

**Master of Science in Cyber Security**

**COURSEWORK**

**MODULE : Secure Design and Development FT**

**CODE : 7032CE**

**WEIGHTING : 70%**

**DUE DATE : March 1 2025**

**Submitted By:**

**Name : Ou Zheliang**

**Index No. : 14090055**

**Word Count :**

Table of Content

[1. Introduction 4](#_Toc191563160)

[2. Gantt Chart 4](#_Toc191563161)

[3. Methodology 5](#_Toc191563162)

[4. Requirement Analysis 5](#_Toc191563163)

[4.1 Actors 5](#_Toc191563164)

[4.2 Modules 5](#_Toc191563165)

[4.3 Functional Requirements 5](#_Toc191563166)

[4.4 Nonfunctional Requirements 5](#_Toc191563167)

[4.5 Technical Requirements 5](#_Toc191563168)

[5. System Design 6](#_Toc191563169)

[5.1 Assumptions 6](#_Toc191563170)

[5.2 UML 6](#_Toc191563171)

[5.2.1 Use Case Diagram 6](#_Toc191563172)

[5.2.2 Domain Class Diagram 7](#_Toc191563173)

[5.3 Wireframe 8](#_Toc191563174)

[5.4 Design Principles 8](#_Toc191563175)

[5.5 Security Principles 8](#_Toc191563176)

[5.6 Legislation 8](#_Toc191563177)

[6. Formal Methods 9](#_Toc191563178)

[6.1 Automata 9](#_Toc191563179)

[6.2 Petri Net Model 10](#_Toc191563180)

[7. System Development 10](#_Toc191563181)

[7.1 Summary of Development Approach 10](#_Toc191563182)

[7.2 Development Considerations 10](#_Toc191563183)

[8. Security Testing and Analysis 10](#_Toc191563184)

[9. Conclusion 10](#_Toc191563185)

[10. Reference 10](#_Toc191563186)

[11. Appendix 10](#_Toc191563187)

# Introduction

During an era where computer systems increasingly deal with sensitive personal and operational data, the need for secure, user-centric web applications has taken centre stage—especially in industries dealing with vulnerable groups like children and elderly users.

This report will discuss the development process of the dance studio management website. This project addressed the unique requirements of a dance and performance group that required an online system to manage training records, attendance, injury reports, and performance data for artists aged 7 to 70. The system has two objectives: delivering ease of use to a wide age range while being extremely secure.

# Gantt Chart

The background of this Gantt Chart is to develop a dance studio management website. This Gantt Chart contains six main parts: Requirement Analysis, System Design, Implementation, Testing, Deployment, and Maintenance.

Requirements Analysis only has one task, which captures all the system requirements and was completed on January 31st 2025. The objective of the system design is to prepare the system design, which was finished on February 6th, 2025. During the implementation, five sub-tasks must be completed: register and login function, show records list, edit and update function, settings modules, and style implementation. All the implementation tasks were completed on February 20th 2025. Three tests need to be conducted: system testing, unit testing, and beta testing. All the tests were completed on February 23rd 2025. After that, the website completes the deployment on the same day and starts maintenance.

# Methodology

The management system is developed by using the waterfall model. Waterfall model has six developments stages: Requirements, Design, Implementation, Testing, Deploying, and Maintenance. All requirements are captured in the requirement analysis stage, and all the system design are defined during the design stage. According to the requirements and design, all the functions and webpages are developed in the implementation stage. After implementation, the system needs to conduct the system test, unit test, and beta test. If all the tests are passed, the system can start deploy and maintenance.

