

# **Docket Management System -** **Documentation**

## **1. INTRODUCTION**

### **1.1. Project Purpose and Scope**

The Docket Management System is a web-based application designed to streamline the process of issuing and verifying examination dockets for students at an educational institution. The system aims to replace the manual, paper-based process, reduce administrative overhead, and improve efficiency.

**Key objectives of the system include:**

- Providing students with a self-service portal to view their eligibility and generate exam dockets.
- Enabling administrators to manage student data, control docket access based on financial clearance, and set active examination periods.
- Implementing a secure and reliable method for verifying the authenticity of dockets using QR codes.
- Offering offline verification capabilities for university staff, ensuring functionality even in areas with poor or no internet connectivity.

The scope of this project covers the entire lifecycle of a docket, from student eligibility checks to final verification by an invigilator.

### **1.2. Target Audience**

This document is intended for:

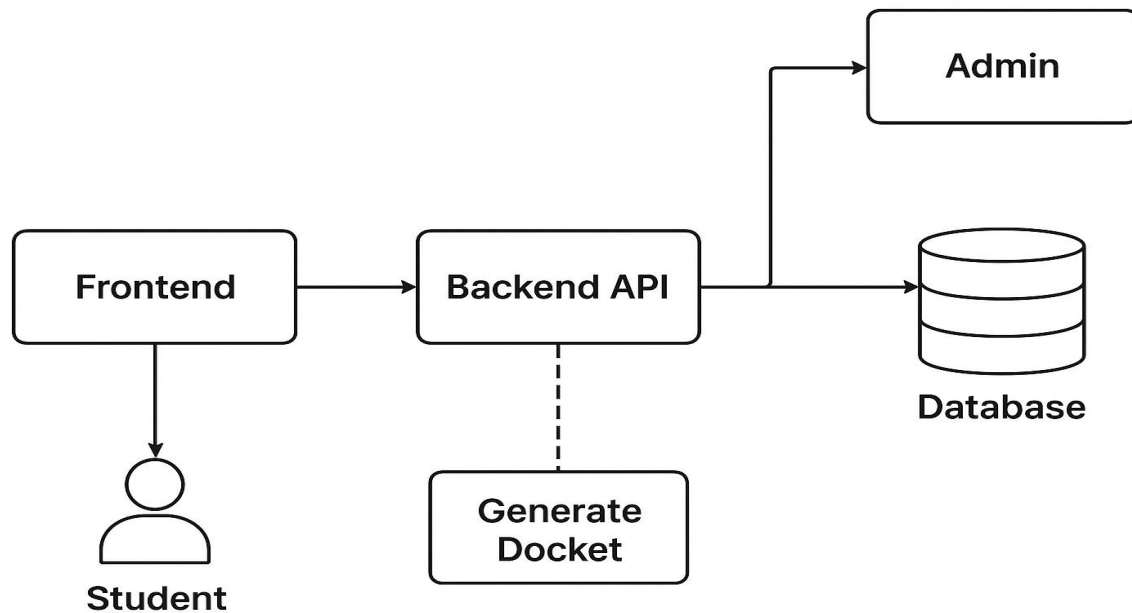
- **System Administrators and IT Staff:** Who are responsible for deploying, maintaining, and managing the system.
- **Software Developers:** Who may need to extend, modify, or fix the application.
- **University Administrators and Staff:** To understand the system's features and how to use the admin dashboard.
- **Students:** To understand how to use the student portal to access their dockets.

### 1.3. System Architecture Overview

The system is built using a client-server architecture.

- **Backend:** A Python Flask application that serves a RESTful API. It handles all business logic, including user authentication, database interactions, docket generation, and QR code creation.
- **Frontend:** A traditional multi-page application built with HTML, CSS, and JavaScript. It provides the user interface for both students and administrators.
- **Database:** A MySQL database that stores all application data, including student information, payment records, and docket details.

The frontend is served directly by the Flask backend, simplifying deployment and development.



#### 1.4. Technologies Used

- **Backend:** Python, Flask, Gunicorn
- **Database:** MySQL
- **Frontend:** HTML5, CSS3, JavaScript (ES6+)
- **Key Python Libraries:**
  - Flask-CORS: For handling Cross-Origin Resource Sharing.
  - PyJWT: For implementing JSON Web Token authentication.
  - passlib: For hashing passwords.
  - qrcode: For generating QR codes.
  - reportlab: For creating PDF dockets.
  - mysql-connector-python: For connecting to the MySQL database.

## 2. FRONT END

The frontend is everything you see and interact with from the driver's seat: the dashboard, the steering wheel, the buttons, and the windows. It is the **face** of the application.

The frontend is what students and administrators click on, look at, and use to get their tasks done. Its primary job is to provide a clear and intuitive user interface. It is responsible for:

- **Presenting Information:** It takes the data provided by the backend (like student names, payment status, and eligibility) and displays it in a structured and easy-to-understand way.
- **Capturing User Input:** It provides forms for logging in, search bars for finding students, and buttons for generating dockets. It collects what the user wants to do.
- **Communicating with the Backend:** When a user clicks a button, the frontend sends a message to the backend (the "engine") to perform an action. It then waits for the response and shows the result to the user (e.g., showing a success message or displaying the generated docket).
- **Providing an Offline Experience:** For administrators, the frontend has a special ability to work even without an internet connection, allowing them to verify dockets anywhere on campus.

This document will guide you through the visual parts of the system, explaining the journey of each user, and how the frontend brings the application to life.

### **The Building Blocks of the Frontend**

Our frontend is built with three fundamental web technologies:

- **HTML (HyperText Markup Language):** This is the skeleton of the website. It defines the structure and content of each page, such as titles, paragraphs, forms, and buttons.
- **CSS (Cascading Style Sheets):** This is the clothing and makeup of the website. It controls all the visual aspects—colors, fonts, layout, and spacing.

CSS is what makes the application look professional and polished. We have two main CSS files: styles.css for the login pages and dashboard.css for the main application dashboards.

- **JavaScript:** This is the brain and muscles of the frontend. It adds interactivity to the website. When you click a button and something happens without the page reloading, that's JavaScript at work. It's responsible for fetching data from the backend, updating the page dynamically, and handling user actions.

## 2.1. Students' portal

### Logging In

The first step for any student is to log in.

1. The student opens the Student Portal page (`students-portal.html`).
2. They are greeted with a clean login form asking for their Student Number and Password.

#### Cavendish Student Portal



[Admin Login](#)

[Forgot password?](#)

After filling in their details and clicking "Login", the JavaScript on this page sends the credentials to the backend's `/login` endpoint. If successful, the student is redirected to their dashboard.

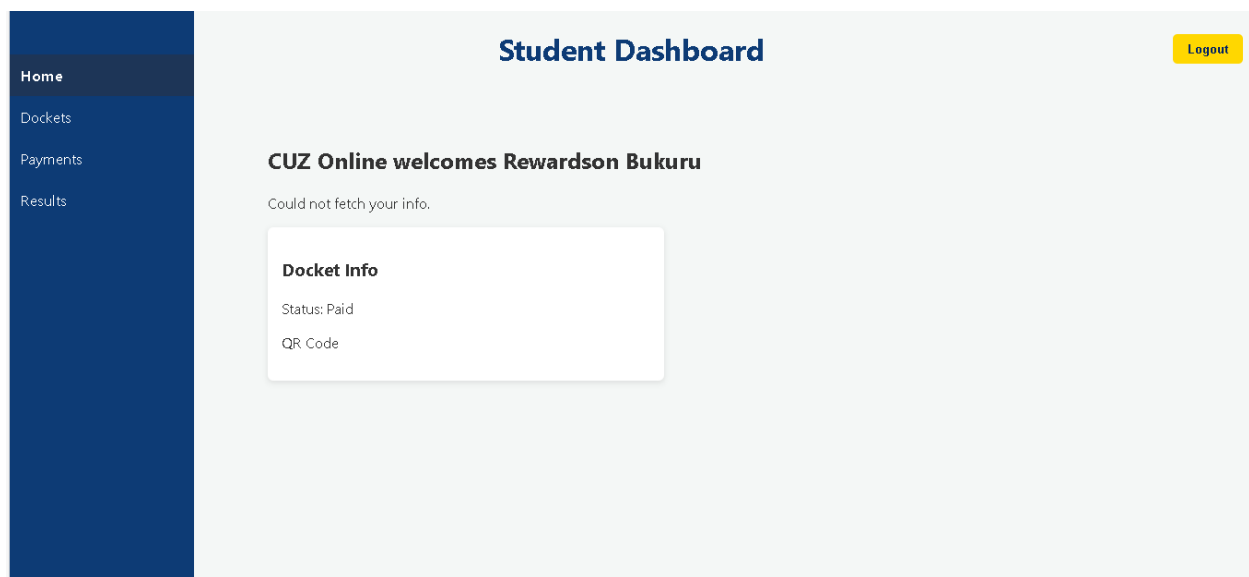
## Student dashbaord

After logging in, the student arrives at their personal dashboard (`dashboard.html`).

This page serves as a central hub. It displays a warm welcome message, the student's full name, and their registered academic programme (e.g., "Bachelor of Science in Computer Science").

A navigation sidebar on the left allows the student to access all other parts of the portal.

Which are the Dockets, Payments, Results and Home.



## Dockets

This is the most important feature for the student.

1. The student clicks on "Dockets" in the sidebar, which takes them to the `dockets.html` page.

2. Here, JavaScript immediately sends a request to the backend to check the student's eligibility for all exam types (CA1, CA2, Final Exam).
3. The page displays the eligibility status for each exam. If a student is cleared financially, they will see an "Eligible" status and an active "Generate Docket" button. If not, they might see a "Not Eligible" status.

## Not eligible status

The screenshot shows a web application interface for 'Dockets'. On the left is a dark blue sidebar with navigation links: 'Home', 'Dockets' (highlighted), 'Payments', and 'Results'. The main content area has a light blue header with the title 'Dockets' and a 'Logout' button. Below the header, the section is titled 'Your Exam Dockets'. It contains two lines of text: 'Downlaod your Docket For Upcoming or on going exams below.' and 'If you are not able to download your docket kindly visit the Rententions office.' Below this is a table with three columns: 'EXAM TYPE', 'STATUS', and 'ACTION'. The table has three rows, all showing 'Not Eligible' status and a message to visit the Retentions Office.

EXAM TYPE	STATUS	ACTION
CA1	Not Eligible	Please visit Retentions Office
CA2	Not Eligible	Please visit Retentions Office
EXAM	Not Eligible	Please visit Retentions Office

## Eligible status .

Home
Dockets
Payments
Results

## Dockets

Logout

### Your Exam Dockets

Download your Docket For Upcoming or on going exams below.

