Team Design Proposal

Unit 6

June 09, 2025

Contents

1. Introduction

The domain that we have chosen to design our system for is a school grading system. We felt this was a suitable choice given this is an environment we are all familiar with. Furthermore, there are a minimal set of very well defined roles. It was thought that this would therefore make the process of deciding what access rights each user has more simple.

2. Security Vulnerabilities

In the unsecure mode of operation, the system shall be vulnerable to common mehods of attack such as a brute force or Denial of Service attack on the local network system and API injection attack.

2.1. Brute force attack

Defined by Curtin (2005) as "simply try[ing] every single possible combination until finding the one that works". This attack is often used against security mechanisms such as passwords and combination locks, where the set of possible solutions is finite and known. An attacker merely has to exhaust all permutations of the input until they happen upon the correct one.

2.1.1. Vulnerable mode

While in the insecure state, the system shall encode user passwords in plaintext. This is a common vulnerability as listed by the Open Worldwide Application Security Project (OWASP), which can lead to the comprimising of a system (OWASP, no date).

2.1.2. Secure mode

OWASP recommends that passwords should be hashed rather than encrypted, as hashing is a one-way process that cannot be reversed. This prevents attackers from recovering plaintext passwords, even if they gain access to the hashes.

In contrast, encryption is reversible and therefore not suitable. If the decryption key is compromised, all passwords can be exposed, undermining security.

To strengthen password storage, a unique salt should be added to each password before hashing. This ensures identical passwords yield different hashes, making attacks like precomputed hash lookups ineffective. (OWASP, no date).

2.2. Denial of Service

A Denial of Service (DoS) attack involves making a computer system, service or network unavailable to intended users by flooding it with illegitimate requests. As a result users may experience slow performance while the large volume of requests are processed (Mirkovic et al., 2004).

2.2.1. Vulnerable mode

The suggested method of transmitting user input is through a network socket, a software abstraction representing one endpoint of a two-way communication link between networked systems (Van Winkle, 2019).

The server must handle TCP SYN packets from clients, responding with SYN/ACKs. Normally, the client completes the handshake with a final ACK, establishing the connection. In a DoS attack, the client omits this final ACK, leaving the connection half-open. By initiating many such connections, the attacker consumes server resources, potentially rendering it unresponsive (Cloudflare, no date).

2.2.2. Secure mode

Secure mode expects floods from many IPs, so we enable SYN-cookies and cap each /24 subnet to 50 half-open connections every 30 seconds, anything above that is briefly dropped.

2.3. API injection

API injection occurs when valid API requests include malicious input executed by the backend, potentially compromising all system data (Ball, 2022). Systems handling personal data, especially under regulations such as the General Data Protection Regulation (GDPR), must guard against such attacks.

2.3.1. Vulnerable mode

In this mode, inputs are not sanitised. Without sanitisation, defined by Olmsted (2024, para 4) as "cleaning and validating user inputs", attackers can send harmful commands via API requests to gain unauthorised data access.

2.3.2. Secure mode

With input sanitisation enabled, all API requests are validated early, following OWASP's Input Validation Cheat Sheet. This blocks malicious input before it reaches core processing. However, the system must remain functional (OWASP, no date).For example filtering that is too strict, rejecting names containing `'`, would be inappropriate for a school context.

3. UML

3.1. Class Diagram

At the core of the system is the Core module, which defines the structure of a request. This includes the HTTP method, the target endpoint (e.g. message/send), the payload containing the necessary data and the session identifier. Incoming requests are handled by corresponding Actions, where the execute() method is invoked to initiate processing. This typically involves executing the logic defined in process_request() and recording operational logs.

Each API endpoint can be associated with specialised action subtypes, such as LoginAction, SendMessageAction or CreateMarkAction. These classes derive their data from the request payload, with fields automatically populated during instantiation by the Action Factory.

