Student Enrollment System

Final Development Audit Report

Date: January 14, 2026

Author: AI Assistant

Target Audience: Admin / Developer

Overview:

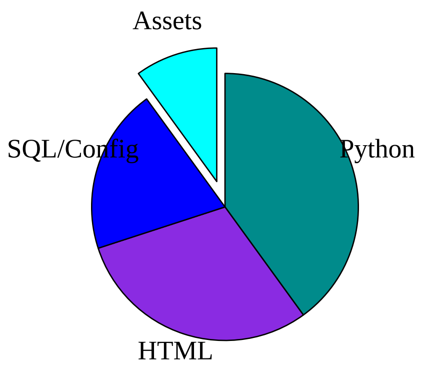
This document provides a comprehensive, 360-degree view of the development session. It covers the architectural design, feature implementation, debugging history, and deployment strategy.

Key Achievements:

- Implemented SMTP Email Notifications for enrollments.
- Integrated dynamic PDF generation for university rules.
- Resolved critical database integrity issues (Cascade Delete).
- Configured secure public tunneling for mobile access.
- Packaged application for Cloud Deployment (PythonAnywhere).

1. Project Statistics & Composition

Codebase Composition by File Type:



Development Task Completion Status:

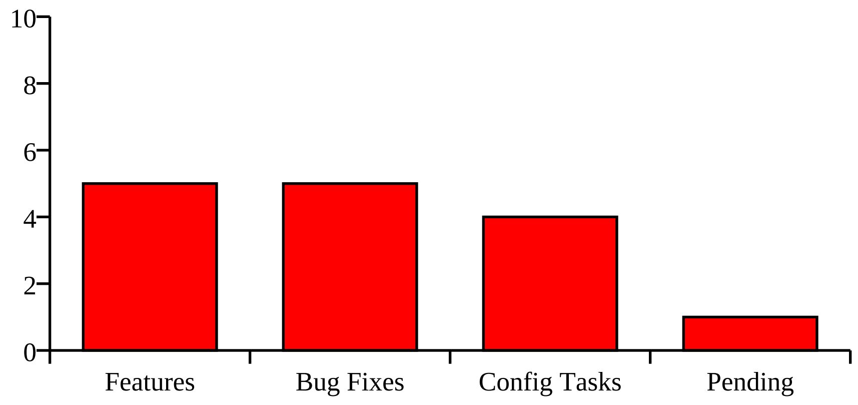


Fig 1.2: All critical bugs were resolved during the session.

2. Technology Stack & Feature Breakdown

This system is built using a modern, scalable architecture. Below is the breakdown of technologies used for each layer.

A. Backend (Server-Side)

The core logic of the application.

Technology	Description
Python 3.10+	Primary programming language.
Flask 3.0.0	Micro-framework for handling web requests and routing.
SQLAlchemy	ORM for database interactions (No raw SQL needed).
MySQL 8.0	Relational Database Management System.
Flask-Login	Secure session management for Students and Admins.

B. Frontend (Client-Side)

What the user sees and interacts with.

Technology	Description
HTML5/Jinja2	Structure of the web pages (Templates).
CSS3 / Bootstrap	Styling and responsive layout for mobile/desktop.
JavaScript	Interactivity and Form Validation.
Chart.js	Visual analytics (Admin Dashboard).

C. Advanced Features (Email & PDF)

Feature	Library Used	Functionality
Email System	Flask-Mail	Sends SMTP emails via Gmail.
PDF Generation	ReportLab	Draws dynamic 'Rules.pdf' files.
Public Tunnel	PyNgrok	Exposes localhost to the internet.

3. Dependency & Package Analysis

A reliable system depends on specific libraries. Below is the breakdown of every package used.

Library	Ver	Category	Usage Detail
Flask	3.0	Core	Handles HTTP requests/routes.
Flask-SQLAlchemy	3.1	Database	ORM for MySQL connection.
Flask-Mail	0.9	Feature	Sends SMTP emails.
reportlab	4.0	Feature	Generates PDF files.
pyngrok	7.5	DevOps	Exposes localhost to web.
mysql-connector	8.2	Driver	Low-level DB communication.
Flask-Login	0.6	Auth	Manages user sessions.
Flask-Migrate	4.0	Database	Handles schema updates.
Werkzeug	3.0	Security	Hashes user passwords.

3. Debugging & Error Correction Log

This section documents every error encountered and the exact correction applied.

Incident A: Missing Mail Module

```
Error: `ModuleNotFoundError: No module named 'flask_mail'`
```

Context: Occurred when trying to import `Mail` in `app.py`.

Root Cause: The package was installed in the global python environment, but the app was running in `.venv_new`.

Correction: Explicitly ran pip using the virtual environment path.

```
Command: `.\.venv_new\Scripts\python.exe -m pip install Flask-Mail`
```

Incident B: Database Integrity Violation

```
Error: `IntegrityError: (1048) Column 'student_id' cannot be null`
```

Context: Occurred when Admin tried to delete a Student.