If you are not able to download your docket kindly visit the Retentions office.

EXAM TYPE	STATUS	ACTION
CA1	Eligible	<a href="#">Preview &amp; Download</a>
CA2	Not Eligible	Please visit Retentions Office
EXAM	Not Eligible	Please visit Retentions Office

When the student clicks the **Generate Docket** button, they are taken to a preview page (`docket-preview.html`) where the generated PDF is displayed. They can then choose to download or print it.

Docket Preview
Download Docket

1 of 1

**Cavendish University Zambia Ltd.**  
Faculty of Business and Information Technology  
Bachelor of Business Administration

**CA1 DOCKET**

---

**Date Issued:** 07/11/2025  
**Student Name:** Rewardson Bukuru  
**Student Number:** 104775

Code	Module	Date	Time	Venue	INVIGILATOR'S SIGNATURE
BBA103	Business Mathematics				
BBA104	Communication Skills				
BBA101	Introduction to Business				
BBA102	Principles of Management				


## The Admins Portal

Administrators have a more complex interface that allows them to manage the entire system.

### **Logging in**

Once the Admins clicks the admin login link, they are directed to the admins portal , where they have to login , with there username and admin password .

**Cavendish Admin Portal**

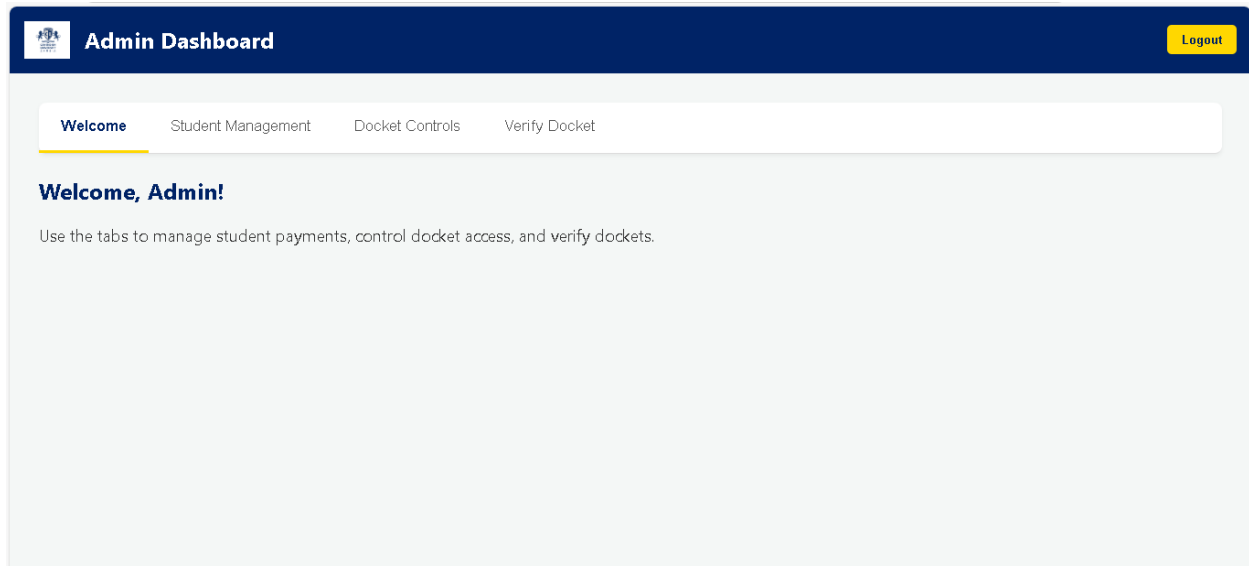


**Admin Login**

Login as Admin

### **Admin Dashboard**

After logging in through their dedicated portal ( `admin-login.html` ), the admin is directed to the `admin-dashboard.html` page. This dashboard is organized into tabs for different functions. Which are Student management, Docket Controls , Verify Docket.




## Managing Students

Under the "Student Management" tab, an admin can look up any student.

A search bar allows the admin to find a student by name or student number.

Once a student is found, the admin can view their financial details and, if necessary, update their payment status. This action directly affects whether a student is eligible to generate a docket.

The admin can also block or unblock a student's docket due to disciplinary issues, by the school management.

**Admin Dashboard**

Logout

Welcome

**Student Management**

Docket Controls

Verify Docket

104775

Search

**Rewardson Bukuru**  
**Student Number:** 104775  
**Programme:** Bachelor of Business Administration

**Total Fee:** 10000.00  
**Amount Paid:** 9000.00  
**Balance:** 1000.00

**Update Payment**  
  

Update Payment

**Update Payment**  
  

Update Payment

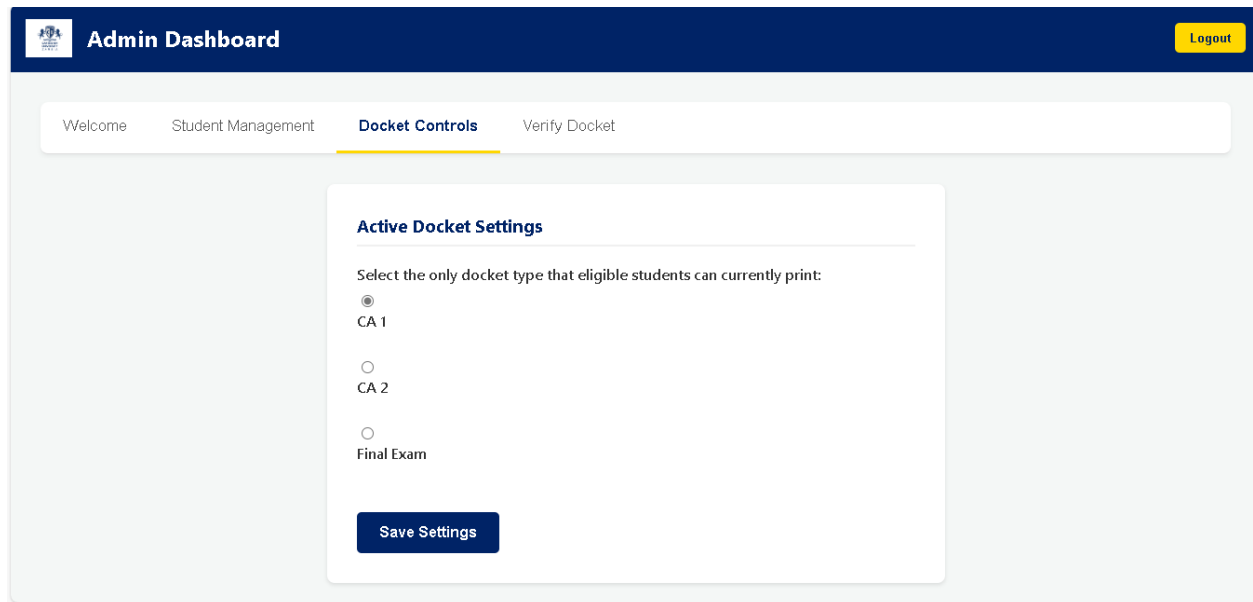
**Docket Status**  
Status: **Active**  

Block Student

## Docket Controls

Under this tab, an admin has powers:

**Set Active Docket:** The admin can use a dropdown menu to select which exam is currently active for the entire university. This ensures that students can only generate dockets for the relevant exam period. Which is either CA, CA2 , or The Exam.

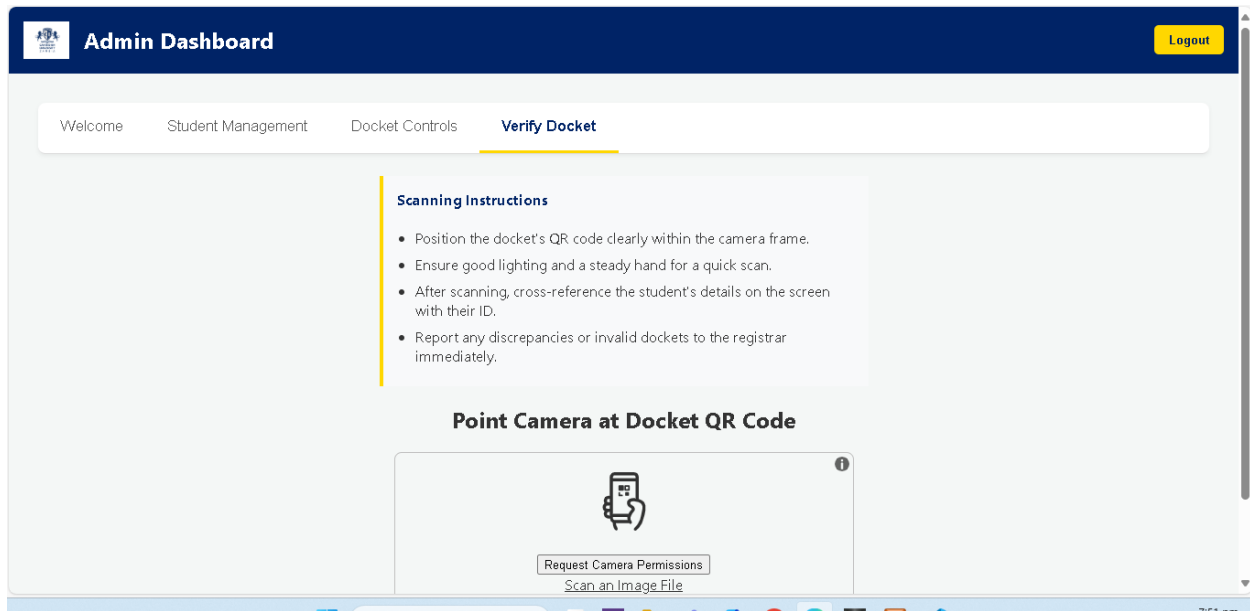


The screenshot shows the Admin Dashboard interface. At the top, there is a dark blue header with the university logo and the text "Admin Dashboard" on the left, and a yellow "Logout" button on the right. Below the header is a light gray navigation bar with four tabs: "Welcome", "Student Management", "Docket Controls" (which is highlighted with a yellow underline), and "Verify Docket". The main content area is a light gray box containing a white card titled "Active Docket Settings". Inside this card, there is a heading "Active Docket Settings" followed by the instruction "Select the only docket type that eligible students can currently print:". Below this instruction are three radio button options: "CA 1" (which is selected), "CA 2", and "Final Exam". At the bottom of the card is a dark blue button labeled "Save Settings".

