Secure Software Development
Suggested format for Design Document

1. **Introduction:** provide a short summary of the application and the task
2. **Background of the application:** provide a short summary of the application
3. **Rationale:** provide a brief justification for your chosen application
4. **System requirements and Specifications** of the application
5. **Functional Requirements** of the application
    a. Jordell
6. **Legal Requirements** of the application
7. **GDPR**
8. **The Payment Services Regulations**
    a. **Georgia**
9. **Security Requirements**
10. **Project Assumptions**
    a. **what we expect from the app, from the functionality**
    b. **Tom**
11. **Solution Proposal**
12. **UML Specifications**
    a. **John**
13. **Tools and Technologies** (Version control, Linters, Static Analysis Tools, IDE, Programming Language)
    a. git, pylint, pylint, vscode, python
    b. justification as we use de-facto industry standards - brief explanation
14. **Libraries for encryption and testing**
    a. **Alex**
15. **Methodology** like Secure Scrum model, TDD

UML models for User: Provide a list of tasks or activities that user's would like to do through the application

- with class diagram for DB entities and their relations

UML models for Attackers: Provide a list of tasks or activities that attacker's would like to do through the application

- stick to injection flow attack

UML models for Admin: Provide a list of tasks or activities that admin would like to do through the application

UML Designs: Provide class diagram and Misuse case diagram for your application

Application Security Verification Standards like Application Security Verification Standards (**ASVS**) levels in Open Web Application Security Project (**OWASP**)

OWASP Proactive Controls: 10 Proactive controls

Threat Modelling: including STRIDE and DREAD

Conclusion

References

- at least two (or more) refs from everybody

Appendix

N.B.: Try to address all the points above briefly within the recommended word limits. In case if you go over the word limits, then put additional words under appendix.

School

Agenda

17.08 - discussion of the initial proposal

bring outline, questions

31.08 - final design discussion

Minutes:

meetings every Sat; communication in channel

Application requirements:

The agreed criteria for successful development are:

-
- The data repository should be created in Python. You are required to use at least three different structures to store the data within the data repository.
    - dict (users dict with id - user structure), list, tree (for sorting by lastname)
- The application should be accessed via a terminal, using a Command Line Interface as the User Interface.
    - describe that users are supposed to work with app via CLI
    - describe possible commands and how to call them (with flags, attributes…)
- All functionality of the system should adhere to GDPR (anon, n.d.a).
    - on Georgia
- The system should be designed so that a design pattern may be applied.
    - check if we can propose some specific design pattern here (based on our classes model)
- Mechanisms should be deployed to minimise the attack surface of the solution. At a minimum, the application should include authentication, authorisation, data encryption and event monitoring. Capabilities should also be enabled to protect against the hacker attacks (details below).
    - authentication - password based auth system (email - password) - user can signup using their email and provide with pass - then pass is being salted and encrypted - put in the
    - students, teachers, admin roles (for authorization)
    - encryption - password by default (let's stick with pass only for now)
    - event monitoring - propose event logging (for every API call) with metrics to monitoring service (like DataDog) and we can attach monitors and alarms in case of incidents
- The system should support ability to turn secure capability on and off.
    - have a single env var flag to turn on-off the securityEnabled feature
    - there is a middleware in place to intercept requests and validate them if flag is on
    - on Alex
- Object-oriented programming practices should be used.
    - we use classes for our entities (student, tutor…) with related methods
- Write unit tests to examine the effectiveness and completeness of your programme. These should be integrated as part of a testing suite, additionally including system, integration, and user acceptance.
    - propose tests types and functionalities to cover our solutions (theoretically)

    - provide with test cases (input - assertion - expected output) to cover app

Legitimate User Requirements:

- A user should have ability to perform CRUD functions (anon, n.d.b).
    - create-delete user (for teachers); search user (for all); update user (some attributes for students, other for teachers)
    - describe in design who can do what based on authorization mechanism
    - Jordell - describe CRUD API
        - -> run school --teacher (--student)
        - -> create --name Alex --email qq@mail.com --pass 123&
- A user should be able to create a secure account.
    - email-password auth by default

Hacker Requirements:

- A hacker should have the ability to carry out a brute force attack on the local network system.
- A hacker should be able to carry out a Denial of Service attack on the local network system.
- A hacker should be able to carry out an API injection attack (Sani et al., 2022).
    - SQL injection validation - request sanitanization; without secure ON - just get users input as it is (with possible SQL statement in there)
    - read source Sani et al., 2022

    - clarify with Anup do we need to implement it in the second part
    - clarify with Anup how to make injection attack without DB in place (or just simulation)

DB STRUCTURE

Student

- id - key
- password - string (salted hash)
- email - string
- name
- surname
- classes - Array[Class]
- …

Tutor

- classes - Array[Class]
- …

Class

- id - key
- name
- tutor - Tutor
- students - Array[Student]
- …

ROLES

Tutor

- can create/delete/update students

Student

- can read limited data from other students/teachers

Admin

- can create/delete/update teachers and students

ALEX SECTION start

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.
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).
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.