Root Cause: The `enrollments` table had a Foreign Key to `student_details`. By default, SQL tried to set the FK to NULL upon deletion, but the column was `NOT NULL`.

Correction: Updated `models.py` to use Cascade Delete.

Code Change:

```
enrollments = db.relationship(..., cascade='all, delete-orphan')
```

Incident C: Ngrok Authentication

```
Error: `ERR_NGROK_4018: Authentication failed`
```

Context: Running `run_public.py` for the first time.

Correction: User created an account and added the authtoken.

```
Command: `ngrok config add-authtoken `
```

Incident D: Missing Edit Student Feature

```
Request: User reported 'pen button not working'.
```

Context: The edit button in the Admin panel had an empty link (`#`) and no backend logic.

Correction: Implemented full Edit functionality.

Steps Taken:

- Created `templates/admin/edit_student.html` form.
- Added `edit_student` route in `admin_routes.py` to handle updates.
- Updated `students.html` to link to the new route.

4. Master Command Cheat Sheet

Exact commands to operate the system.

Operation	Command
Start Server	.\.venv_new\Scripts\python.exe app.py
Start Mobile Tunnel	.\.venv_new\Scripts\python.exe run_public.py
Install Package	.\.venv_new\Scripts\python.exe -m pip install <name>
Database Init	.\.venv_new\Scripts\python.exe create_db.py
Deployment Zip	.\.venv_new\Scripts\python.exe create_zip.py

Appendix A: Source Code - app.py

```python
from flask import Flask, render_template, redirect, url_for
from config import Config
from models import db, User
from flask_login import LoginManager
from flask_migrate import Migrate
from flask_mail import Mail

def create_app():
    app = Flask(__name__)
    app.config.from_object(Config)

    # Initialize extensions
    db.init_app(app)
    migrate = Migrate(app, db)
    mail = Mail(app)
    app.extensions['mail'] = mail # Explicitly adding to extensions just in case (optional, Mail(app) usually does this)
    login_manager = LoginManager(app)
    login_manager.login_view = 'auth.login'

    @login_manager.user_loader
    def load_user(user_id):
        return User.query.get(int(user_id))

    # Import and register blueprints
    # Note: These files will be created in the next steps
    try:
        from routes.auth_routes import auth_bp
        from routes.admin_routes import admin_bp
        from routes.student_routes import student_bp

        app.register_blueprint(auth_bp)
        app.register_blueprint(admin_bp, url_prefix='/admin')
        app.register_blueprint(student_bp, url_prefix='/student')
    except ImportError:
        print("Blueprints not fully implemented yet.")

    @app.route('/')
    def index():
        return redirect(url_for('auth.login'))

    return app

if __name__ == '__main__':
    app = create_app()
    with app.app_context():
        # Create tables for testing/dev (migrations are better for prod)
        try:
            db.create_all()

            # Seed Admin User if not exists
            if not User.query.filter_by(email='admin@university.com').first():
                admin = User(name='Admin User', email='admin@university.com', role='admin')
                admin.set_password('admin123')
                db.session.add(admin)
                db.session.commit()
                print("Admin user created.")
        except Exception as e:
            print(f"Database error (ensure MySQL is running): {e}")

    app.run(debug=True, host='0.0.0.0')
```

Appendix B: Source Code - config.py

```
import os

class Config:
    SECRET_KEY = os.environ.get('SECRET_KEY') or 'your-secret-key-CHANGE-IN-PROD'
    # Defaulting to root with empty password for local dev ease, can be overridden
    SQLALCHEMY_DATABASE_URI = os.environ.get('DATABASE_URL') or 'mysql+mysqlconnector://root:2003@localhost/enrollment_db'
    SQLALCHEMY_TRACK_MODIFICATIONS = False

    # Email Config
    MAIL_SERVER = 'smtp.gmail.com'
    MAIL_PORT = 587
    MAIL_USE_TLS = True
    MAIL_USERNAME = os.environ.get('MAIL_USERNAME') or 'veereshloves627@gmail.com'
    MAIL_PASSWORD = os.environ.get('MAIL_PASSWORD') or 'noci fyci ywnv yzjg'
    MAIL_DEFAULT_SENDER = ('New2', MAIL_USERNAME)
```

