A document with writing on it

AI-generated content may be incorrect.

**Face Recognition Attendance System for University Students**

**Company and Problem Description**

Universities face high student absenteeism, which affects academic performance and increases dropout rates. Traditional attendance methods like manual registers are time-consuming, error-prone, and allow proxy attendance. Lecturers spend valuable class time marking attendance instead of teaching, especially in large classes.

A Face Recognition Attendance System addresses these problems by automatically recognizing students as they enter the lecture hall. It records attendance in real time, stores data securely, and eliminates human error.

Key Benefits:

* Saves class time and reduces manual work.
* Prevents proxy attendance and fraud.
* Provides accurate, tamper-proof attendance records.
* Allows lecturers to monitor trends and allocate marks fairly.
* Helps universities intervene early for students at risk of poor attendance.

This system ensures transparency, efficiency, and improved student engagement while supporting better academic outcomes.

**Process Model**

The system will follow the Agile Model because it is iterative, flexible, and user focused. Agile allows incremental development, continuous testing, and stakeholder feedback.

Steps:

1. Requirement Gathering

* Surveys and workshops with students, lecturers, and administrators.
* Identify needs: automatic recognition, real-time reporting, secure data storage.

2. Defining Features

* Face recognition and automatic attendance marking.
* Prevention of proxy attendance.
* Attendance database management.
* Lecturer dashboard for monitoring.

3. Sprint Planning

Short sprints for incremental development:

Sprint 1: Face recognition.

Sprint 2: Database integration.

Sprint 3: Dashboard and reporting.

4. System Design

* Components: camera module, AI/ML face recognition engine, secure database, lecturer dashboard.

5. Development and Testing

* Incremental implementation per sprint.
* Testing: unit, integration, user acceptance, and security tests.

6. Deployment

* Pilot rollout with feedback collection.
* Full-scale deployment after refinement.

7. Maintenance

* Continuous improvement, performance optimization, and feature updates (e.g., mobile access).

Benefits of Agile:

* Early delivery of functional features.
* Quick adjustments based on feedback.
* Ensures system reliability, user satisfaction, and long-term success.

**Requirements elicitation**

This chapter discusses the foundational requirements for the Face Recognition Attendance System (FRAS), based on a methodological survey of the project outline and current issues identified in academic research. The focus here is to go beyond a list of features to build a structure that captures key functional needs, vital quality attributes, and important ethical aspects from the very beginning.

1. User Requirements: Stakeholder Perspectives

User requirements capture the higher-level needs of all user stakeholders within natural language.

* For Lecturers:

1. The FRAS must automatically record attendance without needing rollcall, so they can save this time in their lectures and eliminate instances of proxy attendance.
2. Lecturers need a way to generate and export attendance reports to track engagement over periods of time for their modules.
3. A mechanism to configure how attendance contributes to the "Duly Performed" mark as part of the assessment of students, is an important assessment mechanism in academic.

* For Students:

1. Students should be able to easily view their attendance history for all modules to have a chance to self-scan and see if they are on track.
2. The attendance process must be as passive and unobtrusive as possible, requiring no physical interactions by the student aside from having taken a seat at the spot in the lecture room.

* For University Administration:

1. The system must allow the University to collect aggregated, anonymized data about attendance trends to identify faculty wide issues and assist in student retention efforts.
2. Attendance records must be incapable of being corrupted and auditable from an integrity standpoint as an academic institution. Each entry or modification of attendance needs an auditable log of the information recorded.
3. System Requirements: Architectural Scope

System Requirements define the top-level capabilities and boundaries in the solution.

1. The FRAS system shall provide real time facial recognition in a lecture room environment via the installed camera hardware.
2. The FRAS shall keep in a secure, encrypted database only the student face features (embeddings) not raw images to mitigate privacy risk. A secure, encrypted database of student facial features (embeddings), not raw images, will be stored by the system to mitigate privacy risk.
3. The system must interface with the university’s existing Student Information System (SIS) to automatically synchronize timetables and registrations to modules.
4. A web-accessed dashboard will be provided for all stakeholders to allow access to relevant functions and data.
5. Functional Requirements: System Functionality.

Functional requirements (FRs) provide a detailed specification of what the system must do and are often expressed in the form "shall".

