README: Secure eShop Application

Table of Contents

Overview

This project is a secure eShop application designed to simulate and protect against legitimate user operations and hacker attacks.

How to Run the App

Prerequisites

- Python version: 3.10 or later
- Required Python packages to import or install (if not in-built in Python):

  - **Flask**: Used for building the web application and API (Pallets, 2010).

- **Pytest**: For writing and executing unit and integration tests (Holger et al., 2015).
- **Requests**: To make HTTP requests for testing APIs (Reitz, 2024).
- **Werkzeug**: Required by Flask for WSGI server and utility functions (Pallets, 2007).
- **Hashlib**: For hashing values before storing.
- **Unittest.mock**: For mocking in tests.
- **Logging**: Python built-in module for logging and monitoring events.
- **Threading**: Enables concurrency (running multiple operations at once).
- **Json**: Handles data serialization and storage.
- **Time**: Tracks and manages time-based operations.
- **Random**: Generates random values for security and functionality.

(Python Software Foundation, 2025)

"pip/pip3 install flask pytest requests werkzeug" – running this should be sufficient to account for all external packages.

Running the Application

1. **Start the Flask API Server and Cli UI:** The server is integrated within the main application. To run:
    - ‘’’python main9.py’’’

This will launch the Flask server at http://127.0.0.1:5000 and the CLI interface simultaneously.

2. **Interact with the Application:**
    - CLI will prompt you to execute user operations or simulate attacks. You can simulate hacker attacks in Secure (where security features are activated) and Insecure (where security features are turned off) environments.
3. **Run Programmatic Tests:**.
    - ‘pytest unit_testing.py‘ – performs unit testing on the main code (Holger et al., 2015).
    - ‘pytest Integration_test.py’ – runs integration tests on the main code (Holger et al., 2015)..
    - ‘python3 penetration_tests.py’ – run in the command line to perform penetration tests.
4. **Run Linting:**
    - Ruff - a linter designed to enforce PEP 8 compliance and detect common Python issues in style and format.
        1. ‘pip install ruff’ – run in command line to install Ruff.
        2. ‘ruff check main9.py’ – runs ruff on main code.
    - Bandit - identifies common security vulnerabilities in Python code.
        1. ‘pip install bandit’ – install Bandit.
        2. ‘bandit -r main9.py’ – runs Bandit on the main code.
    - Flake8 - python linter, checks for style guide violations, undefined variables, and general code quality.
        1. ‘pip install flake8’
        2. ‘flake8 main9.py’
    - Pyre - validates type hints throughout the code.

1. ‘pip install pyre-check’
2. Run ‘pyre init’ and then ‘pyre check’

Functionalities of the eShop

User Roles and Operations

1. **Admin:**
    - Create, edit, view, and delete user accounts.
    - Log in and log out.
2. **Clerk:**
    - Manage inventory (create, update, read, and delete items).
    - Log in and log out.
3. **Customer:**
    - View items in the shop.
    - Add, view, and remove items in the cart.
    - Purchase items in the cart.
    - Update profile details.
    - Request account deletion per GDPR.

Hacker Simulations

- **Brute Force Attack:** Attempt to crack a user’s password.
- **DoS Attack:** Overload the server with requests.
- **API Injection Attack:** Attempt to bypass authentication using malicious payloads.

(Marie, 2023)

Security Features and Their Purpose

1. **Authentication:**
    - Login and user verification with Username and Password.
    - Hash-based password storage using SHA256.
    - Multi-factor authentication (OTP) to protect against unauthorized access.
    - REST API is leveraged to do company ID verification during the login.
2. **Authorization:**
    - Role-based access control (RBAC) to restrict functionalities based on the user's role (Admin, Customer, or Clerk).
3. **Rate Limiting:**
    - Maintains a log of requests per IP address. If the number of requests exceeds the limit in a set time, further requests are blocked to prevent DoS attacks.
4. **Brute Force Protection:**
    - Counts failed login attempts for each user. After a specified number of failures, the user receives a message that he's locked from the App.
5. **Session management:**
    - Sessions are tracked using randomly generated unique tokens that are created during the user's Login and stored in the repo. The unique token is deleted immediately after the user logout.
6. **Input Validation:**
    - All user inputs are sanitized, preventing SQL or script injections.
    - Regex is used to validate inputs - only letters are accepted as first and last names of a user. Weak passwords are rejected – passwords

should include at least six characters, at least one letter, and one number.

7. **The SECURITY_ENABLED toggle**
    - Allows turning Attack Mitigation features on/off for testing purposes.
8. **GDPR Compliance**
    - Users have the right to request deletion only for their account and all of their data will be removed from the repository and the user will exit the App (GDPR, 2025).
9. **Logging**
    - Crucial events are logged to detect any functional failures or security threats.

Attack Simulation & Mitigation features

1. **Brute Force Attack:**
    - The hacker attempts to guess the password of a user by iterating through a predefined list of passwords.
    - Retries are prevented after a maximum of two failed login attempts in secure mode, and a lockout period of 60 seconds is printed on the UI.
2. **Denial of Service (DoS):**
    - The hacker sends a large number of requests to the /verify_company_id endpoint to overload the system.
    - Rate limiting restricts each IP to 10 requests in a 30-second window. Excessive requests return a 429 Too Many Requests status code, and requests are stopped.
3. **API Injection Attack:**

- The hacker sends malicious payloads to bypass authentication checks or exploit vulnerabilities.
- Input sanitization prevents harmful payloads from being executed. For instance, 'OR '1'='1'' payloads are treated as strings rather than executable code (W3 Schools, 2025).

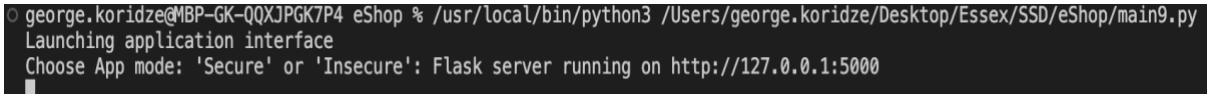
(Marie, 2023)

App demonstration and User Acceptance Test

1) Upon launching the application, the user is prompted to select one of two modes:

Secure Mode **-** enables all attack mitigation features, such as rate-limiting, brute force protection, and input sanitization.

Insecure Mode **-** disables these mechanisms.



The Flask API server and CLI interface are launched parallelly.

2) The user is asked with which role to log in:

3) Admin flow:

- *Authentication*: The admin provides credentials, and performs Multi-factor authentication with OTP..

```
Enter your role (Admin, Clerk, Customer, Hacker) or 'Exit': admin
Admin username: admin
Password: AdminPass
OTP sent: 189217
Enter the OTP sent to your registered email: 189217
User admin logged in successfully!
```

- *Authorization*: Role-based access control ensures only admins can manage user accounts. There is additional verification in the code based on the user's 'Role' attribute.

```
Options: Create User, Edit User, View User, Delete User, Log Out:
```

- *Input Validation* – regex is used to only allow letters to be inputted in first and last names and block weak passwords. The first two attempts are blocked due to these reasons. The third attempt passed with correct inputs.

```
Options: Create User, Edit User, View User, Delete User, Log Out: Create User
Enter new username: User1
Enter new password: UserPass
Enter company ID: Company1
Enter first name: Nick123
Enter last name: Koridze
Enter role (Admin, Clerk, Customer): Customer
Error: First name should only contain letters.
Options: Create User, Edit User, View User, Delete User, Log Out: Create User
Enter new username: User1
Enter new password: UserPass
Enter company ID: 12345
Enter first name: Nick
Enter last name: Koridze
Enter role (Admin, Clerk, Customer): Customer
Error: Password must be at least 6 characters long, contain at least one letter and one number.
Options: Create User, Edit User, View User, Delete User, Log Out: Create User
Enter new username: User1
Enter new password: UserPass123
Enter company ID: Company12
Enter first name: Nick
Enter last name: Koridze
Enter role (Admin, Clerk, Customer): Customer
User User1 created successfully!
```

