Development Individual Project

Project Overview

The project implements a school grading system with user-role-based access. Depending on their roles, users can create, view, update, and delete records such as user accounts, study groups, marks, and messages.

The application includes server and client components. It runs in either secure or non-secure mode. A malicious client is also provided to test the system's security features.

Running the Application

1. Ensure Python 3 is present in the system (the application was tested with Python 3.12).
2. In the project root, execute main.py:

```
python main.py --server --client
```

This command will set up the server in the secure mode and run the client interface. To disable security, add **`--non_secure`** to the list of arguments. To run the malicious client, use **`--malicious`**. To see all keys, run the application with the **`--help`** key.

Server Application

The server application initialises the HTTP server, data repositories, and the **`ActionFactory`**. The factory acts as a router: it receives requests, generates specific Action implementations, and executes them. Using the *Factory* design pattern helps organize and manage supported actions, separates HTTP request handling from business logic, and decouples the server logic from action implementations, providing a

single entry point for the supported actions (Ellis, Stylos and Myers, 2007). The core architecture of the application is shown in the UML class diagram in Figure 1. The server provides a *Representational State Transfer Application Programming Interface* (REST API) that is uniform, cacheable, and stateless (Fielding, 2000; IBM, 2025).

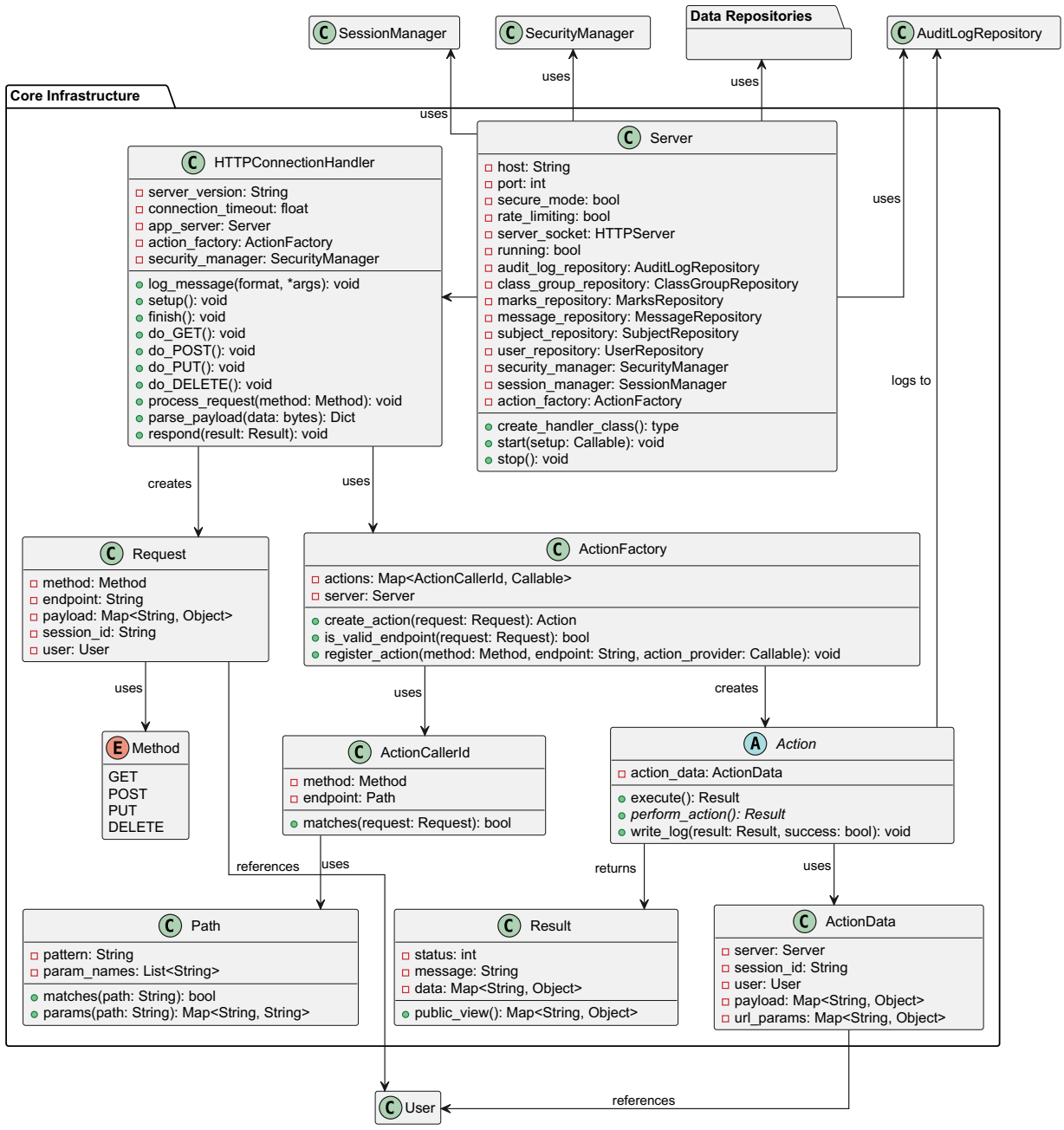

Figure 1. UML Class diagram for the application core

The specific actions exposed via API endpoints implement *Create, Read, Update, and Delete* (CRUD) functionality for the repositories containing users, marks, subjects, etc. (CrowdStrike, 2022) Although a relational database would be more suitable for a production application, this implementation stores data in JSON files when the application shuts down. This choice is supported by the standard Python library's built-in handling of this format and allows to avoid overhead of managing a database engine (Python Software Foundation, 2025b).

Server Testing

The **`server_tests`** module implements a comprehensive testing strategy using Python's standard *unittest* library (Python Software Foundation, 2025c). It includes unit and integration tests, with the *mock* library used to emulate external dependencies and verify class interactions (Python Software Foundation, 2025d).

The server tests are run with the following command:

```
python -m unittest discover server_tests
```

Expected output:

```
[...]
----------------------------------------------------------------------
Ran 241 tests in 26.656s
OK
```

Client Application

A command-line interface (CLI) client is implemented in the client module. The client application supports caching to improve data display and allows users to log in to the

system and run commands associated with the different endpoints on the server, as displayed on the Figure 2.

```
You are not logged in.
Would you like to login? (yes/no): y
Username: admin
Password: admin