* FR1: Attendance Capture.

The system should identify faces from a video stream, ensure liveness to prevent spoofing, matching the face to students enrolled in that class with a confidence level of at least 98%, and create a timestamped attendance record.

* FR2: Data Management & Reporting.

The system shall provide a means for lecturers to filter attendance records by date, student, and module, and export records. The system shall calculate attendance percentage for everyone.

* FR3: Authentication & Authorization.

The system shall authenticate with university credentials to log in to the system, and must employ Role Based Access Control (RBAC), to limit users to see data only relevant to their role

* FR4: Audit & Integrity.

The system shall preserve a tamper-proof audit trail that logs all occasions of an attendance record being created, edited or deleted. The audit trail shall log user, time, and reason for editing.

1. Non-Functional Requirements: Quality Attributes.

Non-functional requirements (NFRs) describe how well the system performs its operations.

* NFR1: Performance & Accuracy.

Attendance must be logged within 3 seconds of a student being visible. The system should have a False Acceptance Rate (FAR) of less than 0.001%, and a high True Acceptance Rate (TAR).

* NFR2: Security & Privacy.

All biometric data must be encrypted both at-rest and in-transit. The system should be designed to be compliant with GDPR/POPIA, to include the principles of data minimization and purpose limitation.

* NFR3: Reliability.

The system should be available during lectures at a 99% rate.

* NFR4: Usability.

The lectures and administrations dashboard interface should be intuitive enough to use with minimum training. The student experience should be entirely passive.

Face Recognition Attendance System for Students

1. Introduction

Universities often face challenges with

student absenteeism, which contributes to poor performance and dropout rates.

Traditional methods of marking attendance are manual, time-consuming, and prone

to manipulation. A Face Recognition Attendance System automates this process by

leveraging biometric technology to detect and recognize students’ faces in

real-time.

This project will ensure:

- Automatic attendance marking without human intervention.

- Secure storage of attendance records (tamper-proof).

- Lecturer access to attendance records and ability to allocate marks.

- Improved efficiency and reduction in student absenteeism.

2. Project Technology

2.1 Software Technologies

- Programming Languages: Python, JavaScript

- Frameworks & Libraries: OpenCV, TensorFlow/Keras, Flask/Django, React.js

- Database: PostgreSQL/MySQL

- APIs: REST APIs for system integration, Face API

2.2 Hardware Technologies

- High-definition/IR cameras

- Local or cloud servers

- Lecturer/Admin devices (laptops or smartphones)