- Admin can view all users, edit attributes on any user, and delete users from the repo.

```
Options: Create User, Edit User, View User, Delete User, Log Out: Edit User
Enter username to edit: User1
Enter new password (leave blank to skip): Giorgi123
Enter new company ID (leave blank to skip):
Enter new first name (leave blank to skip):
Enter new last name (leave blank to skip):
Enter new role (leave blank to skip):
User User1 updated successfully.
Options: Create User, Edit User, View User, Delete User, Log Out: View User
Current users in the system:
Username: admin, First Name: Admin, Last Name: User, Role: Admin, Company ID: 1234
Username: clerk1, First Name: Clerk, Last Name: One, Role: Clerk, Company ID: 5678
Username: customer1, First Name: Customer, Last Name: One, Role: Customer, Company ID: 91011
Username: User1, First Name: Nick, Last Name: Koridze123, Role: Customer, Company ID: Company12
Options: Create User, Edit User, View User, Delete User, Log Out: Delete User
Enter username to delete: User1
User User1 deleted successfully.
Options: Create User, Edit User, View User, Delete User, Log Out: view user
Current users in the system:
Username: admin, First Name: Admin, Last Name: User, Role: Admin, Company ID: 1234
Username: clerk1, First Name: Clerk, Last Name: One, Role: Clerk, Company ID: 5678
Username: customer1, First Name: Customer, Last Name: One, Role: Customer, Company ID: 91011
```

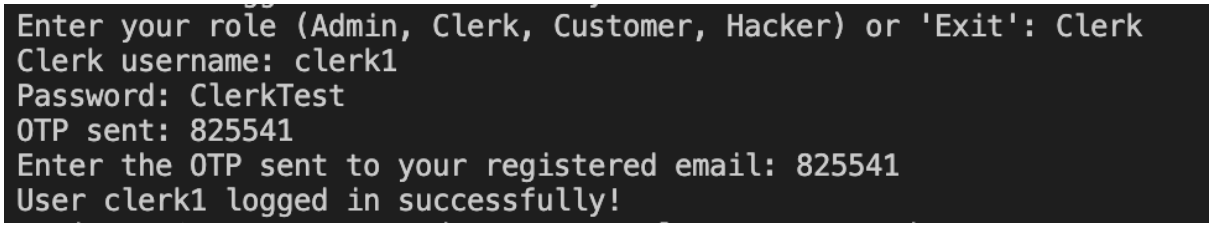
4) All of such changes in user profiles are persisted in the Data Repository.


```
{
    "users": {
        "admin": {
            "password": "0161e13f3124ae3455747b1a9ed78aa231253ae5c543cd28b9a6605835148299",
            "company_id": "1234",
            "first_name": "Admin",
            "last_name": "User",
            "role": "Admin",
            "cart": {}
        },
        "clerk1": {
            "password": "db9664c4d12af908907b27329688882cabc5b321d5343059900f5a18b8a3a61c",
            "company_id": "5678",
            "first_name": "Clerk",
            "last_name": "One",
            "role": "Clerk",
            "cart": {}
        },
        "customer1": {
            "password": "0cca9350cff04c9342cf5da79016d83fe1fa58c8a64bc5cdf6a8c0fa59131b23",
            "company_id": "91011",
            "first_name": "Customer",
            "last_name": "One",
            "role": "Customer",
            "cart": {}
        },
        "User1": {
            "password": "ab766898fa5066e105a5cf4c71be48e2016d0282dd13669eac088e5f22c6e7e1",
            "company_id": "Company12",
            "first_name": "Nick",
            "last_name": "Koridze",
            "role": "Customer"
        }
```

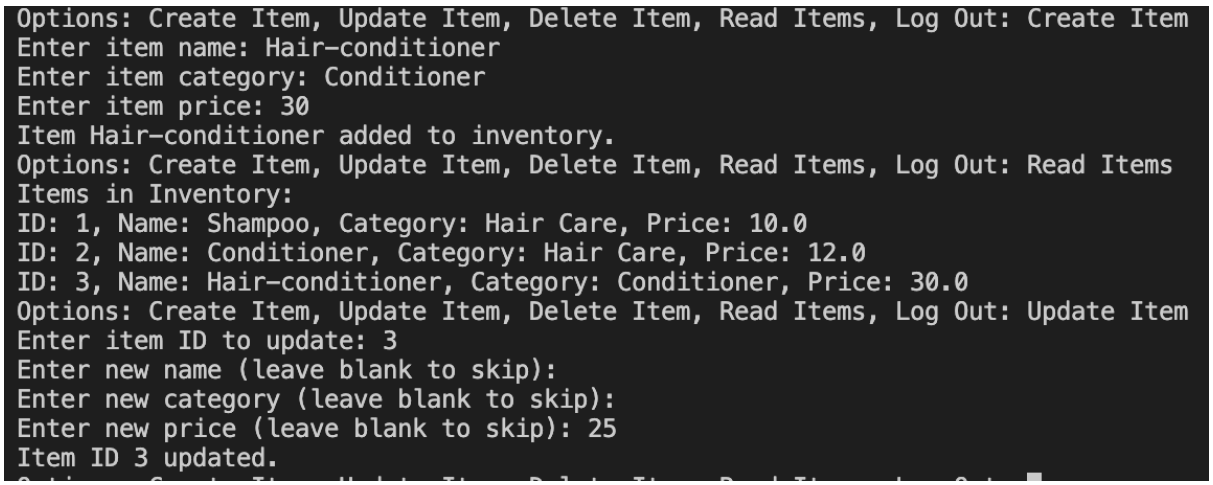
5) For all types of users (admin, customer & clerk), a session token is created and stored in the Repository. It's erased after the logout. After login:

After logout:

6) Clerk Flow:

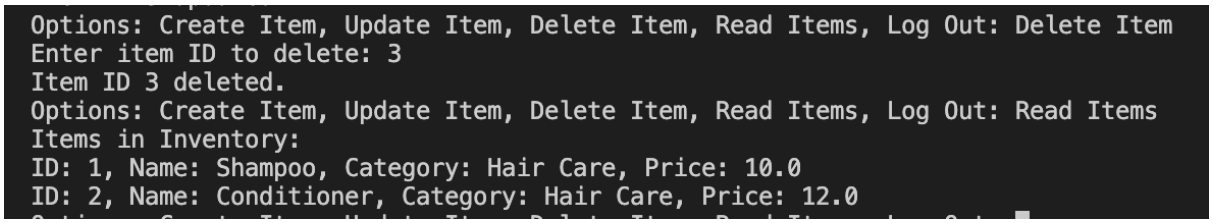


Clerk creating a new item, updating attributes in the existing item, and listing all of the items in the Repository:



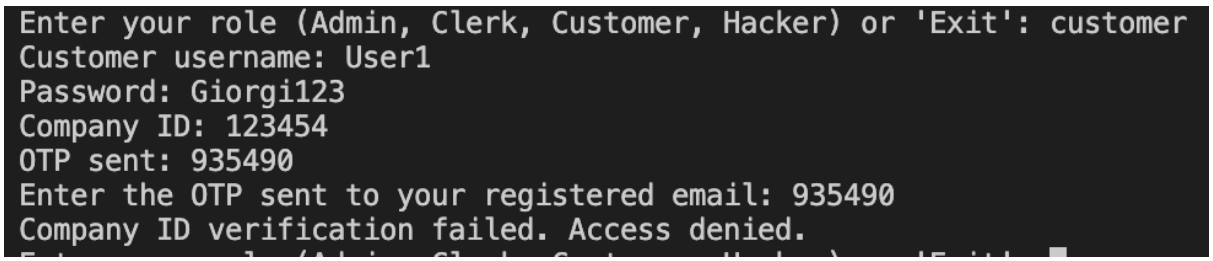
Changes in the Repository:

