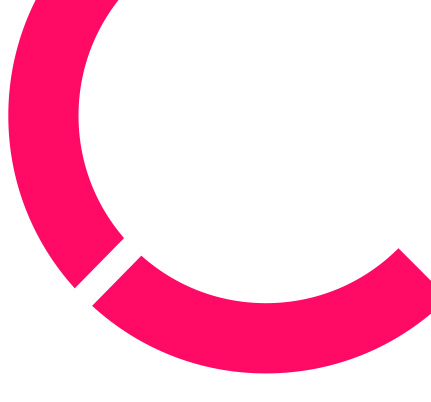
**Rajiv Dhakal, Md Hossain, Komal Gautam, Obioma Nduka**

**DATA WAREHOUSE**

**CENTRIA UNIVERSITY OF APPLIED SCIENCES**

**September 2025**

**CONTENTS**

[1 Abstract 1](#_Toc208348151)

[2 INTRODUCTION 2](#_Toc208348152)

[3 Technologies Used 3](#_Toc208348153)

[3.1 Frontend 3](#_Toc208348154)

[3.2 Backend 4](#_Toc208348155)

[3.3 Database 4](#_Toc208348156)

[3.4 Hosting (Dev) 5](#_Toc208348157)

[4 Key Features 6](#_Toc208348158)

[4.1 Sensor Data Management 6](#_Toc208348159)

[4.2 Sensor Data Analysis 7](#_Toc208348160)

[4.3 Video Upload & Preview 7](#_Toc208348161)

[4.4 Storage Monitoring 8](#_Toc208348162)

[4.5 Export ML Training Dataset 8](#_Toc208348163)

[5 File Structure 9](#_Toc208348164)

[6 Testing & Validation 11](#_Toc208348165)

[7 UI Improvements 12](#_Toc208348166)

[8 Security Considerations 14](#_Toc208348167)

[8.1 Local Access Restriction 14](#_Toc208348168)

[8.2 Validation of File Type and Size 14](#_Toc208348169)

[8.3 Input Validation with FastAPI and Pydantic 14](#_Toc208348170)

[8.4 Directory Isolation for Uploaded Files 15](#_Toc208348171)

[8.5 Error Handling and Logging 15](#_Toc208348172)

[8.6 Export Function Restrictions 15](#_Toc208348173)

[8.7 Session Isolation 15](#_Toc208348174)

[8.8 Code-Level Security Practices 16](#_Toc208348175)

[9 Future Enhancements 17](#_Toc208348176)

[10 CONCLUSION 18](#_Toc208348177)

[REFERENCES 19](#_Toc208348178)

# Abstract

This project focuses on designing and implementing an Admin Dashboard for managing sensor data, video uploads, and storage usage within a local data warehouse system. The dashboard is built with a modern technology stack: Fast API for the backend, SQLite for data storage, and a simple frontend using HTML, CSS, JavaScript, and Chart.js for data visualization.

With this tool, users can monitor and interact with datasets in real time. Key features include entering and viewing sensor data, uploading and previewing videos directly in the browser, and automatically tracking how storage space is being used.

Testing showed that the dashboard performs reliably across its main functions, handling data correctly, displaying charts accurately, and supporting smooth video uploads and previews. The result is a user-friendly interface that helps prepare and manage datasets for machine learning tasks. Looking ahead, the dashboard can be enhanced with features like cloud deployment, user authentication, chart exporting, and integration with external sensor APIs, making it a flexible foundation for future development.

# INTRODUCTION

This project was given by the university to develop an Admin Dashboard that can manage sensor data, video files, and storage usage simply and effectively. The main goal of the project is to create a web-based system where data can be entered, viewed, and monitored easily, while also preparing the base for future use in machine learning and data analysis.

The Admin Dashboard allows users to add and view sensor information, upload and preview videos, and check how much storage is being used. It also includes charts and tables to make the data easier to understand. The system is built using FastAPI for the backend, SQLite as the database, and HTML, CSS, and JavaScript for the frontend. Chart.js is used to display the data visually.

Although the project is running locally during development, it includes some important security practices, such as checking file types, validating user inputs, and keeping uploaded files in a safe location. These features make the system more reliable and prepare it for possible future deployment in a public environment.

# Technologies Used

|  |  |
| --- | --- |
| Component | Technology |
| Frontend | HTML, CSS, JavaScript, Chart.js |
| |  |  | | --- | --- | | Backend |  | | FastAPI (Python) |
| Database | SQLite |
| Hosting (Dev) | Localhost via Uvicorn |

## Frontend

The frontend of the Admin Dashboard is the part that users see and interact with. It was developed using HTML, CSS, and JavaScript. These technologies were chosen because they are lightweight, easy to use, and suitable for building simple web applications.

