Republic of the Philippines
Camarines Sur Polytechnic Colleges
Nabua, Camarines Sur
ISO 9001:2015
COLLEGE of COMPUTER STUDIES

Video Merger and Uploader

Submission for

Application Development and Emerging Technologies

Information Assurance and Security

Group Members

Jave A. Bacsain

Carl Gerald J. Parro

Marc Justin N. Prestado

Part I

Application Development and Emerging Technologies

I. Project Overview and Problem Statement

The Video Merger and Uploader addresses a specific bottleneck in the content creation industry. Streamers, educators, and YouTubers currently waste significant time manually organizing video files, opening complex editing software simply to join clips together, and waiting for rendering to finish before they can interact with the YouTube upload interface.

The primary problem is the fragmented workflow. Switching between file explorers, video editors, and web browsers. Our application solves this by providing a unified pipeline. It automates the selection, merging, metadata application, and uploading of video content, effectively reducing a workflow that typically takes an hour of intermittent attention down to five minutes of active interaction.

II. Feature List and Scope Table

A. Feature List

The system is divided into core features that provide baseline functionality and advanced features that utilize emerging technologies:

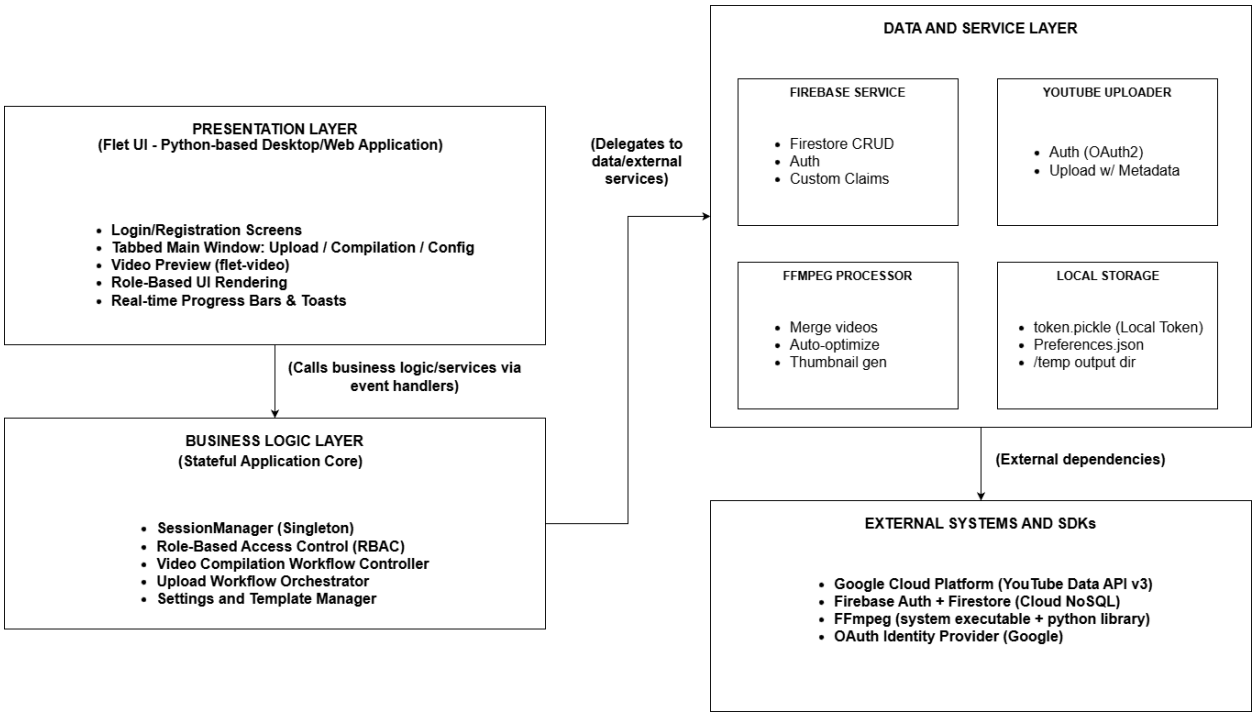
- **Core Features:** Secure Google Login, drag-and-drop video selection, sequential merging via FFmpeg, direct YouTube uploading, and real-time progress tracking.
- **Advanced Features:** Reusable metadata templates, a comprehensive User Role System, and a secure Admin Dashboard for user management.