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.
	todo: add libs from hacker requirements section

Hacker Requirements and Countermeasures

Brute Force Attack

A hacker may attempt to gain unauthorized 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. The rate-limiting will be enforced by tracking the number of failed login attempts per IP address within a specific time window. If the number of attempts exceeds a defined threshold, further login attempts from that IP will be temporarily blocked.

Implementation: Use Redis to track login attempts (INCR, EXPIRE) with a Python Redis client library such as redis-py.

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, ensuring that the system remains responsive and available to legitimate users.

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

API Injection Attack

A hacker could attempt an API injection attack by sending malicious input designed to execute unintended commands or access unauthorized data. To counter this, input validation and sanitation will be applied to all incoming data. The validation

ensures that inputs match expected structures, e.g. that a name field contains a string of 2-20 characters without special symbols or SQL reserved words.

Implementation: 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.

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, making it ideal for managing entities where quick access is needed.
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, preventing duplicates and offering O(1) average time complexity for membership checks and insertions.
list	Used for managing collections of students within Class entities. The list provides efficient storage and access to groups of students, offering a straightforward way to manage multiple elements.

Data Repository Design

Code samples with the data structures 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)
```

todo:

- add refs
- make hacker section more concise?

ALEX SECTION end

Tom

Project Assumptions

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.

OWASP Level 2 ASVS - https://owasp-aasvs.readthedocs.io/en/latest/level2.html

OWAPS Top 10 SeProactive Controls

OWASP Top 10 Proactive Controls describes the most important control and control categories that every architect and developer should absolutely, 100% include in every project. The Top 10 Proactive Controls are by developers for developers to assist those new to secure development.

https://owasp.org/projects/spotlight/historical/2021.02.10/

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.

OWASP Proactive Control	Explanation
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 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.

https://owasp.org/www-community/Threat_Modeling_Process#stride-threat-list

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.

STRIDE Risk	Potential Impacts

https://www.eccouncil.org/cybersecurity-exchange/threat-intelligence/dread-threat-modeling-intro/

