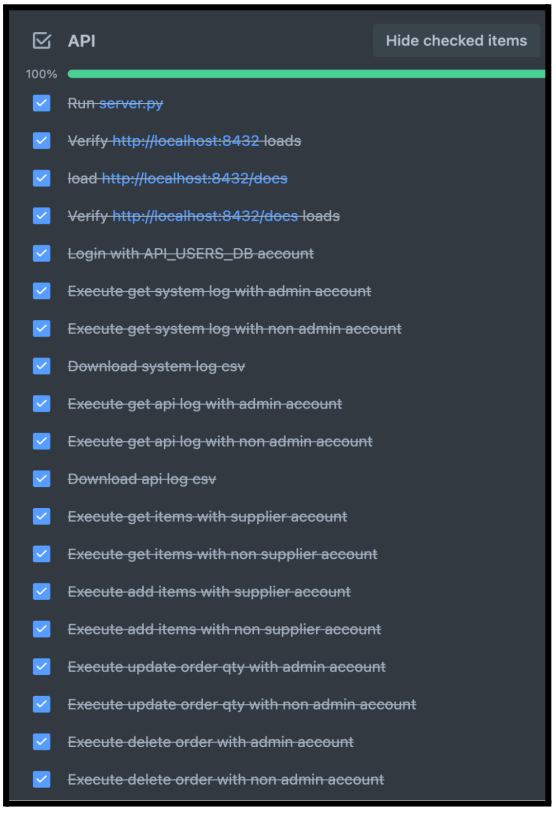
‘main.py’ modules:

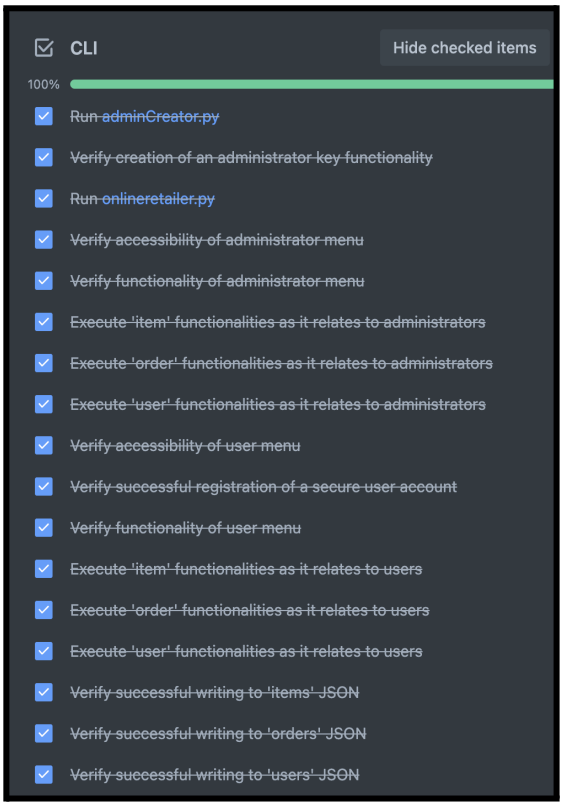
```python
'''Online Retailer API - written by team red as part of team project for SSD delivery using
tutorials and references from FASTAPI'''

# Importing necessary modules
import os
import json
from datetime import datetime
from typing import Union
from pydantic import BaseModel
from typing_extensions import Annotated
from fastapi import FastAPI
from fastapi.responses import FileResponse
from fastapi import Depends, FastAPI, HTTPException, status
from fastapi.security import OAuth2PasswordBearer, OAuth2PasswordRequestForm
from fastapi import File, UploadFile

```

Trello

Trello was utilised as a collaborative tool that allowed the development team to employ the agile scrum development process. It was used to create tasks and assign them to cards that would then be pushed through the progressive stages of development. Access the development board via: https://shorturl.at/lACQX

The following evidences functional testing carried out for the application API and CLI and is illustrated on the development board linked above…