B. Scope Table

Feature / Module	In Scope	Out of Scope	Notes
Video Merging	Basic concatenation, Resolution standardization, Format Conversion	Trimming, Color Correction, Multi-track editing	The focus is on speed and automation, not creative non-linear editing
Authentication	Google OAuth 2.0, Role Management	Facebook, Twitch, Custom Email Login	YouTube is the target platform, making Google Auth the logical choice

Uploading	Visibility, Title, Tags, Description	Scheduling, YouTube Shorts tools	Planned for next or future iteration
Data Storage	User Roles, Usage Stats, Audit Logs	Cloud Video Storage	Videos are processed locally to maintain privacy and reduce hosting costs.

III. Architecture Diagram



IV. Data Model

The application utilizes a NoSQL data model hosted on Firebase Firestore. The schema is designed for flexibility and real-time synchronization.

- User Collection *(`users`)*
  - This is the primary collection. Each document is keyed by the user's *`uid`*. It contains fields for *`email`* (string), *`role`* (guest, free, premium, developer, admin), *`stats`* (a map containing integers for *`upload_count`* and *`merge_count`*), and *`last_login`* (timestamp).

- Audit Logs Collection (*audit_logs*)
  - This collection stores security events. Each document contains a unique *log_id*, the *action* performed (e.g., "PROMOTE_USER"), the *actor_email* (who did it), the *target_email* (who it was done to), and a server-side *timestamp*.

V. Emerging Technologies

This project Integrates two distinct categories of emerging technologies to enhance functionality and user experience

1. Smart Codec Detection (FFmpeg Integration)

We moved beyond simple file concatenation by implementing an intelligent codec analysis layer. The application uses ffprobe to scan input videos and analyze metadata such as codec, resolution, and framerate. Based on this analysis, the system automatically determines the optimal merge strategy: using fast stream copy for videos with identical codecs (instant preview generation) or full re-encoding for mixed-codec videos (with compatibility warnings). This prevents merge failures and provides users with clear feedback about video compatibility issues before processing.

2. Cloud-Based User Management (Firebase Firestore)

Unlike traditional desktop apps that store user data locally, our application utilizes Firebase Firestore for cloud-based user persistence. User profiles, roles, usage statistics, and metadata templates are stored in the cloud, allowing users to access their data across multiple devices. When an Administrator changes a user's role in the database, the change is reflected when the user next logs in or refreshes their session, providing centralized user management capabilities within a desktop application environment.

3. Smart Video Processing with Intelligent Caching

We implemented an automated codec analysis and caching system that goes beyond simple video concatenation. The application uses ffprobe to analyze video metadata (codec, resolution, framerate) and intelligently determines the optimal processing strategy. For videos with identical codecs, the system uses fast stream copy to generate instant previews (1-2 seconds), enabling real-time preview while processing. For mixed-codec videos, it skips preview generation and warns users about compatibility issues, preventing merge failures and wasted processing time. This dynamic decision-making eliminates manual codec selection and provides immediate visual feedback.

4. **Micro-Analytics with Automated Usage Tracking**

   The application features a lightweight analytics system that tracks user behavior and enforces role-based limits without external analytics platforms. The system automatically tracks daily arrangement counts per user, implements midnight UTC automatic reset logic, calculates remaining usage in real-time, and displays usage statistics with reset countdowns in the UI. This purposeful analytics integration enables fair-use enforcement for free users while providing admins with usage insights for feature prioritization and resource planning.