Login successful!
Loading entities...
Entities loaded successfully!

Enter command (or 'help' for available commands, 'exit' to quit): list_users
User(id=1, username='admin', role='ADMIN')
User(id=2, username='teacher', role='TEACHER')
User(id=3, username='student1', role='STUDENT')
User(id=4, username='student2', role='STUDENT')

Enter command (or 'help' for available commands, 'exit' to quit):
```

Figure 2. Using the client application

Client Testing

The client test suite includes integration tests to ensure correct request execution and processing. The tests are run using the following command:

```
python -m unittest discover client_tests
```

Expected output:

```
[...]
----------------------------------------------------------------------
Ran 16 tests in 6.267s
OK
```

Security Overview

The system implements login functionality to authenticate users and partially complies to the *General Data Protection Regulation* (GDPR), specifically to the Articles 25, 30, and 32: access to personal data is restricted by permissions; unauthenticated and unauthorised access is blocked. For example, students can only view their own marks, and teachers and students can only see the messages they sent or received; otherwise, HTTP status *403 Forbidden* is returned. The server logs user actions, and cryptographic software is used to protect connections and passwords. (‘Regulation (EU) 2016/679 of the European Parliament and of the Council (General Data Protection Regulation)’, 2016)

A dedicated **`SecurityManager`** is set up as the entry point for the different parts of the application to provide security-related functionality. The server can run in two modes: with security features enabled or disabled. The **`malicious_client`** module includes a client that performs attacks of the server (Figure 3).

```
Enter command: dos
Running DoS attack on /...
Starting attack on / with 150 connections
Attack will run for 30 seconds with 10s delay between headers
Target: localhost:8000 (HTTP)
Attack in progress: 6/150 connections active, 5.1/30s elapsed
Attack in progress: 6/150 connections active, 10.1/30s elapsed
Attack in progress: 6/150 connections active, 15.2/30s elapsed
Attack in progress: 6/150 connections active, 20.3/30s elapsed
Attack in progress: 6/150 connections active, 25.4/30s elapsed
Attack duration reached. Stopping all connection threads...
Attack completed. Forcefully closing 5 connections...
Attack completed in 30.00 seconds (plus 0.53s cleanup)
Connections: 6 established, 144 failed
Headers sent: 18
Maximum concurrent connections: 6

Enter command:
```

Figure 3. Running attacks using the specialised client

A dedicated test suite in **`malicious_client_tests`** automatically runs attack scenarios on secure and non-secure server instances. The test suite is executes using the following command:

```
python -m unittest discover malicious_client_tests
```

Expected output:

```
[...]
----------------------------------------------------------------------
Ran 6 tests in 25.133s
OK
```

HTTP Traffic Encryption

In secure mode, the HTTP server will be set up with Transport Level Security (TLS) support, the industry standard for protecting HTTP connections. It encrypts the transported data and prevents *Man in the Middle* (MiTM) attacks.