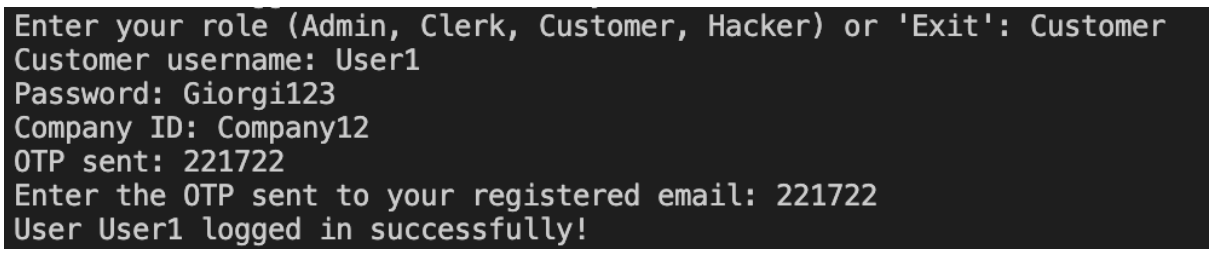
Deletion – deletes in runtime and in the Repository:

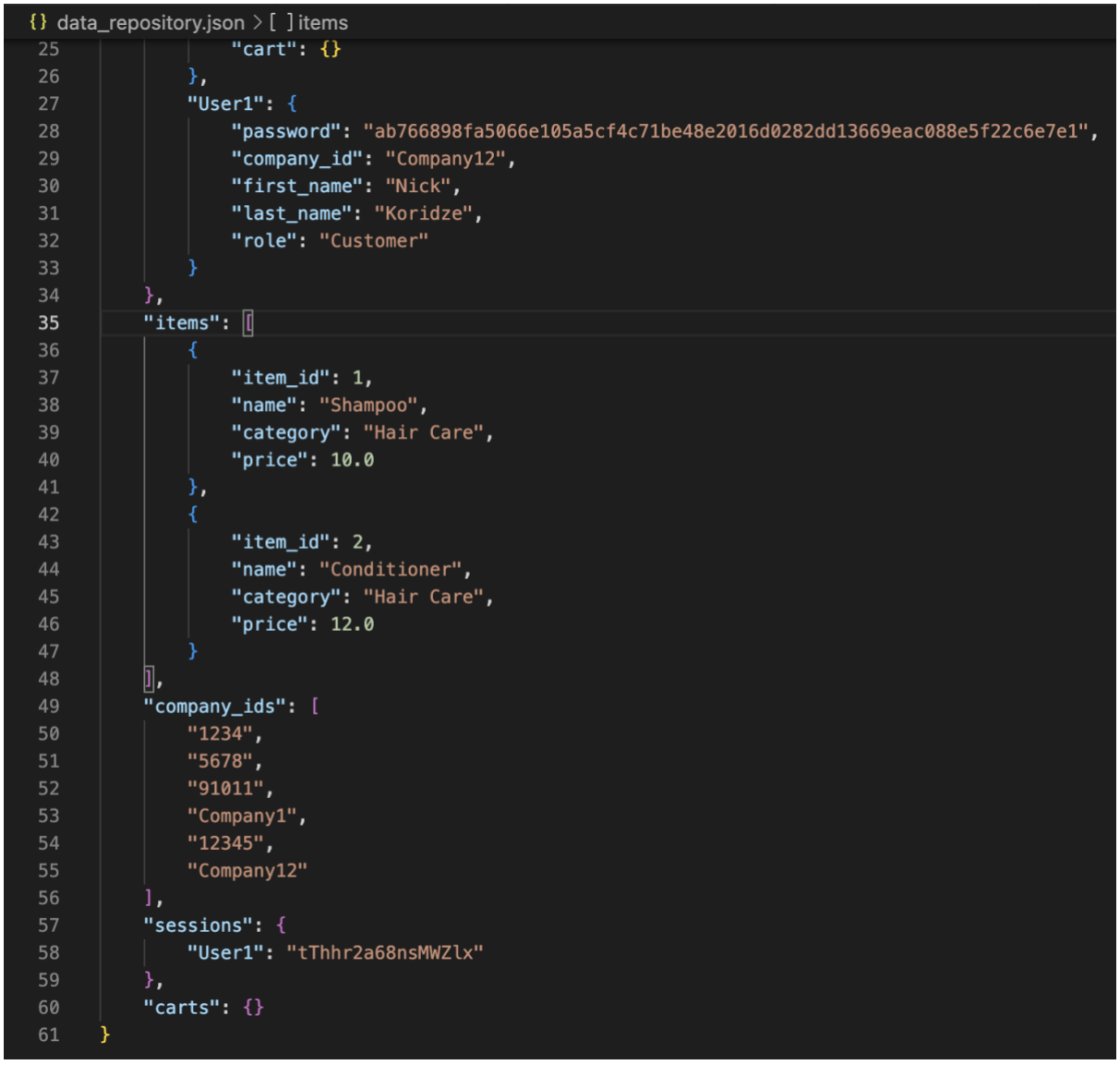


7) Customer flow:

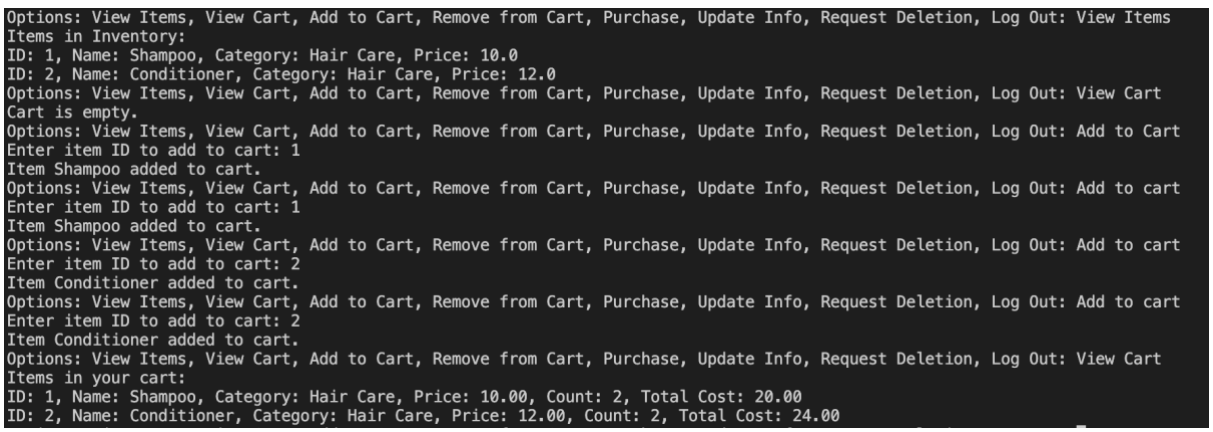
- *Authentication* - in addition to other verifications, REST API is leveraged to verify inputted Company ID with the repository.



