
Music School CRM - Application Report

Version: Final

Date: 30.05.2025

Done by: Denis K., Pavel F., Danil T.

1. Introduction and Objectives

Introduction:

This report details the structure, features, and technical implementation of the Music School CRM web application. Built using the Flask framework in Python, this application serves as a centralized platform for managing key aspects of a music school's operations, specifically focusing on student records, course administration, and user authentication. It leverages extensions like Flask-SQLAlchemy for database interaction, Flask-Login for user session management, and Flask-WTF for secure form handling.

Objectives:

The primary objectives of the Music School CRM application are:

- **User Management:** Provide a secure system for user registration and login.
- **Student Management:** Enable administrators/users to perform CRUD (Create, Read, Update, Delete) operations on student records.
- **Course Management:** Allow users to manage course information, including details, start dates, and associated learning materials (via file uploads).
- **Enrollment Tracking:** Facilitate the enrollment of students into specific courses and display enrollment details.
- **Data Persistence:** Securely store and retrieve application data using a relational database (MySQL specified in config).
- **Modular Design:** Structure the application code logically using Flask Blueprints for maintainability and scalability.
- **Secure File Handling:** Implement secure uploading, storage, and serving of course materials.
- **User-Friendly Interface:** Provide a clean, intuitive web interface using Bootstrap for styling.

2. Description of Features

The Music School CRM application offers the following core features:

1. Authentication:

- **User Registration:** New users can create an account with a unique username and password (/register).
- **User Login:** Registered users can log in to access protected areas (/login). Uses password hashing for security.
- **Session Management:** Flask-Login manages user sessions, including a "Remember Me" option.
- **Logout:** Authenticated users can securely log out (/logout).
- **Access Control:** Most application routes are protected using @login_required, redirecting unauthenticated users to the login page.

2. Dashboard (/):

- Serves as the main landing page after login.
- Provides quick navigation links to the Students and Courses sections.

3. Student Management (/students):

- **List & Search:** View a paginated list of all students. Search students by first or last name (/students/).
- **Add Student:** Add new students with first name, last name, and email (/students/add). Form validation is included.
- **Edit Student:** Update existing student details (/students/<id>/edit).
- **Delete Student:** Remove a student record from the system (/students/<id>/delete). Requires POST request and CSRF protection (implicitly via WTForms in the form submission).
- **View Student Details:** See individual student information and a list of courses they are enrolled in (/students/<id>).

4. Course Management (/courses):

- **List Courses:** View a list of all available courses, showing title, start date, and links to materials (/courses/).
- **Add Course:** Create a new course, including title, start date, and optional course material upload (/courses/add). The logged-in user is automatically assigned as the course creator (user_id).

- **Edit Course:** Modify existing course details and update/replace course materials (/courses/<id>/edit). Shows the current material file.
- **Delete Course:** Remove a course record and its associated material file from the server (/courses/<id>/delete). Requires POST request and CSRF protection.
- **View Course Details:** Display comprehensive course information, including the creator's username, start date, links/players for viewing/downloading materials, and a list of currently enrolled students (/courses/<id>).
- **View Course Material Summary:** Display summary of text-type uploaded material via free Google AI API (Gemma 12B). Separate button with summary is available if your material is of .pdf or .docx type
- **Material Handling:**
 - Uploads allowed for specific file types (docx, .pdf, .mp3, .mp4).
 - File size limits are enforced.
 - Secure filename handling is used.
 - Files are served via a dedicated route (/uploads/<filename>).
 - Appropriate display methods (links, audio/video players) are used in templates based on file type.

5. Enrollment Management:

- **Enroll Student:** From the course detail page, users can enroll available students into the displayed course (/courses/<course_id>/add_student/<student_id>). Prevents duplicate enrollments. Requires POST and CSRF protection.
- **View Enrollments:** Enrollments are visible on both the student detail page (courses the student is in) and the course detail page (students enrolled in the course).