The Security module includes the SecurityManager, which serves as the main interface for security-related operations. It also incorporates the SessionManager and several utility classes. To ensure secure communication, the system could employ asymmetric encryption, whereby the public key is made accessible via the RSAKeyManager through a GetPublicKeyAction. During request processing, the private key would then be used to decrypt the incoming payload.

The domain-specific logic is currently straightforward, though there may be opportunities to simplify it further given the structural similarities across its various components.

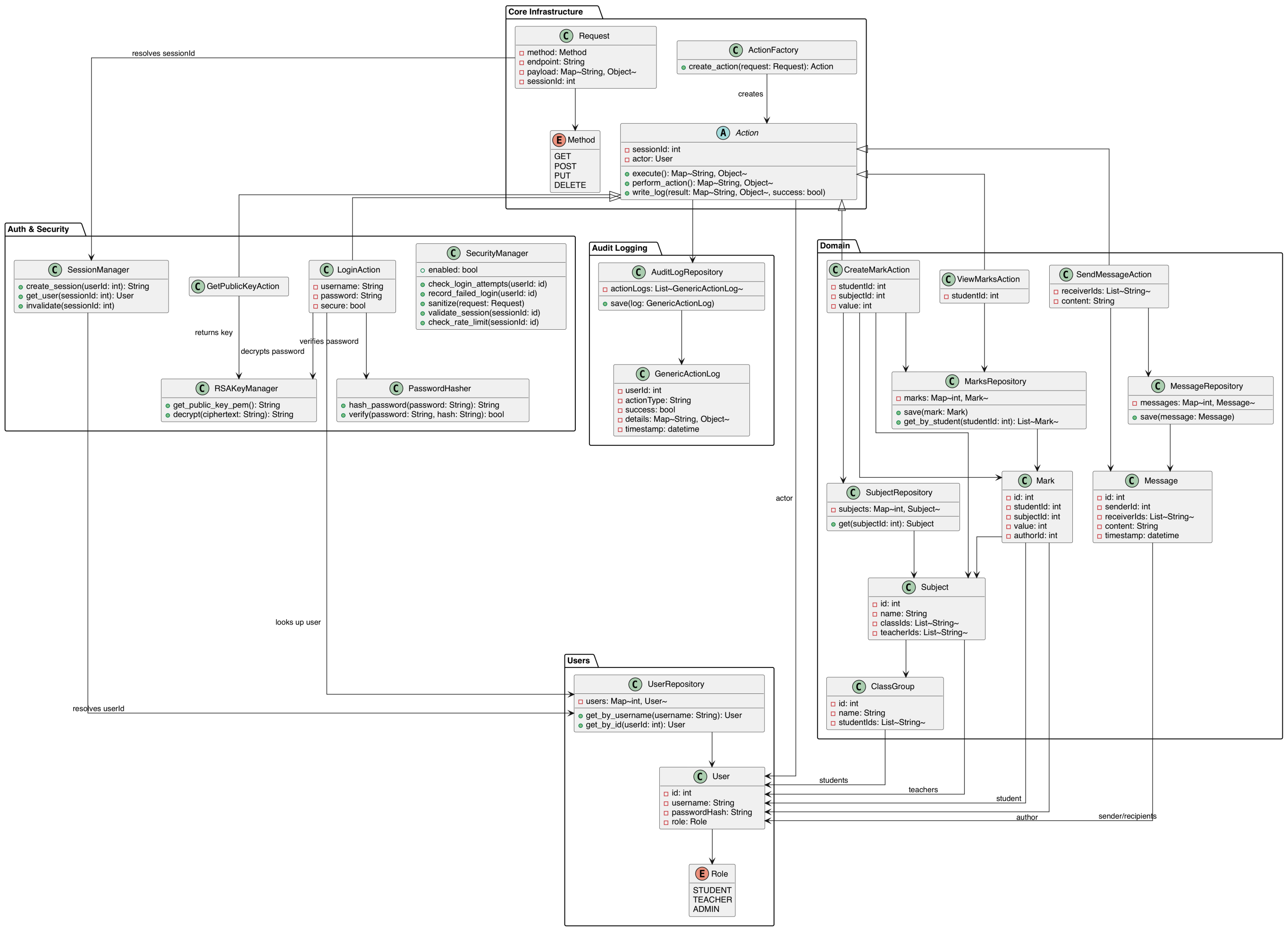

Figure 1: Class Diagram

3.2. Misuse Diagrams

The below diagrams describe the various actions available to both a legitimate user and a hacker. They supplement the above descriptions of how security vulnerabilities can be exploited by an attacker.