VI. Setup and Run Instructions

To set up the development environment, the following steps must be taken:

1. **Prerequisites:** The system requires Python 3.10 or later. FFmpeg must be installed on the host machine and added to the system PATH
2. **Configuration:** Security credentials must be configured manually. The *`client_secret.json`* (for OAuth) and *`firebase-admin-key.json`* (for Firestore) must be placed in the *`configs/`* directory
3. **Dependencies:** All Python libraries can be installed via the package manager using the command: *`pip install -r requirements.txt`*
4. **Execution:** To launch the application in desktop mode, navigate to the source directory (*`cd src`*) and run the Flet command: *`flet run`*
5. **Platform Support:** The application is fully validated on Windows 10/11 and retains compatibility with macOS and Linux due to the cross-platform nature of Python and Flet

VII. Testing Summary

Quality Assurance was conducted using a mix of automated and manual testing strategies. We utilized *pytest* for our automated testing suite, focusing on the critical business logic.

- **Automated Coverage:** We achieved comprehensive test coverage on the Role-Based Access Control logic, verifying that all four roles (Guest, Free, Premium, Admin) interact correctly with the permission sets. The Session Manager's lifecycle (login, token refresh, logout) was also fully covered by integration tests.

- **Manual Testing:** User Interface responsiveness and the accuracy of the upload progress bars were tested manually to ensure a smooth user experience.
- **Execution:** The full test suite can be verified by running pytest tests/ -v in the terminal. Individual test modules can be run separately (e.g., *`pytest tests/test_roles.py -v`* for role-specific tests).

VIII. Team Roles and Contribution Matrix

Contributions can be seen throughout the whole process, but it is divided specifically into:

Member	Role	Responsibilities	Contribution (by percentage %)
Bacsain, Jave A.	Project Lead	Responsible for main initializations. Implemented Core Architectures, and oversees the Role-Based Access Control as a Super Admin	50%
Parro, Carl Gerald J.	Data & Integration Engineer QA / Test Coordinator Documentation & Release Manager	Enhanced the GUI for better usability and fixed minor bugs such as non-functional buttons	20%
Prestado, Marc Justin N.	Data & Integration Engineer QA / Test Coordinator Documentation & Release Manager	Bug and Enhancement issues handler. Responsible for many different features that enhances User Experience	30%

IX. Risk / Constraint Notes and Future Enhancements

Constraints

- The application operates under specific technical constraints. It has a strict dependency on a stable internet connection for both authentication and uploading. Additionally, development and testing were constrained by Google's API quotas (limiting requests to 10,000 units/day), which required careful management of testing cycles.

Future Enhancements

- If several difficulties and complications arise, to improve the deployment experience, we plan to containerize the application using Docker, which would remove the need for users to manually install FFmpeg. We also aim to implement an "Offline Queue," allowing users to merge videos without an internet connection and automatically upload them once connectivity is restored.
- In the next iteration of the development, the focus will be the awaiting existing bugs and enhancements in the GitHub Repository:
    - Better Error Handling / Feedback: User Friendly error messages should appear when an error occurs and not a string of stack trace.
    - Encode individual clip in merge screen: Preprocessing clips to resolve format discrepancies before merging
    - Rate Limiting: Controls network traffic to prevent brute force attacks, mitigate system abuse, and ensure compliance with outgoing API quotas
    - Ad borders that is dependant on the ad content to avoid GUI clutter
    - Persistent Google Authentication when logged in as guest
    - Make Payment work using Payment APIs (currently using mock payment)
    - Make Ad work using Googleads API