2.3 Security

- Biometric encryption

- Authentication & role-based access

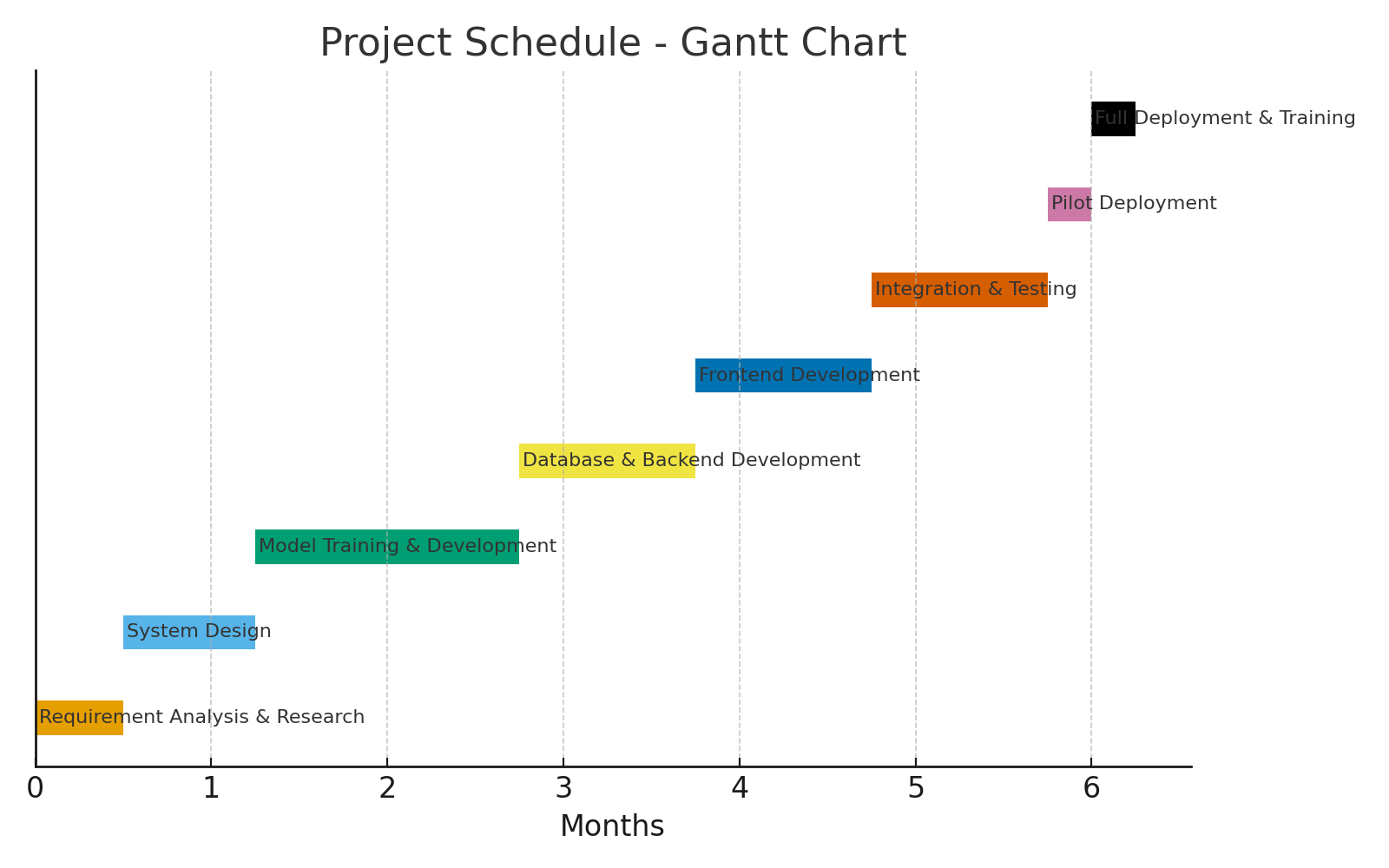
- Compliance with POPIA/GDPR regulations

3.Risks

| **Risk** | **Impact** | **Likelihood** | **Mitigation Strategy** |
| --- | --- | --- | --- |
|  |  |  |  |
| Poor camera performance in dim classrooms | High | Medium | Use IR cameras with low-light support |
| Hardware/software cost constraints | Medium | High | Develop in phases; seek funding/grants |
| Data privacy concerns (biometric misuse) | High | High | Encrypt face data; comply with POPIA/GDPR |
| False recognition (twins, masks, glasses) | Medium | Medium | Improve model training with large datasets |
| System downtime during lectures | Medium | Low | Cloud backup and redundant servers |
| Resistance from students | Low | Low | Awareness sessions and transparent data policies |

Timeline

The project duration is estimated at 6 months with the following phases:



Project Costs (Estimated)

|  |  |
| --- | --- |
| Item | Cost (ZAR) |
| High-resolution cameras (10 × R2,500) | R25,000 |
| Local server setup / cloud subscription | R20,000 |
| Software development (team of 4 devs, 6 months) | R600,000 |
| Database licensing & hosting | R10,000 |
| Security & encryption tools | R5,000 |
| Training & documentation | R5,000 |
| Maintenance & support (1 year) | R30,000 |
| Total Project Cost | R695,000 |

**Tools and environment**

**Web/mobile platform**

Face Recognition attendance system is architected as a responsive web application to ensure maximum accessibility and cross-platform compatibility. This approach allows authorized users, lecturers and students to access the system from a wide range of devices, including desktop PCs, laptops, and tablets, via a standard web browser. The interface will provide functionalities for real-time attendance viewing, generating detailed reports, and managing student profiles

**Programming Languages**

Face Recognition attendance system will be developed in HTML, CSS and JavaScript, chosen for its robust ecosystem of libraries ideal for data processing and machine learning tasks. Face Recognition For performing real-time face detection and recognition. As the web framework to handle routing, requests, and server-side logic. Flask offers flexibility for a focused application, while Django provides a more batteries-included approach.