In non-secure mode, the requests are sent over HTTP without encryption and can be read by a third party. Figure 4 demonstrates a login request intercepted using Wireshark, a network protocol analyser used to capture local traffic (Wireshark Foundation, no date). In the *Open Worldwide Application Security Project (OWASP) Top 10* classification, this would classify as *A02:2021 "Cryptographic Failures"* (OWASP Top 10 Team, 2021a). Sending vulnerable information over plain HTTP is also categorised in the *Common Weakness Enumeration* (CWE) catalog under *CWE-319 "Cleartext Transmission of Sensitive Information"* (The MITRE Corporation, no date d).

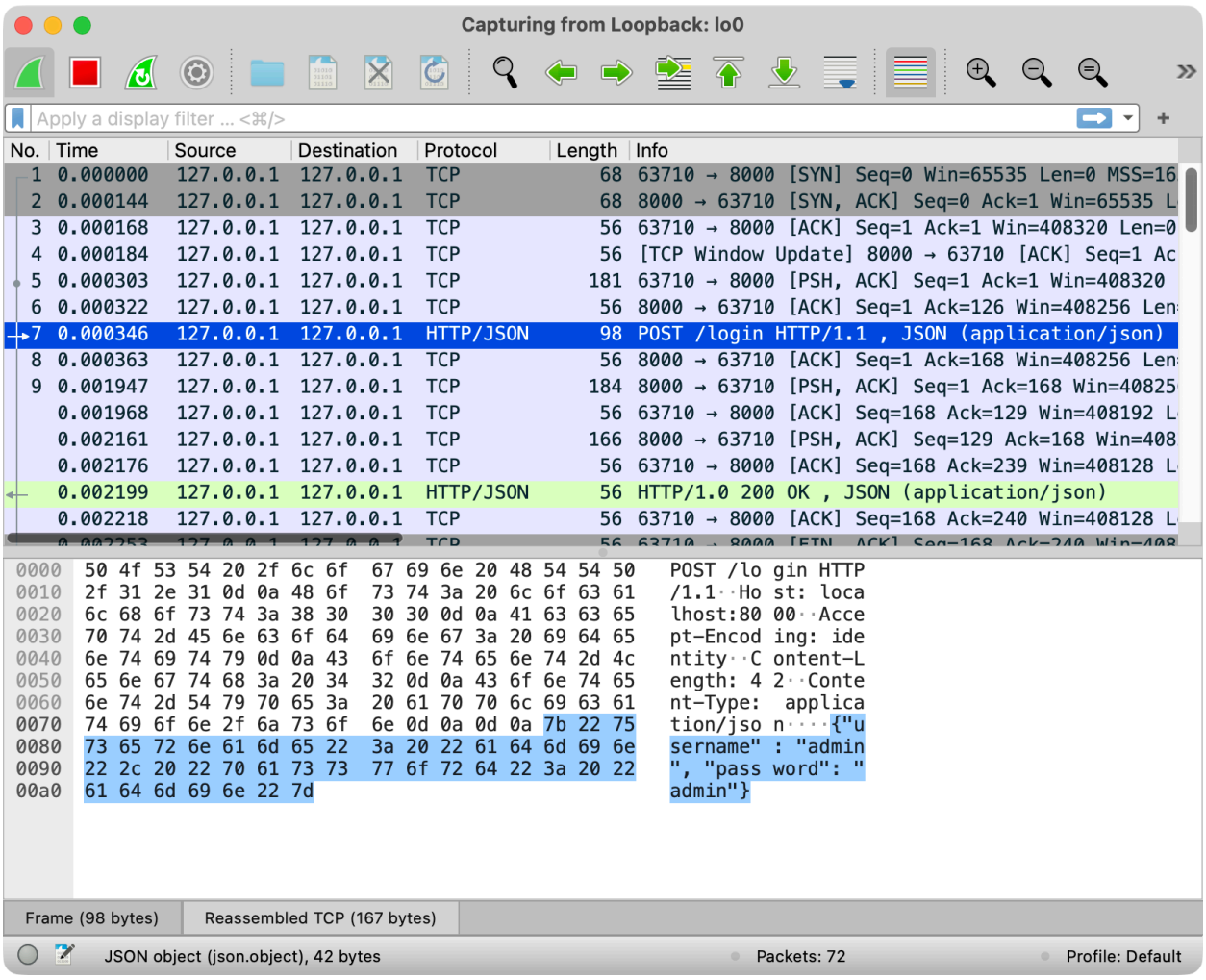

Figure 4. Intercepting user credentials using Wireshark

Rate Limiting and Session Management

The server records recent requests and enforces rate limiting to prevent *Denial of Service* (DoS) attacks and *Brute Force* attacks. After several failed logins, further requests are denied. This prevents the attacks categorised as *A07:2021 "Identification and Authentication Failures"* and *CWE-307 "Improper Restriction of Excessive Authentication Attempts" (OWASP Top 10 Team, 2021c; The MITRE Corporation, no date b)*.

If a session ID is leaked, it cannot be used indefinitely, as sessions expire after a timeout, preventing *CWE-613 "Insufficient Session Expiration"* (The MITRE Corporation, no date g).