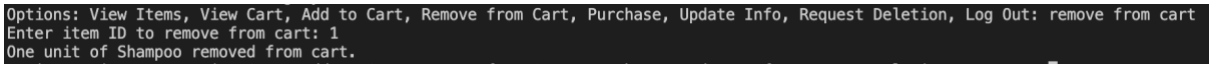


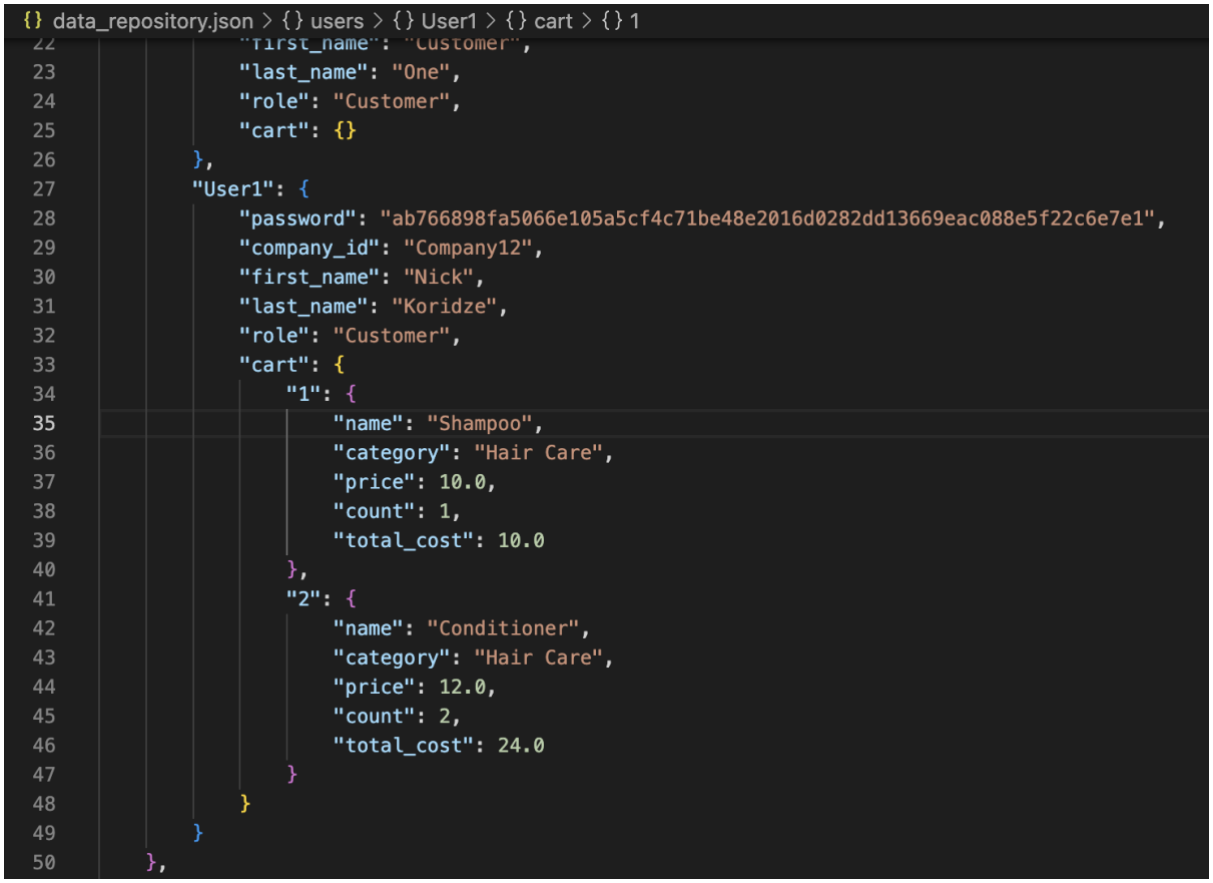


- Users can view all items in the shop, add items to their cart, and view their cart. The count and sum of items in the cart are calculated.



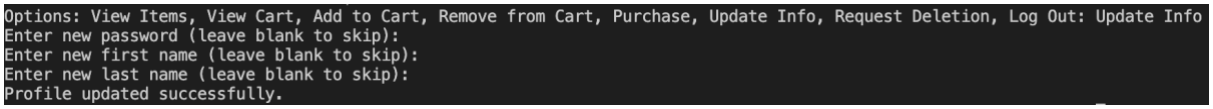
- After removing.





- After Purchase, total amount spent is printed and all the items are deleted from the Cart.

- The user can update his specific profile attributes in the repository and request the deletion of his account (which automatically logs him out).



```
Options: View Items, View Cart, Add to Cart, Remove from Cart, Purchase, Update Info, Request Deletion, Log Out: Request Deletion
User User1 logged out successfully!
Account deleted and logged out. Exiting...
Enter your role (Admin, Clerk, Customer, Hacker) or 'Exit':
```

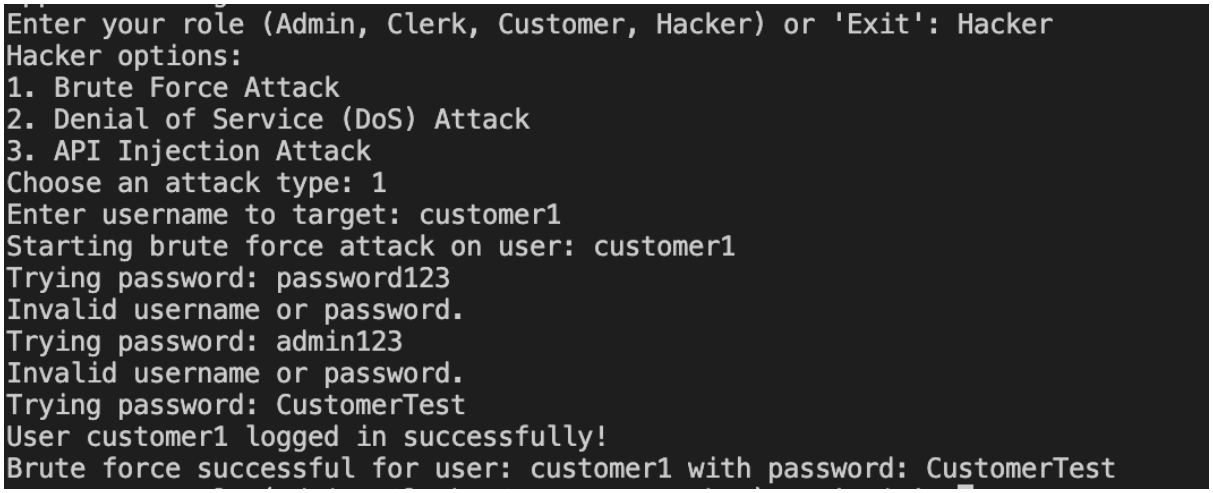
8) Event Logging:

```
application.log
2025-01-20 10:23:02 - WARNING - Failed login attempt for user customer1.
2025-01-20 10:23:04 - WARNING - Failed login attempt for user customer1.
2025-01-20 10:23:06 - INFO - User customer1 logged in successfully.
2025-01-20 11:52:29 - INFO - 127.0.0.1 - - [20/Jan/2025 11:52:29] "POST /verify_company_id HTTP/1.1" 200 -
2025-01-20 11:52:29 - INFO - 127.0.0.1 - - [20/Jan/2025 11:52:29] "POST /verify_company_id HTTP/1.1" 200 -
2025-01-20 11:52:29 - INFO - 127.0.0.1 - - [20/Jan/2025 11:52:29] "POST /verify_company_id HTTP/1.1" 200 -
2025-01-20 11:52:29 - INFO - 127.0.0.1 - - [20/Jan/2025 11:52:29] "POST /verify_company_id HTTP/1.1" 200 -
2025-01-20 11:52:29 - INFO - 127.0.0.1 - - [20/Jan/2025 11:52:29] "POST /verify_company_id HTTP/1.1" 200 -
2025-01-20 11:52:29 - INFO - 127.0.0.1 - - [20/Jan/2025 11:52:29] "POST /verify_company_id HTTP/1.1" 200 -
2025-01-20 11:52:29 - INFO - 127.0.0.1 - - [20/Jan/2025 11:52:29] "POST /verify_company_id HTTP/1.1" 200 -
2025-01-20 11:52:29 - INFO - 127.0.0.1 - - [20/Jan/2025 11:52:29] "POST /verify_company_id HTTP/1.1" 200 -
2025-01-20 11:52:29 - INFO - 127.0.0.1 - - [20/Jan/2025 11:52:29] "POST /verify_company_id HTTP/1.1" 200 -
2025-01-20 11:52:29 - INFO - 127.0.0.1 - - [20/Jan/2025 11:52:29] "POST /verify_company_id HTTP/1.1" 200 -
2025-01-20 11:52:29 - INFO - 127.0.0.1 - - [20/Jan/2025 11:52:29] "␛[31m␛[1mPOST /verify_company_id HTTP/1.1␛[0m" 429 -
2025-01-20 14:33:02 - WARNING - Failed login attempt for user da.
2025-01-20 18:26:24 - INFO - User customer1 logged in successfully.
2025-01-20 18:29:24 - INFO - User clerk1 logged in successfully.
2025-01-20 18:31:19 - INFO - 127.0.0.1 - - [20/Jan/2025 18:31:19] "POST /verify_company_id HTTP/1.1" 200 -
2025-01-20 18:31:19 - INFO - User customer1 logged in successfully.
2025-01-20 20:24:15 - INFO - User admin logged in successfully.
2025-01-20 20:39:58 - ERROR - Invalid first name: Nick123. Only letters are allowed.
2025-01-20 20:40:46 - ERROR - Invalid password provided for new user User1.
2025-01-20 20:41:36 - INFO - User User1 created by admin.
2025-01-20 20:48:34 - INFO - User User1 created by admin.
2025-01-20 21:04:45 - INFO - User clerk1 logged in successfully.
2025-01-20 21:12:03 - INFO - 127.0.0.1 - - [20/Jan/2025 21:12:03] "POST /verify_company_id HTTP/1.1" 200 -
2025-01-20 21:12:03 - INFO - User User1 logged in successfully.
2025-01-20 21:25:39 - INFO - 127.0.0.1 - - [20/Jan/2025 21:25:39] "␛[31m␛[1mPOST /verify_company_id HTTP/1.1␛[0m" 400 -
2025-01-20 21:38:00 - INFO - 127.0.0.1 - - [20/Jan/2025 21:38:00] "POST /verify_company_id HTTP/1.1" 200 -
2025-01-20 21:38:00 - INFO - User User1 logged in successfully.
2025-01-20 21:39:30 - INFO - Item Shampoo added to cart for user User1.
2025-01-20 21:39:35 - INFO - Item Shampoo added to cart for user User1.
```