SQL Alchemy as an Object-Relational Mapping (ORM) tool to abstract and simplify all database interactions.

**Database**

Face Recognition attendance will utilize a **SQL-based relational database management system (RDBMS)** such as MongoDB. This is essential for maintaining structured data integrity across related tables storing information on students, lecturers, courses, attendance records, and facial encodings. The database will enforce strict relationships and constraints to prevent data anomalies. All data will be stored securely with access controlled by robust authentication and authorization mechanisms, ensuring that only authorized personnel can view or modify sensitive attendance records.

**Operating Environment**

The system is designed for deployment in a standard server environment common to universities. A cloud-based Infrastructure-as-a-Service (IaaS) platform like **Microsoft Azure** or **Amazon AWS** is proposed for hosting. This provides scalable computing resources, high availability, and enhanced security measures to protect sensitive biometric data. The client-side requires only a modern web browser with a camera, making it easy to deploy in lecture halls, tutorial rooms, and computer labs without the need for specialized hardware on user devices.

**Integrated development environment**

**Visual Studio Code (VS Code)** will be the primary IDE for development. It offers excellent support for Python through extensions, providing features like intelligent code completion, debugging, and an integrated terminal. Extensions for database management (e.g., SQLite Viewer) will allow developers to inspect and query the database directly within the IDE. VS Code’s built-in Git integration and extensive plugin ecosystem make it a productive and efficient environment for building the Face Recognition attendance system.

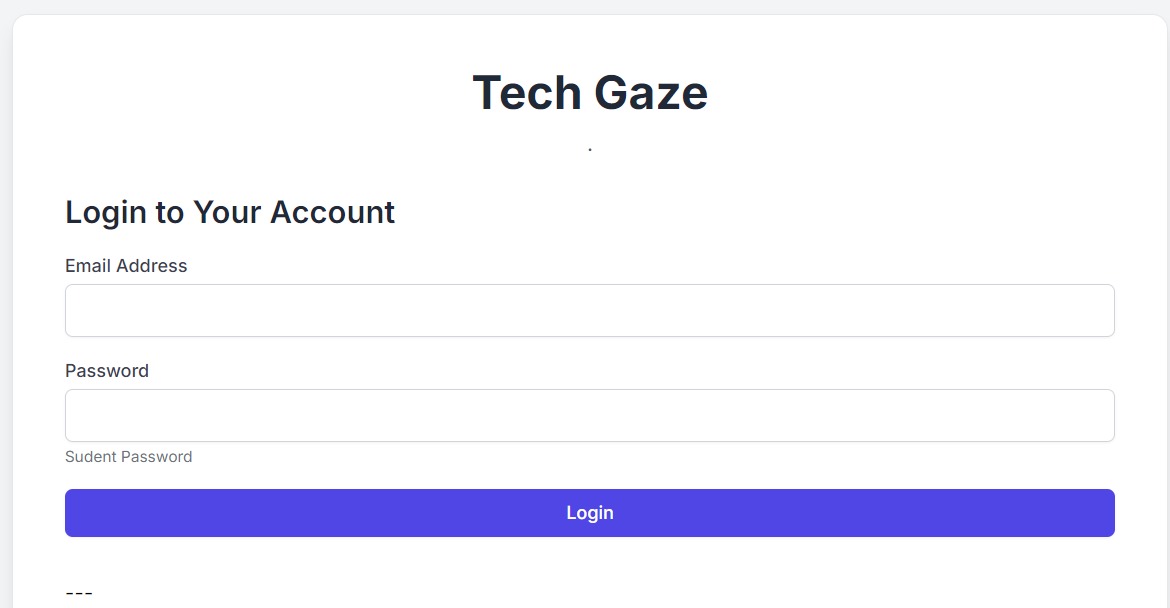
Software Design

WIREFRAME DIAGRAM

A screen shot of a device

AI-generated content may be incorrect.

* It shows a **camera interface** that scans students’ faces in real time as they enter class.
* When a face is recognized, the system automatically marks that student as present in the attendance database.
* This eliminates manual signing and reduces the chance of errors or proxy attendance.