The malicious client simulates a *Brute Force* attack by trying commonly used passwords revealed in the research of the vendor Nord Security (2024). Figure 5 demonstrates that when rate limiting is enabled on the server, the attack fails, and a session ID is not acquired.

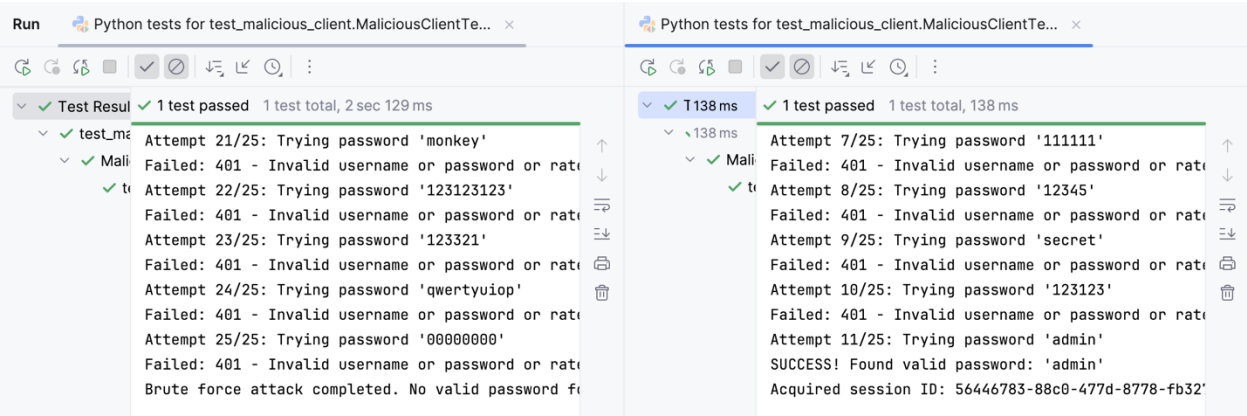

Figure 5. Execution of a Brute Force attack with secure (left) and non-secure (right) server

Connection Timeouts

The Python HTTP server processes requests sequentially in a single thread, making it susceptible to *Slowloris*-style DoS attacks. This type of attack involves sending partial requests to the server and never completing them, which can block the processing of further requests (Damon *et al.*, 2012). This is categorised as *CWE-400 "Uncontrolled Resource Consumption"* (The MITRE Corporation, no date e).

To mitigate such attacks, a lower connection timeout can be set so connections close sooner (Shorey *et al.*, 2018). As the application does not send or receive large amounts

of data, it will not affect the server's functionality. When run in secure mode, the attack only causes longer response times, not full server unavailability.

Figure 6 demonstrates that in secure mode the server keeps responding to a legitimate user during the attack. In non-secure mode, a connection cannot be established.