# Requirement Analysis

The requirement asks me to develop an application for directors and coaches to manage the data of artists, performances, and training. Directors and coaches can also update attendance and injury records. The dance group includes children and adults from the age of 7 to 70. The actors and modules can be defined as follows:

## 4.1 Actors

* Director
* Coach
* Artist

## 4.2 Modules

* User Management Module
* Data Entry Modul
* Security Audit Module
* Report generation module
* Notification and communication module

## 4.3 Functional Requirements

|  |  |  |
| --- | --- | --- |
| **Module** | **Requirement** | **Security constraints** |
| User Management Module | The system should allow directors and coaches to create, edit, and delete their account | Contains audit logs of operation records |
| User Management Module | When users log in, the system will authenticate their username and password. The password must meet the complexity requirements (At least 8 characters including letters and numbers | Password stores using PBKDF2 algorithm and SHA256 hash |
| User Management Module | To recover passwords for director and coach accounts, a one-time password (TOTP) generated by SMS or an authenticator app must be used | Require multi-factor authentication |
| Data Entry Module | Directors can upload and modify artists’ personal information, their performance attendance, and performance data | Verify the input to prevent the XSS attack |
| Data Entry Module | Coaches can upload and modify artists’ training attendance, and training data | Verify the input to prevent the XSS attack |
| Data Entry Module | The system should automatically record the operating user, time and content of each data modification, and generate a version history for rollback. | Modification records cannot be tampered with |
| Data Entry Module | Directors can generate statistics for performance attendance, and coaches can generate statistics for training attendance | RBAC permission control |
| Security Audit Module | Record abnormal operations every time they occur and log them to a document | Modification records cannot be tampered with |
| Report generation module | Generate attendance visualization chart for performance and training | Sensitive information needs to be shielded |
| Report generation module | Generate data statistics for performance and training data | Sensitive information needs to be shielded |
| Notification and communication module | Guardian confirmation requests and system maintenance notifications will be sent to users three days before maintenance | Communication content needs to be encrypted and sensitive information needs to be shielded |

## 4.4 Nonfunctional Requirements

|  |  |
| --- | --- |
| **Module** | **Requirement** |
| Performance | Average response time less than 2 seconds |
| Performance | File upload (less than 100MB) processing time less than 30 seconds |
| Performance | When the database query has less than 100,000 records, the execution time is less than 5 seconds |
| Security | All sensitive data (such as identify number) must be stored with AES-256 encryption |
| Security | The system must pass OWASP ASVS Level 2 certification and fix all Critical/High level vulnerabilities |
| Usability and Accessibility | The system's annual availability is more than 99.9%, and the unplanned downtime is less than 8.76 hours per year |
| Usability and Accessibility | The interface for children (7-12 years old) must pass COPPA compliance review, and the text reading level must be less than 6th grade in elementary school |
| Reliability and Fault Tolerance | When a single node fails, the system automatically switches to the backup server, with a recovery time objective of less than 5 minutes |
| Reliability and Fault Tolerance | When the network is interrupted, the front end needs to cache uncommitted data and automatically retry after recovery |
| Reliability and Fault Tolerance | Persistent data storage must guarantee 99.9999999% reliability |
| Compliance | EU user data must meet GDPR requirements, provide a "data deletion" function, and complete the request within 72 hours |

## 4.5 Technical Requirements

|  |  |  |
| --- | --- | --- |
| **Components** | **Technology Selection** | **Reason** |
| Front-end | HTML, CSS, JavaScript | Simple to understand and easy to use |
| Backend | Django (Python) | Built-in security features (CSRF, ORM anti-SQL injection) |
| Database | MariaDB (XAMPP) | Easy to install and simple to operate |
| Code Editor | Visual Studio Code | Quick installation, flexible selection of features and plugins |

# System Design

## Assumptions

1. We assume that the directors and coaches would be able to access the system.
2. We assume that the directors and coaches would be the admin of the system.
3. We assume that the directors would be allowed to do an admin creation for artist module and performance module.
4. We assume that the coaches would be allowed to do an admin creation for training module.
5. We assume that Artists would not access to the system.
6. We assume that Performance and training have no dependencies.