README.md

OWASP (2021) Security Risk Considerations

Development sought to consider the security risks defined by OWASP (The Open Worldwide Application Security Project) (2021). Analysis of adherence to satisfying OWASP definitions:

OWASP Web Application Security Risk	***Application Implementations That Curtail The Risk***
A03:2021 - Injection	- Input sanitisation.
A07:2021 - Identification & Authentication Failures	- Password complexity requirements. - Logging failed login attempts.
A02:2021 - Cryptographic Failures	- Cryptographic encryption of login information for users & administrators using Fernet cryptography. - Storage of encrypted hashed strings.
A01:2021 - Broken Access Control	- Robust roles and allocated permissions. - No role inheritance. - Separation of login portal and access of it dependent on role.
A05:2021 - Security Misconfiguration	- Implementation of only necessary, OWASP-approved and known libraries. - Ensuring correct configuration of all technologies.
A06:2021 - Vulnerable & Outdated Components	- Implementation of a tool that checks for dependencies and security vulnerabilities. - Usage and implementation of the latest version of all tools involved.
A09:2021 - Security Logging & Monitoring Failures	- Implementation of an effective and efficient API logging system. - Storing log records in API database, which is protected by authentication requirements and optional SSL certificates.

Application Security Testing

The system is prepared to particularly prevent 3 common types of attack: brute force attack (Luxemburk et al., 2021; OWASP, n.d.), denial of service attack (Humayun et al., 2020; OWASP, n.d.) and API injection (IEEE, 2022; OWASP, 2023).

1. *API Injection Attack*
    - In order to prevent API Injection Attacks, the system features input sanitisation measures to ensure that the data inputted is only of a particular type when asked (such as inputs only accepting strings when asked or integers when asked). This security measure therefore prevents characters or strings that could inject harmful code into the system.
    - The secure API file (called main.py) contains an “add-item” function that restricts authorised users to add items to the items.json file. This feature helps to prevent API injection attacks.
    - In addition, an unsecure API file (called mainunsecure.py) has been created to show the security vulnerabilities of the Online Retailer API that can arise from API injection attacks. This file does not contain the security measures that are featured in the main.py file and therefore would represent an injection attack as it could inject unrestricted data into the json databases.

```
            user_edit_choice = str(input('Input the corresponding number to the option you wish to choose:
#Defining editing of user password function & exit option
            if user_edit_choice == str(1):
                print('')
                new_user_password = str(input('Input the new password: '))
                User.edit_user(self, user_id_query=user_id_query, user_edit_choice=user_edit_choice,
                               new_user_password=new_user_password, new_user_username="",
                               new_user_name="", new_user_email_address="",
                               new_user_house_number="", new_user_street="", new_user_town="",
                               new_user_country="", new_user_postcode="", new_user_bank_name="",
                               new_user_bank_account_name="", new_user_bank_account_bsb="",
                               new_user_bank_account_number="", new_user_card_name="",
                               new_user_card_number="", new_user_card_expiry="",
                               new_user_card_cvc="")
                user_menu(self)
#Defining editing of username function & exit option
            elif user_edit_choice == str(2):
```

<<<

This code demonstrates input sanitisation (here, ensuring that the inputted data is only string type).

```
@app.put("/add-item")
def add_items(token: Annotated[str, Depends(OAUTH2_SCHEME)], username: str,
              item_id: str, item_name: str, item_price: str,
              item_description: str, item_stock: str):
    """Adds items to the items json file as a supplier"""
    date_now = datetime.now()
    date_str = date_now.strftime("%d/%m/%Y %H:%M:%S")
    user_dict = API_USERS_DB.get(username)
    user = UserInDB(**user_dict)
    existing_items = []
    if user.disabled is False:
        if user.role == 'S':
            for item in I_DATA['items']:
                existing_items.append(item['itemID'])
            if item_id in existing_items:
                write_api_log('ADDITEMS', 'ERROR', date_str,
                              'Item already exists in db - '
                              + user.username  , '409')
                raise HTTPException(status_code=status.HTTP_409_CONFLICT,
```

<<<

The “add-item” function is limited only to authorised users and restricts users to add items to the items.json file.