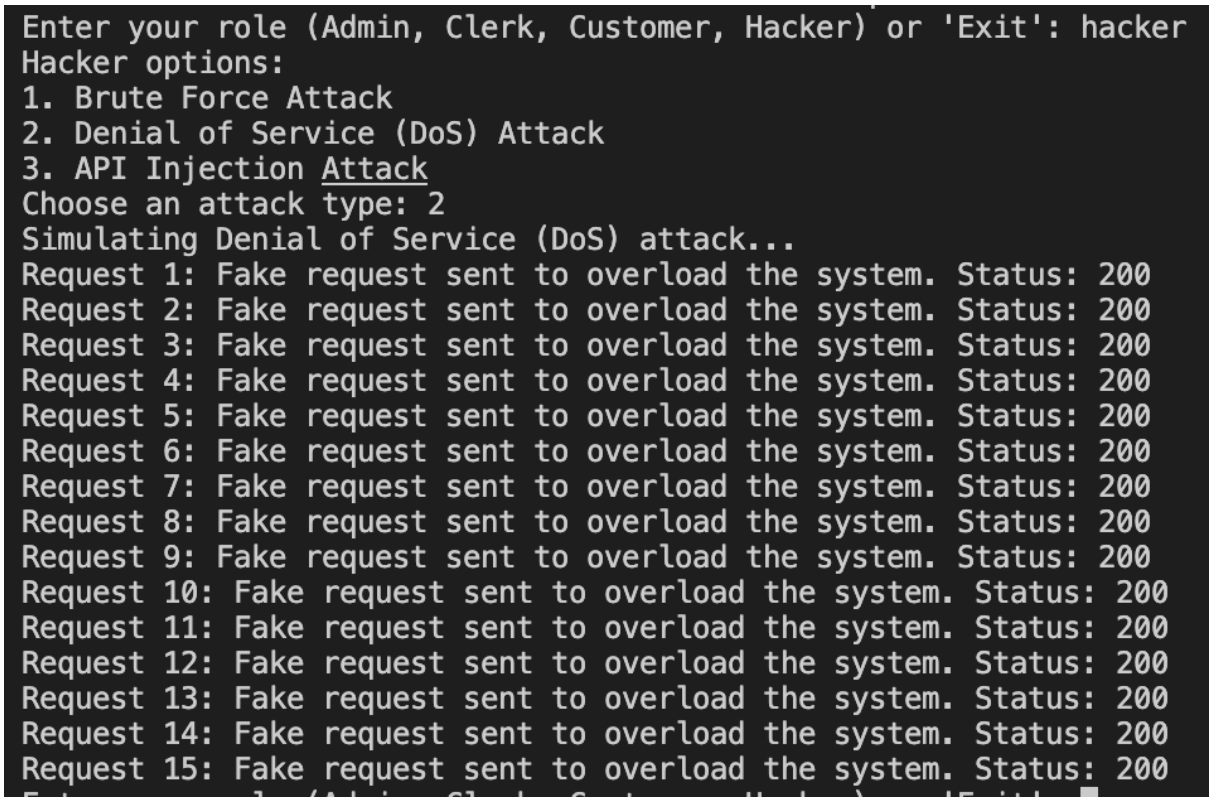
9) Hacker flow (when attack mitigation features are off):

```
Enter your role (Admin, Clerk, Customer, Hacker) or 'Exit': Hacker
Hacker options:
1. Brute Force Attack
2. Denial of Service (DoS) Attack
3. API Injection Attack
```

- *Brute Force Attack succeeds*



- *DoS attack on REST API succeeds*



- *API Injection fails* (but due to storing data in Json repo instead of SQL database)

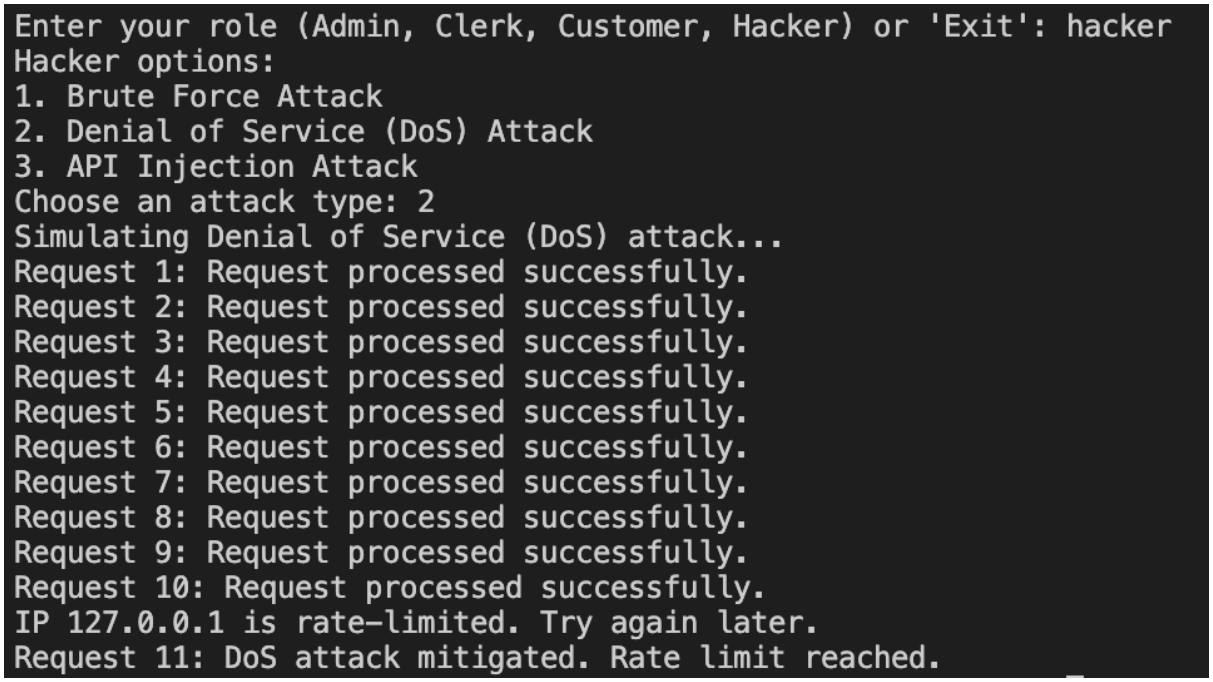
```
Enter your role (Admin, Clerk, Customer, Hacker) or 'Exit': hacker
Hacker options:
1. Brute Force Attack
2. Denial of Service (DoS) Attack
3. API Injection Attack
Choose an attack type: 2
Simulating Denial of Service (DoS) attack...
Request 1: Fake request sent to overload the system. Status: 200
Request 2: Fake request sent to overload the system. Status: 200
Request 3: Fake request sent to overload the system. Status: 200
Request 4: Fake request sent to overload the system. Status: 200
Request 5: Fake request sent to overload the system. Status: 200
Request 6: Fake request sent to overload the system. Status: 200
Request 7: Fake request sent to overload the system. Status: 200
Request 8: Fake request sent to overload the system. Status: 200
Request 9: Fake request sent to overload the system. Status: 200
Request 10: Fake request sent to overload the system. Status: 200
Request 11: Fake request sent to overload the system. Status: 200
Request 12: Fake request sent to overload the system. Status: 200
Request 13: Fake request sent to overload the system. Status: 200
Request 14: Fake request sent to overload the system. Status: 200
Request 15: Fake request sent to overload the system. Status: 200
```