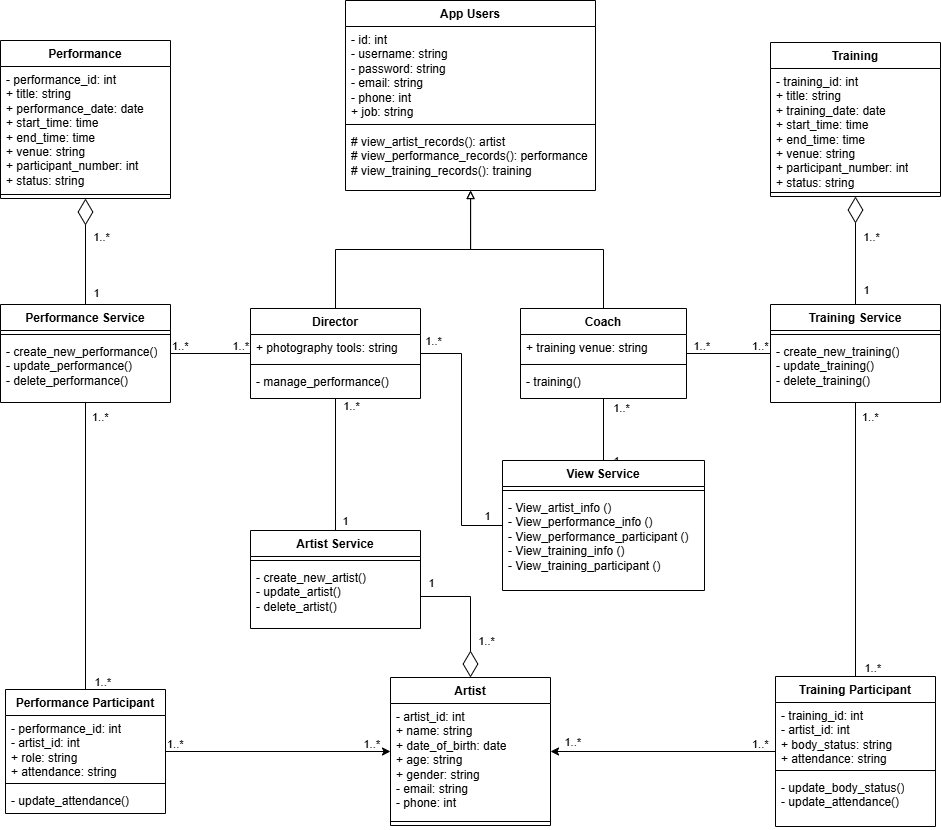
## UML

### Use Case Diagram

This Use Case Diagram focuses on the interaction between users (directors and coaches) and dance studio management platform, including following key elements:

* Actor: admin, director, coach.
* Use Case: sign up, login, change password, view artist info, view performance info, view training info, modify artist data, modify performance data, and modify training data.
* Relationship: All functions are directly interactive by the user