2. *Denial-of-Service (DoS) Attack*
    - The secure API file (called main.py) contains features that help to prevent Denial of Service (DoS) attacks. The “upload-file” function is limited only to authorised users and limits the file size and extension of the file. This would prevent a DoS attack.
    - In addition, the unsecure API file (called mainunsecure.py) does not contain the security measures that are featured in the main.py file and therefore leaves the API vulnerable to a DoS attack.

```python
@app.post("/upload-file")
async def upload(token: Annotated[str, Depends(OAUTH2_SCHEME)], username: str,
                 file: UploadFile = File(...)):
    """For customers to upload a file"""
    content_type = file.content_type
    cur_dir = os.getcwd() + '/uploads'
    date_now = datetime.now()
    date_str = date_now.strftime("%d/%m/%Y %H:%M:%S")
    user_dict = API_USERS_DB.get(username)
    user = UserInDB(**user_dict)
    if user.disabled is False:
        if user.role == 'C':
            if len(await file.read()) >= 1000:
                write_api_log('UPLOADFILE', 'ERROR', date_str,
                              'File is too large - ' + file.filename + ' - Size (KB) = '
                              + str(file.size) + ' - '
                              + user.username  , '400')
                raise HTTPException(status_code=status.HTTP_400_BAD_REQUEST,
```

<<<

The “upload-file” function is limited only to authorised users and limits the file size to 1000KB.

3. *Brute Force Attack*
    - The ‘onlineretailer.py’ contains features that help to prevent Brute Force Attacks including password complexity requirements, password encryption and administrator authentication.
    - Password complexity requirements include making sure the minimum password length is 8 characters long as well as there being a 10 second time limit to input a password or else it will log you out.
    - Administrator authentication requires the admin to authenticate themself again if using CRUD functions - the system will prompt the admin to “Input administrator key to continue: ”
    - These features can be turned off (commented out) which would make it more likely for Brute Force Attacks to occur.

```
#Setting a password, depositing it and encrypting it
        i = 1
        while i == 1:
            user_password_query = input('Input a password (must be at least 8 characters): ')
            if len(user_password_query) < 8 or len(user_password_query) > 30:
                print('Try again. Too short or too long.')
            else:
                break
        User.write_password_deposit(self, user_password_query)
        User.write_user_key(self)
        key = User.load_user_key(self)
        User.encrypt(self, 'encrypted_password.csv', key)
```

^^^ Code demonstrating the password complexity requirements (making sure the minimum password length is 8 characters long)

```
ID:332
Name:flower pot
Price:2.99
Description:small brown flower pot
Stock:50

------------------------------------------------------------
Admin authentication...
Input administrator key to continue:
```

<<<

Prompting an administrator to enter their admin key to continue…

Application Linting

Pylint Linting

The Pylint linter was used to identify stylistic, syntactical and programming errors and implement their solutions. Some examples of our Pylint linting…

