Online School System Design

Introduction

Modern education increasingly relies on School Management Systems (SMS) to effectively manage essential student data, including enrolment, grades, and attendance. The adoption of these systems has surged, particularly during the COVID-19 pandemic, making them a cornerstone of digital education (Falana et al., 2021). However, SMS platforms handle sensitive student information and academic records, making them vulnerable to cyberattacks. As a result, institutions not only require highly functional SMS but also demand robust security measures to safeguard the data stored within them (RSI Security, 2021).

Background of the Application

SMS are essential for modern schools since they ease duties such as course management and assessment. However, they suffer cyber dangers like malware and unauthorised access, rendering standard security measures ineffective. Improving SMS security is critical for safeguarding sensitive data.

Rationale

The growing number of cyberattacks on educational institutions emphasises the need for more secure SMS (Soykan and Şimşek, 2014). As hackers target these systems more frequently, SMS platform security has become a major worry (Amin et

al., 2020). The development of a safe and secure SMS is thus critical to developing robust digital learning environments capable of overcoming emerging cybersecurity challenges.

GDPR and legal requirements for the application

- Companies must protect the data of their users.
- The way data is accessed and interacted with must be changed depending on the type of user.
- All data must be stored securely with only selected parties being allowed access.
- All information must be deleted after a 3 year inactive period.
- All data must be shared with the individual when requested.
- Software must be regularly updated and screened for potential bugs.
- Software must be checked for unauthorised use by users or other parties. (GDPR, 2016; Welsh Government, 2021)

The application has been designed so that GDPR and legality are considered in each step to ensure the security of its users' data. This has been done through carefully considering the risks to the application and planning ahead for these eventualities.

Solution Proposal

The proposed school management application will securely manage student, tutor, and administrative data while utilising Python's object-oriented programming for robust development (Falana et al., 2021; Phillips, 2018). It will have a command-line

interface and security measures to combat threats. The Observer design pattern (Pillai, 2017) will notify users when data changes, resulting in more efficient updates and notifications.

Detailed project assumptions breakdown can be found in Appendix E.

UML Specifications

The Unified Modelling Language (UML) provides tools for analysis, design and implementation of software-based systems. The UML Specifications (found in Appendix A) contains detailed activity diagrams for the various roles and scenarios within the system. The diagram ensures accurate analysis for both legitimate user activities and potential misused cases, highlighting the system's features and security measures.