## Verifying Dockets (Online and Offline)

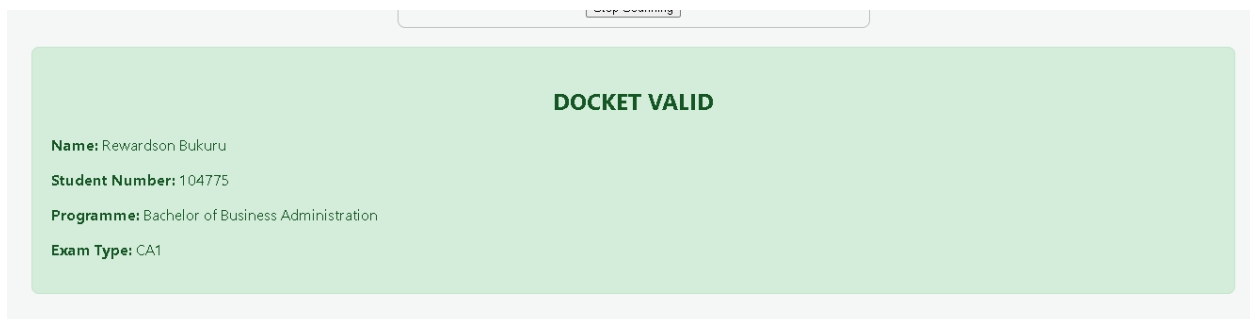
This is a critical feature found in the "Verify Docket" tab.

1. **Syncing for Offline Use:** Before going to an exam hall where internet may be unavailable, the admin must click the **"Sync for Offline Use"** button. In the background, JavaScript fetches all student and docket data from the backend and saves it into the browser's own storage (called IndexedDB). This is the magic that enables offline verification.
2. **Scanning:** The admin clicks "Scan QR Code" to open the device's camera. They point it at the QR code on a student's docket.

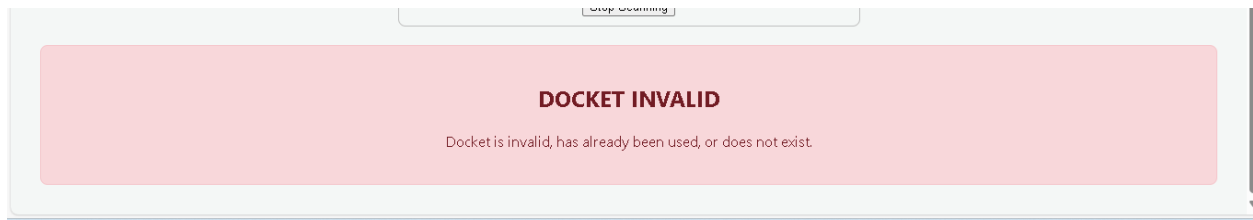


3. Verification: The frontend takes the information from the QR code and checks it against the data it has (either from the backend if online, or from its local storage if offline). It then displays the result instantly: a large "VERIFIED" or "INVALID" message, along with the student's photo and name for visual confirmation.

Verified docket .



Invalid or already used docket.



## Frontend File Structure

Here is a list of the key frontend files and what they do:

### HTML Files:

students-portal.html` & `admin-login.html`: The login pages.

dashboard.html` & `admin-dashboard.html`: The main landing pages after login.

dockets.html: Where students generate their dockets.

payments.html: Where students view their financial status.

### CSS Files:

-styles.css: Styles for the simple login pages.

- dashboard.css: More complex styles for the dashboards and inner pages.

### JavaScript Files:

sw.js (Service Worker): A special script that runs in the background. It helps the application work offline by saving the main files (like the verification page) so they can be opened without internet.

db.js: A helper file that makes it easy to work with the browser's offline database (IndexedDB).

## 3. BACKEND

The backend of the Docket Management System is built using the Flask web framework in Python. It follows a modular structure, organizing different functionalities into Blueprints. This approach helps in keeping the code clean, organized, and easy to maintain.

These are the detailed components of the backend :

## 1.app.py

The `app.py` script is responsible for four primary functions:

1. **Application Initialization:** It creates and configures the Flask application instance.
2. **Middleware and Service Configuration:** It sets up Cross-Origin Resource Sharing (CORS) for frontend communication and defines the application's core configuration, such as JWT settings.
3. **Core Route Definition:** It defines fundamental, cross-cutting routes for authentication (`/login`, `/logout`) and user session management (`/me`).
4. **Blueprint Registration:** It acts as a hub, importing and registering modular components (Blueprints) that contain feature-specific routes (`dockets`, `verification`, `admin`).

## 2. Initialization and Configuration

The script begins by importing necessary libraries and setting up the Flask application environment.

### 2.1. Flask Application Setup

```
``python
# Snippet 1: Flask App Initialization
backend_dir = os.path.dirname(os.path.abspath(__file__))
project_root = os.path.dirname(backend_dir)
frontend_dir = os.path.join(project_root, "Docket-system-frontend", "frontend")
```

```
app = Flask(
    __name__,
    static_folder=frontend_dir, # Use the correct absolute path
    static_url_path=""
)
CORS(app, supports_credentials=True, resources={r"/*": {"origins": "*"}})
```

### Explanation:

**Path Resolution:** The code first programmatically determines the absolute path to the `frontend` directory. This is a robust method that avoids hardcoding paths, ensuring the application can correctly locate its static assets (HTML, CSS, JS) regardless of where the script is run from.

- `Flask(...)`: An instance of the Flask application is created.
- `__name__`: A special Python variable that helps Flask locate resources.
- `static_folder`: This parameter tells Flask where to find static files. By pointing it to the `frontend` directory, the backend can directly serve the entire frontend application.
- `static_url_path=""`: This makes the static files accessible from the root URL (e.g., `http://127.0.0.1/styles.css` instead of `http://127.0.0.1/static/styles.css`).
- `CORS(app, ...)`: This line initializes the `Flask-CORS` extension. It configures the server to send the correct CORS headers, allowing web pages from any origin (`"origins": "\*"`) to make API requests to this backend. This is essential for the frontend to communicate with the backend during development and deployment.

## 2.2. JWT and Database Configuration

```
python
# Snippet 2: JWT and Database Configuration
# JWT configuration
JWT_SECRET = os.getenv("JWT_SECRET", "change-me-please-and-use-long-random")
JWT_ALGO = "HS256"
```

```

JWT_EXP_SECONDS = int(os.getenv("JWT_EXP_SECONDS", 60 * 60 * 8)) #
8 hours

# Database Connection
def get_db_connection():
    return mysql.connector.connect(
        host=os.getenv("DB_HOST", "localhost"),
        user=os.getenv("DB_USER", "root"),
        password=os.getenv("DB_PASSWORD", ""),
        database=os.getenv("DB_NAME", "docket_system"),
        autocommit=False
    )
'''

```

### Explanation:

- `os.getenv(...)`: The application retrieves its configuration from environment variables using `os.getenv`. This is a security best practice that separates configuration from code. Sensitive data like the `JWT_SECRET` and database credentials are not hardcoded. Default values are provided for development convenience.

### - JWT Settings:

- `JWT_SECRET`: The secret key used to sign and verify JSON Web Tokens. A compromised secret key would allow an attacker to forge valid tokens.

- `JWT_ALGO`: The signing algorithm used, `HS256` (HMAC using SHA-256).

- `JWT_EXP_SECONDS`: The token's validity period (8 hours). After this time, the token expires and the user must log in again.

- `get_db_connection()`: This function encapsulates the logic for creating a database connection. By centralizing it, we ensure that all parts of the application connect to the database consistently. Setting `autocommit=False` is important for routes that need to perform multiple database operations in a single, atomic transaction.

### 3. Authentication and Authorization

The `jwt\_required` decorator is the cornerstone of the application's security model.

```
```python
# Snippet 3: JWT Authentication Decorator
def jwt_required(role=None):
    def decorator(f):
        @wraps(f)
        def wrapper(*args, **kwargs):
            # 1. Extract Token
            auth = request.headers.get("Authorization", "")
            token = None
            if auth.startswith("Bearer "):
                token = auth.split(" ", 1)[1]
            else:
                token = request.cookies.get("access_token")

            if not token:
                return jsonify({"ok": False, "error": "Missing token"}), 401

            try:
                # 2. Decode and Validate Token
                payload = jwt.decode(token, JWT_SECRET, algorithms=[JWT_ALGO])
            except jwt.ExpiredSignatureError:
                return jsonify({"ok": False, "error": "Token expired"}), 401
            except Exception as e:
                return jsonify({"ok": False, "error": f"Invalid token: {e}"}), 401

            # 3. Role-Based Access Control (RBAC)
            if role and payload.get("role") != role:
                return jsonify({"ok": False, "error": "Forbidden"}), 403
        return wrapper
    return decorator
```

```

# 4. Attach Payload to Request
request.user = payload
return f(*args, **kwargs)
return wrapper
return decorator
'''

```

### Explanation:

This function is a **higher-order function** that returns a decorator. This pattern allows the decorator to accept arguments (like `role``).

**1. Token Extraction:** The decorator first attempts to find the JWT. It supports two common methods:

- **Bearer Token:** From the ``Authorization`` HTTP header. This is standard for REST APIs and is often used by mobile or non-browser clients.

- **Cookie:** From an HTTP cookie named ``access_token``. This is common for traditional web applications where the browser automatically sends cookies with each request.

**2.Token Decoding and Validation:** It uses ``jwt.decode()`` to validate the token. This single function call implicitly performs several critical checks:

- **Signature Verification:** It re-calculates the token's signature using the ``JWT_SECRET`` and compares it to the signature on the token. If they don't match, the token is invalid.

- **Expiration Check:** It checks the ``exp`` (expiration) claim in the token's payload against the current time.

**3. Role-Based Access Control (RBAC):** If a ``role`` argument (e.g., ``"admin"``) is passed to the decorator, it checks the ``role`` claim within the token's payload. If the user's role does not match the required role, it returns a ``403 Forbidden`` error, even if the token is otherwise valid.

**4. Payload Injection:** If all checks pass, the entire decoded payload (containing user ID, role, etc.) is attached to Flask's global ``request`` object as ``request.user``. The protected route can then access this information to perform its logic.

## 4. Core Routes

### 4.1. `/login`

```
```python
# Snippet 4: Login Route
@app.route("/login", methods=["POST"])
def login():
    # ... (credential extraction) ...

    # Database query based on role
    if role == "admin":
        cur.execute("SELECT ... FROM admins WHERE username=%s",
(username,))
    else:
        cur.execute("SELECT ... FROM students WHERE student_number=%s",
(username,))
    user = cur.fetchone()

    # ... (credential verification using bcrypt.verify) ...

    # Payload creation
    now = datetime.datetime.utcnow()
    payload = {
        "sub": str(user["id"]),
        "role": role,
        "iat": now,
        "exp": now + datetime.timedelta(seconds=JWT_EXP_SECONDS)
    }

    # Token encoding
    token = jwt.encode(payload, JWT_SECRET, algorithm=JWT_ALGO)
```

```
# ... (response creation with optional cookie setting) ...  
'''
```

### Explanation:

The login route is the gateway to the application's secure areas.