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.

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.

Jordel Section

1. Introduction

Modern education depends on the Learning Management System (LMS), which helps institutions to effectively manage student data such as enrollment, grades, and attendance. Adoption of it has skyrocketed, especially during the COVID-19 epidemic, so it is a pillar of digital education (Falana et al., 2021). But since LMS systems handle private student information and academic records, depending more on them raises security issues. Protection of educational institutions from cyberattacks depends on a safe LMS (Security R, 2021) .

2. Background of the Application

Learning Management Systems (LMS) have become integral to modern education, facilitating course management and student assessments in digital learning environments. These platforms, which are essential in smart schools, offer numerous benefits but are also susceptible to various cyber threats, such as denial of service attacks, malware, and unauthorized access. Traditional security measures, like password authentication, are no longer sufficient. Therefore, enhancing LMS security is crucial to protect sensitive student data and academic records from potential cyberattacks.

3. Rationale

The reason for creating a safe LMS for educational institutions is the growing need for strong digital learning environments shielding against cybersecurity concerns

(Soykan and Şimşek, 2014). As hackers attack educational institutions more and more, LMS platform security has taken the stage (Amin et al., 2020). The suggested solution will offer a safer, more dependable platform for administrators, teachers, and students by addressing security issues, therefore enabling the continuous change toward digital learning.

4. System Requirements and Specifications

- Platform: Python-based application accessible via Command-Line Interface (CLI).
- Data Structures: Utilizes dictionaries, lists, and sets for efficient management of student data.

Functional Requirements

- 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

Non-Functional Requirements

- 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 LMS data (including student records, course materials, and logs), with scalable options for future growth.
- Peripherals: Keyboard only for CLI access.

GDPR and legal requirements for the application.

School management information systems have to follow the GDPR guidelines to protect and safeguard the data of users (Welsh Government, 2021). This is a legal requirement for software used in schools due to the sensitive nature of the data collected. Under the GDPR guidelines companies must protect the data of their users and must protect the way data is accessed and interacted with by different parties (GDPR, 2016). Under GDPR all data must be stored securely with only selected parties being allowed access to stored information. This information must be deleted after 3 years and the data must be shared with the individual when requested (GDPR, 2016). It is also legally required for software containing such information within a school setting to be regularly updated and screened for potential bugs in order to prevent potential leaks of personal data (Welsh Government, 2021). Legally, it is also required that software is monitored to check for unauthorised use by users or by other parties (Welsh Government, 2021).