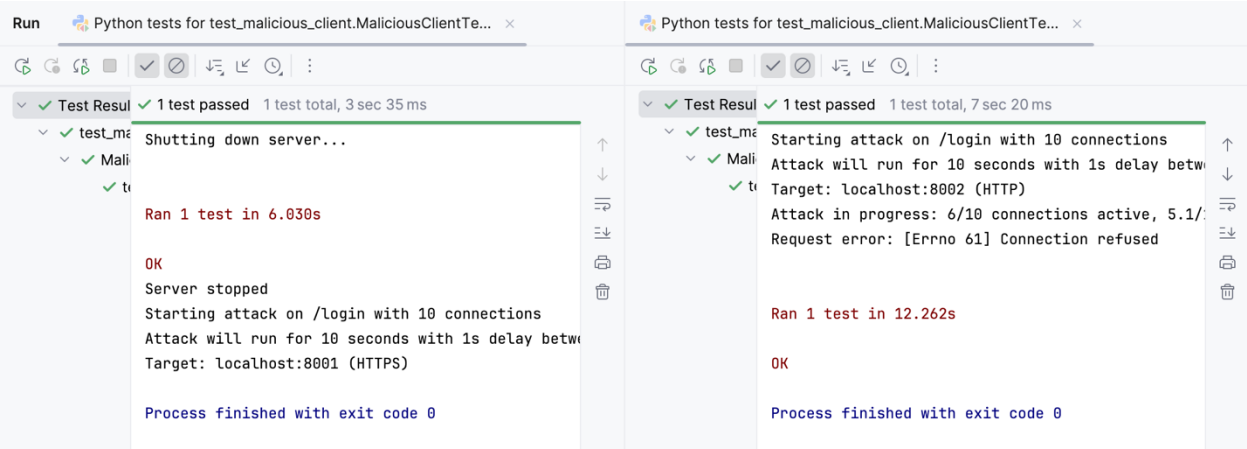

Figure 6. Execution of a Slowloris-style DoS attack with secure (left) and non-secure (right) server

Password Hashing

Stored passwords are hashed using the *bcrypt* algorithm, which uses random salt and requires significant processing power to reverse. This prevents malicious actors from using passwords or *Rainbow tables* — lists of previously calculated password hashes — in case the database is leaked (Skanda, Srivatsa and Premananda, 2022). The *bcrypt* algorithm is trusted by security-oriented software vendors such as Proton (Proton AG, 2024). An excerpt demonstrating the hashed passwords in the user data repository is below:

```
{
  "user_id": 1,
  "username": "admin",
  "password_hash":
"$2b$12$H48k9so0pBRoOctKWpiYqu32TWNLgShu4x34vA97zzgRjt1qn0uVG",
  "role": "ADMIN"
}
```

In non-secure mode, passwords are stored in plain text, leading to *CWE-312 "Cleartext Storage of Sensitive Information"* (The MITRE Corporation, no date c). In case a malicious actor gains access to the storage, all user passwords are exposed:

```
{
  "user_id": 1,
  "username": "admin",
  "password ": "admin",
  "role": "ADMIN"
}
```

Preventing Injection Attacks

Unlike the original design, this server implementation does not deserialize requests directly into **`Action`** objects. Instead, request content is passed as an argument during action execution. The standard Python library's **`json.loads()`** function is used to process incoming payload. This function is considered safe, as it will not execute arbitrary code. If the input is not valid JSON, it raises an error (Python Software Foundation, 2025b). However, a large JSON file could overwhelm the application and consume excessive resources.

In contrast, non-secure mode uses Python's **`eval()`** function to create objects from JSON strings, but it will also execute any arbitrary code (Python Software Foundation, 2025a). As part of an injection attack, the malicious client sends a Python script that

traverses server directories, extracts user credentials, and sends them to a remote server controlled by an attacker. Similarly, other configuration files or environment variables with access keys may be exposed.

This is described as *A03:2021 "Injection"*, *CWE-94 "Improper Control of Generation of Code"* (application constructs code of unvalidated input), *CWE-502 "Deserialization of Untrusted Data"* (input is deserialised without any pre-processing) (OWASP Top 10 Team, 2021b; The MITRE Corporation, no date a, no date f).

Figure 7 demonstrates that the script is not executed in the secure mode and no data is received. In non-secure mode, the script successfully extracts the sensitive information from the server.