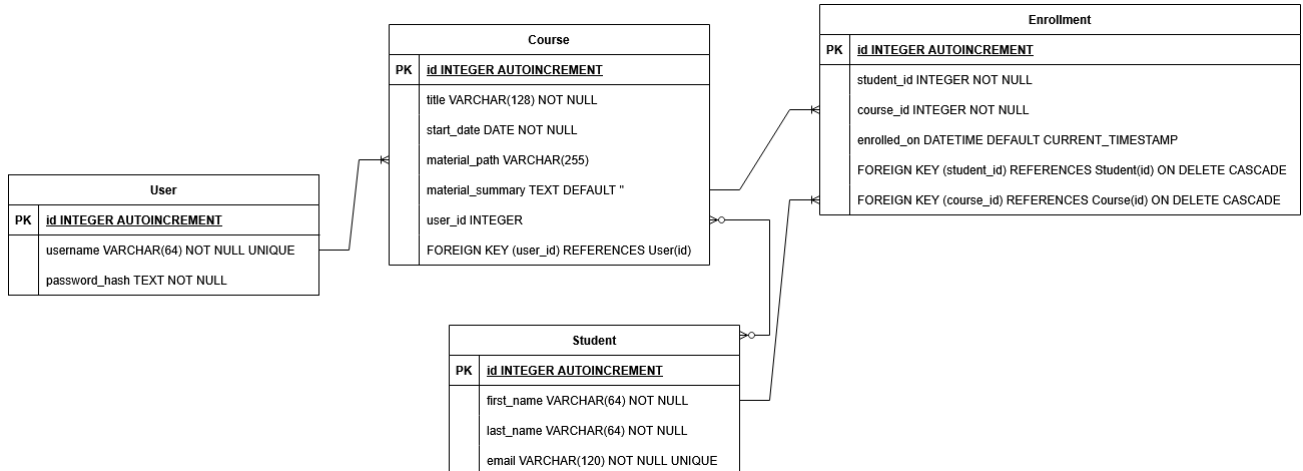
6. Security:

- **CSRF Protection:** Flask-WTF provides Cross-Site Request Forgery protection on all forms.
- **Password Hashing:** User passwords are securely hashed using werkzeug.security.
- **Input Validation:** Forms use WTForms validators (DataRequired, Email, Length, EqualTo).

- **Secure File Serving:** send_from_directory is used to prevent directory traversal issues when serving uploaded files.

3. ER Diagram of the Database

Database schema consists of four main tables: User, Student, Course, and Enrollment.



Relationships:

- * **User <-> Course (One-to-Many):** One User can create many Courses. Each Course is created by one User (`user_id` links to `User.id`). The `nullable=True` allows courses to exist without a creator initially or if migrating old data.
- * **Student <-> Enrollment (One-to-Many):** One Student can have many Enrollments (enroll in many courses).
- * **Course <-> Enrollment (One-to-Many):** One Course can have many Enrollments (have many students enrolled).
- * **Student <-> Course (Many-to-Many):** The `Enrollment` table acts as a junction table, creating a many-to-many relationship between Students and Courses. A student can enroll in multiple courses, and a course can have multiple students. The `Enrollment` table also stores the date (`enrolled_on`) when the enrollment occurred.
- * **Cascading Deletes:** Deleting a Student or a Course will automatically delete associated Enrollment records due to `cascade="all, delete"` on the relationships.

4. Explanation of Code Structure (OOP & Blueprints)

The application follows standard Flask project structure conventions, emphasizing modularity and Object-Oriented Programming principles.

Object-Oriented Programming (OOP):

OOP is utilized primarily in the following areas:

1. **Models (models.py):** Each database table (User, Student, Course, Enrollment) is represented by a Python class inheriting from `db.Model` (Flask-SQLAlchemy). These classes encapsulate the data (attributes like `id`, `username`, `title`, etc.) and associated behavior (like `set_password`, `check_password` methods in the User model). Relationships between tables are also defined within these classes using `db.relationship`.
2. **Forms (forms.py):** Web forms are defined as classes inheriting from `FlaskForm` (Flask-WTF). Each form class defines the form fields (`StringField`, `PasswordField`, `FileField`, etc.) and their associated validation rules (`DataRequired`, `Email`, etc.). This encapsulates form logic and validation.
3. **Flask Extensions:** The core Flask application and its extensions (SQLAlchemy, LoginManager, CSRFProtect) are themselves implemented using classes, providing an object-oriented interface for configuration and use.

Flask Blueprints:

Blueprints are used to organize the application into distinct functional components, making the codebase more manageable and scalable.

1. **Purpose:** Instead of defining all routes in the main `app.py`, routes related to specific features are grouped into separate files (Blueprints).
2. **Implementation:**
 - **routes/auth.py (auth_bp):** Contains routes for user authentication (login, logout, register).
 - **routes/students.py (students_bp):** Handles all routes related to student management (list, add, edit, delete, detail). Registered with the URL prefix `/students`.
 - **routes/courses.py (courses_bp):** Manages routes for course operations (list, add, edit, delete, detail, enroll student). Registered with the URL prefix `/courses`.
 - **routes/main.py (main_bp):** Defines the main index/dashboard route (`/`).
3. **Registration:** These Blueprints are imported and registered with the main Flask application instance within the `create_app` function in `app.py`. This links the routes defined in the Blueprints to the application's routing system.