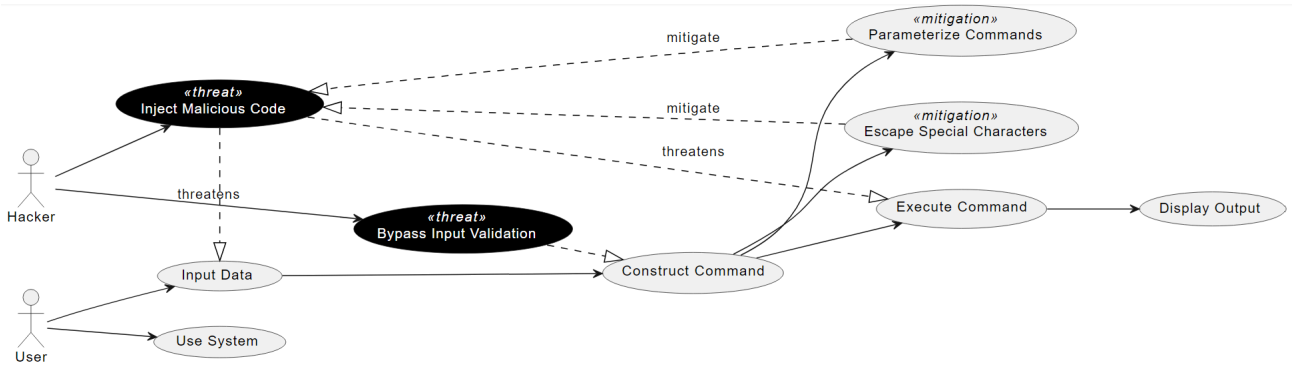


Figure 1: Attacker Misuse Case Diagram

Security Requirements

Requirement	Implementation Details
Authentication	Authentication is login-password based. Passwords are stored as salted hashes using the bcrypt algorithm.
Authorization	Role-Based Access Control (RBAC) is implemented with roles such as student, tutor, and admin, each with specific access rights.
Data Encryption	Sensitive data, including Social Security Numbers (SSNs), is encrypted using the AES-256 algorithm for protection both at rest and in transit (Bowne, 2018).
Event Monitoring	Significant actions, particularly related to authentication, authorization, and data access, are logged and monitored. Additionally, custom metrics are added to detect anomalies. A monitoring system can be integrated at any point (e.g., Kibana or DataDog) (Blokdyk, 2021).
Security Toggle	A global environment variable SECURITY_ENABLED controls the activation of security features. When enabled, middleware intercepts and validates requests based on security policies.

Table 1: App Security Requirements

Tools and Technologies

All tools and technologies listed are either de facto industry standards or widely adopted solutions (e.g., VSCode). Others are directly derived from the requirement to use Python as the development language (e.g., Pytest, Pylint).

Tool	Purpose
Git	Version control for tracking changes and managing collaboration.
VSCode	Integrated Development Environment (IDE) with strong Python support.
Python	*initial requirement*
Pylint	Enforces code quality and identifies potential issues early.
Pytest	Framework for unit, integration, and acceptance testing.
cryptography	Handles encryption of sensitive data, such as SSNs, using AES-256.
bcrypt	Generates secure salted hashes for passwords.
uuid	Generates unique identifiers for entities like Student, Tutor, and Class.
redis	In-memory storage solution used for rate-limiting.
pydantic	Validates clients input.

Table 2: Tools and Technologies used

Hacker Requirements and Countermeasures

Brute Force Attack

A hacker may attempt to gain unauthorised access by systematically trying different password combinations. To counter this, the system will implement rate-limiting based on the IP address of the client making the request.

Mitigation: Use Redis to track login attempts (INCR, EXPIRE) with a Python Redis client library such as redis-py (John & Mathew, 2021).

Denial of Service (DoS) Attack

A hacker may try to overwhelm the system with a flood of requests, leading to service unavailability. The countermeasure for this threat is to implement general rate-limiting for all requests. This will throttle the number of requests an IP address can make within a specific time frame.

Mitigation: Use Redis to manage request counts (INCR, EXPIRE) and enforce rate-limiting, ensuring that no IP can exceed the allowed number of requests (Nguyen et al., 2023).

API Injection Attack

A hacker could attempt an API injection attack by sending malicious input designed to execute unintended commands or access unauthorised data. To counter this, input validation and sanitation will be applied to all incoming data (Mitropoulos & Spinellis, 2017).

Mitigation: Use the pydantic library for input validation, or employ regular expressions to ensure inputs adhere to the required format; use SQL ORM to perform requests to the DB (Pulkkinen, 2024).

OWASP Top 10 Proactive Controls

The OWASP Top 10 Proactive Controls are a list of security techniques that software architects and designs should utilise when developing projects. These should be implemented at the earliest stages of a project and should be referenced throughout all stages of design and development. (OWASP, 2024). Top 10 controls breakdown can be found in Appendix B.

STRIDE and DREAD Models

STRIDE is an acronym that references different types of attacks that hackers may use to try and compromise a system and its data (Conklin, 2024). DREAD builds on this and provides a quantitative way of determining the risks associated with these attacks, culminating in a score that determines whether the risk of attack is low, medium or high (Kirtley, 2023). STRIDE and DREAD models breakdowns can be found in Appendix C.

Data Repository Design

Data Structure	Justification
dict	Used for storing Student, Tutor, and Class records with fast lookup by id. The dict provides O(1) average time complexity for lookups, inserts, and deletions.
set	Used for managing unique associations, such as classes in Student and Tutor entities. The set ensures that each student or tutor is associated with unique classes and offers O(1) average time complexity.
list	Used for managing collections of students within Class entities. The list provides efficient storage and access to groups of students.