‘users.py’ - Scored 3.22/10 ***before*** Pylint linting…

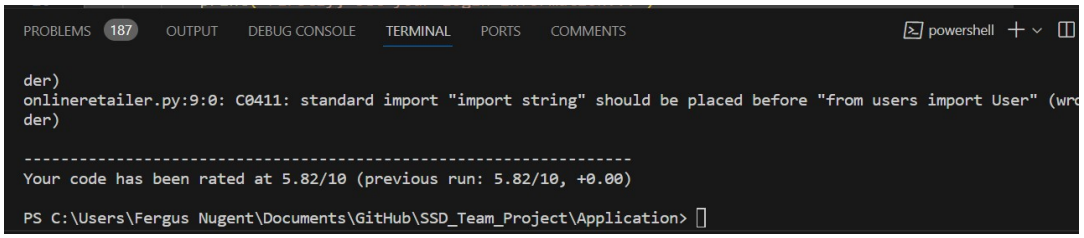
```
PROBLEMS 78   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS                powershell

t-order)
users.py:4:0: C0411: standard import "import json" should be placed before "from cryptography.fernet import Ferne
rt-order)
users.py:5:0: W0611: Unused import filelogging (unused-import)

-------------------------------------------------------------------
Your code has been rated at 3.22/10 (previous run: 3.22/10, +0.00)
```

then, ‘users.py’ - Scored 9.51/10 ***after*** Pylint linting…

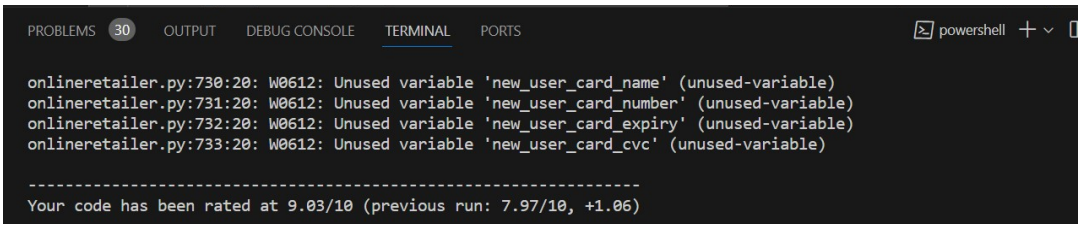
```
PROBLEMS 32   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS                powershell

users.py:115:4: R0913: Too many arguments (20/5) (too-many-arguments)
users.py:115:4: R0914: Too many local variables (23/15) (too-many-locals)
users.py:115:4: R0912: Too many branches (13/12) (too-many-branches)
users.py:5:0: W0611: Unused import filelogging (unused-import)

-------------------------------------------------------------------
Your code has been rated at 9.51/10 (previous run: 9.22/10, +0.29)
```

README.md

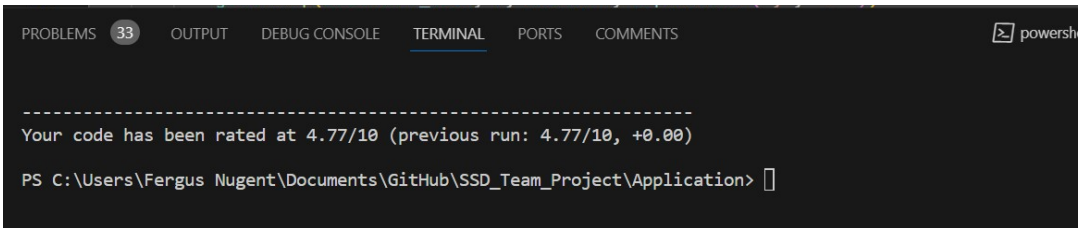
‘onlineretailer.py’ - Scored 5.82/10 ***before*** Pylint linting…



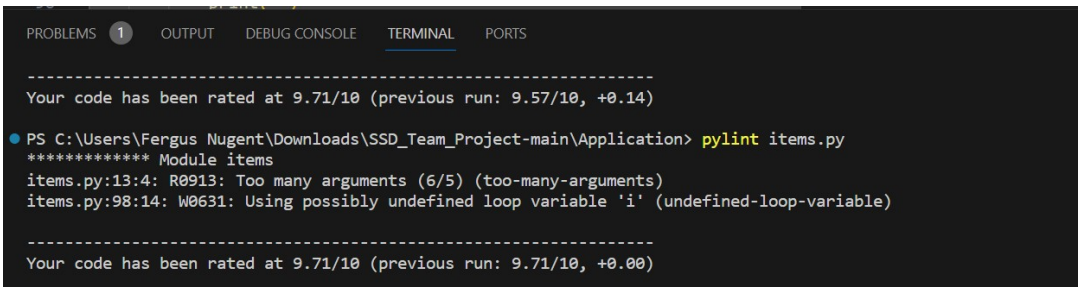
‘onlineretailer.py’ - Scored 9.03/10 ***after*** Pylint linting…