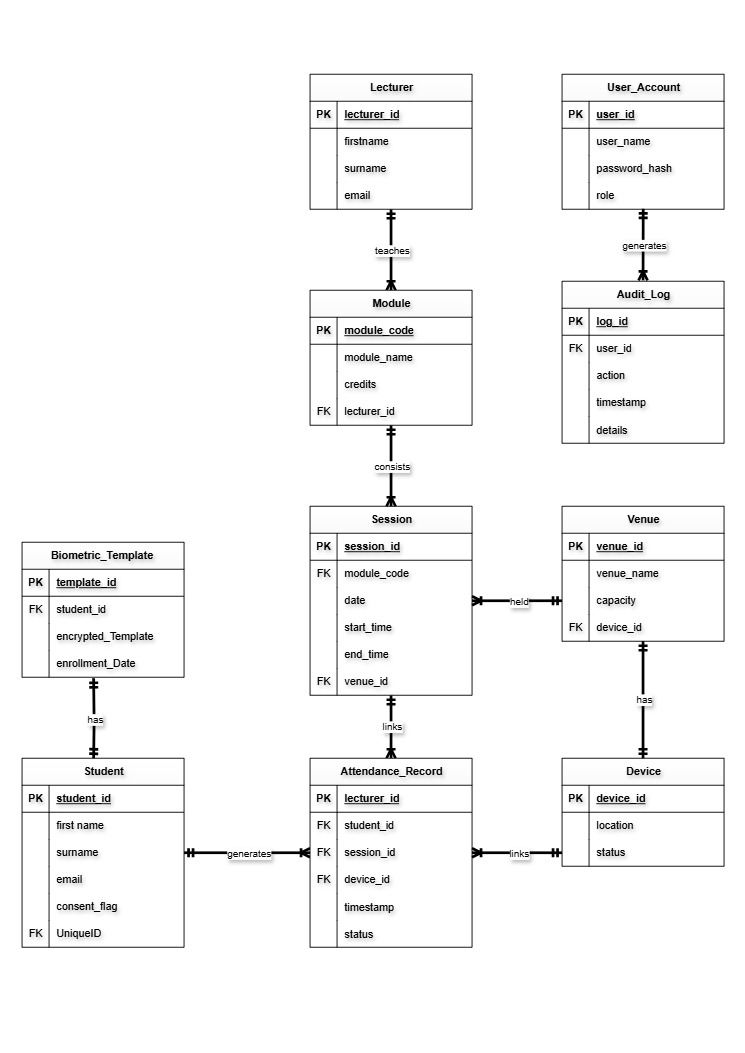
Students can log in

A screenshot of a computer

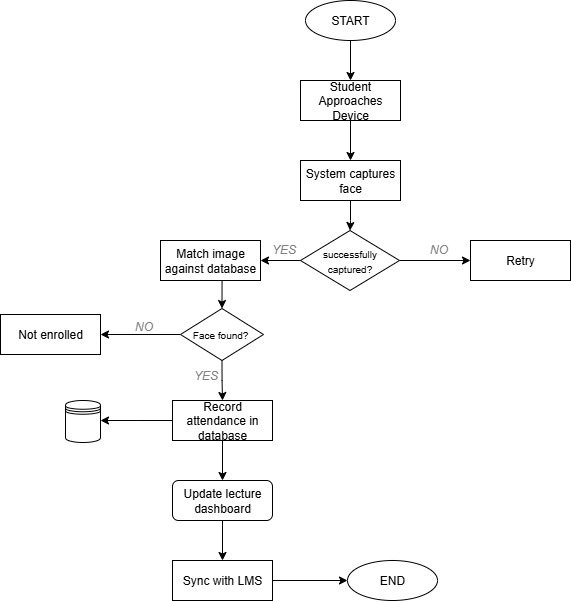
AI-generated content may be incorrect.

* It includes **login fields** for username and password to ensure only authorized users can access the system.
* After login, the user is directed to a **dashboard** where they can view class lists, attendance records, and system notifications.
* This design helps secure the system and gives quick access to important attendance data.

Entity Relationship Diagram



**Sequence Diagram**

****

**Software construction**

**Backend Functions**

Backend Functions

All Face API processing (detection and descriptor extraction) occurs server-side through Node.js, thus allowing us to carry out operations on frames with faces without the need of a browser.