Table 3: Main Data Structures

Code samples with the data structures used can be found in Appendix F.

References

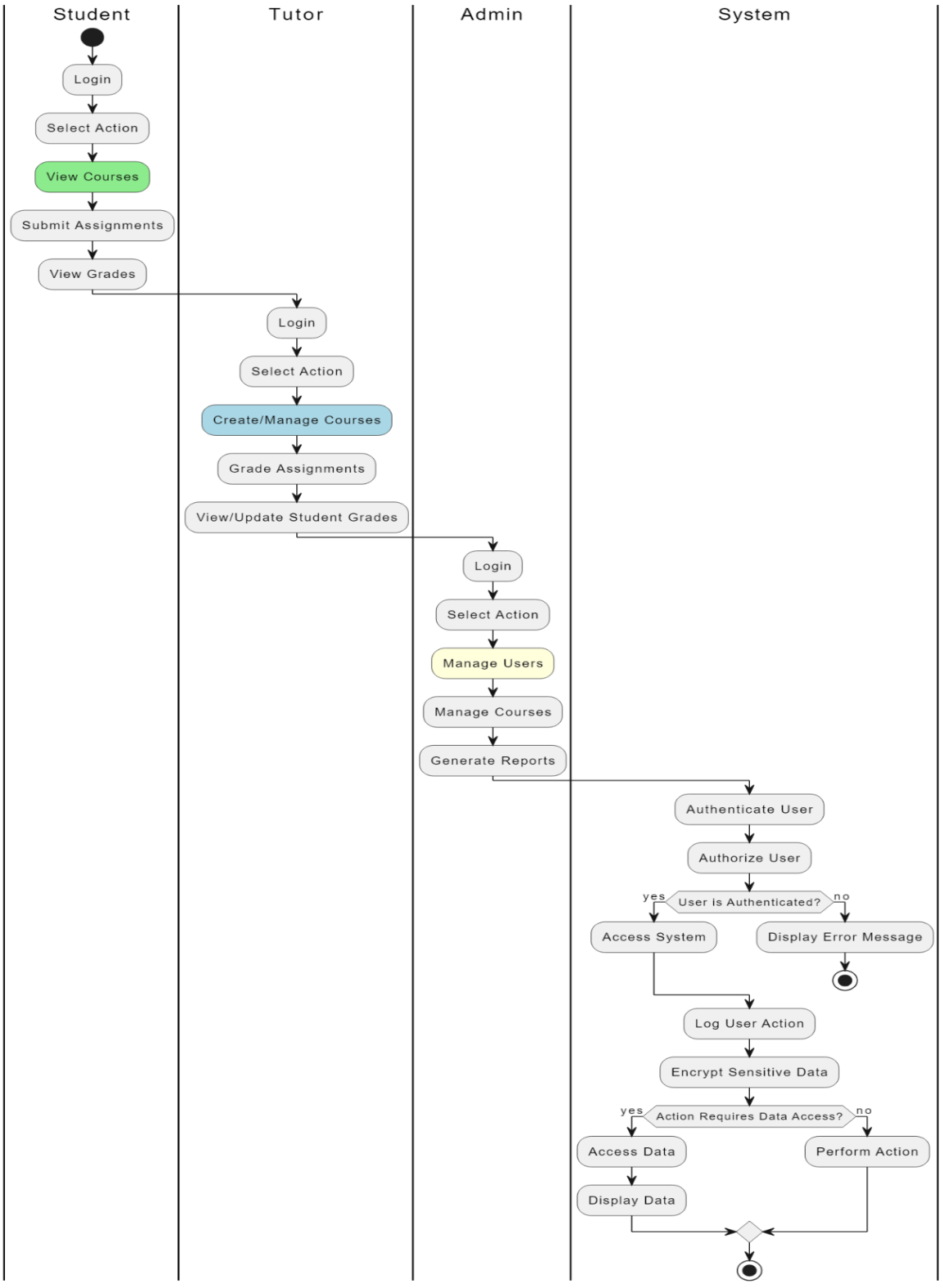
- Amin, R., Kunal, S., Saha, A., Das, D. & Alamri, A., 2020. CFSec: Password based secure communication protocol in cloud-fog environment. Journal of Parallel and Distributed Computing, 140, pp.52-62. Available from: https://www.researchgate.net/publication/350838302_Se-LMS_Secured_learning_management_systems_for_smart_school [Accessed 10 August 2024].
- Blokdyk, G. (2021) *Event Monitoring Second Edition*. AA World Services.
- Bowne, S. (2018) *Hands-On Cryptography with Python: Leverage the power of Python to encrypt and decrypt data*. Packt Publishing Ltd.
- Conklin, L. (2024) *Threat Modelling Process*. Available from: https://owasp.org/www-community/Threat_Modeling_Process#stride [Accessed 4 September 2024].
- Department of Education and Skills. (2021) *Guidance for schools to implement the information management strategy*. Cardiff: Welsh Government Llywodraeth Cymru.
- European Parliament and Council of the European Union. *Regulation (EU) 2016/679 of the European Parliament and of the Council on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)*. Available from: https://eur-lex.europa.eu/eli/reg/2016/679/oj [Accessed 21 August 2024].
- Falana, O.J., Ebo, I.O. & Odom, I.S. (2021) Se-LMS: Secured e-learning management systems for smart school. International Journal of Software Engineering and Computer Systems, 7(1): 36-46.
- John, L.A. & Mathew, J. (2021) 'Efficient Brute Force Attack Handling: Server Virtualization', in *Proceedings of the National Conference on Emerging Computer Applications (NCECA)*, p. 218.
- Kirtley, N. (2023) DREAD Threat Modelling. Available from: https://threat-modeling.com/dread-threat-modeling/ [Accessed 4 September 2024].
- Mitropoulos, D. & Spinellis, D. (2017) 'Fatal injection: a survey of modern code injection attack countermeasures', *PeerJ Computer Science*, 3, p. e136.
- Nguyen, H.-P. et al. (2023) 'An Approach to Prevent DDoS Attack Using Real-Time Access Logs Analysis', in *Recent Challenges in Intelligent Information and Database Systems*. Springer Nature Switzerland, pp. 92–105.