10) Hacker flow (with mitigation features):

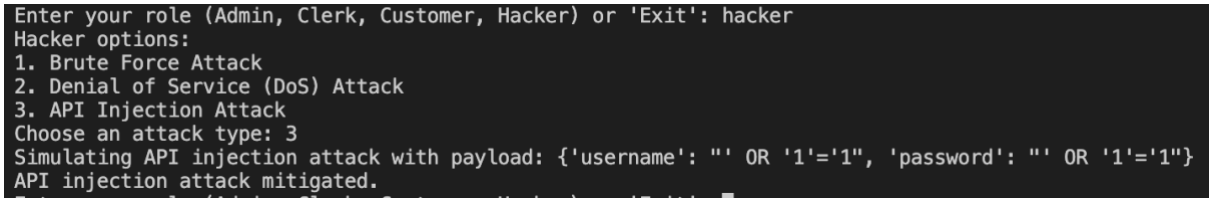
- *Brute Force Protection*: after a set number of failed attempts, the user account is locked.

```
Choose App mode: 'Secure' or 'Insecure': Flask server running on http://127.0.0.1:5000
Secure
App is running in Secure Mode.
Enter your role (Admin, Clerk, Customer, Hacker) or 'Exit': Hacker
Hacker options:
1. Brute Force Attack
2. Denial of Service (DoS) Attack
3. API Injection Attack
Choose an attack type: 1
Enter username to target: customer1
Starting brute force attack on user: customer1
Trying password: password123
Invalid username or password.
Trying password: admin123
Invalid username or password.
User customer1 is locked out for 60 seconds.
User customer1 is locked out until Mon Jan 20 22:01:00 2025.
Brute force attack blocked: User customer1 is now locked out.
```

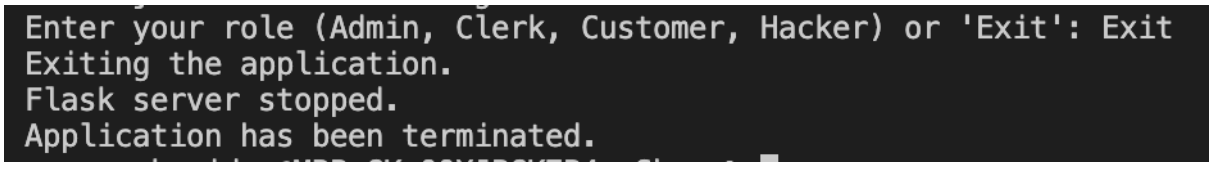
- *Rate Limiting*: limits the number of API requests per IP within a specified time window to mitigate DoS attacks.



- *Input Sanitization:* cleans malicious inputs in the API to mitigate injection attacks.

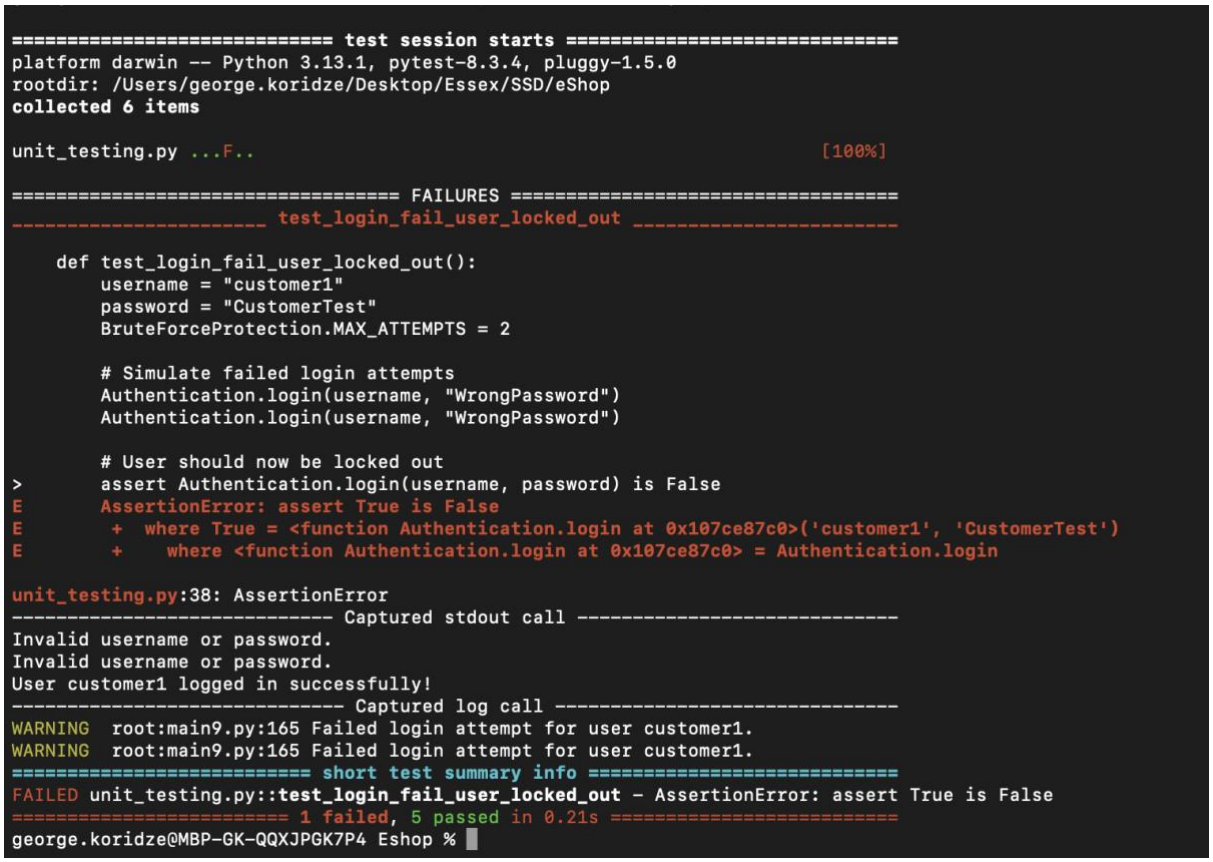


11) Click ‘Exit’ to end the App. The Flask API server and CLI interface are exited parallelly, for better user experience.