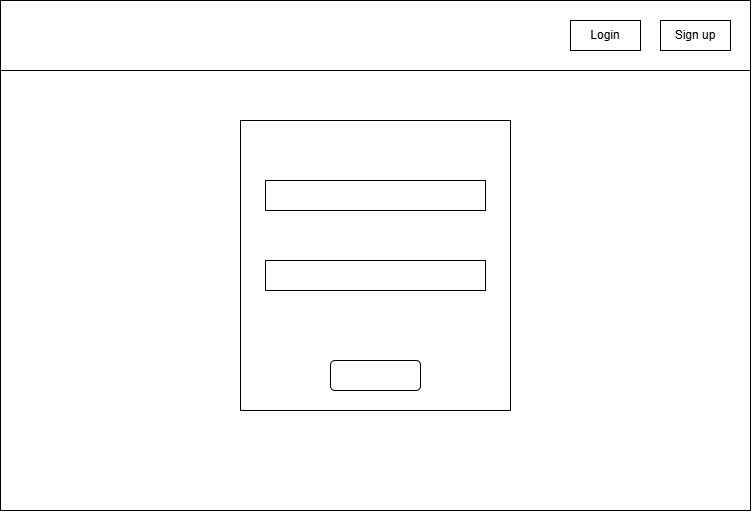
### Domain Class Diagram

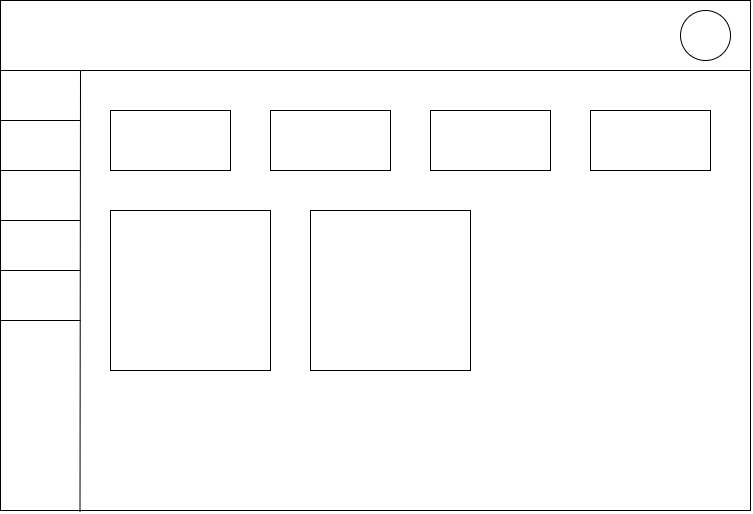


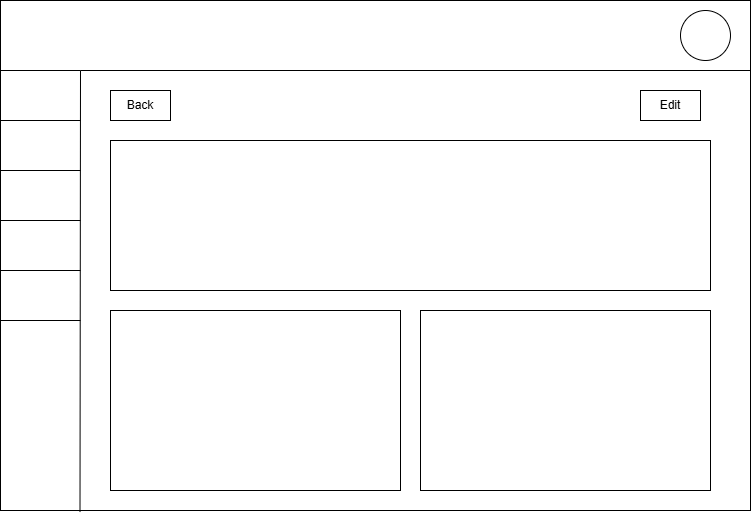
This is a Domain Class Diagram for dance studio management system, including following key elements:

* Class:
* App Users: (Attributes: id, username, password, email, phone, job; Methods: view\_artist\_records(), view\_performance\_records(), view\_training\_records())
* Director: (Attributes: photography tools; Methods: manage\_performance())
* Coach: (Attributes: training venue; Methods: training())
* Artist: (Attributes: artist\_id, name, date\_of\_birth, age, gender, email, phone)
* Performance: (Attributes: performance\_id, title, performance\_date, start\_time, end\_time, venue, participant\_number, status)
* Training: (Attributes: training\_id, title, training\_date, start\_time, end\_time, venue, participant\_number, status)
* Perofrmance Participant: (Atrtibutes: - performance\_id, artist\_id, role, attendance; Methods: update\_attendance())
* Training Participant: (Atrtibutes: - training\_id, artist\_id, body\_status, attendance; Methods: update\_body\_status(), update\_attendance())
* Performance Service: (Methods: create\_new\_performance, update\_performance(), delete\_performance())
* Training Service: (Methods: create\_new\_training, update\_training(), delete\_training())
* Artist Service: (Methods: create\_new\_artist, update\_artist(), delete\_artist())
* View Service: (Methods: View\_artist\_info (), View\_performance\_info (), View\_performance\_participant (), View\_training\_info (), View\_training\_participant ())
* Relationships:
* Director and Coach inherit App User
* Director and Coach associate View Service
* Director associate Artist Service and Performance Service
* Coach associate Training Service
* Performance Service associate Performance Participant
* Training Service associate Training Participant
* Performance Participant and Training Participant associate Artist
* Performance aggregate Performance Service
* Training aggregate Training Service
* Artist aggregate Artist Service