X. Individual Reflection

Bacsain, Jave A.	Being the lead developer on this project taught me so much. I made it a habit to document every change in the repo and codebase, which actually helped me keep track of progress and avoid getting lost later. I also got to practice managing a repo properly by making branches every time I tried to fix something or add a new feature. GitHub's pull requests and issues were super useful for staying organized. Honestly, my favorite part was using git checkout, git merge, git push, and git pull because those commands are just the most satisfying and useful when managing the code. Working on the app also made me realize how important it is to constantly do unit and integration tests instead of just manual testing. I wasted so much time waiting for things to load while testing manually, especially with login and other processes. On top of that, I learned a lot about organizing file structures, keeping naming consistent, and figuring out if new code actually adds value to the project or not. This project also helped me practice UI and UX because I was always thinking about how the end user would want to use the app and how to make it as smooth and intuitive as possible. This project also gave me a real look at what it is like to be a full stack developer. It is frustrating at times, but now I get how much juggling front end and back end work actually involves, and it made me respect the role a lot more.
Parro, Carl Gerald J.	I worked on improving the GUI to make it more user-friendly and fixed minor bugs like non-working buttons. This taught me the importance of usability and attention to detail.
Prestado, Marc Justin N.	Working on this project presented me in an environment that is full of ideas, plans, problems, stress, and difficulties. It brings you into the sense of bringing your ideas to life, realizing how different it is from your mind compared to the real thing. It makes you realize just how difficult it is despite having development tools and engineering strategy. But despite all of the hardships, the feeling of making your code work as intended brings you pure happiness and makes you forget about the hardships from earlier, just then to add another feature and the cycle repeats. Not only did this project make me think deep into software development, it also taught me how to collaborate, be vocal with the development, and the right mindset to figure out the difficulties behind every bug and conflict.

Part II

Information Assurance And Security

Executive Summary

The Video Merger and Uploader is an open-source desktop application designed for streamers and video editors to automate the merging and uploading of video clips and VODs to YouTube. It eliminates repetitive manual tasks, including renaming, editing, and settings configuration, while offering optional automated highlight compilations. The project demonstrates secure role-based access control, cloud integration with Firebase, and AI-assisted video processing via FFmpeg.

Framework Chosen & Rationale

We chose Flet (Python GUI framework) as it is stated in the requirements specification and course instruction. It features its simplicity, cross-platform support, reactive UI, and easy integration with Python backends. It allows modular GUI design with event-driven architecture and reactive controls, ideal for streamers needing responsive, intuitive interfaces.

Implemented Features (baseline + enhancements)

- **Baseline:** Video merging (FFmpeg), YouTube uploads, session management, Firebase-based authentication, role-based access control (5 roles), progress tracking, error handling, multi-page UI.
- **Enhancements:** OAuth2 Google login, advanced RBAC with granular permissions, usage tracking, audit logs, AI-assisted video processing, CSV export of video manifests.

Architecture & Module Overview (with diagram)

- **Presentation Layer:** /app/gui (Flet UI components, multi-page layout, reactive controls)
- **Business Logic:** /access_control (session and role management)
- **Data Access:** /storage (local file handling, Firebase Firestore integration)
- **External Integration:** /uploader (YouTube API, OAuth, FFmpeg processing)

Threat Model & Security Controls

- STRIDE model: Spoofing (OAuth2), Tampering (Firestore rules), Repudiation (audit logs), Info Disclosure (no secrets in code), DoS (rate limits), Elevation of privilege (RBAC multi-layer enforcement).
- Defense in Depth: UI checks → Backend verification → Firebase rules → Audit logging → Rate limiting.
- Input validation: file type/size, email, schema validation.