- It accepts a `POST` request containing the user's credentials and role.
- It queries the appropriate table (`admins` or `students`) based on the specified role.
- It uses `bcrypt.verify()` to securely compare the provided password against the stored hash. This function is designed to be slow to protect against brute-force attacks.
- Upon successful verification, it constructs the JWT **\*\*payload\*\***. The payload contains standard claims:
  - `sub` (Subject): The user's unique ID.
  - `iat` (Issued At): The time the token was created.
  - `exp` (Expiration Time): The time the token will expire.
- It also includes a custom claim, `role`, which is essential for the RBAC implemented in the `jwt\_required` decorator.
- Finally, it encodes the payload and secret key into a JWT string and returns it to the client.

## 4.2. Serving the Frontend

```
'''python  
# Snippet 5: Frontend Serving Routes  
@app.route("/")  
def serve_index():  
    """Serve student portal HTML"""  
    return send_from_directory(app.static_folder, "students-portal.html")  
  
@app.route("/<path:path>")
```

```
def serve_static_files(path):
    """Serve static frontend files (HTML, CSS, JS, etc.)"""
    # ...
    ...
```

**Explanation:** These routes configure the Flask backend to also function as a web server for the frontend application. The `@app.route("/")` route makes the main student portal the default page when a user navigates to the root URL. The `@app.route("/<path:path>")` is a catch-all route that serves any other file requested from the `static_folder` (our `frontend` directory). This integrated serving model simplifies deployment.

## 5. Blueprint Registration

```
``python
# Snippet 6: Blueprint Registration
from routes.dockets import docket_bp
from routes.verification import verification_bp
from routes.admin_controls import admin_controls_bp
app.register_blueprint(docket_bp, url_prefix="/dockets")
app.register_blueprint(verification_bp, url_prefix="/verification")
app.register_blueprint(admin_controls_bp, url_prefix="/admin")
``
```

### **Explanation:**

This is where the modular design comes together.

- The application imports the Blueprint objects (`docket_bp`, `verification_bp`, etc.) from their respective files in the `routes/` directory.
- `app.register_blueprint()` tells the main application about the routes defined in each blueprint.
- The `url_prefix` parameter is very important. It namespaces all routes within a blueprint. For example, a route defined as `/generate` inside `dockets.py` will be

accessible at the URL `"/dockets/generate"`. This prevents URL conflicts and logically groups the API endpoints.

## dockets.py - Docket & Data Services

The `'dockets_bp'` Blueprint is the heart of the student-facing functionality. It exposes API endpoints that allow students to check their eligibility for exams and generate their dockets as PDF files. It also provides essential data endpoints for the admin and verification clients, including data for offline PWA functionality.

This blueprint directly interacts with both the database (for student, course, and clearance data) and the file system (for system-wide settings like the active exam and student blocklists).

### 2. Core Functionalities

#### 2.1. PDF Generation (`'generate_docket_pdf'`)

This is a helper function that dynamically creates the final docket PDF using the `'reportlab'` library.

- Purpose: To assemble student data, course information, and a unique QR code into a structured, printable A4 document.
- Key Operations:
  - Draws the university logo and document headers.
  - Populates a table with the student's personal information.
  - Creates a main table listing all the courses the student is enrolled in for the semester.
  - Embeds a dynamically generated QR code image for verification.

```
```python
# Snippet 1: Main Courses Table Generation
# This snippet shows how ReportLab is used to construct the central table in the
PDF.
```

```

header = [Paragraph(h, style_bold_header) for h in ['Code', 'Module', 'Date', 'Time',
'Venue', "INVIGILATOR'S SIGNATURE"]]
table_data = [header]
for course in courses:
    table_data.append([Paragraph(c, style_normal) for c in
[course.get('course_code', ""), course.get('course_name', ""), "", "", ""]])

main_table = Table(table_data, colWidths=[0.7*inch, 2.4*inch, 0.6*inch, 0.6*inch,
0.6*inch, 1.4*inch])
main_table.setStyle(TableStyle([
    ('BACKGROUND', (0, 0), (-1, 0), colors.lightgrey),
    ('GRID', (0, 0), (-1, -1), 1, colors.black)
]))
'''

```

## 2.2. Eligibility Check (`/eligibility/<student_id>`)

This endpoint is the first step in the docket generation process, allowing the frontend to display the student's status for each exam type.

- Purpose: To provide a clear, reason-based eligibility status for each exam (CA1, CA2, Final Exam).

- **Business Logic**: A student is determined to be `'eligible'` only if **all** of the following conditions are met:

1. The student's number is **not** in the `'blocked_students.json'` file.
2. The requested `'exam_type'` is the one currently set as active in `'exam_settings.json'`.
3. The student's status for that exam is `'eligible'` in the `'clearances'` table in the database.

- Frontend Interaction: The frontend calls this endpoint upon loading the student dashboard to enable or disable the "Generate Docket" buttons and show an explanatory message if a docket is unavailable.

```
```python
# Snippet 2: Eligibility Response Structure
# The endpoint returns a list with this structure, which the frontend uses to render
the UI.

"eligibility": [
    {
        "exam_type": "ca1",
        "eligible": True,
        "reason": ""
    },
    {
        "exam_type": "ca2",
        "eligible": False,
        "reason": "Docket not currently active."
    },
    {
        "exam_type": "exam",
        "eligible": False,
        "reason": "Not cleared by Finance. Please visit the Retentions Office."
    }
]
```
```

### 2.3. Docket Generation (/generate)

This is the main endpoint for creating and issuing a docket PDF.

- Purpose: To perform a final eligibility check, generate a secure verification token, record the transaction in the database, and return the final PDF document.

- Security-Critical Operation: The most important step in this function is the secure token generation.

1. A cryptographically secure random token is created using ``secrets.token_urlsafe()``.
2. This token is then hashed using ``hashlib.sha256()``.
3. The raw token is placed into the QR code data.
4. The hashed token is stored in the ``docket_tokens`` database table.

- **Why this is important:** This pattern ensures that the database only stores a secure hash of the token. The verification process works by hashing the token from a scanned QR code and comparing it to the stored hashes. This prevents anyone with database access from being able to forge a valid docket.

```
```python
# Snippet 3: Secure Token Generation and Hashing
# This is the core security mechanism for docket verification.

# 1. Generate a random token for the QR code
token_value = secrets.token_urlsafe(16)

# 2. Hash the token for secure storage in the database
token_hash = hashlib.sha256(token_value.encode()).hexdigest()

# 3. The raw token is used in the QR data, the hash is stored in the DB
qr_data = f'{student["student_number"]}|{exam_type}|{token_value}'

# ... later in the code ...
cur.execute("""
    INSERT INTO docket_tokens (docket_id, token_hash, status)
    VALUES (%s, %s, 'active')
""", (docket_id, token_hash))
```
```

## **2.4. Offline Data Sync (`/sync/students` & `/sync/tokens`)**

These endpoints are designed to support the offline capabilities of the QR code verification Progressive Web App (PWA).

- Purpose: To allow the verification client to download all necessary data so it can continue to function without an internet connection.
- `/sync/students`: Returns a JSON list of all students (ID, name, student number). The PWA uses this to display student details after a successful offline scan.
- `/sync/tokens`: Returns a JSON list of all **active token hashes**. The PWA stores these hashes locally. When a QR code is scanned offline, the PWA can hash the token from the QR code and check if it exists in its local list of active hashes

## **3. Database Interaction**

This blueprint is database-intensive and interacts with the following key tables:

- Reads: `students`, `programmes`, `courses`, `enrollments`, `curriculum`, `clearances`, `student\_balances`.
- Writes: `dockets` (to record the issued docket), `docket\_tokens` (to store the secure token hash), `payments` (for admin payment updates).

## **verification.py` - QR Code Verification Services**

The `verification\_bp` Blueprint provides the API endpoints necessary for staff members (admins) to scan

student dockets and confirm their validity. A key feature is its robust handling of concurrent

verification attempts and support for offline scanning synchronization.

## **2. Core Functionalities**

### **2.1. Docket Verification (/verify)**

This is the central endpoint for processing a scanned QR code and determining the validity of a student's docket.

- Purpose: To securely validate a docket presented via a QR code, prevent double-scanning, and log the verification event.

- Access Control: This endpoint is protected by the `@jwt_required(role="admin")` decorator, ensuring only authenticated administrators can perform verifications.

- Verification Process:

1. QR Data Parsing: The endpoint receives `qr_data` (a string like "student\_number|exam\_type|token\_value") from the frontend. It parses this string to extract the individual components.

2. Token Hashing: For security, the `token_value` extracted from the QR code is immediately hashed using SHA-256. This hash is then used to query the `docket_tokens` table, ensuring that the raw token is never directly exposed or compared.

3. Concurrency Control (SELECT ... FOR UPDATE): This is a critical security and integrity feature. When the system searches for the `docket_tokens` entry, it uses `SELECT ... FOR UPDATE`. This SQL clause locks the database row corresponding to the token for the duration of the transaction.

- Why it's important: This prevents a "race condition" where two different verifiers might scan the same valid docket simultaneously. The first verifier to acquire the lock proceeds, marks the token as 'used', and commits. The second verifier's attempt to acquire the lock will either wait or fail, preventing the same docket from being verified twice.

4. Status Update & Logging: If a valid and active token is found, its status in the `docket_tokens` table is updated to 'used', and a record of the verification is inserted into the `verifications` table.

5. Response: The endpoint returns the student's details (first name, last name, student number, program) for immediate visual confirmation by the verifier.

## Secure Token Lookup and Concurrency Control

This code demonstrates the use of hashing and database row locking for robust verification.