## Wireframe







## Design Principles

Design principles are the core guidelines for system architecture and development, and must be combined with business goals, security requirements, and user experience.

|  |  |  |
| --- | --- | --- |
| **Design Principles** | **Definition** | **Implementation** |
| Least Privilege | Users and system components have only the minimum permissions required to complete their tasks | Different role users can only modify the data they be assigned |
| Defence in Depth | Reduce single point failure risk through multi-layer protection mechanism | Transmission encryption (HTTPS) + Storage encryption (AES-256) + Backup encryption (AWS KMS) |
| Privacy by Design | Embed privacy protection from the early stage of system design | Anonymization of children’s data; Collect only necessary information |
| Age-Adaptive UI | Dynamically adjust interface complexity and interaction methods based on user age | Children's interface: large icons, simplified operation process;  Elderly interface: high contrast theme, font enlargement, reduced pop-up window interference |
| Modular Architecture | The system is split into independent modules according to functions to reduce coupling | Split the features by using various Class and functions |
| Configuration over Hardcoding | Externalize mutable parameters (such as secret keys) as configuration files | Third-party service keys are injected through environment variables |
| Regulatory Compliance | Comply with data protection regulations to ensure users’ right to know and control | Provide "data deletion" function to permanently delete all user information |
| Audit Trail | Record key operation logs to support post-event tracing | Record user login, data modification, file download and other operations |

## Security Principles

All security principles are involved CIA (Confidentiality, Integrity, Availability).

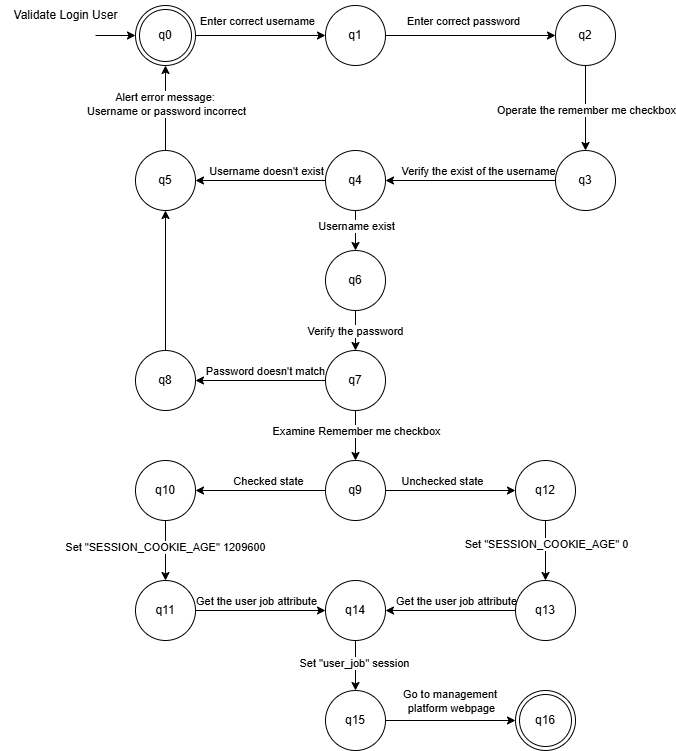
|  |  |
| --- | --- |
| **Security Principles** | **Description** |
| Least privilege | Director and coaches can view all data. Director can only create and edit artist and performance data. Coaches can only create and edit training data. Artist can’t access any data. |
| Input validation | All input will be validated on the server side. |
| Human-machine verification | The system uses RECAPTCHA to verify whether the user is a human or a machine. |
| Password encryption | Django framework use PBKDF2 algorithm and SHA256 hash function with random salt to encrypt the user password. |
| XSS protection | Django framework use “autoescape” to perform simple HTML escaping. |
| CSRF Protection | Django framework provide CSRF token to protect CSRF attack. |
| SQL injection protection | Django framework provide Object-Relational Mapping to protect SQL injection. |