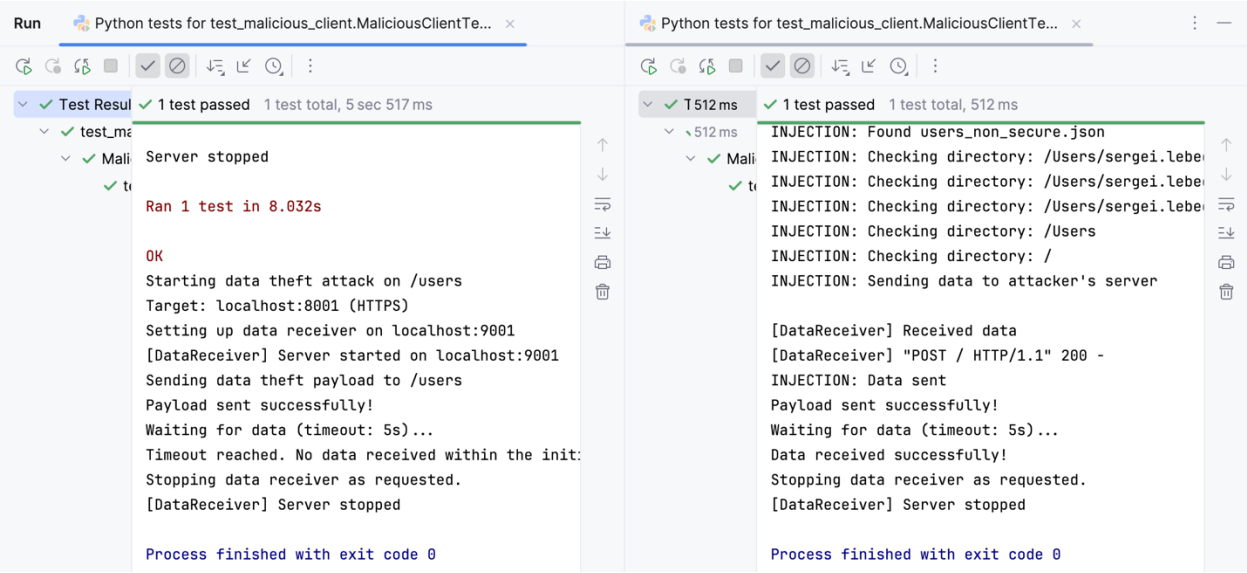

Figure 7. Execution of an injection attack with secure (left) and non-secure (right) server

Development Process and Reflection

The initial design offered a solid outline of the server application but lacked some detail. It was necessary to expand object states adding necessary fields and introducing methods to facilitate communication between class objects.

Some security features differ from the original design, but the final software reflects equivalent design considerations. Additionally, the requirement to run the server in non-secure mode encouraged research into attack vectors and their implementation, as well as a deeper understanding of proper security setup in networked software.

Implementing server tests streamlined development, ensured endpoint consistency, and prevented regressions. It also accelerated client development by ensuring API stability and providing request examples used in the client.

Using testing frameworks to demonstrate attack effects (or lack thereof) streamlined the development of malicious scripts and removed the need for manual verification.

References

CrowdStrike (2022) *What Is CRUD? Create, Read, Update, and Delete*, *CrowdStrike.com*. Available at: https://www.crowdstrike.com/en-us/cybersecurity-101/observability/crud/ (Accessed: 13 July 2025).

Damon, E. *et al.* (2012) 'Hands-on denial of service lab exercises using SlowLoris and RUDY', in *Proceedings of the 2012 Information Security Curriculum Development Conference*. New York, NY, USA: Association for Computing Machinery (InfoSecCD '12), pp. 21–29. Available at: https://doi.org/10.1145/2390317.2390321.

Ellis, B., Stylos, J. and Myers, B. (2007) 'The Factory Pattern in API Design: A Usability Evaluation', in *29th International Conference on Software Engineering (ICSE'07)*. *29th International Conference on Software Engineering (ICSE'07)*, pp. 302–312. Available at: https://doi.org/10.1109/ICSE.2007.85.

Fielding, R.T. (2000) *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation. University of California. Available at:

https://ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm (Accessed: 1 July 2025).

IBM (2025) *What Is a REST API (RESTful API)?* Available at: https://www.ibm.com/think/topics/rest-apis (Accessed: 1 July 2025).