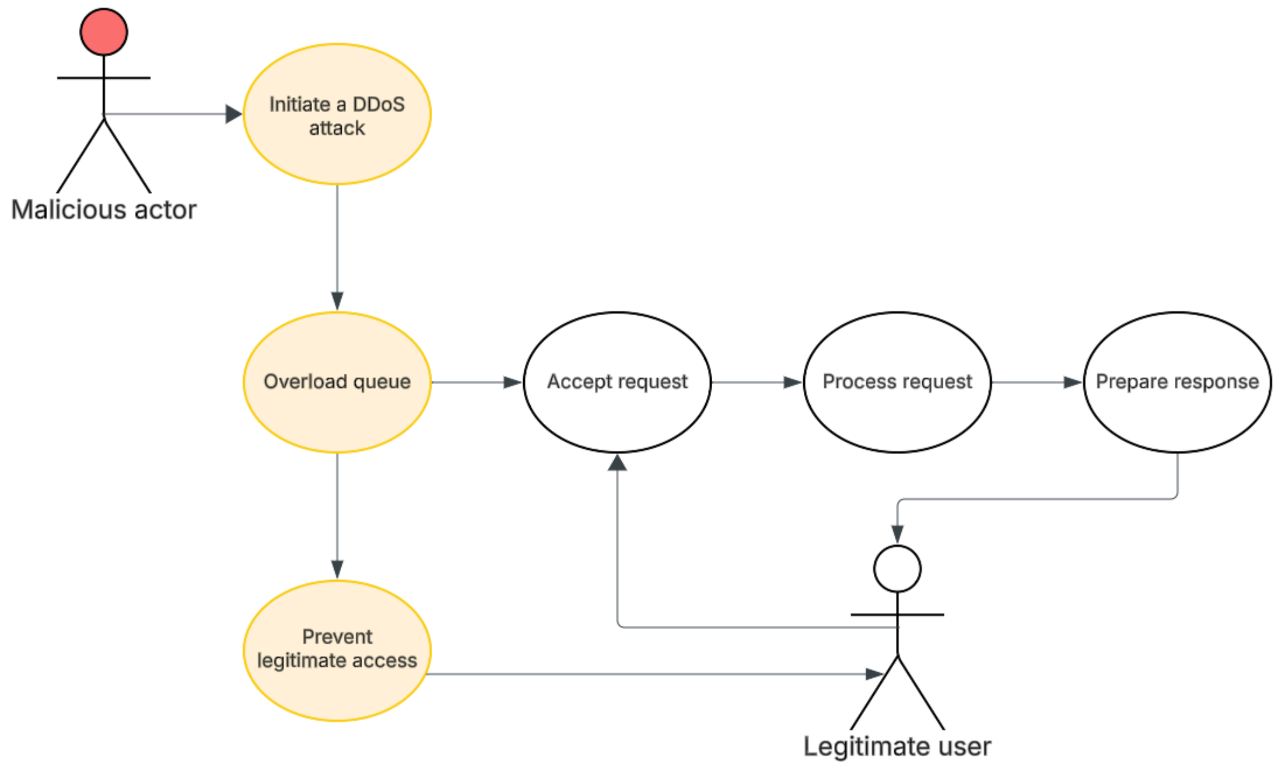

Figure 2: DDoS Misuse Diagram

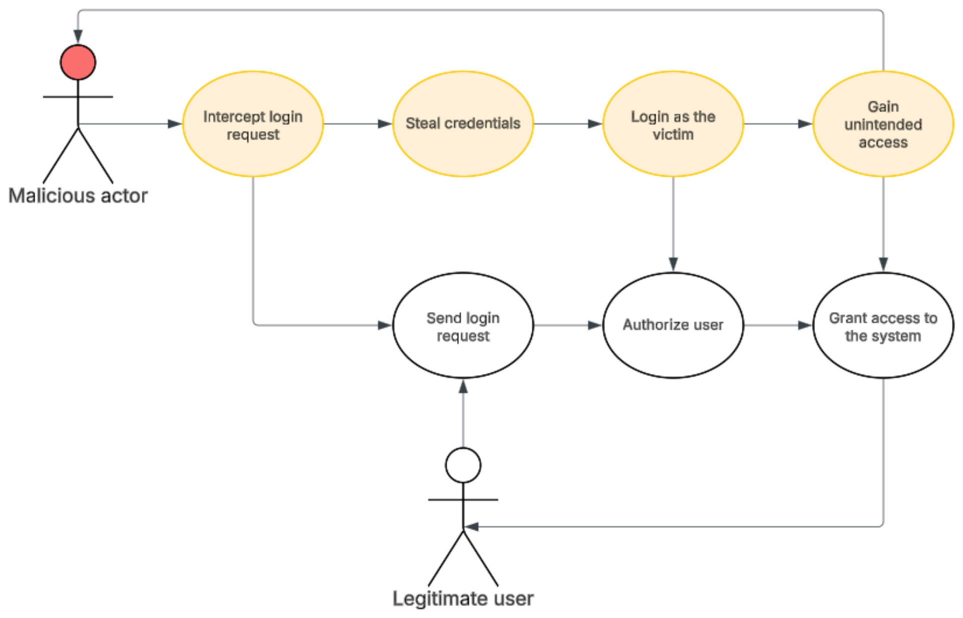

Figure 3: Man in the Middle Misuse Diagram

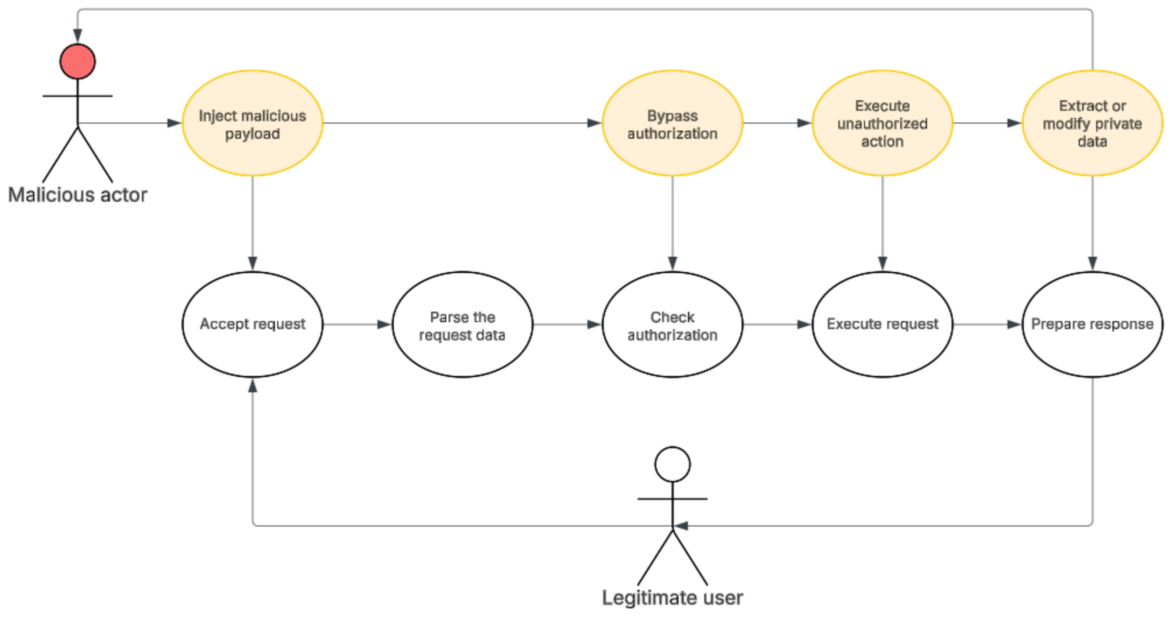

Figure 4: API Injection Misuse Diagram

4. API

The attached OpenAPI schema defines a RESTful API for a School Management System, covering users, subjects, coursework, and grading. It uses standard HTTP methods and organizes endpoints by user roles (admin, tutor, student) for clarity.

Security is handled via a JSON Web Token (JWT) based authentication, using a bearerAuth scheme. Clients must include a valid token in the Authorization header, enabling role-based access (e.g., only tutors can assign marks, only admins can create users).

Although designed to work with an external Identity Provider, the schema handles authentication internally in the interests of simplicity. This streamlines development but would typically be replaced by a dedicated IdP in production. The result is a secure, role-aware API for managing school operations.

5. Testing