## Legislation

I adopted the General Data Protection Regulation (GDPR) as my legislation because its scope is organizations that process EU citizens' data, regardless of location. The system must provide data access, correction, deletion (right to be forgotten) and portability to the users. The critical operation must obtain user consent (such as guardian consent for children's data). If Data was breached, the system must report significant breaches to regulators within 72 hours.

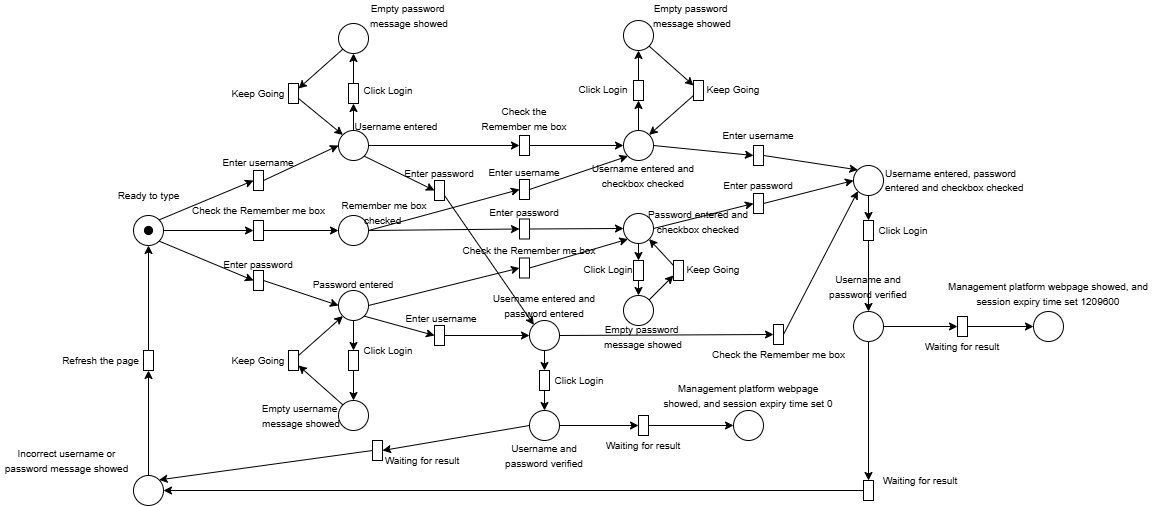
# Formal Methods

## 6.1 Automata

Login Process

The above Automata describe the process of entering login details into the system. It begins with a ready login state in the login webpage; the users enter the username and password, select the remember me check box, and click the login button to validate the username and password.

## 6.2 Petri Net Model

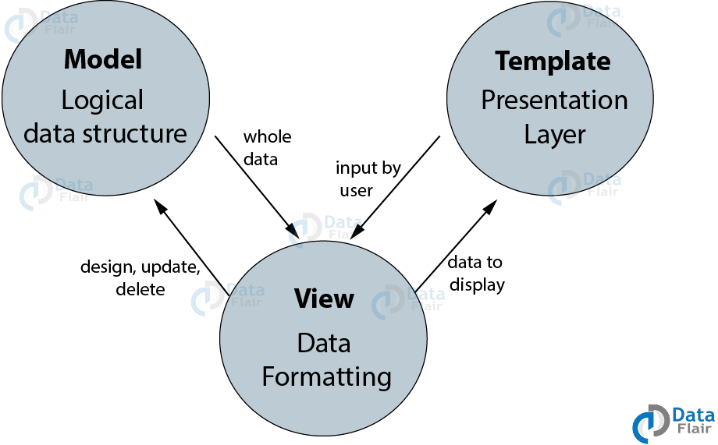


Once the Finite Automata is created, I base on it and complete the Petri net diagram. This Petri net describes the state changes, event triggering, and resource flows in distributed, concurrent systems.