```
token_hash = hashlib.sha256(token_value.encode()).hexdigest()

conn.start_transaction() # Begin a database transaction

cur.execute("""
    SELECT dt.token_id, dt.docket_id, d.student_id
    FROM docket_tokens dt
    JOIN dockets d ON dt.docket_id = d.docket_id
    JOIN students s ON d.student_id = s.id
    WHERE dt.token_hash = %s
    AND s.student_number = %s
    AND d.exam_type = %s
    AND dt.status = 'active'
    LIMIT 1 FOR UPDATE -- CRITICAL: Locks the row to prevent race
conditions
""", (token_hash, student_number, exam_type))
token_row = cur.fetchone()

if not token_row:
    conn.rollback() # Release lock and abort transaction
    ... handle invalid/used token ...
else:
    ... proceed with updating token status and logging verification ...
    conn.commit() # Release lock and save changes
```

## 2.2. Offline Verification Synchronization (`/sync`)

This endpoint facilitates the synchronization of verification records that were performed while the verification device was offline.

- Purpose: To allow the Progressive Web App (PWA) to upload a batch of pending offline verifications to the central database once an internet connection is re-established.
- Process: The endpoint receives a list of ``pending_verifications``. For each item in the list, it performs a similar token hashing and lookup process as the ``/verify`` endpoint. If a token is found and is still active, its status is updated to ``used``, and a verification log entry is created.
- Transaction Management: The entire synchronization process is wrapped in a single database transaction, ensuring atomicity. If any part of the batch fails, the entire operation can be rolled back to maintain data consistency.

### **3. Database Interaction**

This blueprint primarily interacts with the following tables:

- Reads: ``docket_tokens``, ``dockets``, ``students``, ``programmes``.
- Writes: ``docket_tokens`` (to update token status), ``verifications`` (to log scan events).
- Concurrency: Employs database transactions and row-level locking (``FOR UPDATE``) to ensure data integrity during verification.

## **admin controls.py**

### Purpose

This blueprint provides administrators with a simple interface to manage dynamic, system-wide settings

that are not stored in the main database. Its primary function is to control docket eligibility rules that

can be changed frequently without requiring a database migration.

### Interaction with Frontend

The admin dashboard frontend (e.g., `admin-dashboard.html`) is the primary consumer of this blueprint's

API. The frontend contains UI elements that allow an administrator to:

- Set the Active Exam: A dropdown or set of radio buttons would make a `POST` request to the

`/admin/settings` endpoint to change the value in `exam\_settings.json`. This immediately changes which

docket type students are allowed to generate.

- Block/Unblock Students: A search bar and a "Block" button in the admin panel would allow an admin

to find a student and send their student number to the  
`/admin/students/<student\_number>/block` endpoint.

### Interaction with Backend

This blueprint's interaction with the rest of the backend is indirect but critical. It operates by

modifying two key JSON files:

- `exam\_settings.json`: Contains a single key, `"active\_exam"`, which dictates the only exam type

(`"ca1"`, `"ca2"`, or `"exam"`) for which dockets can currently be generated.

- `blocked\_students.json`: Contains a simple list of student numbers.

The `dockets.py` blueprint reads these two files at the beginning of every eligibility check. This design

decouples the administrative controls from the core docket generation logic, allowing for a simple

file-based method of controlling system behavior.

### Interaction with Database

This component has **\*\*no direct interaction with the database\*\***. It operates exclusively on the file system

by reading and writing to the JSON files mentioned above. This is a deliberate design choice for

simplicity. While this approach is effective for a small number of settings, it lacks the transactional

integrity, scalability, and concurrency safety of a database. For instance, if two admins were to change a

setting at the exact same moment, a race condition could occur where one change overwrites the other.

## [utils/qr\\_generator.py](#)

### [Purpose](#)

This is a small, single-purpose utility module. Its sole function is to take a string of data, generate a

QR code image from it, and return that image as a **\*\*base64-encoded string\*\***.

### [Interaction with Frontend](#)

A base64 string is a format that can be directly embedded into an HTML `` tag's `src` attribute. For

example: `

there were a feature where the frontend needed to display a QR code directly on a web page without first

saving it as a file first.

## Interaction with Backend & Database

This utility is not currently used by any part of the main application logic. The ``dockets.py``

blueprint, which is responsible for generating the docket PDF, contains its own separate logic for

creating a QR code and saving it as a temporary image file on the server's disk before drawing it onto the

PDF canvas.

Therefore, ``qr_generator.py`` likely represents either a remnant of a previous design that was superseded

or a utility created for a potential future feature. It has no interaction with the database.

## ``scripts/`` Directory (Maintenance Scripts)

### Purpose

The ``scripts/`` directory contains standalone Python scripts that are **\*\*not\*\*** part of the running Flask web

application. They are intended for one-off administrative or data maintenance tasks and must be executed

manually from the command line by a developer or system administrator with access to the server.

### `scripts/create_admin.py``

This script is used to securely add a new administrator to the system.

- Interaction: When run, it prompts the user in the command line terminal for a new admin username and password. It then securely **\*\*hashes\*\*** the provided password using the ``bcrypt`` algorithm. Finally, it connects to the database and

inserts the new administrator's username and hashed password into the `admins` table. It has no interaction with the frontend or the running web application.

## scripts/hash\_password.py` & hash\_admin\_passwords.py

These are data migration scripts designed to enhance the security of an existing database.

- Purpose: Their function is to find any user (both students and admins) whose password is still stored in plaintext in the database and convert it into a secure `bcrypt` hash.
- Interaction: The scripts connect to the database and query the `students` and `admins` tables for any records where the `password\_hash` column is empty but a plaintext `password` exists. For each such record, it computes the secure hash and updates the `password\_hash` field, often nullifying the old plaintext password column afterward. These scripts are essential if the system is being upgraded from a less secure, plaintext password storage scheme. They have no interaction with the frontend or the web application.

## `.env` File (Environment Configuration)

### **Purpose**

The `.env` file is a text file used to store environment variables for local development. It is the foundation of the application's configuration, allowing for a clean separation between code and configuration.

### **Interaction with Backend**

The `python-dotenv` library is used at the very beginning of `app.py` to load the key-value pairs from the `.env` file into the application's operating system environment. Throughout the application, the backend code retrieves these configuration values using `os.getenv("VARIABLE\_NAME")`. This mechanism is used for:

- Database Credentials: `DB\_HOST`, `DB\_USER`, `DB\_PASSWORD`, `DB\_NAME`.
- Security Secrets: `JWT\_SECRET` for signing and verifying authentication tokens.
- Application Behavior: `JWT\_EXP\_SECONDS` to control token lifespan.

### **Interaction with Database & Frontend**

- The `.env` file has no direct interaction with the frontend.
- Its interaction with the database is fundamental. The `get\_db\_connection()` function in `app.py`
  - relies entirely on the database credentials stored in this file to know how and where to connect.

### **Security**

The `.env` file is critical for security. Because it contains sensitive information like database passwords and secret keys, it **must never be committed to version control (e.g., Git)**. The project's `.gitignore` file should always contain an entry for `.env` to prevent accidental exposure. Instead, a template file named `.env.example` is typically included in the repository, which lists the required variables but with placeholder or empty values.

## **4.DATABASE**

The Docket Management System Database is designed to securely store, manage, and verify student academic docket, course enrollments, examination clearances, and verification tokens. This database serves as the backbone of the system's backend, ensuring integrity, security, and traceability of all records. It is implemented using MySQL and consists of 17 interrelated tables.

## 1. Database Structure Overview

The database consists of 17 primary tables, each serving a distinct purpose. They are linked through foreign key relationships to maintain referential integrity and prevent duplication or tampering of records. The major relational flow connects students, their programmes, courses, clearances, dockets, and verification tokens.

## 2. Table Descriptions and Relationships

### students

Stores personal and academic details of each student. Linked to the 'programmes' table through `programme_id`.

Showing rows 0 - 24 (100 total, Query took 0.0011 seconds.)

SELECT \* FROM `students`

☐ Profiling

[\[ Edit inline \]](#)

[\[ Edit \]](#)

[\[ Explain SQL \]](#)

[\[ Create PHP code \]](#)

[\[ Refresh \]](#)

1

>

>>

☐ Show all

Number of rows: 25

Filter rows:

Sort by key: None

Extra options

|                          |                      |                      | id                     | student_number | first_name | last_name | email  | programme_id                      | current_year | current_semester |
|--------------------------|----------------------|----------------------|------------------------|----------------|------------|-----------|--------|-----------------------------------|--------------|------------------|