Nord Security (2024) *Top 200 Most Common Passwords*, *NordPass*. Available at: https://nordpass.com/most-common-passwords-list/ (Accessed: 13 July 2025).

OWASP Top 10 Team (2021a) *A02 Cryptographic Failures - OWASP Top 10:2021*. Available at: https://owasp.org/Top10/A02_2021-Cryptographic_Failures/ (Accessed: 13 July 2025).

OWASP Top 10 Team (2021b) *A03 Injection - OWASP Top 10:2021*. Available at: https://owasp.org/Top10/A03_2021-Injection/ (Accessed: 13 July 2025).

OWASP Top 10 Team (2021c) *A07 Identification and Authentication Failures - OWASP Top 10:2021*. Available at: https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/ (Accessed: 13 July 2025).

Proton AG (2024) *What is password hashing and salting?*, *Proton*. Available at: https://proton.me/blog/password-hashing-salting (Accessed: 13 July 2025).

Python Software Foundation (2025a) *Built-in Functions*, *Python documentation*. Available at: https://docs.python.org/3/library/functions.html (Accessed: 13 July 2025).

Python Software Foundation (2025b) *json — JSON encoder and decoder*, *Python documentation*. Available at: https://docs.python.org/3/library/json.html (Accessed: 13 July 2025).

Python Software Foundation (2025c) *unittest — Unit testing framework*, *Python documentation*. Available at: https://docs.python.org/3/library/unittest.html (Accessed: 13 July 2025).

Python Software Foundation (2025d) *unittest.mock — mock object library*, *Python documentation*. Available at: https://docs.python.org/3/library/unittest.mock.html (Accessed: 13 July 2025).

'Regulation (EU) 2016/679 of the European Parliament and of the Council (General Data Protection Regulation)' (2016). Official Journal of the European Union. Available at: https://eur-lex.europa.eu/eli/reg/2016/679/oj/eng (Accessed: 19 January 2025).

Shorey, T. *et al.* (2018) 'Performance Comparison and Analysis of Slowloris, GoldenEye and Xerxes DDoS Attack Tools', in *2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. *2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pp. 318–322. Available at: https://doi.org/10.1109/ICACCI.2018.8554590.

Skanda, C., Srivatsa, B. and Premananda, B.S. (2022) 'Secure Hashing using BCrypt for Cryptographic Applications', in *2022 IEEE North Karnataka Subsection Flagship International Conference (NKCon)*. *2022 IEEE North Karnataka Subsection Flagship International Conference (NKCon)*, pp. 1–5. Available at: https://doi.org/10.1109/NKCon56289.2022.10126956.

The MITRE Corporation (no date a) *CWE - CWE-94: Improper Control of Generation of Code ('Code Injection') (4.17)*. Available at: https://cwe.mitre.org/data/definitions/94.html (Accessed: 13 July 2025).

The MITRE Corporation (no date b) *CWE - CWE-307: Improper Restriction of Excessive Authentication Attempts (4.17)*. Available at: https://cwe.mitre.org/data/definitions/307.html (Accessed: 13 July 2025).

The MITRE Corporation (no date c) *CWE - CWE-312: Cleartext Storage of Sensitive Information (4.17)*. Available at: https://cwe.mitre.org/data/definitions/312.html (Accessed: 13 July 2025).

The MITRE Corporation (no date d) *CWE - CWE-319: Cleartext Transmission of Sensitive Information (4.17)*. Available at: https://cwe.mitre.org/data/definitions/319.html (Accessed: 13 July 2025).

The MITRE Corporation (no date e) *CWE - CWE-400: Uncontrolled Resource Consumption (4.17)*. Available at: https://cwe.mitre.org/data/definitions/400.html (Accessed: 13 July 2025).

The MITRE Corporation (no date f) *CWE - CWE-502: Deserialization of Untrusted Data (4.17)*. Available at: https://cwe.mitre.org/data/definitions/502.html (Accessed: 13 July 2025).

The MITRE Corporation (no date g) *CWE - CWE-613: Insufficient Session Expiration (4.17)*. Available at: https://cwe.mitre.org/data/definitions/613.html (Accessed: 13 July 2025).

Wireshark Foundation (no date) *Wireshark*, *Wireshark*. Available at: https://www.wireshark.org/ (Accessed: 13 July 2025).