John's Section
Solution Proposal

The proposed system is a school management application designed to provide secure and efficient management of student, faculty, and administrative data. It will leverage Python's capabilities for robust development and security features. The application will be accessed via a command-line interface (CLI) and will include mechanisms to protect against various security threats (Falana et al., 2021).

The data repository will utilize three in-memory data structures namely dictionary, set and list. To protect against security threats, several security measures are put in place including:

- **Authentication:** Require users to provide a username and password for login. Implement password hashing to store passwords securely.
- **Authorization:** Assign roles (e.g., student, faculty, admin) to users and grant them appropriate permissions.
- **Encryption:** Encrypt sensitive data (e.g., student personal information, financial records) using strong encryption algorithms.
- **Event Monitoring:** Track system activity and log suspicious events for analysis.
- **Brute Force Protection:** Implement rate limiting to prevent excessive login attempts.
- **Denial of Service Protection:** Use techniques like request validation and resource throttling to mitigate DoS attacks.
- **API Injection Protection:** Validate input parameters and sanitize data before processing (Sani et al., 2022).

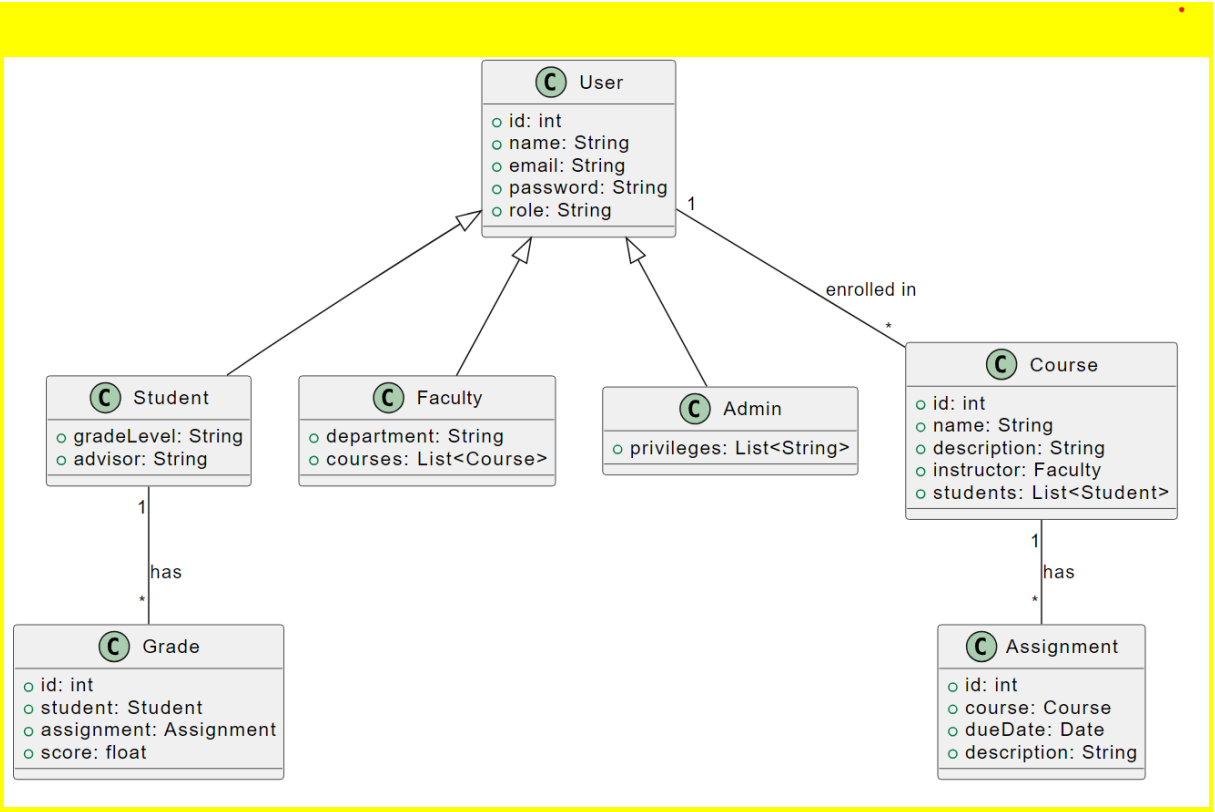
The application will utilize the **Observer** pattern to notify interested parties (e.g., students, faculty) of changes in relevant data (e.g., grades, announcements). This allows for an efficient way to handle updates and notifications within the system.

UML Specifications

1. User Tasks/Activities

- Create an account
- Log in
- View personal information
- Update personal information
- View course schedule
- Access course materials
- Submit assignments
- View grades
- Communicate with instructors

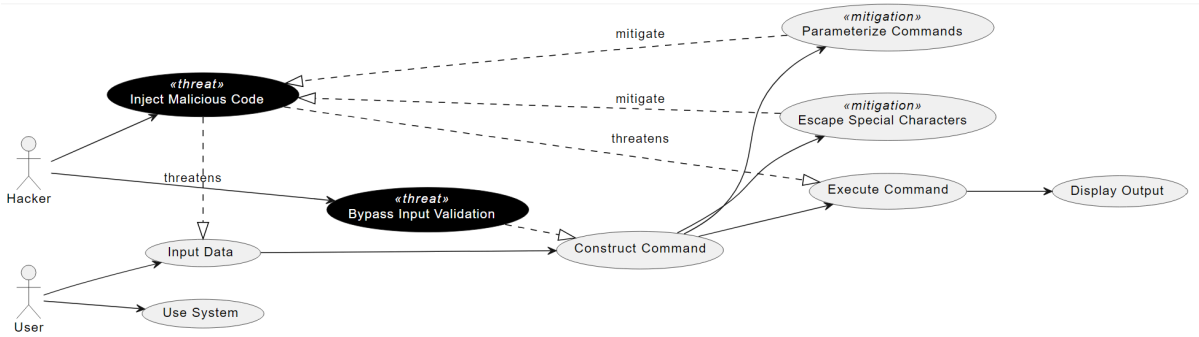
Class Diagram for the Proposed System:



2. Attacker Tasks/Activities

- **Brute Force:** Attempt to guess valid credentials.
- **Denial of Service:** Send excessive requests to overload the system.
- **API Injection:** Inject malicious code into API requests.

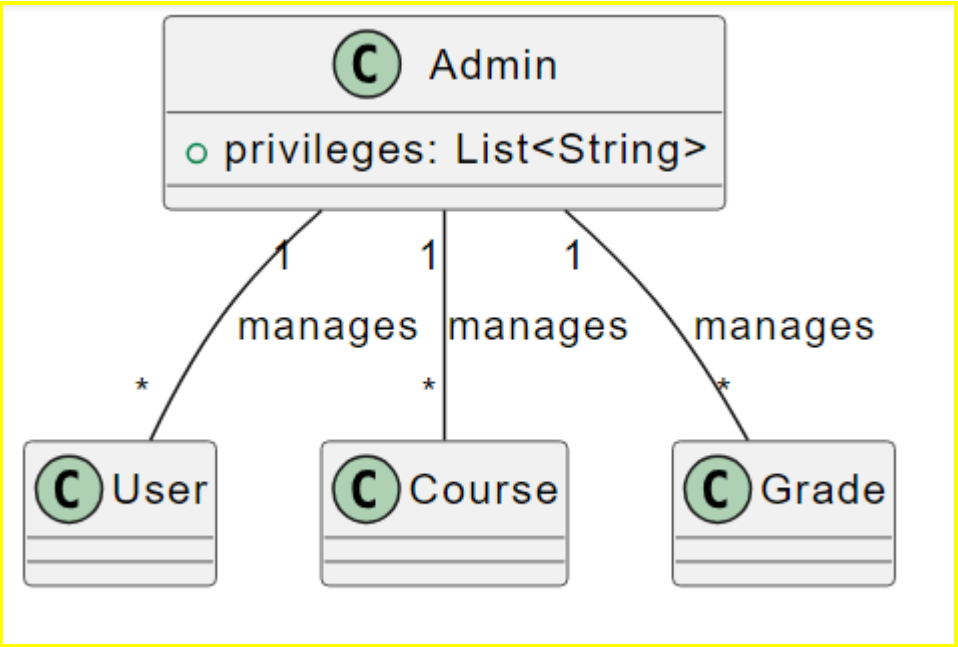
Misuse Case Diagram for Injection Flow Attack:



3. Admin Tasks/Activities

- Manage user accounts
- Create and manage courses
- Assign students to courses
- Grade assignments
- Generate reports
- Monitor system logs
- Enable/Disable security features

Class Diagram for Admin Tasks:



REFERENCES