| <input type="checkbox"/> | <a href="#">Edit</a> | <a href="#">Copy</a> | <a href="#">Delete</a> | 1              | 104775     | Rewardson | Bukuru | rb104775@students.cavendish.co.zm | 1            | 1                |
| <input type="checkbox"/> | <a href="#">Edit</a> | <a href="#">Copy</a> | <a href="#">Delete</a> | 2              | 104776     | Denise    | Seti   | ds104776@students.cavendish.co.zm | 2            | 1                |
| <input type="checkbox"/> | <a href="#">Edit</a> | <a href="#">Copy</a> | <a href="#">Delete</a> | 3              | 104777     | James     | Mwale  | jm104777@students.cavendish.co.zm | 3            | 1                |
| <input type="checkbox"/> | <a href="#">Edit</a> | <a href="#">Copy</a> | <a href="#">Delete</a> | 4              | 104778     | Agness    | Zulu   | az104778@students.cavendish.co.zm | 4            | 1                |
| <input type="checkbox"/> | <a href="#">Edit</a> | <a href="#">Copy</a> | <a href="#">Delete</a> | 5              | 104779     | Peter     | Tembo  | pt104779@students.cavendish.co.zm | 5            | 1                |
| <input type="checkbox"/> | <a href="#">Edit</a> | <a href="#">Copy</a> | <a href="#">Delete</a> | 6              | 104780     | Joseph    | Mumba  | jm104780@students.cavendish.co.zm | 6            | 1                |
| <input type="checkbox"/> | <a href="#">Edit</a> | <a href="#">Copy</a> | <a href="#">Delete</a> | 7              | 104781     | Linda     | Chanda | lc104781@students.cavendish.co.zm | 7            | 1                |
| <input type="checkbox"/> | <a href="#">Edit</a> | <a href="#">Copy</a> | <a href="#">Delete</a> | 8              | 104782     | Henry     | Nnoma  | hn104782@students.cavendish.co.zm | 8            | 1                |

### programmes

Holds details about academic programmes offered by the university.

Showing rows 0 - 9 (10 total, Query took 0.0004 seconds.)

SELECT \* FROM `programmes`

Profiling [ Edit inline ] [ Edit ] [ Explain SQL ] [ Create PHP code ] [ Refresh ]

Show all | Number of rows: 25 | Filter rows: Search this table | Sort by key: None

Extra options

|                                           | programme_id | programme_name                      | programme_code | total_years | semesters_per_year | created_at          | updated_at          | duration_year |
|-------------------------------------------|--------------|-------------------------------------|----------------|-------------|--------------------|---------------------|---------------------|---------------|
| <input type="checkbox"/> Edit Copy Delete | 1            | Bachelor of Business Administration | BBA            | 4           | 2                  | 2025-09-17 15:34:18 | 2025-09-26 12:30:29 |               |
| <input type="checkbox"/> Edit Copy Delete | 2            | Bachelor of Computer Science        | BCS            | 4           | 2                  | 2025-09-17 15:34:18 | 2025-09-26 12:30:29 |               |
| <input type="checkbox"/> Edit Copy Delete | 3            | Bachelor of Accounting and Finance  | BAF            | 4           | 2                  | 2025-09-17 15:34:18 | 2025-09-26 12:30:29 |               |
| <input type="checkbox"/> Edit Copy Delete | 4            | Bachelor of Public Health           | BPH            | 4           | 2                  | 2025-09-17 15:34:18 | 2025-09-26 12:30:29 |               |
| <input type="checkbox"/> Edit Copy Delete | 5            | Bachelor of Nursing                 | BNUR           | 4           | 2                  | 2025-09-17 15:34:18 | 2025-09-26 12:30:29 |               |
| <input type="checkbox"/> Edit Copy Delete | 6            | Bachelor of Education               | BEDU           | 4           | 2                  | 2025-09-17 15:34:18 | 2025-09-26 12:30:29 |               |
| <input type="checkbox"/> Edit Copy Delete | 7            | Bachelor of Engineering             | BENG           | 4           | 2                  | 2025-09-17 15:34:18 | 2025-09-26 12:30:29 |               |

Console

## courses

Contains all course records offered under each programme.

Showing rows 0 - 24 (25 total, Query took 0.0135 seconds.)

SELECT \* FROM `courses`

Profiling [ Edit inline ] [ Edit ] [ Explain SQL ] [ Create PHP code ] [ Refresh ]

1 | > >> | Show all | Number of rows: 25 | Filter rows: Search this table | Sort by key: None

Extra options

|                                           | course_id | course_code | course_name              | programme_id | year_of_study | semester | credits | created_at          | updated_at          |
|-------------------------------------------|-----------|-------------|--------------------------|--------------|---------------|----------|---------|---------------------|---------------------|
| <input type="checkbox"/> Edit Copy Delete | 1         | BBA101      | Introduction to Business | 1            | 1             | 1        | 3       | 2025-09-17 16:31:09 | 2025-09-17 16:31:09 |
| <input type="checkbox"/> Edit Copy Delete | 2         | BBA102      | Principles of Management | 1            | 1             | 1        | 3       | 2025-09-17 16:31:09 | 2025-09-17 16:31:09 |
| <input type="checkbox"/> Edit Copy Delete | 3         | BBA103      | Business Mathematics     | 1            | 1             | 1        | 3       | 2025-09-17 16:31:09 | 2025-09-17 16:31:09 |
| <input type="checkbox"/> Edit Copy Delete | 4         | BBA104      | Communication Skills     | 1            | 1             | 1        | 3       | 2025-09-17 16:31:09 | 2025-09-17 16:31:09 |
| <input type="checkbox"/> Edit Copy Delete | 5         | BBA105      | Financial Accounting 1   | 1            | 1             | 2        | 3       | 2025-09-17 16:31:09 | 2025-09-17 16:31:09 |
| <input type="checkbox"/> Edit Copy Delete | 6         | BBA106      | Business Economics       | 1            | 1             | 2        | 3       | 2025-09-17 16:31:09 | 2025-09-17 16:31:09 |
| <input type="checkbox"/> Edit Copy Delete | 7         | BBA107      | Organizational Behavior  | 1            | 1             | 2        | 3       | 2025-09-17 16:31:09 | 2025-09-17 16:31:09 |
| <input type="checkbox"/> Edit Copy Delete | 8         | BBA108      | Business Law             | 1            | 1             | 2        | 3       | 2025-09-17 16:31:09 | 2025-09-17 16:31:09 |

## curriculum

Maps courses to specific programmes and academic years.

Showing rows 0 - 24 (320 total, Query took 0.0141 seconds.)

SELECT \* FROM `curriculum`

Profiling [ Edit inline ] [ Edit ] [ Explain SQL ] [ Create PHP code ] [ Refresh ]

1 > >> | ☐ Show all | Number of rows: 25 | Filter rows: Search this table | Sort by key: None

Extra options

|                          |                  | curriculum_id | programme_id | course_id | year_of_study | semester | created_at          |
|--------------------------|------------------|---------------|--------------|-----------|---------------|----------|---------------------|
| <input type="checkbox"/> | Edit Copy Delete | 5             | 1            | 1         | 1             | 1        | 2025-09-18 21:33:51 |
| <input type="checkbox"/> | Edit Copy Delete | 6             | 1            | 2         | 1             | 1        | 2025-09-18 21:33:51 |
| <input type="checkbox"/> | Edit Copy Delete | 7             | 1            | 3         | 1             | 1        | 2025-09-18 21:33:51 |
| <input type="checkbox"/> | Edit Copy Delete | 8             | 1            | 4         | 1             | 1        | 2025-09-18 21:33:51 |
| <input type="checkbox"/> | Edit Copy Delete | 13            | 1            | 5         | 1             | 2        | 2025-09-18 21:36:21 |
| <input type="checkbox"/> | Edit Copy Delete | 14            | 1            | 6         | 1             | 2        | 2025-09-18 21:36:21 |
| <input type="checkbox"/> | Edit Copy Delete | 15            | 1            | 7         | 1             | 2        | 2025-09-18 21:36:21 |
| <input type="checkbox"/> | Edit Copy Delete | 16            | 1            | 8         | 1             | 2        | 2025-09-18 21:36:21 |
| <input type="checkbox"/> | Edit Copy Delete | 17            | 1            | 9         | 2             | 1        | 2025-09-18 21:36:21 |
| <input type="checkbox"/> | Edit Copy Delete | 18            | 1            | 10        | 2             | 1        | 2025-09-18 21:36:21 |
| <input type="checkbox"/> | Edit Copy Delete | 19            | 1            | 11        | 2             | 1        | 2025-09-18 21:36:21 |
| <input type="checkbox"/> | Edit Copy Delete | 20            | 1            | 12        | 2             | 1        | 2025-09-18 21:36:21 |

l&db=docket\_system3&table=curriculum&pos=0

## enrollments

Tracks which course each student is enrolled in. Links 'students' and 'curriculum'.

Showing rows 0 - 24 (400 total, Query took 0.0012 seconds.)

SELECT \* FROM `enrollments`

Profiling [ Edit inline ] [ Edit ] [ Explain SQL ] [ Create PHP code ] [ Refresh ]

1 > >> | ☐ Show all | Number of rows: 25 | Filter rows: Search this table | Sort by key: None

Extra options

|                          |                  | enrollment_id | student_id | curriculum_id | year_of_study | semester | enrollment_status | created_at          | updated_at          |
|--------------------------|------------------|---------------|------------|---------------|---------------|----------|-------------------|---------------------|---------------------|
| <input type="checkbox"/> | Edit Copy Delete | 1             | 1          | 5             | 1             | 1        | active            | 2025-09-23 03:18:51 | 2025-09-23 03:18:51 |
| <input type="checkbox"/> | Edit Copy Delete | 2             | 1          | 6             | 1             | 1        | active            | 2025-09-23 03:18:51 | 2025-09-23 03:18:51 |
| <input type="checkbox"/> | Edit Copy Delete | 3             | 1          | 7             | 1             | 1        | active            | 2025-09-23 03:18:51 | 2025-09-23 03:18:51 |
| <input type="checkbox"/> | Edit Copy Delete | 4             | 1          | 8             | 1             | 1        | active            | 2025-09-23 03:18:51 | 2025-09-23 03:18:51 |
| <input type="checkbox"/> | Edit Copy Delete | 8             | 2          | 73            | 1             | 1        | active            | 2025-09-23 03:57:30 | 2025-09-23 03:57:30 |
| <input type="checkbox"/> | Edit Copy Delete | 9             | 2          | 74            | 1             | 1        | active            | 2025-09-23 03:57:30 | 2025-09-23 03:57:30 |
| <input type="checkbox"/> | Edit Copy Delete | 10            | 2          | 75            | 1             | 1        | active            | 2025-09-23 03:57:30 | 2025-09-23 03:57:30 |