HTML is used to structure the content of the dashboard, such as forms, tables, and charts.

CSS is applied to style the interface, making it more user-friendly and visually clear. This includes adjusting layouts, colors, fonts, and spacing to ensure a cohesive and visually appealing design. JavaScript is used to add interactivity, such as updating charts, handling button clicks, and previewing uploaded videos without requiring page refreshes.

For visualizing data, the frontend also utilizes Chart.js, a JavaScript library that facilitates the creation of interactive charts. With this, the system can display sensor trends, pie charts of data distribution, and storage usage clearly and understandably.

## Backend

The backend of the Admin Dashboard was developed using FastAPI, which is a modern and lightweight Python web framework. It is responsible for handling the system's main logic, processing requests from the frontend, and communicating with the database. The backend manages tasks such as submitting and deleting sensor data, uploading and previewing video files, and monitoring the total storage usage of the system.

In this project, the backend provides API endpoints that the frontend can call whenever the user interacts with the system. For example, when a new sensor entry is added or a video is uploaded, the request is sent to the backend, which processes it and updates the database or file storage accordingly. The backend also calculates real-time statistics, such as storage usage, and sends the results back to the frontend for display. The application runs on Uvicorn, a high-performance server for Python applications, which makes the backend fast and reliable during local development.

## Database

The Admin Dashboard uses SQLite as its database system. SQLite was chosen because it is lightweight, easy to configure, and does not require a separate server, which makes it highly suitable for local development and smaller projects. In this project, the database is responsible for storing sensor data, such as ID, time range, value, and unit, as well as records of uploaded video files and their locations. It also supports storage monitoring by keeping track of data that is later used to calculate and display storage usage. Since SQLite stores all information in a single file, it is simple to manage, portable, and reliable. At the same time, it provides the advantages of SQL queries for handling structured data. This ensures that all information entered through the dashboard is saved securely and can be retrieved efficiently for visualization, management, or export purposes.

## Hosting (Dev)

During development, the Admin Dashboard is hosted locally using Uvicorn, which is a fast ASGI server for running FastAPI applications. Local hosting means the system runs only on the developer’s own computer and can be accessed through a web browser. This setup is useful for testing and development because it is safe. By running locally, the system avoids risks from unauthorized users and allows quick debugging during development.

Although the project is currently hosted on localhost, it has been designed so that it can later be deployed to a cloud platform (e.g., Vercel, Render, or AWS) for wider access if needed.

# Key Features

## Sensor Data Management

There is a milestone feature in the Data Warehouse admin Dashboard. The Sensor data management provides for handling sensor information with precision and control. Users can entry new sensor data by specifying key attributes for example, Sensor ID, start time, end time, Value and unit. To maintain data clean and conserve storage, users can delete old

A screenshot of a computer

AI-generated content may be incorrect.Pic 1: Submit Sensor Entry and delete old Sensor Data

data sensor data by timestamp. Delete function enables selective removal of older data without affecting other data.

A screenshot of a computer

AI-generated content may be incorrect.Submitted data stored in database and CSV file. All stored data are visible in Admin dashboard as table. The tabular format allows user to scan quickly, compare sensor values and unit. The table is optimized for clarity and performance, even when managing large datasets.

Pic 2: Sensor data table in Admin dashboard.

## Sensor Data Analysis

The sensor data analysis provides visual insight into the behaviour for sensor activities. This module is designed to help users interpret quickly and efficiently. The pie chart displays the proportional distribution for different type of sensors for instance, temperature, humidity and other sensors. The Pie chart allows to understand sensors contribution to most data pool. The Line chart illustrates the daily trend of the values for specific sensor. User can choose for the specific sensor from the dropdown Manu and the line chart dynamically update over the time.

A white background with black dots

AI-generated content may be incorrect.

Pic 3: Pie chart and line chart in the Sensor Analysis module

## Video Upload & Preview

A screenshot of a computer

AI-generated content may be incorrect.The Video upload and preview feature enables users to control or manage directly form dashboard. It offers a seamless handling experience for uploading, previewing and deleting individual video file. Each video file includes delete option. Uploaded video stores in the

Pic 4: Video Upload and preview module

Database and user can see those as tabular form in the admin dashboard. By clicking on the watch button, video will play in new browser tab. In the action column of the table, user can delete video individually.