- OWASP (2024) OWASP Top 10 Proactive Controls. Available from: https://top10proactive.owasp.org/the-top-10/introduction/ [Accessed 4 September 2024].
- Phillips, D. (2018) *Python 3 Object-Oriented Programming: Build Robust and Maintainable Software with Object-Oriented Design Patterns in Python 3.8* Packt Publishing Ltd.
- Pillai, A.B. (2017) *Software architecture with Python*. Packt Publishing Ltd.
- Pulkkinen, H. (2024) *Design of a Software Configuration Tool for a Test System in Mass Production.* Available from: https://trepo.tuni.fi/bitstream/handle/10024/154999/PulkkinenHenri.pdf?sequence=2 [Accessed 25 August 2024].
- RSI Security, 2019. Common cybersecurity threats in education. [online] Available from: https://blog.rsisecurity.com/common-cyber-security-threats-in-education/ [Accessed 10 August 2024].
- Soykan, F. & Şimşek, B., 2017. Examining studies on learning management systems in SSCI database: A content analysis study. Procedia Computer Science, 120, pp.871-876. Available from: https://www.researchgate.net/publication/350838302_Se-LMS_Secured_learning_management_systems_for_smart_school [Accessed 10 August 2024].

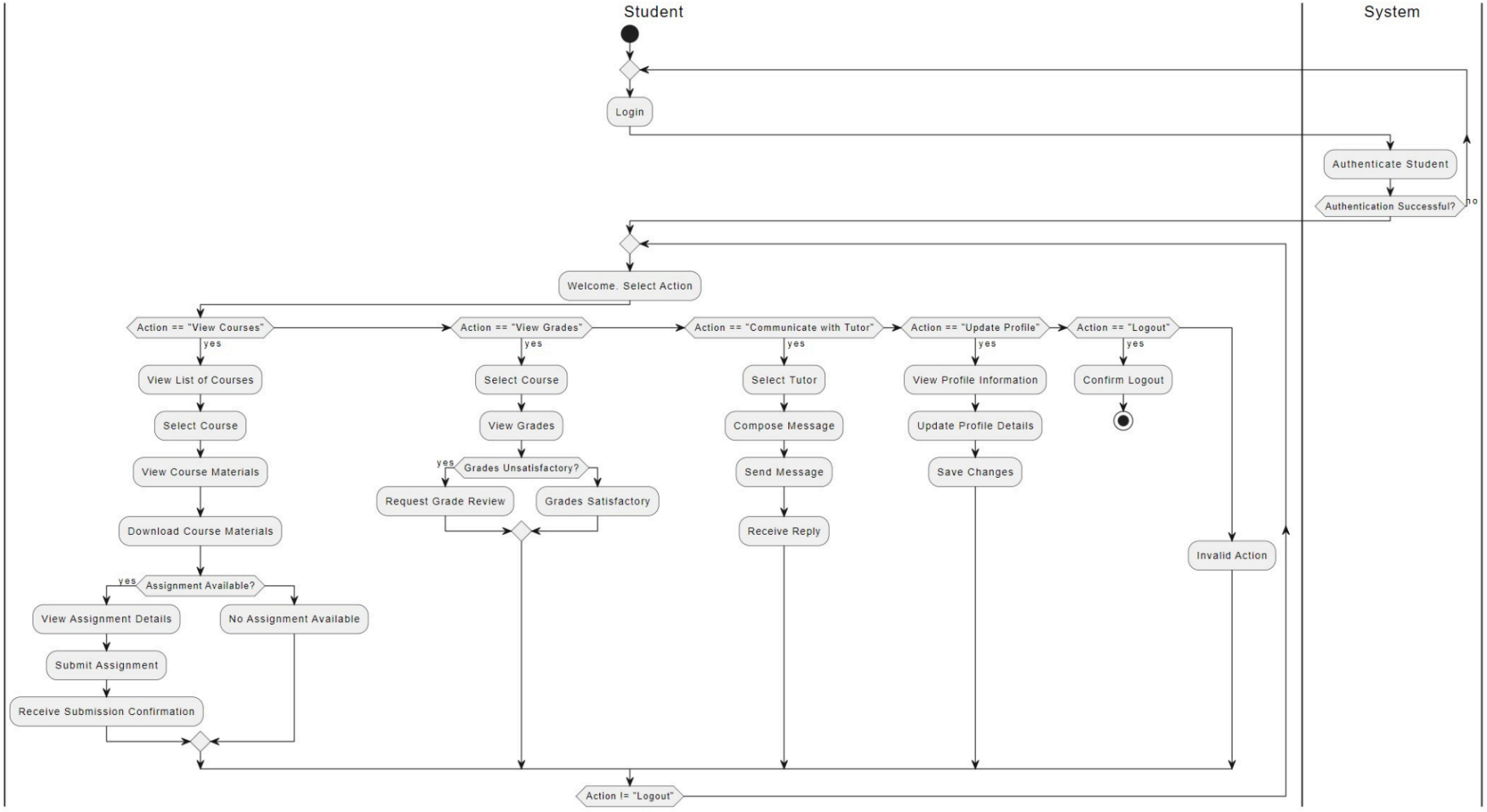
Appendices

Appendix A: UML Specifications

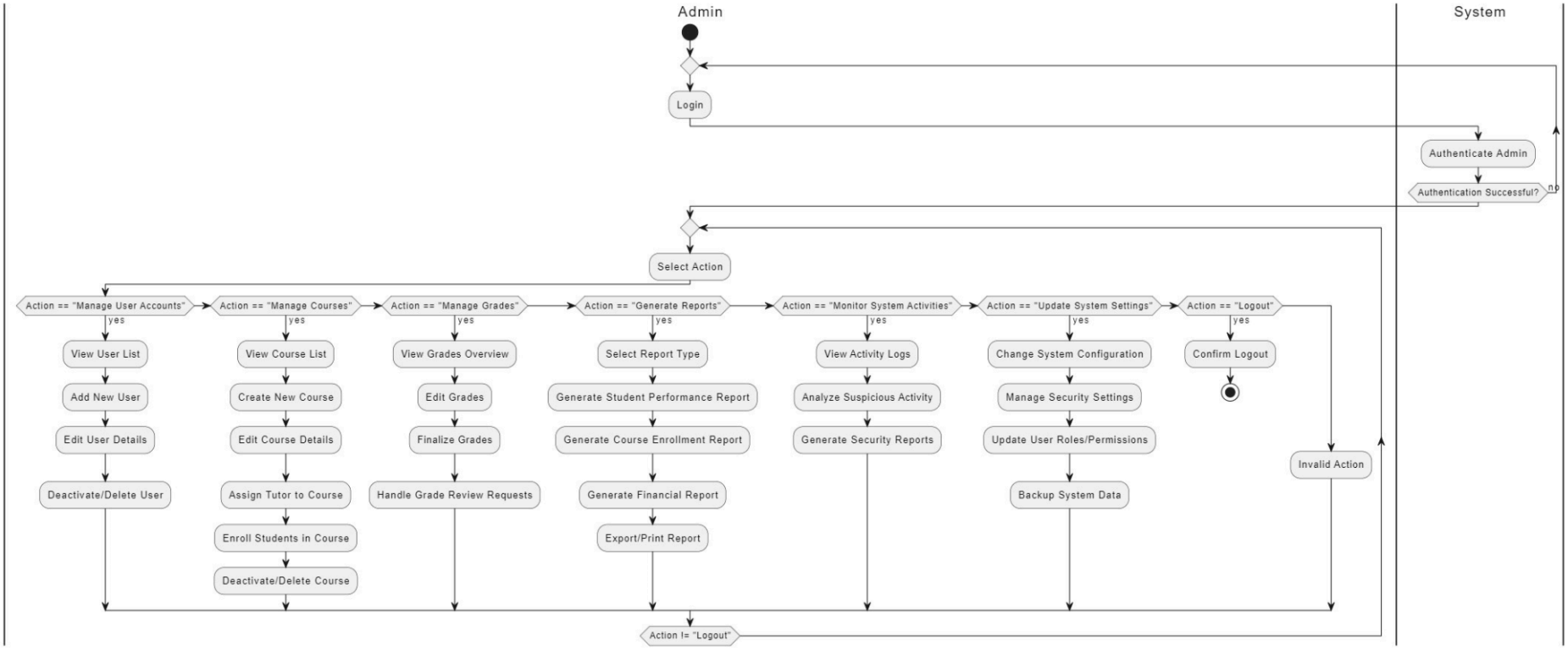
A.1. Activity Diagram for the Proposed System