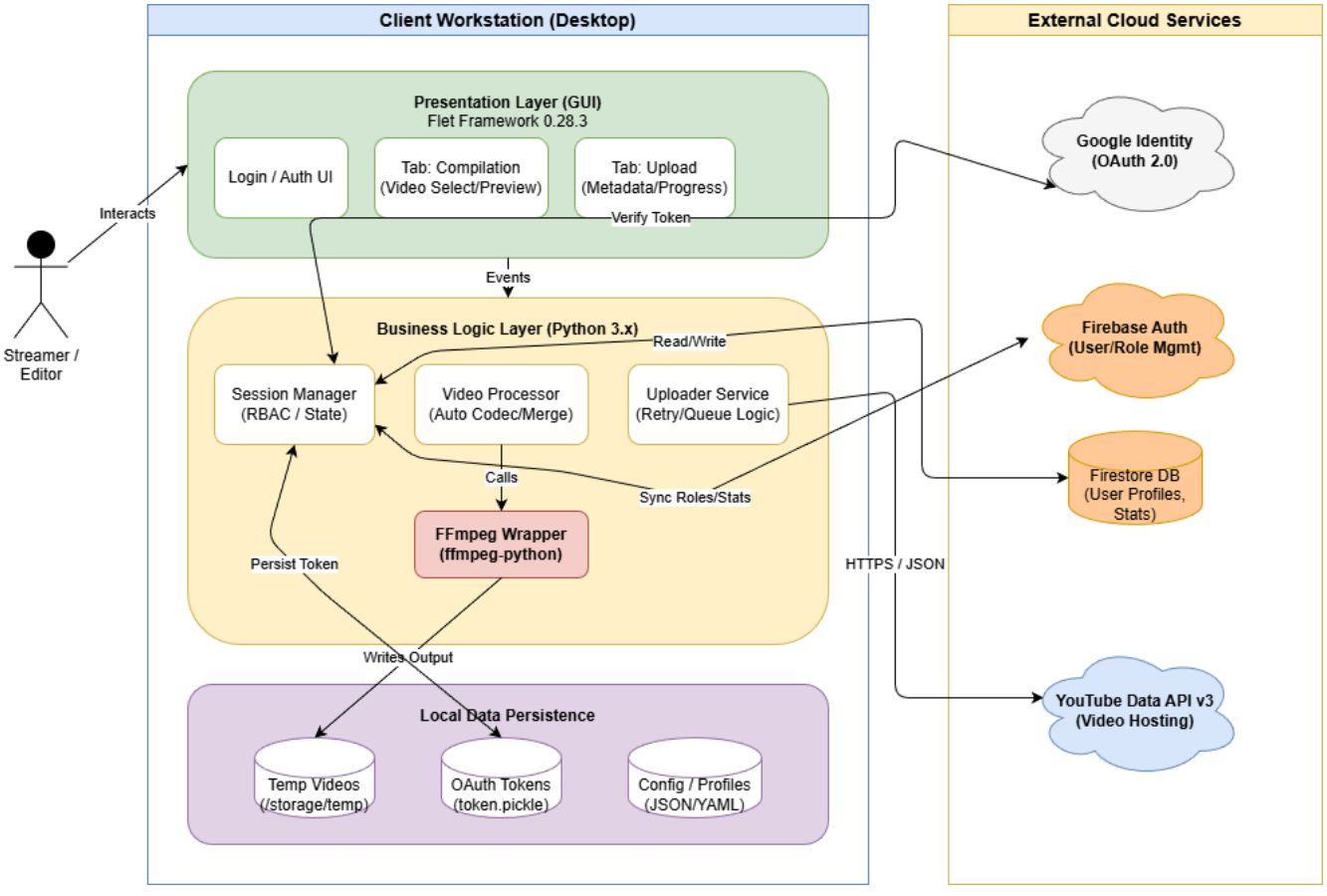
Design Decisions / Trade-offs

- Chose Flet over Electron for lighter dependencies.
- Firebase for cloud persistence, but added local JSON fallback.
- OAuth2 for authentication simplifies password handling but requires careful credential management.

Limitations & Future Work

- Payment system: Implemented as a mock service, due to the high time cost and complexity of integrating live payment gateways (e.g., Stripe/PayPal)
- Ads: Implemented as a mock service, due to; Flet not supporting WebView, and setting up Google ads involves high time cost complexity.
- Audit log filtering and export UI pending full development.
- Potential integration with SQLite for local history and cloud storage (Drive/S3) for backups.