This structure promotes separation of concerns, making it easier to locate code related to specific features and simplifying future development or modification.

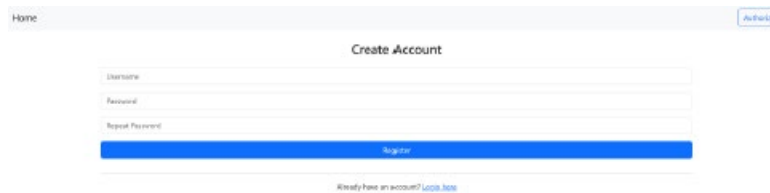
5. Screenshots of Key Pages

- Login Page (/login)



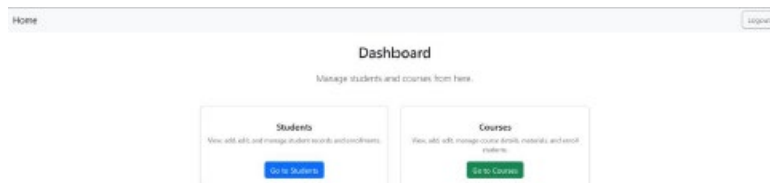
The screenshot shows the Login page. At the top, there is a navigation bar with a 'Home' link on the left and an 'Authorise' button on the right. The main heading is 'Login'. Below it, there are three input fields: 'Username', 'Password', and 'Remember me'. A blue 'Login' button is positioned below these fields. Below the button, there is a link 'Don't have an account?' and a green 'Create New Account' button.

- Registration Page (/register)



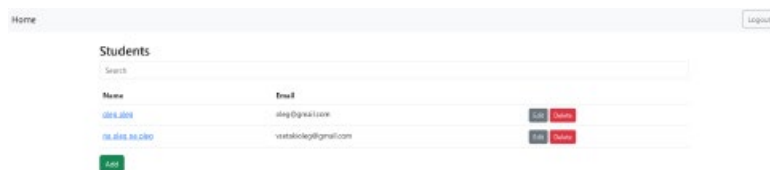
The screenshot shows the Create Account page. At the top, there is a navigation bar with a 'Home' link on the left and an 'Authorise' button on the right. The main heading is 'Create Account'. Below it, there are three input fields: 'Username', 'Password', and 'Repeat Password'. A blue 'Register' button is positioned below these fields. Below the button, there is a link 'Already have an account? [Login here](#)'.

- Dashboard / Home Page (/)



The screenshot shows the Dashboard page. At the top, there is a navigation bar with a 'Home' link on the left and a 'Logout' button on the right. The main heading is 'Dashboard'. Below it, there is a sub-heading 'Manage students and courses from here.' and two cards. The first card is titled 'Students' and contains the text 'View, add, edit and manage student records and enrolments.' with a blue 'Go to Students' button. The second card is titled 'Courses' and contains the text 'View, add, edit, manage course details, materials and enrol students.' with a green 'Go to Courses' button.

- Student List Page (/students/)



The screenshot shows the Students page. At the top, there is a navigation bar with a 'Home' link on the left and a 'Logout' button on the right. The main heading is 'Students'. Below it, there is a search bar. Below the search bar, there is a table with two columns: 'Name' and 'Email'. The table contains two rows of data. The first row has 'John Doe' in the Name column and 'john.doe@example.com' in the Email column. The second row has 'Jane Smith' in the Name column and 'jane.smith@example.com' in the Email column. To the right of each row, there are two buttons: 'Edit' (blue) and 'Delete' (red). Below the table, there is a green 'Add' button.

- Add/Edit Student Form (/students/add or /students/<id>/edit)

Home [Logout](#)

Add Student

First Name

Last Name

Email

[Cancel](#) [Add Student](#)

Home [Logout](#)

Edit Student

First Name

Last Name

Email

[Cancel](#) [Edit Student](#)

- Student Detail Page (/students/<id>)

Home [Logout](#)

oleg oleg

[oleg@gmail.com](#)

Enroll in Available Courses

[21124 Olego 1111-11-11](#) [+ Enroll](#)

Currently Enrolled Courses

This student is not currently enrolled in any courses.