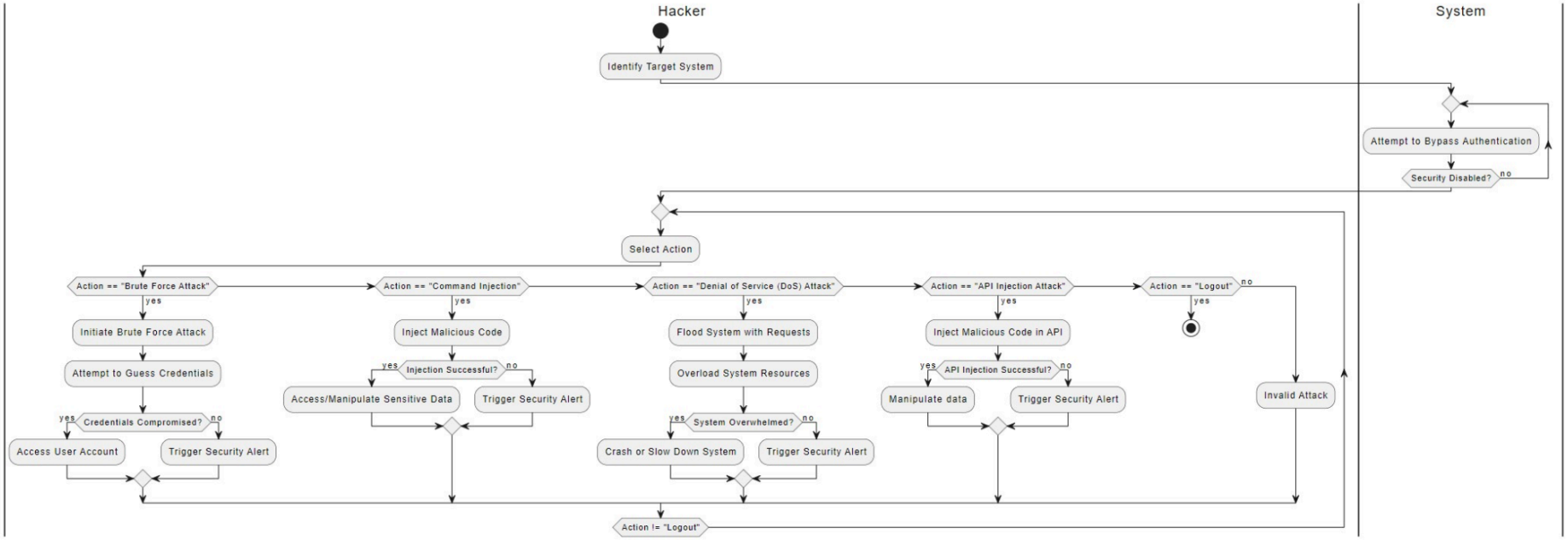
A.2 Student Tasks/Activities



A.3 Admin Tasks/Activities



A.4 Attacker Tasks/Activities



Appendix B: OWASP Controls Breakdown

OWASP Proactive Control	Explanation
C1: Implement Access Control	This ensures that access to systems and data is only given to those with a legitimate need for this, with access being revoked once this is no longer required.
C2: Use Cryptography the proper way	This ensures that passwords, secrets and sensitive data are property encrypted, utilising methods and tools that meet the requirements of standards and regulations, such as GDPR. OWASP recommends using standard libraries to do this rather than trying to implement an individual scheme.
C3: Validate all Input & Handle Exceptions	By using syntactic (in the expected format) validity and semantic (within an accepted range) validity, systems are able to withstand attacks such as SQL and RCE Injection.
C4: Address Security from the Start	Security should be an essential part of system development, ensuring that this is simple to understand for both developers and users.
C5: Secure By Default Configurations	Security should be an essential feature of any security and should not be locked behind a paywall. This ensures that systems are secure out of the box and no further manipulation is needed.
C6: Keep your Components Secure	When using libraries and frameworks, it is essential to ensure that security is a key focus of these. This also avoids the needs to use additional security libraries that can make applications complex and more vulnerable.
C7: Implement Digital Identity	Utilising tools such as passwords and multi-factor authentication ensure that sensitive and confidential data is not easy to obtain for those who shouldn't have access

OWASP Proactive Control	Explanation
	to it.
C8: Leverage Browser Security Features	Browsers are the gateway to the internet for a lot of users, and as such they can be vulnerable to attacks. By using secure protocols and tools which using browser based applications, data becomes harder to obtain.
C9: Implement Security Logging and Monitoring	Logging and monitoring allows for security and program concerns to be identified in real time, ensuring that timely fixes can be implemented. This reduces the window for an attack and allows attacks to be identified more easily.
C10: Stop Server Side Request Forgery	Server Side Request Forgery tried to coerce a server into performing requests on behalf of an attacker, thus showing that the server and not an attacker is performing the action.