# System Development

## 7.1 Summary of Development Approach

This project was developed using Python Django MTV (Model-Template-View), HTML, CSS, JavaScript, and Visual Studio Code software.

The Django framework provides advantages by abstracting database operations through the built-in ORM and following the DRY (Don't Repeat Yourself) principle to reduce redundant code. Developers only need to focus on business logic, and rich middleware, form tools, and out-of-the-box security mechanisms (such as CSRF protection) help them build scalable database-driven Web applications.

The above picture shows how the MTV model works. When a user initiates a request, Django directs the request to the corresponding View through routing (URL configuration). The View is responsible for business logic processing, and then passes the data to the Template for dynamic rendering, and finally generates a complete HTML page and returns it to the user, realizing a full-link closed loop from data access to interface display.

## 7.2 Development Considerations

1. All security features on develop based on the Python Django framework.
2. Password are stored by using PBKDF2 algorithm hashing with random salt.
3. The website design should keep user-friendly.
4. Keep XSS and CSRF protection open on every necessary location.
5. The system provides the OTP when users need to recover the password.

# Security Testing and Analysis

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Modules** | **Use Case Scenario** | **Sub Scenario** | **Expected Result** | **Result** | **Solutions** |
| Director and coach | Login | Login with SQL injection | Reject login request | Successful | - |
| Login | Forgot password | Receive OTP and recover password | Successful | - |
| Insert a new record | Fill out text input with Cross-Site Scripting | Display the whole code and not run | Successful | - |
| Modify sensitive data | Users visit third-party sites containing malicious code (Cross-Site Request Forgery) | Reject all unauthorized data modification requests | Successful | - |
| Register and Change password | Use normal password (Full number or full letters) | Reject the request and ask for strong password | Successful | - |
| Director | View training records | Try to modify the training data | Do not provide the modify functions | Successful | - |
| Coach | View performance records | Try to modify the performance data | Do not provide the modify functions | Successful | - |

# Conclusion

In conclusion, I successfully developed a web-based system to manage the dance studio. This system integrates security-by-design principles and formal verification methods (automata and Petri net) to manage multi-age user data. It also uses Django’s built-in protections (CSRF tokens, ORM anti-injection) to protect against SQL, XSS, and CSRF attacks. It complies with GDPR and maintains usability for children and elderly users.

Some security principles are not achieved because of technical limits, such as auto-backup database and generating security reports. This will be enhanced by the developer in the future.

# Reference

Infinity, B. (2023, July 12). *Waterfall model in software Engineering | Board Infinity*. Board Infinity. https://www.boardinfinity.com/blog/a-quick-guide-to-arrays-in-python/

GeeksforGeeks. (2024, September 12). *Introduction of finite automata*. GeeksforGeeks. https://www.geeksforgeeks.org/introduction-of-finite-automata/

Kirankishore. (2022, June 22). Petri Net — simplified - geek culture - medium. *Medium*. https://medium.com/geekculture/petri-net-simplified-3460a27cb1dd

Kirvan, P., Lutkevich, B., & Lewis, S. (2024b, November 15). *What is a Waterfall model? Definition and guide*. Search Software Quality. <https://www.techtarget.com/searchsoftwarequality/definition/waterfall-model>

Team, D. (2021, June 21). *Django MTV Architecture – Understanding Components & Working*. DataFlair. https://data-flair.training/blogs/django-mtv-architecture/

Trapani, K. (2023, March 3). *Security by design: 7 application security principles you need to know*. Cprime. https://www.cprime.com/resources/blog/security-by-design-7-principles-you-need-to-know/

# Appendix