## Storage Monitoring

The Storage Monitoring feature provides real-time view for the system usage. It displays live calculation of total used storage in megabytes (MB). It updates dynamically during add or delete data. The Pie chart illustrates the proportional distribution of storage across categories.

There is two more dynamic Bar chart Sensor count and Video count by size which are displaying individually by their categories. Both bar chart displays absolute values for each category. The Storage Monitoring feature ensures transparency and control over data infrastructure.

A graph with blue squares

AI-generated content may be incorrect.

Pic 5: Storage Overview

## Export ML Training Dataset

The Export ML Training Dataset is a feature that enables the pre-processing of sensor data for machine learning purposes. User can select a particular sensor and define start time and end time, after that the data can be pulled based on the time in the targeted manner. If the sensor data is selected, the system consolidates the into model-trainable format like excel. This competence enables predictive analytics, anomaly detection, and performance forecasting.

A screenshot of a computer

AI-generated content may be incorrect.

Pic 5: Export ML Training Dataset

# A screenshot of a computer AI-generated content may be incorrect.File Structure

A screen shot of a computer

AI-generated content may be incorrect.

# Testing & Validation

* Verified sensor data submission and deletion
* Confirmed video upload and preview functionality
* Validated storage breakdown accuracy via FastAPI logs
* Ensured charts render correctly and update live

All main functionalities were thoroughly tested to ensure performance and stability. Sensor upload and delete were tested for integrity and correctness. Video upload and in-line preview were made to function seamlessly on supported media types. Storage breakdown calculation check was tested for compliance with actual consumption via FastAPI logs. Chart components like pie, bar, and line charts were tested for correct rendering and real-time refresh, ensuring a responsive and interactive user experience throughout the dashboard.

# UI Improvements

There were also some enhancements made to make the visual design and usability of the dashboard more streamlined. Spacing between tables and graphs was reduced to create a tighter and more streamlined appearance. Pie graphs were aligned on the left to create a better visual balance and transitions between sections. The heights of charts were reduced without sacrificing readability to maintain the space-efficiency of the dashboard. Duplicate canvas elements were discovered and eliminated in an effort to prevent rendering issues and maintain code cleanliness. Additionally, tables and form controls were formatted for readability and consistency to promote overall usability and create a clean, professional-looking interface

A screenshot of a computer

AI-generated content may be incorrect.

Pic 6: Initial UI

A screenshot of a computer

AI-generated content may be incorrect.

Pic 7: Updated UI

# Security Considerations

In the implementation of the Admin Dashboard for managing sensor data, video uploading, and machine learning data, multiple pragmatic security measures were incorporated to ensure secure and error-free functioning in a local environment. Despite the fact that the system is executed in a local server, the following security protocols were implemented to ensure data integrity, prevent misuse, and be prepared for any future public release.

## Local Access Restriction

The dashboard is locally hosted and accessible only within a safe network. This automatically limits the location vulnerable to harm. Steer clear of public hosting while in development also ensures that the possibility of unauthorized access, brute force, and data exposure is drastically reduced.

## Validation of File Type and Size

For preventing misuse of video uploading, validation was made very strict:

- Only `.mp4` are permitted.

- The size of the file is limited to a particular threshold so that the server is not overwhelmed by it or even malicious code.

This ensures that the system processes only relevant and safe content.

## Input Validation with FastAPI and Pydantic

All form inputs and API requests are validated using FastAPI’s built-in support for Pydantic models. This enforces strict data types and formats, reducing the risk of injection attacks or malformed data entries. For example, sensor data submissions are checked for correct structure before being stored or visualized.

## Directory Isolation for Uploaded Files

Uploaded video files are stored in a dedicated directory whose permissions for access are limited. This prevents the files from being run accidentally or exposed and ensures that the file system remains clean and secure.

## Error Handling and Logging

Basic error management mechanisms were implemented to prevent the system from crashing due to unexpected inputs or activities. Logs are generated for significant operations like file uploads, exporting data, and the generation of charts. These logs help track usage and identify potential misuse or bugs.

## Export Function Restrictions

The machine learning dataset export capability is limited to specific formats (e.g., CSV) and includes file protection against unauthorized access. Pre-approved datasets are allowed for download, and download activity is monitored to prevent excessive or bulk downloads.

## Session Isolation

User authentication is not present yet, but the dashboard is such that it will be able to run in isolated sessions. Every user action is stateless, and no sensitive information is cached between sessions. This minimizes the attack vector of session hijacking or exposure of data.