| <input type="checkbox"/> | Edit Copy Delete | 11            | 2          | 76            | 1             | 1        | active            | 2025-09-23 03:57:30 | 2025-09-23 03:57:30 |
| <input type="checkbox"/> | Edit Copy Delete | 12            | 3          | 105           | 1             | 1        | active            | 2025-09-23 03:57:30 | 2025-09-23 03:57:30 |
| <input type="checkbox"/> | Edit Copy Delete | 13            | 3          | 106           | 1             | 1        | active            | 2025-09-23 03:57:30 | 2025-09-23 03:57:30 |
| <input type="checkbox"/> | Edit Copy Delete | 14            | 3          | 107           | 1             | 1        | active            | 2025-09-23 03:57:30 | 2025-09-23 03:57:30 |

## Student balances

Keeps track of all student balance, it is linked to the payments table once a student makes a payment the student balance is automatically, updated using the sql triggers

Showing rows 0 - 24 (397 total, Query took 0.0014 seconds.)

`SELECT * FROM `student_balances``

☐ Profiling [\[ Edit inline \]](#) [\[ Edit \]](#) [\[ Explain SQL \]](#) [\[ Create PHP code \]](#) [\[ Refresh \]](#)

1 > >> ☐ Show all Number of rows: 25 Filter rows: Search this table Sort by key: None

Extra options

|                          |                                                                  | balance_id | student_id | programme_id | year_of_study | semester | total_fee | amount_paid | balance   | last_updated        |
|--------------------------|------------------------------------------------------------------|------------|------------|--------------|---------------|----------|-----------|-------------|-----------|---------------------|
| <input type="checkbox"/> | <a href="#">Edit</a> <a href="#">Copy</a> <a href="#">Delete</a> | 4          | 1          | 1            | 1             | 1        | 10000.00  | 0.00        | 10000.00  | 2025-10-11 10:50:22 |
| <input type="checkbox"/> | <a href="#">Edit</a> <a href="#">Copy</a> <a href="#">Delete</a> | 5          | 2          | 2            | 1             | 1        | 12000.00  | 0.00        | 12000.00  | 2025-09-26 12:36:44 |
| <input type="checkbox"/> | <a href="#">Edit</a> <a href="#">Copy</a> <a href="#">Delete</a> | 6          | 2          | 2            | 1             | 1        | 12000.00  | 0.00        | 12000.00  | 2025-09-26 12:36:44 |
| <input type="checkbox"/> | <a href="#">Edit</a> <a href="#">Copy</a> <a href="#">Delete</a> | 7          | 2          | 2            | 1             | 1        | 12000.00  | 0.00        | 12000.00  | 2025-09-26 12:36:44 |
| <input type="checkbox"/> | <a href="#">Edit</a> <a href="#">Copy</a> <a href="#">Delete</a> | 8          | 2          | 2            | 1             | 1        | 12000.00  | 0.00        | 12000.00  | 2025-09-26 12:36:44 |
| <input type="checkbox"/> | <a href="#">Edit</a> <a href="#">Copy</a> <a href="#">Delete</a> | 9          | 3          | 3            | 1             | 1        | 11000.00  | 22000.00    | -11000.00 | 2025-10-11 09:36:11 |
| <input type="checkbox"/> | <a href="#">Edit</a> <a href="#">Copy</a> <a href="#">Delete</a> | 10         | 3          | 3            | 1             | 1        | 11000.00  | 22000.00    | -11000.00 | 2025-10-11 09:36:11 |
| <input type="checkbox"/> | <a href="#">Edit</a> <a href="#">Copy</a> <a href="#">Delete</a> | 11         | 3          | 3            | 1             | 1        | 11000.00  | 22000.00    | -11000.00 | 2025-10-11 09:36:11 |
| <input type="checkbox"/> | <a href="#">Edit</a> <a href="#">Copy</a> <a href="#">Delete</a> | 12         | 3          | 3            | 1             | 1        | 11000.00  | 22000.00    | -11000.00 | 2025-10-11 09:36:11 |
| <input type="checkbox"/> | <a href="#">Edit</a> <a href="#">Copy</a> <a href="#">Delete</a> | 13         | 4          | 4            | 1             | 1        | 9000.00   | 1000.00     | 8000.00   | 2025-10-11 10:49:41 |
| <input type="checkbox"/> | <a href="#">Edit</a> <a href="#">Copy</a> <a href="#">Delete</a> | 14         | 4          | 4            | 1             | 1        | 9000.00   | 1000.00     | 8000.00   | 2025-10-11 10:49:41 |

## clearances

Maintains continuous assessment (CA1, CA2) and exam eligibility statuses for students.

Showing rows 0 - 24 (400 total, Query took 0.0020 seconds.)

`SELECT * FROM `clearances``

☐ Profiling [\[ Edit inline \]](#) [\[ Edit \]](#) [\[ Explain SQL \]](#) [\[ Create PHP code \]](#) [\[ Refresh \]](#)

1 > >> ☐ Show all Number of rows: 25 Filter rows: Search this table Sort by key: None

Extra options

|                          |                                                                  | clearance_id | student_id | programme_id | year_of_study | semester | ca1_status | ca2_status | exam_status | last_checked        |
|--------------------------|------------------------------------------------------------------|--------------|------------|--------------|---------------|----------|------------|------------|-------------|---------------------|
| <input type="checkbox"/> | <a href="#">Edit</a> <a href="#">Copy</a> <a href="#">Delete</a> | 1            | 1          | 1            | 1             | 1        | blocked    | blocked    | blocked     | 2025-10-11 10:50:22 |
| <input type="checkbox"/> | <a href="#">Edit</a> <a href="#">Copy</a> <a href="#">Delete</a> | 2            | 1          | 1            | 1             | 1        | blocked    | blocked    | blocked     | 2025-10-11 10:50:22 |
| <input type="checkbox"/> | <a href="#">Edit</a> <a href="#">Copy</a> <a href="#">Delete</a> | 3            | 1          | 1            | 1             | 1        | blocked    | blocked    | blocked     | 2025-10-11 10:50:22 |
| <input type="checkbox"/> | <a href="#">Edit</a> <a href="#">Copy</a> <a href="#">Delete</a> | 4            | 1          | 1            | 1             | 1        | blocked    | blocked    | blocked     | 2025-10-11 10:50:22 |
| <input type="checkbox"/> | <a href="#">Edit</a> <a href="#">Copy</a> <a href="#">Delete</a> | 5            | 2          | 2            | 1             | 1        | blocked    | blocked    | blocked     | 2025-09-26 12:53:35 |
| <input type="checkbox"/> | <a href="#">Edit</a> <a href="#">Copy</a> <a href="#">Delete</a> | 6            | 2          | 2            | 1             | 1        | blocked    | blocked    | blocked     | 2025-09-26 12:53:35 |
| <input type="checkbox"/> | <a href="#">Edit</a> <a href="#">Copy</a> <a href="#">Delete</a> | 7            | 2          | 2            | 1             | 1        | blocked    | blocked    | blocked     | 2025-09-26 12:53:35 |

## dockets

Stores information about issued dockets, including the student, programme, exam type, and QR code data.

Showing rows 0 - 2 (3 total, Query took 0.0018 seconds.)

SELECT \* FROM `dockets`

Profiling [ Edit inline ] [ Edit ] [ Explain SQL ] [ Create PHP code ] [ Refresh ]

Show all | Number of rows: 25 | Filter rows: Search this table | Sort by key: None

Extra options

|                                           | docket_id | student_id | programme_id | exam_type | course_id | year_of_study | semester | qr_code                           |
|-------------------------------------------|-----------|------------|--------------|-----------|-----------|---------------|----------|-----------------------------------|
| <input type="checkbox"/> Edit Copy Delete | 1         | 1          | 1            | ca1       | NULL      | 0             | 0        | 104775_ca1_zLQRYI3ipOM7DwCCfO6H2g |
| <input type="checkbox"/> Edit Copy Delete | 2         | 1          | 1            | ca1       | NULL      | 0             | 0        | 104775_ca1_mvMD6cJUlJPJWJWtB3CMbg |
| <input type="checkbox"/> Edit Copy Delete | 3         | 1          | 1            | ca1       | NULL      | 0             | 0        | 104775_ca1_Qi8Duo7QRoxmct5N-jksCA |

Check all | With selected: Edit Copy Delete Export

Show all | Number of rows: 25 | Filter rows: Search this table | Sort by key: None

## docket\_tokens

Holds secure hashed tokens corresponding to each docket for validation and security verification.

Showing rows 0 - 2 (3 total, Query took 0.0007 seconds.)

SELECT \* FROM `docket\_tokens`

Profiling [ Edit inline ] [ Edit ] [ Explain SQL ] [ Create PHP code ] [ Refresh ]

Show all | Number of rows: 25 | Filter rows: Search this table | Sort by key: None

Extra options

|                                           | token_id | docket_id | token_hash                                            | issued_at           | expires_at | status | used_at |
|-------------------------------------------|----------|-----------|-------------------------------------------------------|---------------------|------------|--------|---------|
| <input type="checkbox"/> Edit Copy Delete | 1        | 1         | fd25671bceec8fb291ab766c0b0356b32cb53da2e936c17ee...  | 2025-10-09 11:33:32 | NULL       | active | NULL    |
| <input type="checkbox"/> Edit Copy Delete | 2        | 2         | d08a86c61771e0157d51ced13e71dceb2e91bb8213001a8d9...  | 2025-10-09 15:04:22 | NULL       | active | NULL    |
| <input type="checkbox"/> Edit Copy Delete | 3        | 3         | d5158a8fe246a985cd5337ccc941d6c576134eae0a6722ae7a... | 2025-10-09 15:26:07 | NULL       | active | NULL    |

Check all | With selected: Edit Copy Delete Export

## token\_keys

Contains master-level keys used for QR code verification and encryption processes.

Showing rows 0 - 0 (1 total, Query took 0.0006 seconds.)

SELECT \* FROM `token\_keys`

Profiling [ Edit inline ] [ Edit ] [ Explain SQL ] [ Create PHP code ] [ Refresh ]

Show all | Number of rows: 25 | Filter rows: Search this table

Extra options

|                                           | key_id | key_name                 | secret_key                               | created_at          | status |
|-------------------------------------------|--------|--------------------------|------------------------------------------|---------------------|--------|
| <input type="checkbox"/> Edit Copy Delete | 1      | default_verification_key | Gy2pDHLl8tYxnm3OgAj_rFImQ2pk5_zUS0USPFDQ | 2025-10-09 11:33:32 | active |

Check all | With selected: Edit Copy Delete Export

Show all | Number of rows: 25 | Filter rows: Search this table

## verifications

Logs every verification attempt, marking whether a docket was valid, reprinted, or flagged.

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0144 seconds.)