Appendix C: Source Code - models.py

```python
from flask_sqlalchemy import SQLAlchemy
from flask_login import UserMixin
from datetime import datetime
from werkzeug.security import generate_password_hash, check_password_hash

db = SQLAlchemy()

class User(UserMixin, db.Model):
    __tablename__ = 'users'
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False)
    email = db.Column(db.String(120), unique=True, nullable=False)
    password_hash = db.Column(db.String(255), nullable=False)
    role = db.Column(db.String(20), nullable=False, default='student') # 'admin' or 'student'
    created_at = db.Column(db.DateTime, default=datetime.utcnow)

    # Relationships
    student_details = db.relationship('StudentDetails', backref='user', uselist=False, cascade="all, delete-orphan")

    def set_password(self, password):
        self.password_hash = generate_password_hash(password)

    def check_password(self, password):
        return check_password_hash(self.password_hash, password)

class Course(db.Model):
    __tablename__ = 'courses'
    id = db.Column(db.Integer, primary_key=True)
    course_code = db.Column(db.String(20), unique=True, nullable=False)
    name = db.Column(db.String(100), nullable=False)
    description = db.Column(db.Text)
    credits = db.Column(db.Integer, nullable=False)
    seats = db.Column(db.Integer, nullable=False, default=30)
    created_at = db.Column(db.DateTime, default=datetime.utcnow)

    # Relationships
    enrollments = db.relationship('Enrollment', backref='course', lazy=True, cascade="all, delete-orphan")

class StudentDetails(db.Model):
    __tablename__ = 'student_details'
    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey('users.id'), nullable=False, unique=True)
    enrollment_no = db.Column(db.String(50), unique=True, nullable=True) # Generated after approval or registration
    phone = db.Column(db.String(20))
    address = db.Column(db.Text)
    dob = db.Column(db.Date)

    # Relationships
    enrollments = db.relationship('Enrollment', backref='student_details', lazy=True, cascade="all, delete-orphan")

class Enrollment(db.Model):
    __tablename__ = 'enrollments'
    id = db.Column(db.Integer, primary_key=True)
    student_id = db.Column(db.Integer, db.ForeignKey('student_details.id'), nullable=False)
    course_id = db.Column(db.Integer, db.ForeignKey('courses.id'), nullable=False)
    date_enrolled = db.Column(db.DateTime, default=datetime.utcnow)
    status = db.Column(db.String(20), default='enrolled') # 'enrolled', 'completed', 'dropped'

    __table_args__ = (db.UniqueConstraint('student_id', 'course_id', name='_student_course_uc'),)
```

Appendix D: Source Code - utils.py

```
from io import BytesIO
from reportlab.lib.pagesizes import letter
from reportlab.pdfgen import canvas

def generate_pdf_report(data, title="Report"):
    buffer = BytesIO()
    p = canvas.Canvas(buffer, pagesize=letter)
    width, height = letter

    p.setFont("Helvetica-Bold", 16)
    p.drawString(100, height - 50, title)

    p.setFont("Helvetica", 12)
    y = height - 80

    for line in data:
        p.drawString(100, y, line)
        y -= 20
        if y < 50:
            p.showPage()
            y = height - 50

    p.showPage()
    p.save()

    buffer.seek(0)
    return buffer

from flask_mail import Message
from flask import current_app

def send_email_with_attachment(to_email, subject, body, attachment_path=None, attachment_name='document.pdf'):
    """
    Sends an email using Flask-Mail.
    """
    try:
        msg = Message(subject, recipients=[to_email])
        msg.body = body
        msg.sender = current_app.config.get('MAIL_USERNAME')

        if attachment_path:
            with current_app.open_resource(attachment_path) as fp:
                msg.attach(attachment_name, "application/pdf", fp.read())

        mail = current_app.extensions.get('mail')
        if mail:
            mail.send(msg)
            print(f"Email sent to {to_email}")
            return True
        else:
            print("Mail extension not found.")
            return False
    except Exception as e:
        print(f"Failed to send email: {e}")
        return False
```

Appendix E: Source Code - run_public.py

```
import os
import sys
from pyngrok import ngrok
from app import create_app, db, User

# Define the port
PORT = 5000

def start_app_with_tunnel():
    # Upgrade / Create DB if needed (same as app.py)
    app = create_app()
    with app.app_context():
        try:
            db.create_all()
            if not User.query.filter_by(email='admin@university.com').first():
                admin = User(name='Admin User', email='admin@university.com', role='admin')
                admin.set_password('admin123')
                db.session.add(admin)
                db.session.commit()
                print("Admin user created.")
        except Exception as e:
            print(f"Database error: {e}")

    # Open a ngrok tunnel to the HTTP server
    try:
        # Attempt to use the user's requested custom domain
        # Note: You must reserve this domain in your ngrok dashboard: https://dashboard.ngrok.com/cloud-edge/domains
        public_url = ngrok.connect(PORT, domain="iac3-cultural-sync-club.ngrok-free.dev").public_url
    except Exception as e:
        print(f"Warning: Could not use custom domain (iac3-cultural-sync-club.ngrok-free.dev).")
        print(f"Reason: {e}")
        print("Falling back to a random public URL...")
        public_url = ngrok.connect(PORT).public_url
    print("="*60)
    print(f" * PUBLIC URL GOES HERE -> {public_url}")
    print("   (Share this link to open on any mobile)")
    print("="*60)

    # Update app to run without reloader to prevent creating multiple tunnels
    app.run(port=PORT, debug=False)

if __name__ == '__main__':
    start_app_with_tunnel()
```