‘items.py’ - Scored 4.77/10 ***before*** Pylint linting…



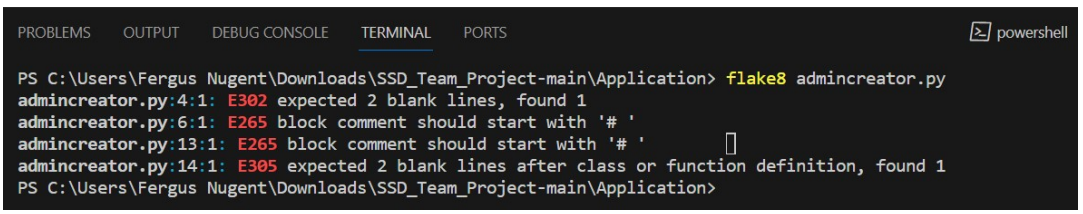
‘items.py’ - Scored 9.71/10 ***after*** Pylint linting…



Flake8 Linting

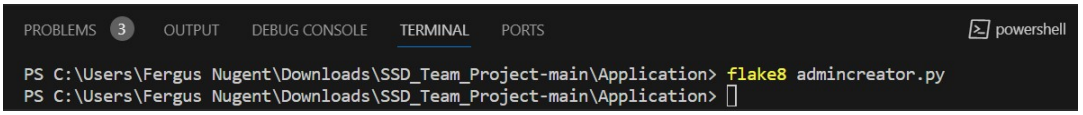
The Flake8 linter was used to prevent things like syntax errors, typos, bad formatting and incorrect styling. Some examples of our Flake8 linting…

‘adminCreator.py’ - ***before*** Flake8 linting…

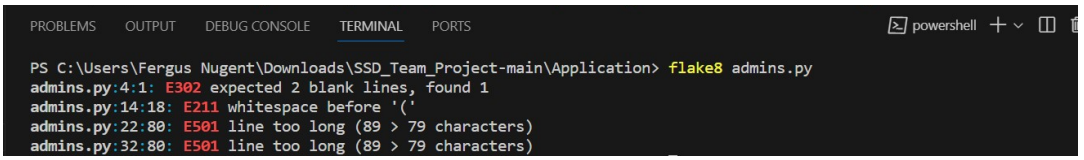


README.md

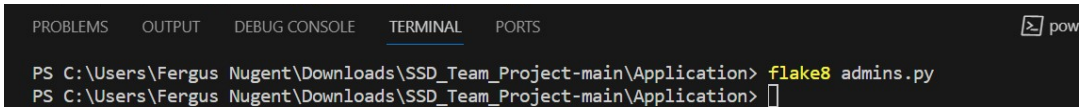
‘adminCreator.py’ - ***after*** Flake8 linting…



‘admins.py’ - ***before*** Flake8 linting…



‘admins.py’ - ***after*** Flake8 linting…



Application GDPR Compliance, Due Diligence & Privacy Policy

The application has been developed according to the stipulations set out by the General Data Protection Regulation (GDPR) (General Data Regulation Protection, 2018), as data processing relates to:

1. Accessing the user data
2. Automated processing, erasure and data portability
3. Data integrity and confidentiality

Security measure implementation has been prioritised throughout the application development process in a best effort to protect user data. OWASP (2021) security risks have been mitigated appropriately.

Collected user data is for intended purpose only. No sharing with 3rd parties will occur. No exposure of the API externally or via network because it is accessed by http. Therefore, it is in our best confidence that the user data is only accessible to authorised members with appropriate and permitted access.