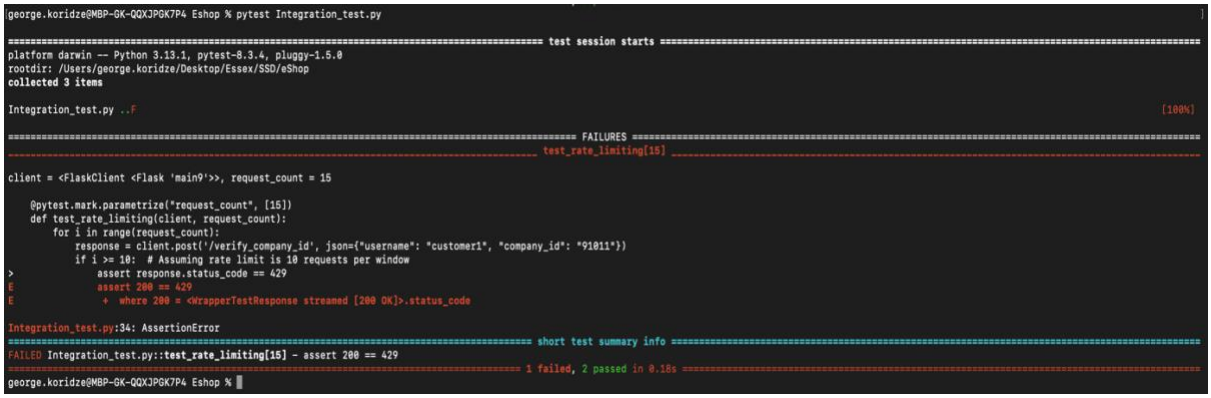


Programmatic Testing

1) Unit Testing – six functions were tested. One of them failed, while six succeeded.



2) Integration Testing – three aspects of API were tested with one failure and two successes.



3) Penetration Testing – SQL injection, fake logins and payload injections were mitigated.

4) Linting: all type and pep8 form tests were passed with Ruff and Pyre.

```
george.koridze@MBP-GK-QQXJPGK7P4 Eshop % pyre init
ƛ Which directory(ies) should pyre analyze? (Default: `.`): .
ƛ Successfully initialized pyre!
  You can view the configuration at `/Users/george.koridze/Desktop/Essex/SSD/eShop/.pyre_configuration`.
  You can now run the type checker with `pyre`.
george.koridze@MBP-GK-QQXJPGK7P4 Eshop % pyre check

ƛ No type errors found
```

```
george.koridze@MBP-GK-QQXJPGK7P4 Eshop % ruff check main9.py

All checks passed!
```

5) There were some security and style errors found with Flake8 and Bandit.

```
george.koridze@MBP-GK-QQXJPGK7P4 Eshop % flake8 main9.py

main9.py:28:80: E501 line too long (126 > 79 characters)
main9.py:36:80: E501 line too long (126 > 79 characters)
main9.py:44:80: E501 line too long (129 > 79 characters)
main9.py:53:80: E501 line too long (82 > 79 characters)
main9.py:54:80: E501 line too long (86 > 79 characters)
main9.py:58:80: E501 line too long (83 > 79 characters)
main9.py:70:80: E501 line too long (105 > 79 characters)
main9.py:72:1: E302 expected 2 blank lines, found 1
main9.py:74:49: E261 at least two spaces before inline comment
main9.py:74:80: E501 line too long (99 > 79 characters)
main9.py:80:22: W291 trailing whitespace
main9.py:82:26: W291 trailing whitespace
main9.py:88:1: E302 expected 2 blank lines, found 1
main9.py:92:80: E501 line too long (81 > 79 characters)
main9.py:100:80: E501 line too long (84 > 79 characters)
main9.py:112:1: E303 too many blank lines (3)
main9.py:113:1: E302 expected 2 blank lines, found 3
main9.py:118:1: E302 expected 2 blank lines, found 1
main9.py:124:1: E303 too many blank lines (3)
main9.py:125:1: E302 expected 2 blank lines, found 3
main9.py:141:1: E303 too many blank lines (3)
main9.py:142:1: E302 expected 2 blank lines, found 3
main9.py:157:80: E501 line too long (92 > 79 characters)
main9.py:165:80: E501 line too long (93 > 79 characters)
main9.py:168:9: E303 too many blank lines (2)
main9.py:175:5: E303 too many blank lines (2)
main9.py:189:80: E501 line too long (94 > 79 characters)
main9.py:214:1: E302 expected 2 blank lines, found 1
main9.py:221:80: E501 line too long (90 > 79 characters)
main9.py:224:80: E501 line too long (83 > 79 characters)
main9.py:224:84: W291 trailing whitespace
main9.py:225:80: E501 line too long (83 > 79 characters)
main9.py:226:80: E501 line too long (136 > 79 characters)
main9.py:230:80: E501 line too long (83 > 79 characters)
main9.py:231:80: E501 line too long (88 > 79 characters)
main9.py:238:80: E501 line too long (94 > 79 characters)
main9.py:247:5: E303 too many blank lines (2)
main9.py:257:80: E501 line too long (115 > 79 characters)
main9.py:259:80: E501 line too long (84 > 79 characters)
main9.py:271:80: E501 line too long (98 > 79 characters)
main9.py:272:80: E501 line too long (94 > 79 characters)
main9.py:280:5: E303 too many blank lines (2)
main9.py:281:80: E501 line too long (137 > 79 characters)
main9.py:281:138: W291 trailing whitespace
main9.py:282:80: E501 line too long (141 > 79 characters)
main9.py:283:80: E501 line too long (123 > 79 characters)
main9.py:283:124: W291 trailing whitespace
main9.py:284:80: E501 line too long (132 > 79 characters)
main9.py:286:80: E501 line too long (141 > 79 characters)
main9.py:287:80: E501 line too long (146 > 79 characters)
main9.py:300:80: E501 line too long (111 > 79 characters)
main9.py:305:80: E501 line too long (84 > 79 characters)
main9.py:311:5: E303 too many blank lines (2)
main9.py:314:80: E501 line too long (96 > 79 characters)
main9.py:320:80: E501 line too long (90 > 79 characters)
main9.py:321:80: E501 line too long (100 > 79 characters)
main9.py:338:80: E501 line too long (91 > 79 characters)
main9.py:342:80: E501 line too long (81 > 79 characters)
main9.py:353:1: E302 expected 2 blank lines, found 1
main9.py:365:80: E501 line too long (106 > 79 characters)
main9.py:383:80: E501 line too long (87 > 79 characters)
```

```
main9.py:353:1: E302 expected 2 blank lines, found 1
main9.py:365:80: E501 line too long (106 > 79 characters)
main9.py:383:80: E501 line too long (87 > 79 characters)
main9.py:389:5: E303 too many blank lines (2)
main9.py:400:80: E501 line too long (85 > 79 characters)
main9.py:414:80: E501 line too long (89 > 79 characters)
main9.py:415:80: E501 line too long (92 > 79 characters)
main9.py:424:80: E501 line too long (80 > 79 characters)
main9.py:432:80: E501 line too long (111 > 79 characters)
main9.py:436:80: E501 line too long (80 > 79 characters)
main9.py:438:80: E501 line too long (110 > 79 characters)
main9.py:444:80: E501 line too long (89 > 79 characters)
main9.py:449:80: E501 line too long (87 > 79 characters)
main9.py:455:80: E501 line too long (80 > 79 characters)
main9.py:456:80: E501 line too long (116 > 79 characters)
main9.py:472:5: E303 too many blank lines (2)
main9.py:483:5: E303 too many blank lines (2)
main9.py:511:80: E501 line too long (119 > 79 characters)
main9.py:540:1: E303 too many blank lines (3)
main9.py:541:1: E302 expected 2 blank lines, found 3
main9.py:548:80: E501 line too long (96 > 79 characters)
main9.py:559:80: E501 line too long (109 > 79 characters)
main9.py:575:5: E303 too many blank lines (2)
main9.py:580:80: E501 line too long (82 > 79 characters)
main9.py:582:80: E501 line too long (109 > 79 characters)
main9.py:586:80: E501 line too long (116 > 79 characters)
main9.py:589:80: E501 line too long (132 > 79 characters)
main9.py:591:26: W291 trailing whitespace
main9.py:600:80: E501 line too long (99 > 79 characters)
main9.py:605:80: E501 line too long (110 > 79 characters)
main9.py:613:80: E501 line too long (94 > 79 characters)
main9.py:614:80: E501 line too long (98 > 79 characters)
main9.py:615:80: E501 line too long (98 > 79 characters)
main9.py:616:80: E501 line too long (96 > 79 characters)
main9.py:617:80: E501 line too long (86 > 79 characters)
main9.py:638:80: E501 line too long (184 > 79 characters)
main9.py:650:5: E303 too many blank lines (2)
main9.py:655:80: E501 line too long (82 > 79 characters)
main9.py:657:80: E501 line too long (109 > 79 characters)
main9.py:661:80: E501 line too long (119 > 79 characters)
main9.py:664:80: E501 line too long (84 > 79 characters)
main9.py:676:80: E501 line too long (86 > 79 characters)
main9.py:677:80: E501 line too long (94 > 79 characters)
main9.py:678:80: E501 line too long (88 > 79 characters)
main9.py:679:80: E501 line too long (136 > 79 characters)
main9.py:695:5: E303 too many blank lines (2)
main9.py:701:80: E501 line too long (82 > 79 characters)
main9.py:714:80: E501 line too long (109 > 79 characters)
main9.py:718:80: E501 line too long (135 > 79 characters)
main9.py:730:66: W291 trailing whitespace
main9.py:744:80: E501 line too long (94 > 79 characters)
main9.py:745:80: E501 line too long (98 > 79 characters)
main9.py:746:80: E501 line too long (96 > 79 characters)
main9.py:749:80: E501 line too long (100 > 79 characters)
main9.py:751:80: E501 line too long (89 > 79 characters)
main9.py:753:80: E501 line too long (87 > 79 characters)
main9.py:769:5: E303 too many blank lines (2)
main9.py:793:73: W291 trailing whitespace
main9.py:794:80: E501 line too long (116 > 79 characters)
```

```
george.koridze@MBP-GK-QQXJPGK7P4 Eshop % bandit -r main9.py

[main]  INFO    profile include tests: None
[main]  INFO    profile exclude tests: None
[main]  INFO    cli include tests: None
[main]  INFO    cli exclude tests: None
[main]  INFO    running on Python 3.13.1
Run started:2025-01-20 12:28:03.883383

Test results:
>> Issue: [B311:blacklist] Standard pseudo-random generators are not suitable for security/cryptographic purposes.
   Severity: Low   Confidence: High
   CWE: CWE-330 (https://cwe.mitre.org/data/definitions/330.html)
   More Info: https://bandit.readthedocs.io/en/1.8.2/blacklists/blacklist_calls.html#b311-random
   Location: ./main9.py:120:19
119         """Generate a random session token."""
120         return ''.join(random.choices(string.ascii_letters + string.digits, k=16))
121

--------------------------------------------------
>> Issue: [B311:blacklist] Standard pseudo-random generators are not suitable for security/cryptographic purposes.
   Severity: Low   Confidence: High
   CWE: CWE-330 (https://cwe.mitre.org/data/definitions/330.html)
   More Info: https://bandit.readthedocs.io/en/1.8.2/blacklists/blacklist_calls.html#b311-random
   Location: ./main9.py:188:22
187             """Simulate sending an OTP to the user."""
188             otp = ''.join(random.choices(string.digits, k=6))
189             print(f"OTP sent: {otp}")  # In a real system, this would be sent to the user securely

--------------------------------------------------
>> Issue: [B113:request_without_timeout] Call to requests without timeout
   Severity: Medium   Confidence: Low
   CWE: CWE-400 (https://cwe.mitre.org/data/definitions/400.html)
   More Info: https://bandit.readthedocs.io/en/1.8.2/plugins/b113_request_without_timeout.html
   Location: ./main9.py:252:23
251             for i in range(15):
252                 response = requests.post(
253                     "http://127.0.0.1:5000/verify_company_id",
254                     json={"username": "customer1", "company_id": "91011"}
255                 )
256                 if not SECURITY_ENABLED:

--------------------------------------------------
>> Issue: [B113:request_without_timeout] Call to requests without timeout
   Severity: Medium   Confidence: Low
   CWE: CWE-400 (https://cwe.mitre.org/data/definitions/400.html)
   More Info: https://bandit.readthedocs.io/en/1.8.2/plugins/b113_request_without_timeout.html
   Location: ./main9.py:271:23
270                 # Send a request to the vulnerable API endpoint
271                 response = requests.post("http://127.0.0.1:5000/verify_company_id", json=fake_payload)
272                 if response.status_code == 200 and "success" in response.json().get("status", ""):

--------------------------------------------------
>> Issue: [B113:request_without_timeout] Call to requests without timeout
   Severity: Medium   Confidence: Low
   CWE: CWE-400 (https://cwe.mitre.org/data/definitions/400.html)
   More Info: https://bandit.readthedocs.io/en/1.8.2/plugins/b113_request_without_timeout.html
   Location: ./main9.py:704:19
703             # Use Flask API to verify company ID
704             response = requests.post(
705                 "http://127.0.0.1:5000/verify_company_id",
706                 json={"username": username, "company_id": company_id}
707             )
708
```

```
--------------------------------------------------

Code scanned:
        Total lines of code: 614
        Total lines skipped (#nosec): 0

Run metrics:
        Total issues (by severity):
                Undefined: 0
                Low: 2
                Medium: 3
                High: 0
        Total issues (by confidence):
                Undefined: 0
                Low: 3
                Medium: 0
                High: 2
Files skipped (0):
```