[Back to Students List](#) [Add Student](#)

- Course List Page (/courses/)

Home [Logout](#)

Courses

Title	Start	Material	
21124	1111-11-11	View Document	Add Delete

[Add](#)

- Add/Edit Course Form (/courses/add or /courses/<id>/edit)

Add Course

Title

Start Date

Course Material

Max file size: 25MB. Allowed types: docx, pdf, mp3, mp4.

If a PDF is uploaded, a summary will be automatically generated.

Edit Course

Title

Start Date

Course Material

Current file: doom.pdf [\(View/Download\)](#)

Upload a new file to replace the current one (optional).

Max file size: 25MB. Allowed types: docx, pdf, mp3, mp4.

If a PDF is uploaded, a summary will be automatically generated.

- Course Detail Page (/courses/<id>) - Showing details, material link/player, and enrolled students.

Home

CheckCRM

Created by: Denis

Start Date: 1111-11-11

Material

Currently Enrolled Students

No students are currently enrolled in this course.

Add Students to Course

[ne oleg ne oleg](#)

(vsetakioleg@gmail.com)

You have no such rights!

6. Challenges Faced and Solutions

Challenge: Implementing File Uploads and Serving

One significant challenge during development was implementing a secure and functional system for uploading, managing, and serving course materials (files). Key aspects of this challenge include:

- Handling file data submitted via HTML forms.
- Validating file types and sizes.
- Storing files securely on the server file system.
- Preventing filename conflicts and security vulnerabilities (e.g., path traversal).
- Associating the correct file with the corresponding course record in the database.
- Handling updates (replacing existing files) and deletions (removing files from disk when a course is deleted).
- Providing a secure way for users to access/download these files via the web interface.
- Displaying different types of media (PDFs, audio, video) appropriately in the browser.

Solution:

The following steps and components were implemented to address the file handling challenge:

1. **Form Field:** A FileField from Flask-WTF was added to the CourseForm in forms.py to handle the file input in HTML forms. The form's enctype was set to multipart/form-data in the templates (templates/courses/form.html).
2. **Configuration:** UPLOAD_FOLDER, ALLOWED_EXTENSIONS, and MAX_CONTENT_LENGTH were defined in config.py and loaded into the Flask app configuration in app.py. This centralizes file-related settings.
3. **Upload Logic (routes/courses.py - add and edit):**
 - The request.files dictionary was accessed to get the uploaded FileStorage object.
 - werkzeug.utils.secure_filename() was used to sanitize the filename, preventing directory traversal attacks and ensuring a safe filename for the filesystem.
 - The file extension was checked against the ALLOWED_EXTENSIONS set.
 - The *sanitized filename* (not the full path) was stored in the material_path column of the Course model in the database.

- The file was saved to the configured UPLOAD_FOLDER using `file.save(save_path)`.
- Error handling (e.g., file save errors, invalid file types) was added using `try...except` blocks and Flask's flash messaging.

4. **File Updates/Deletions (routes/courses.py - edit and delete):**

- In the edit route, if a new file is uploaded, the code checks if an old file exists (based on `course.material_path`) and uses `os.remove()` to delete the old file from the UPLOAD_FOLDER before saving the new one and updating the `material_path` in the database.
- In the delete route, before deleting the Course record from the database, the code checks `course.material_path` and uses `os.remove()` to delete the associated file from the filesystem.

5. **Secure Serving (app.py):**

- A dedicated route (`@app.route('/uploads/<path:filename>')`) was created.
- Flask's `send_from_directory()` function was used within this route. This function is designed to securely serve files from a specific directory (the UPLOAD_FOLDER), preventing users from accessing files outside this designated area.

6. **Template Display (templates/courses/list.html, templates/courses/detail.html):**

- Jinja2 templates check if `course.material_path` exists.
- If it exists, the `url_for('uploaded_file', filename=course.material_path)` function is used to generate the correct URL to the file-serving endpoint.
- Conditional logic based on the file extension (`.pdf`, `.mp3`, `.mp4`, `.docx`) is used to render either a direct link (with a summary button in details), an `<audio>` tag, or a `<video>` tag, providing appropriate ways for users to interact with the material.

7. **CSRF Protection:** Forms involving actions that modify files (like the delete form which triggers file removal) include `{{ csrf_form.hidden_tag() }}` and validation in the backend to prevent CSRF attacks.

This multi-faceted approach ensures that files are handled securely throughout their lifecycle within the application, from upload to deletion, while providing a functional user experience.