## Code-Level Security Practices

Secure coding practices were adhered to all over the codebase:

- Steering clear of hardcoded credentials or secrets.

- Configuration using environment variables.

- Keeping dependencies updated to patch known vulnerabilities.

These security controls provide a solid foundation for the existing use case of the Admin Dashboard. As the project progresses toward public release, more advanced controls such as user authentication, HTTPS encryption, and role-based access control will be included to meet more stringent security needs. The current implementation shows a fair balance between usability and safeguarding, keeping data secure and enabling proper analysis and management.

# Future Enhancements

To bring the dashboard to the next level of functionality and scalability, several enhancements are on the future development roadmap. These include:

* User Authentication & Roles: Incorporating secure login and role-based access control to selectively grant permissions to different types of users.
* Advanced Filtering & Search: Implementing dynamic filters and keyword search to improve data navigation in charts and tables.
* Real-Time Sensor Streaming: Incorporating WebSocket or MQTT protocol support to provide live sensor data updates without page refresh.
* Video Analytics Integration: Automating video tagging, motion detection, and metadata extraction for videos uploaded.
* Export Options Expansion: Enabling multiple formats (CSV, JSON, XLSX) for exporting sensor and video data.
* Mobile Optimization: Streamlining layout and controls for effortless use on tablets and smartphones.
* Alert System: Incorporating threshold-based notifications and alerts for irregular sensor readings or storage capacity.

These enhancements are intended to enhance usability, performance, and level of analysis so that the dashboard remains agile in response to evolving operational needs and future technological integrations.

# CONCLUSION

The Data Warehouse project set out to design and implement an Admin Dashboard for managing sensor data, video uploads, and storage usage in a structured and efficient way. The outcome of the work is a functioning system that integrates a modern backend using FastAPI, a lightweight SQLite database, and a clear frontend with HTML, CSS, JavaScript, and Chart.js. Together, these technologies created a modular solution that allows data to be ingested, stored, visualized, and prepared for further use, particularly in machine learning contexts.

The system successfully supports core functionalities such as sensor data submission and deletion, uploading and previewing videos directly in the browser, and tracking storage usage in real time. Data visualization through line charts, pie charts, and bar charts enhances usability and provides users with immediate insight into both sensor activity and system storage distribution. Furthermore, the dashboard offers dataset export features, enabling structured data to be transformed into machine-learning-ready formats. These outcomes demonstrate that the project achieved its primary goal of building a versatile, user-friendly data management environment.

During development, several technical challenges were encountered and resolved. Ensuring secure file uploads required restricting file formats and sizes, while input validation was handled through FastAPI and Pydantic to enforce correct data types. Storage usage calculations initially presented inconsistencies, but careful validation against backend logs ensured accurate results. Layout adjustments were made to improve the user interface, reducing unnecessary spacing and eliminating duplicate chart elements. These challenges and their solutions not only improved the quality of the system but also strengthened the team’s skills in problem-solving, debugging, and applying secure coding practices.

Security was a key consideration throughout the project. Even though the dashboard is hosted locally during development, strict measures were applied, including file validation, input checks, directory isolation, and error logging. By avoiding hardcoded credentials and maintaining dependencies, the project ensured a safe foundation for future deployment. While advanced features such as authentication and HTTPS encryption were not yet implemented, the system has been designed to accommodate them in the next stages of development.

The project also has clear potential for long-term use and extension. It provides a solid base for future student projects, coursework, and experiments in artificial intelligence. The modular architecture allows for scaling, cloud deployment, and integration with external sensor APIs, making the system relevant beyond its immediate scope. It bridges academic learning with practical application, ensuring that future developers can build upon a strong and flexible platform.

In conclusion, the Data Warehouse project has met its objectives by delivering a secure, functional, and extendable Admin Dashboard. It combines usability with robust technical design, demonstrates effective solutions to real development challenges, and provides

# REFERENCES

Chart.js – JavaScript library used for rendering pie, bar, and line charts. <https://www.chartjs.org>

FastAPI – Backend framework used for API development and data handling. <https://fastapi.tiangolo.com>

HTML5 <video> Tag – Used for inline video preview functionality. <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/video>

CSS Flexbox & Grid – Applied for responsive layout and UI alignment. <https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Flexible_Box_Layout>

JavaScript ES6+ – Used for dynamic chart updates, form handling, and interactivity.

Browser DevTools & Console Logs – Utilized for testing, debugging, and validation.