Appendix C: STRIDE and DREAD Models Breakdown

C.1 STRIDE

STRIDE Risk	Potential Impacts
Spoofing	potential for students in particular to attempt to try and get teacher passwords to access information. Additional authentication can alleviate this.
Tampering	Assuming only admin accounts are able to use CRUD functions, the risk of this should be low. However, it's possible that these admin actors could use the application in bad faith. Tampering during transfer on open networks shouldn't be too much of a risk due to it being localised.
Repudiation	Can be avoided during building by ensure user actions are able to be logged and monitored in case of any prohibited activity
Information Disclosure	Risk of teachers (who can read records) and Admin (with full CRUD access) being able to access records that they shouldn't be able to. Varying permissions for different accounts and records would prevent this.
Denial of Service	Low level of risk associated with this due to it being a small, localised application, however good security should avoid this.
Elevation of Privileges	Elevation of privileges – potential for teachers and other staff to ask for this and admin with poor knowledge of data security may allow this. Could be non malicious but could take place. Low level of risk of further action to gain unauthorised information beyond this.

C.2 DREAD

DREAD Category	Score	Justification
Damage	8 - Non-sensitive user data related to individuals or employer compromised	The system is not designed to hold sensitive data beyond names. Although data disclosure could happen, the classification of this would not be sensitive.
Reproducability	5 - Complex	While the reproducibility of an attack is possible, the system is designed to avoid this.
Exploitability	5 Available attack tools	Due to the build of the system, readily available tools for attacks would potential be effective against this system.
Affected Users	10 - All users	All users are stored in the same place locally, so all would be impacted by an attack.
Discoverability	0 - Hard to discover vulnerability	The system is designed to be as secure as possible with security as the focus.
Total Score	28 - High Risk	The high risk relates to the fact that all information was be compromised if an attack is successful, however this would be difficult.

Appendix D: System Requirements and Specifications

Requirements	Details
Platform	Python-based application accessible via Command-Line Interface (CLI).
Data Structures	Utilises dictionaries, lists, and sets for efficient management of student data.
Account Management	Role-based access control for different user levels.
Data Management	CRUD (Create, Read, Update, Delete) operations for student records.
Security features	Encryption, and Event Monitoring.
Network	100 Mbps Ethernet, web access enabled.
CPU	Dual-core, uses threading for efficient concurrency handling.
RAM	1GB, adequate for modern devices.
Storage	Up to 5GB for SMS data (including student records, course materials, and logs), with scalable options for future growth
Peripherals	Keyboard only for CLI access.

Appendix E: Project Assumptions

Assumption Category	Assumption Details
Resources	1) The software design will be such that it allows for a solo developer to be able to complete the development and testing of the software independently. 2) The developer of the software will be able to utilise Python, as well as secure software development methods, to a standard that will allow the project to be completed accurately. 3) The development process will be able to utilise open-source libraries when required, with no additional financial resources being required to complete a fully realised programme.
Application Design	1) The application will meet the needs of the client (a school) and will be designed in a way that is appropriate to this setting. 2) The application will be designed in a way that complies with all current legislation regarding data security and data rights, as well as internationally recognised standards.
Development	1) Development of the application will be completed by a date which is mutually agreed between the developer and the customer. 2) Development will be completed in accordance with software development best practices such as

	minimising code complexity, reusability and the utilisation of external libraries.
Testing	1) The application will be fully tested, ensuring that the final product meets the requirements set within the design documents, as well as ensuring that all security features are in place. 2) Testing outputs will be recorded and provided as part of the final deliverable.

Appendix F: Code Sample with Data Structure used

```
# Dictionary: Storing Student records
students = {}

# Adding a student
students[student_id] = {
    "id": student_id,
    "password": "hashed_password",
    "email": "john.doe@example.com",
    "name": "John",
    "surname": "Doe",
    "classes": set()  # Using a set for unique class associations
}

# Dictionary: Storing Class records
classes = {}

# Adding a class
classes[class_id] = {
    "id": class_id,
    "name": "Introduction to Computer Science",
    "tutor": tutor_id,
    "students": []  # Using a list to store students in the class
}

# Adding a student to the class
classes[class_id]["students"].append(student_id)
```