System architecture diagram



- Local JSON/YAML config files: video templates, app preferences

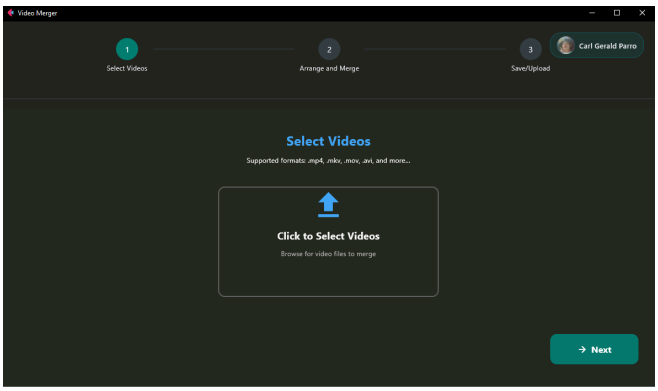
Installation & setup (Flask run / Flet run instructions)

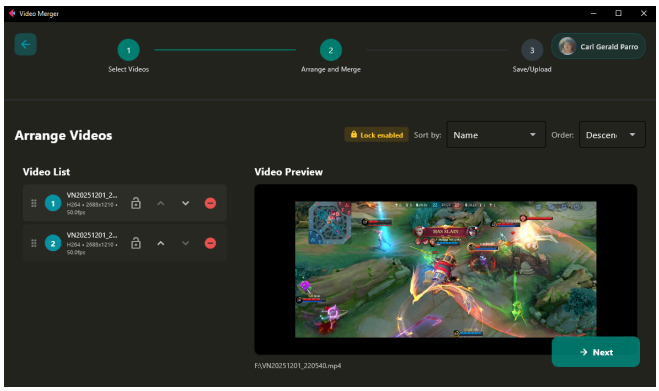
- Python virtual environment setup
- Dependency installation: pip install -r requirements.txt
- API credential placement (configs/ folder)
- Running app: flet run (desktop) or flet run --web

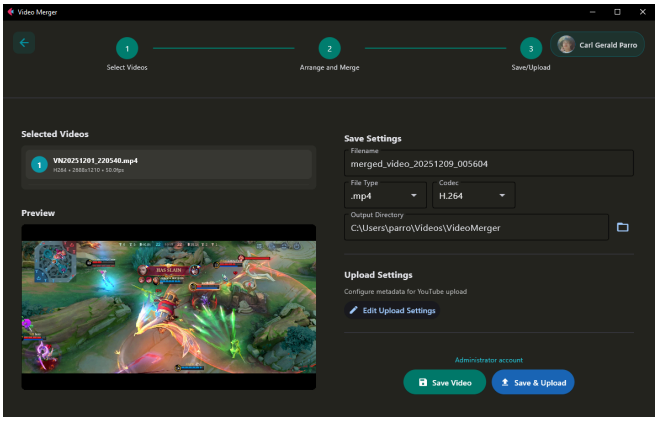
User capabilities

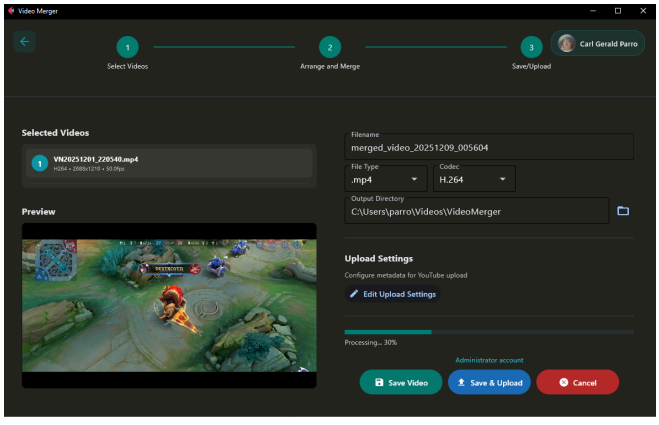
Capabilities / Features	Guest	Free	Premium	Admin
Local Video Merging	✓	✓	✓	✓
YouTube Uploading	✖	✓	✓	✓
Metadata Templates	✖	✓	✓	✓
Ad-Free Experience	✖	✖	✓	✓
Arrangement Tools	✖	5 use per day	✓	✓
Admin Dashboard	✖	✖	✖	✓
Audit Logs and Security	✖	✖	✖	✓