Although, processing of application data remains subject to:

1. Consent of the data subjects
2. Deployment of the application (i.e. commercialised or network deployed)

It is assumed that if this application was prepared for commercial deployment, then the application would be pushed through the correct legal process and the necessary documentation and policies would be written.

References

General Data Protection Regulation. (2023) General Data Protection Regulation Information. Available from: https://gdpr-info.eu/ [Accessed 21 Oct 2023].

Humayun, M., Niazi, M., Jhanjhi, N., Alshayeb, M. & Mahmood, S. (2020) Cyber security threats and vulnerabilities: a systematic mapping study. *Arabian Journal for Science and Engineering*, 45, pp. 3171-3189.

IEEE Computer Society. (2022) API Injection Attacks: What They Are and How to Prevent Them. Available from: https://www.computer.org/publications/tech-news/trends/api-injection-attacks-prevention [Accessed 21 Oct 2023].

Luxemburk, J., Hynek, K. & Cejka, T. (2021) Detection of https brute-force attacks with packet-level feature set. *2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC)*, pp. 114-122.

OWASP. (2023) OWASP Top Ten. Available from: https://owasp.org/www-project-top-ten/ [Accessed 5 Oct 2023].

OWASP. (n.d.) Blocking Brute Force Attacks. Available from: https://owasp.org/www-community/controls/Blocking_Brute_Force_Attacks [Accessed 21 Oct 2023].

OWASP. (n.d.) Denial of Service. Available from: https://owasp.org/www-community/attacks/Denial_of_Service [Accessed 21 Oct 2023].

Van Rossum, G. (2001). Pep 8 - Style Guide for Python Code. Available from: https://peps.python.org/pep-0008/ [Accessed 5 Oct 2023].

Bibliography

Chocolatey. (2023) Chocolatey Software. Available from: https://chocolatey.org/ [Accessed 18 Oct 2023].

CrypTool-Online. (2023) OpenSSL. Available from: https://www.cryptool.org/en/cto/openssl [Accessed 21 Oct 2023].

Devwebtuts. (2023) Crack the Code: Mastering Brute Force Attack Simulations with Python and VS Code. Available from: https://medium.com/@devwebtuts_50448/performing-a-brute-force-attack-simulation-in-python-using-vs-code-cc0bd007fcad [Accessed 21 Oct 2023].

JSONLint. (2023) JSONLint Validator and Formatter. Available from: https://jsonlint.com/ [Accessed 21 Oct 2023].

Python Code Quality Authority (PyCQA). (2023) Fernet (symmetric encryption). Available from: https://github.com/pyca/cryptography/blob/main/docs/fernet.rst#fernet-symmetric-encryption [Accessed 18 Oct 2023].

Python Code Quality Authority (PyCQA). (2022) Flake8. Available from: https://github.com/pycqa/flake8/blob/main/docs/source/index.rst [Accessed 18 Oct 2023].

FiloSottile. (2022) mkcert. Available from: https://github.com/FiloSottile/mkcert [Accessed 18 Oct 2023].

Pylint. (2023) pylint 3.0.1. Available from: https://pypi.org/project/pylint/ [Accessed 18 Oct 2023].

Rajshirolkar. (2021) FastAPI over HTTPS for development on Windows. Available from: https://dev.to/rajshirolkar/fastapi-over-https-for-development-on-windows-2p7d [Accessed 21 Oct 2023].

Tiangolo. (2023) FastAPI. Available from: https://fastapi.tiangolo.com/ [Accessed 18 Oct 2023].

Tutorialspoint. (2023) DoS & DDoS attack. Available from: https://www.tutorialspoint.com/python_penetration_testing/python_penetration_testing_dos_and_ddos_attack.htm [Accessed 21 Oct 2023].

Uvicorn. (n.d.) Available from: https://www.uvicorn.org/ [Accessed 18 Oct 2023].