•Face detection and encoding: Server Side loads the Face API models and applies faceapi. detectSingleFace(input).withFaceLandmarks().withFaceDescriptor() to the incoming video frames to generate the 128-D face descriptor vectors.

•Descriptor comparison: This new descriptor is then compared against the descriptors stored in the MongoDB. Using new faceapi.FaceMatcher(reference Descriptors), the backend calls faceMatcher.findBestMatch(query Descriptor) for an adequate matching. In the scenario where the best match distance is within tolerance, the corresponding student shall be identified.

•Attendance logging: When a student has been matched correctly, the system calls log Attendance(student, lectureId), which inserts a record into the SQL Attendance Log table with the current date/time.

The Face API pipeline (load models, detect face, compute descriptor, match) handles the entire facial recognition logic.

**Database Structure**

Structured data are stored within relational tables in an SQL database, while biometric data (face descriptors) are maintained in the MongoDB collections. Below is an ERD that illustrates this hybrid schema: student and course entities remain in SQL, and face encodings go into a MongoDB collection. In the ER modelling, SQL uses traditional tables and columns, while Mongo uses collections and documents.

Students (SQL) are enrolled in Modules/Lectures, which are recorded in Attendance Log (SQL), and the attendance of each student at each lecture is recorded. Facial encodings are stored as documents within the MongoDB collection called Face Descriptors.

• Students (SQL table): student (PK), name, email, etc. Each student may have one or more face descriptors within Mongo.

• Face Descriptors (MongoDB collection): Stores each document as: { student: …, descriptor: [ ...128 floats... ] }. This collection stores 128-dimensional Face API embeddings corresponding to each enrolled student.

• Modules/Courses (SQL): modulini (PK), module\_name, module\_code, etc.

• Lectures (SQL): lectured (PK), module\_id (FK), date/time, location, etc.

• Attendance Log (SQL table): log\_id (PK), student (FK → Students), lecture\_id (FK → Lectures), timestamp (date/time) – for all attendance events.

This ERD reflects a hybrid database: structured entities (Students, Lectures, Attendance\_Log) use SQL tables, while biometric data (face descriptors) resides in MongoDB. In the ERD, the FaceDescriptors collection references student\_id from the students table, illustrating the embedding of face encoding within the overall schema.

**GUI (Frontend)**

**HTML Structure**

The frontend UI is split into semantic components:

•Login Page: Contains a <form> with <input> fields for username and password and a submit button. There’s a header with a logo for branding. No unnecessary elements are shown.

•Dashboard: Main page after login, including header with system title, and content area. Key UI elements consist of video preview box where live camera feed is shown for face capture, status indicators, and lists or tables of current modules and attendance.

•Report Generation Page: Contains Filters and a prominent “Generate Report” <button> on top, with results shown below.

•Video Preview Element: An HTML <video> or <canvas> element present on the dashboard to display the captured camera stream. It is used to take pictures or video frames of student's faces for recognition.

Each component is placed within <div> containers having a clear ID or class for styling; the layout follows best practices: header/navigations, content sections, and responsive wrappers.

**CSS Styling**

CSS styling will ensure a clean yet responsive interface. We use CSS Grid/Flexbox along with media queries to change the layout for different devices. Some style guidelines include:

• Responsiveness: Use @media queries to change the layout for a mobile screen. For example, set breakpoints such that your tables collapse into lists on a phone, and buttons or icons are resized appropriately.

• Colour Schemes and Layout: Choose palette colours defined in CSS variables. Layout sections—header, content, sidebar—are set up with grid or flex containers and given spacing. Navigation bars and buttons pick their own colours from the theme.

• Element Styling: Classes for UI elements are defined, e.g., .btn-primary for the blue 'Generate Report' button,. status-present for green, and. status-absent for the color red. Headings are bold; the content text is legible (font sizes scale with the viewport).

• Video Box: #video-preview is styled as a fixed aspect ratio box centered on the page. For smaller screens, the box provides fluid resizing to ensure full visibility.

All these CSS rules make the interface look great on any desktop, tablet, or phone. More so, the use of @media specifies different styles depending on screen width, as documented for responsive design.