```
SELECT * FROM `verifications`
```

☐ Profiling [\[ Edit inline \]](#) [\[ Edit \]](#) [\[ Explain SQL \]](#) [\[ Create PHP code \]](#) [\[ Refresh \]](#)

**verification\_id docket\_id scanned\_by scanned\_at scan\_result device\_id remarks**

Query results operations

[Create view](#)

[Bookmark this SQL query](#)

Label:  ☐ Let every user access this bookmark

## admins

Stores system administrator credentials and permissions

Showing rows 0 - 1 (2 total, Query took 0.0008 seconds.)

```
SELECT * FROM `admins`
```

☐ Profiling [\[ Edit inline \]](#) [\[ Edit \]](#) [\[ Explain SQL \]](#) [\[ Create PHP code \]](#) [\[ Refresh \]](#)

☐ Show all | Number of rows: 25 | Filter rows:  | Sort by key: None

Extra options

|                                                                                           | admin_id | username  | password_hash                                            | role        | status | created_at          | updated_at          |
|-------------------------------------------------------------------------------------------|----------|-----------|----------------------------------------------------------|-------------|--------|---------------------|---------------------|
| <input type="checkbox"/> <a href="#">Edit</a> <a href="#">Copy</a> <a href="#">Delete</a> | 1        | REWARDSON | \$2b\$12\$ERdPCHmVWjVyxBC6wUyvR_dkjuu3hxa13AGqxQTg96m... | super_admin | active | 2025-10-09 11:18:59 | 2025-10-09 14:31:06 |
| <input type="checkbox"/> <a href="#">Edit</a> <a href="#">Copy</a> <a href="#">Delete</a> | 2        | DENISE    | \$2b\$12\$xQ4OBxk8ID77H.n5S2nJyuTxCC3iGky0.qul14eOs...   | super_admin | active | 2025-10-09 11:18:59 | 2025-10-09 14:31:07 |

[↑](#) ☐ Check all | With selected: [Edit](#) [Copy](#) [Delete](#) [Export](#)

☐ Show all | Number of rows: 25 | Filter rows:  | Sort by key: None

Query results operations

## Payments

Stores all payments made by the student on the frontend, that have been updated by the admin, Payments made are automatically. Updated in the Student balances.

Showing rows 0 - 10 (11 total, Query took 0.0017 seconds.)

SELECT \* FROM `payments`

☐ Profiling [\[ Edit inline \]](#) [\[ Edit \]](#) [\[ Explain SQL \]](#) [\[ Create PHP code \]](#) [\[ Refresh \]](#)

☐ Show all | Number of rows: 25 | Filter rows: Search this table | Sort by key: None

Extra options

|                                                                                           | payment_id | student_id | programme_id | course_id | amount   | payment_type | payment_date        | payment_status | receipt_number |
|-------------------------------------------------------------------------------------------|------------|------------|--------------|-----------|----------|--------------|---------------------|----------------|----------------|
| <input type="checkbox"/> <a href="#">Edit</a> <a href="#">Copy</a> <a href="#">Delete</a> | 1          | 1          | 1            | NULL      | 8000.00  | General      | 2025-10-11 09:18:11 | completed      | NULL           |
| <input type="checkbox"/> <a href="#">Edit</a> <a href="#">Copy</a> <a href="#">Delete</a> | 2          | 3          | 3            | NULL      | 11000.00 | General      | 2025-10-11 09:36:11 | completed      | NULL           |
| <input type="checkbox"/> <a href="#">Edit</a> <a href="#">Copy</a> <a href="#">Delete</a> | 3          | 1          | 1            | NULL      | 10000.00 | General      | 2025-10-11 09:42:19 | completed      | NULL           |
| <input type="checkbox"/> <a href="#">Edit</a> <a href="#">Copy</a> <a href="#">Delete</a> | 4          | 1          | 1            | NULL      | 8000.00  | General      | 2025-10-11 10:03:18 | completed      | NULL           |
| <input type="checkbox"/> <a href="#">Edit</a> <a href="#">Copy</a> <a href="#">Delete</a> | 5          | 1          | 1            | NULL      | 1000.00  | General      | 2025-10-11 10:12:14 | completed      | NULL           |
| <input type="checkbox"/> <a href="#">Edit</a> <a href="#">Copy</a> <a href="#">Delete</a> | 6          | 1          | 1            | NULL      | 1000.00  | General      | 2025-10-11 10:25:15 | completed      | NULL           |
| <input type="checkbox"/> <a href="#">Edit</a> <a href="#">Copy</a> <a href="#">Delete</a> | 7          | 1          | 1            | NULL      | 1000.00  | General      | 2025-10-11 10:34:30 | completed      | NULL           |
| <input type="checkbox"/> <a href="#">Edit</a> <a href="#">Copy</a> <a href="#">Delete</a> | 8          | 1          | 1            | NULL      | 1000.00  | General      | 2025-10-11          | completed      | NULL           |

Console [\[ Edit inline \]](#) [\[ Edit \]](#) [\[ Explain SQL \]](#) [\[ Create PHP code \]](#) [\[ Refresh \]](#)

8db=docket\_system38table=pay...

## Fee schedule

Keeps track of the percentage , a each student has to pay to be eligible for the clerances in order for them to have acess to there docket. 20 percent for CA1, 75 percent for CA2, and 100 percent for EXAMS.

Showing rows 0 - 3 (4 total, Query took 0.0001 seconds.)

SELECT \* FROM `fee\_schedule`

☐ Profiling [\[ Edit inline \]](#) [\[ Edit \]](#) [\[ Explain SQL \]](#) [\[ Create PHP code \]](#) [\[ Refresh \]](#)

☐ Show all | Number of rows: 25 | Filter rows: Search this table | Sort by key: None

Extra options

|                                                                                           | schedule_id | exam_type | required_percentage | created_at          |
|-------------------------------------------------------------------------------------------|-------------|-----------|---------------------|---------------------|
| <input type="checkbox"/> <a href="#">Edit</a> <a href="#">Copy</a> <a href="#">Delete</a> | 1           | CA1       | 20.00               | 2025-09-17 02:48:16 |
| <input type="checkbox"/> <a href="#">Edit</a> <a href="#">Copy</a> <a href="#">Delete</a> | 2           | CA2       | 75.00               | 2025-09-17 02:48:16 |
| <input type="checkbox"/> <a href="#">Edit</a> <a href="#">Copy</a> <a href="#">Delete</a> | 3           | EXAM      | 100.00              | 2025-09-17 02:48:16 |

☐ Check all | With selected: [Edit](#) [Copy](#) [Delete](#) [Export](#)

☐ Show all | Number of rows: 25 | Filter rows: Search this table | Sort by key: None

## 4. Table Relationships

The relationships among the tables are primarily based on student activity and verification of integrity:

- A student (students) belongs to a programme (programmes).
- Each programme includes multiple courses (courses) via the curriculum.

- Enrollments link students and their respective courses.
- Clearances record eligibility for CA1, CA2, and final exams.
- Dockets are generated for eligible students, linked by `student_id` and `programme_id`.
- Each docket has a corresponding token in `docket_tokens`, hashed for security.
- `Token_keys` store master verification keys for generating and validating QR codes.
- Verifications log all QR code scans and validation attempts.
- Admins, users, and logs tables track authentication and system activities.

## 5. Database and Backend Integration

The Flask backend interacts with this MySQL database through secure connectors and environment variables. Key endpoints handle issuing, storing, and verifying dockets. Each docket generated is stored in the `dockets` and `docket_tokens` tables, while verifications are logged in the `verifications` table. `Token_keys` are used to cross-check authenticity during QR scans, preventing forgery or reuse of old tokens. When a student reprints a docket, the system invalidates the previous token and generates a new one, ensuring unique verification integrity.

The database schema enforces ACID properties (Atomicity, Consistency, Isolation, Durability) and uses foreign keys to maintain relational integrity. Indexing is applied to frequently queried columns like `student_number`, `docket_id`, and `token_hash` for optimized performance.

## 6. Security and Data Integrity

To maintain security and data integrity:

- Tokens are hashed using SHA-256.
- QR codes are generated dynamically per docket.
- Verification requests log timestamps and IPs.
- Access to the database is protected via environment variables and restricted credentials.

The Docket Management System Database ensures data consistency, security, and smooth integration between students, administrators, and verification officers.

With its relational structure and security mechanisms, it prevents forgery, ensures verifiable records, and supports scalability for future university needs.

## 5. Project Conclusion

The Cavendish University Docket Management System represents a holistic and robustly engineered solution to a significant administrative challenge. By systematically digitizing the process of exam docket issuance and verification, the project successfully replaces a manual, paper-based workflow with a secure, efficient, and user-centric digital ecosystem. This conclusion synthesizes the architectural design, key technical implementations, and the synergistic relationship between the system's three core pillars: the database, the backend, and the frontend.

### The Architectural Synergy

The success of this system lies not in any single component, but in the seamless integration of its three distinct layers, each designed with a clear separation of concerns.

### The Database: The Foundation and Single Source of Truth

At its core, the MySQL database serves as the authoritative foundation for all operations. The relational schema is intelligently designed to model the real-world entities and business rules of the university. Tables such as students, courses, programmes, and enrollments establish the academic context. More critically, the clearances and student\_balances tables directly implement the financial business logic, providing a clear, queryable source for student eligibility.

The database design extends beyond simple data storage to become an active participant in the application's security and integrity. The docket\_tokens table, which stores only the secure hashes of verification tokens, and the verifications table, which provides a permanent audit trail of all scanning activities, are testaments to a security-first mindset. The database is not merely a passive repository; it is the bedrock upon which all application logic is reliably and securely built.

## The Backend: The Central Nervous System

The Python Flask backend acts as the central nervous system of the application. It is the engine that executes all business logic, enforces security, and serves as the vital intermediary between the raw data in the database and the user's interactions on the frontend. Its architecture is defined by modularity and security.

- **Modularity:** The use of Flask Blueprints (dockets, verification, admin\_controls) effectively compartmentalizes the application's features. This separation makes the codebase easier to understand, maintain, and extend.
- **Security:** The backend is the primary guardian of the system's integrity. Security is not an afterthought but is woven into the fabric of the application through multiple layers:
  - **Authentication & Authorization:** JSON Web Tokens (JWT) provide a stateless, secure method for authenticating users and managing sessions, with role-based access control strictly enforced by the `jwt_required` decorator.
  - **Data Protection:** Passwords are never stored in plaintext, instead utilizing the industry-standard bcrypt hashing algorithm. Likewise, docket verification tokens are hashed with SHA-256 before being stored.
  - **Concurrency Control:** The implementation of SELECT ... FOR UPDATE within the verification process is a sophisticated and crucial feature that prevents race conditions and ensures that a single docket cannot be fraudulently used multiple times in quick succession.
- **API Abstraction:** The backend exposes a clean, well-defined RESTful API that the frontend consumes. This abstracts away the complexity of the database, providing simple, secure endpoints for the client-side application to perform its functions.

## The Frontend: The Intuitive User Interface

The frontend, built with standard HTML, CSS, and JavaScript, is the face of the system and is meticulously tailored to its different user groups. It translates the powerful backend functionality into an accessible and intuitive experience.

- **Student Portal:** This interface is designed for clarity and ease of use, allowing students to effortlessly view their eligibility status and generate their dockets with a single click. It empowers students by giving them direct, on-demand access to critical examination documents.
- **Admin Dashboard:** This serves as the system's command center, providing administrators with the tools needed to manage the examination cycle. Functionality such as setting the active exam and blocking students is made simple through a dedicated user interface, which leverages the backend's file-based configuration for real-time control.
- **Verification PWA:** Perhaps the most innovative aspect of the frontend is the QR code verification tool, designed as a Progressive Web App (PWA). This demonstrates a deep understanding of the end-user's environment, where internet connectivity may not always be reliable in an examination hall. By syncing student and token data for offline access, the PWA ensures that the verification process is resilient and can function seamlessly under any network conditions.

## Final Synthesis

In conclusion, the Docket Management System is a successful and well-rounded software engineering project. It demonstrates a comprehensive understanding of full-stack development, from relational database design and secure backend architecture to user-focused frontend implementation. The project's strengths lie in its robust security measures, its modular and maintainable codebase, and its thoughtful consideration of the end-user experience, particularly with the inclusion of offline capabilities. It stands as a complete, functional, and secure solution that effectively addresses and solves the complex challenges of managing and verifying examination dockets in a modern academic institution.

## **6. References and Further Reading**

For a deeper understanding of the core technologies and concepts employed in this project, the following resources are recommended:

- Flask Web Framework: Flask Documentation (<https://flask.palletsprojects.com/>)
- JSON Web Tokens (JWT): RFC 7519 (<https://tools.ietf.org/html/rfc7519>)
- Password Hashing: Bcrypt Hashing Function (<https://en.wikipedia.org/wiki/Bcrypt>)
- Progressive Web Apps (PWA): MDN Web Docs ([https://developer.mozilla.org/en-US/docs/Web/Progressive\\_web\\_apps](https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps))
- ReportLab PDF Library: ReportLab Documentation (<https://www.reportlab.com/docs/reportlab-userguide.pdf>)
- Database Transaction Isolation: MySQL Documentation on SELECT ... FOR UPDATE (<https://dev.mysql.com/doc/refman/8.0/en/innodb-locking-reads.html>)