The testing strategy shall comprise of differing levels of testing with unit, integration and user testing being performed. Unit testing shall be performed using Python's `unittest` framework, which supports test creation and execution. It also provides implementation for mocks, a way of allowing controlled simulation of external dependencies to test isolated logic components (O'Reilly, chapter 12, 2024).

Integration testing will consist of tests that utilise functionality of multiple sub-systems as defined in the class diagram. For example mocking user input to create an API request. This action would require action from more than one unit, the result of which can then be checked against an expected result in order to verify end-to-end functionality.

User testing will involve real users executing predefined tasks to confirm correct behavior and ensure a smooth, intuitive user experience.

6. References

Ball, C. (2022) *Hacking APIs*, O’Reilly: No Starch Press. Available at: https://learning.oreilly.com/library/view/hacking-apis/9781098130244/c12.xhtml° (Accessed: 24 May 2025).

Cloudflare. (no date) *SYN flood attack*. Available at: https://www.cloudflare.com/learning/ddos/syn-flood-ddos-attack/° (Accessed: 19 May 2025).

Mirkovic, J. et al. (2004) *Internet Denial of Service: Attack and Defense Mechanisms*, O’Reilly: Pearson. Available at: https://learning.oreilly.com/library/view/internet-denial-of/0131475738/ch02.html#ch02° (Accessed: 21 May 2025).

Olmsted, A. (2024) *Security-Driven Software Development*, O’Reilly: Packt Publishing. Available at: https://learning.oreilly.com/library/view/security-driven-software-development/9781835462836/° (Accessed: 25 May 2025).

OWASP. (no date) *Password Plaintext Storage*. Available at: https://owasp.org/www-community/vulnerabilities/Password_Plaintext_Storage° (Accessed: 23 May 2025).

OWASP. (no date) *Password Storage Cheat Sheet*. Available at: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html° (Accessed: 21 May 2025).

Van Winkel, L. (2019) *Hands-On Network Programming with C*, O’Reilly: Packt Publishing. Available at: https://learning.oreilly.com/library/view/hands-on-network-programming/9781789349863/72011b8c-93de-48c9-a0ee-787a2f513473.xhtml° (Accessed: 20 May 2025).

Westerveld, D. (2024) *API Testing and Development with Postman*. 2nd edn. O’Reilly: Packt Publishing. Available at: https://learning.oreilly.com/library/view/api-testing-and/9781804617908/Text/Chapter_12.xhtml#_idParaDest-250° (Accessed May 28 2025).

Index of Figures