Further Developments

- Errors were identified during linting and testing. They have been documented above, but due to time constraints, they were not corrected.
- Although the application is already functional and mostly secure, further refinements are planned to ensure complete adherence to linting, pep8, as well as code structure and security standards.

References

GDPR. (n.d.) *General Data Protection Regulation*. Available from: https://gdpr.eu/tag/gdpr/ [Accessed 20 January 2025].

Holger, K. & Pytest-dev team (2015) Pytest: helps you write better programs. Available from: https://docs.pytest.org/en/stable/ [Accessed 19 January 2025].

Reitz, K. (2024) Requests: HTTP for Humans™. Available from: https://requests.readthedocs.io/en/latest/ [Accessed 19 January 2025].

Pallets (2010) Flask. Available from: https://flask.palletsprojects.com/en/stable/ [Accessed 19 January 2025].

Pallets (2007) Werkzeug. Available from: https://werkzeug.palletsprojects.com/en/stable/ [Accessed 19 January 2025].

Python Software Foundation (2025) hashlib — Secure hashes and message digests. Available from: https://docs.python.org/3/library/hashlib.html [Accessed 19 January 2025].

Python Software Foundation (2025) unittest — Unit testing framework. Available from: https://docs.python.org/3/library/unittest.html [Accessed 19 January 2025].

Python Software Foundation (2025) logging — Logging facility for Python. Available from: https://docs.python.org/3/library/logging.html [Accessed 19 January 2025].

Python Software Foundation (2025) random — Generate pseudo-random numbers. Available from: https://docs.python.org/3/library/random.html [Accessed 19 January 2025].

Python Software Foundation (2025) time — Time access and conversions. Available from: https://docs.python.org/3/library/time.html [Accessed 19 January 2025].

Python Software Foundation (2025) json — JSON encoder and decoder. Available from: https://docs.python.org/3/library/json.html [Accessed 19 January 2025].

Python Software Foundation (2025) threading — Thread-based parallelism. Available from: https://docs.python.org/3/library/threading.html [Accessed 19 January 2025].

Marie, J. (2023) The Anatomy of an API Abuse Attack: A Hacker's Process Unveiled. Available from: https://www.traceable.ai/blog-post/the-anatomy-of-an-api-abuse-attack-a-hackers-process-unveiled [Accessed 19 January 2025].

W3 Schools (2025) SQL Injection. Available from: https://www.w3schools.com/sql/sql_injection.asp [Accessed 19 January 2025].