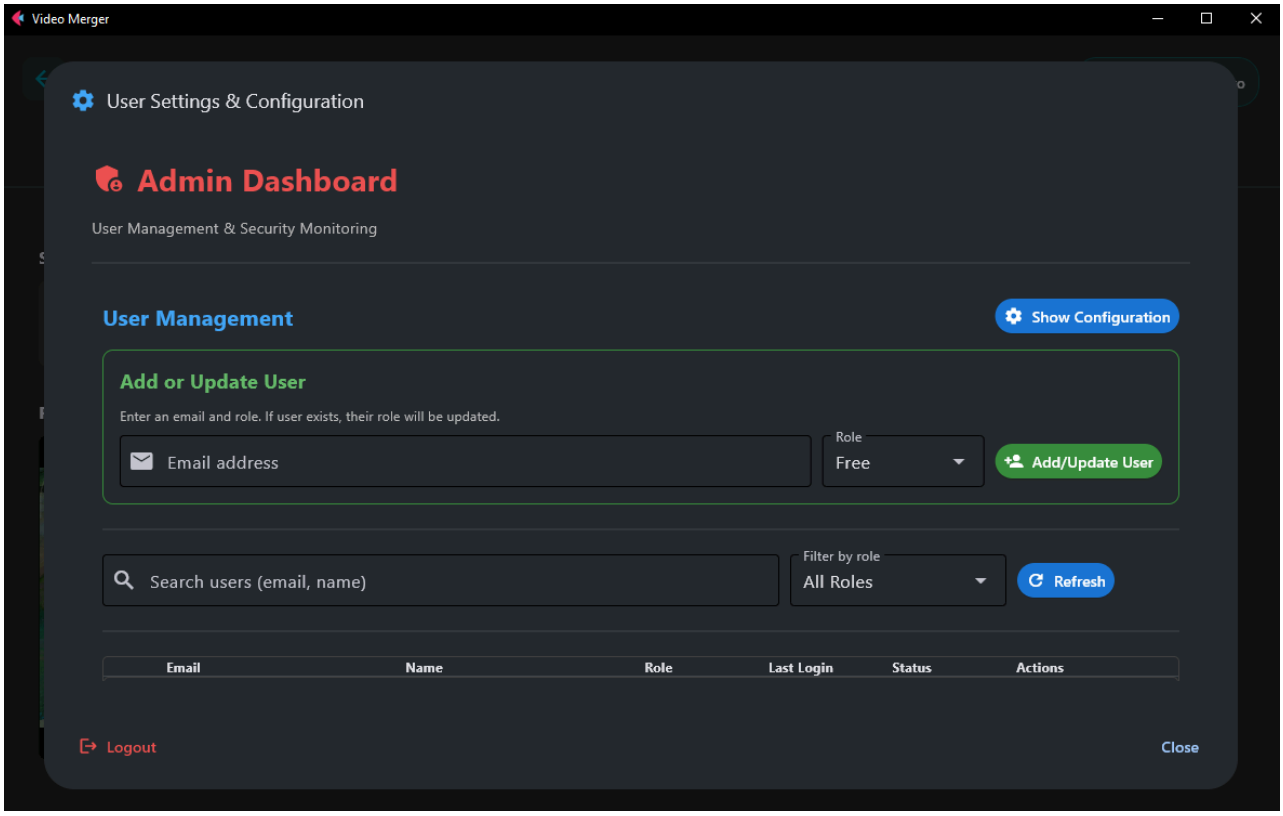
Screenshots / annotated UI











README with quick start, features list, chosen enhancements

requirements.txt (or pyproject.toml if using Poetry)

Inline docstrings for core services