(1) Voice Bot with Visual Avatar for IT Department Assistance

---

1. Introduction

Managing frequent queries from students, parents, and visitors in an academic department is often repetitive and time-consuming. Common questions such as *faculty seating location, whom to contact for scholarships, exam seating arrangements, lab schedules, and placement details* usually burden staff and faculty.

This project aims to build an **AI-powered Voice Bot with a Visual Character** that can interact naturally with users, answer FAQs, and guide them with proper information. The bot will be deployed as a **Python desktop application with GUI** and can be optionally extended as a **Flask-based web/mobile app** for real-time student/visitor assistance.

The interactive **visual avatar (character)** will make the experience engaging, especially for first-time visitors and parents.

---

2. Objectives

- Provide an intelligent voice bot that can answer frequently asked questions in the IT department.
- Support both **voice input/output** and **text-based chat**.
- Display a **visual character/avatar** synchronized with speech.
- Offer a searchable knowledge base (faculty seating, scholarships, exam details, etc.).
- Extend via Flask for **web/mobile usage** in real-time.

---

3. Features

🔷 Python Desktop Version (Main Requirement)

1. **Input Sources:**
   - Voice (Microphone input → Speech Recognition).
   - Text (User can also type query).
2. **Voice Bot Features:**

    - Speech-to-Text (Google SpeechRecognition / Vosk).
    - Query processing (Rule-based + FAQ database + NLP).
    - Text-to-Speech (pyttsx3 / gTTS).
3. **Visual Character (Avatar):**
    - 2D animated character (Tkinter/PyQt canvas with simple mouth animations).
    - Avatar lip-syncs with text-to-speech.
    - Expressions (smile, blink) for engagement.
4. **Knowledge Base (FAQ System):**
    - Faculty seating chart (Room no. → Faculty name).
    - Scholarship guidance (Whom to meet / Office location).
    - Exam seating arrangements (Uploaded CSV/PDF lookup).
    - Placement/Training details (Coordinator info).
    - Department facilities (Labs, classrooms).
    - General IT dept. info (HoD, office timings, etc.).
    - Stored in SQLite / JSON for easy updates.
5. **GUI Dashboard (Tkinter/PyQt):**
    - Chat window (text + voice support).
    - Avatar display with animations.
    - Current query + answer display.
    - Admin panel to update FAQs.
6. **Reports & Logs:**
    - Save all queries in SQLite for analytics.
    - Show most frequently asked questions.

🔷 Flask/Mobile App Extension (Optional)

1. **Web Dashboard:**
    - Text/voice-based query system.
    - Avatar displayed in web (WebGL / animated GIF / video character).
2. **Mobile Features:**
    - Ask FAQ queries from phone.
    - Get instant answers (voice + text).
    - View faculty seating map & exam arrangements.
3. **Optional AI Features:**
    - NLP-powered question answering (Hugging Face / spaCy).
    - Intent recognition for new/unstructured queries.
    - Predictive answering (learns from repeated questions).

---

4. Technology Stack

🔷 Python (Desktop Version)

- **Speech Recognition**: Google SpeechRecognition, Vosk, or Whisper.
- **Text-to-Speech**: pyttsx3, gTTS.
- **Visual Avatar**: Tkinter/PyQt + OpenCV (simple face animations).
- **Database**: SQLite (FAQ + logs).
- **NLP/AI**: NLTK / spaCy (for query processing).
- **GUI**: Tkinter or PyQt5.

🔷 Flask/Mobile Version

- Flask → Backend.
- HTML/CSS/JS → Frontend.
- Web Speech API → Voice input/output.
- Chart.js/D3.js → Analytics (FAQ trends).

- SQLite/PostgreSQL → Knowledge base & logs.

---

5. Workflow (4-Day Roadmap)

Day 1 – Voice Bot Basics

- Implement Speech-to-Text and Text-to-Speech pipeline.
- Build simple rule-based responses from FAQ database.

Day 2 – Visual Avatar & GUI

- Create GUI with chat window.
- Add animated avatar character synced with speech.
- Connect query input/output to GUI.

Day 3 – Knowledge Base & Query Processing

- Build SQLite FAQ database (faculty seating, scholarships, exam details, etc.).
- Implement query matching (exact + fuzzy search).
- Store logs of all interactions.

Day 4 – Reports & Extensions

- Show analytics (most asked questions).
- Add PDF/CSV import for exam seating data.
- Polish GUI (admin panel, alerts).

✅ **Extra Credit (Flask Extension):**

- Deploy web version with voice support.
- Add mobile access for students/parents.

---

6. Expected Outcomes

- A Python desktop application with:
    - Voice-enabled chatbot.

    - Interactive visual avatar.
    - Department-specific FAQ answering.
    - Admin panel to update FAQs.
    - Logs & analytics of queries.
- An extended Flask-based web/mobile app for broader usage.

---

7. Possible Enhancements

- Multi-language support (English, Hindi, Gujarati, etc.).
- More realistic 3D avatar integration (Unity/Blender → WebGL).
- Integration with **college website / student portal**.
- Live map of department seating chart.
- ML-based intent recognition to answer unseen questions.
- Integration with WhatsApp/Telegram bot for FAQs.

(2) Blockchain-Based Digital Certification System with Automated Email Distribution

---

1. Introduction

Traditional event certification often faces challenges like manual distribution, duplication, forgery, and lack of a secure verification mechanism. With growing academic, technical, and cultural events, it is essential to provide participants with **tamper-proof digital certificates** that can be **easily validated online**.

This project proposes a **Blockchain-integrated Digital Certification System** where certificates are automatically generated for each participant, distributed via email, and validated using blockchain. The blockchain ensures **immutability and authenticity**, preventing fraudulent claims, while automated distribution enhances efficiency.

---

2. Objectives

- Generate personalized **digital certificates** for event participants.
- Distribute certificates automatically to participants' email IDs.
- Store certificate hashes on **blockchain for validation**.
- Provide a **web-based validation system** for third parties to verify authenticity.
- Ensure scalability for different academic/cultural/technical events.

---

3. Features

🔷 **Certificate Creation & Distribution**

1. **Certificate Template Management**
   - Upload or design event-specific certificate templates.
   - Add placeholders (Name, Event Name, Date, Achievement, Signature).
2. **Bulk Certificate Generation**
   - Import participant list (CSV/Excel).

   - Auto-generate certificates with participant details.
3. **Automated Email Distribution**
   - Send certificates as **PDF attachments** to participant email IDs.
   - Email content customization (event name, thank-you note).

🔷 Blockchain-Based Validation

1. **Certificate Hashing**
   - Generate SHA-256 hash of each certificate.
   - Store hashes on blockchain (Ethereum/Hyperledger/Polygon testnet).
2. **Validation Portal**
   - Web page where certificate ID/QR code can be entered.
   - System compares certificate hash with blockchain record.
   - Displays **valid/invalid** certificate status.
3. **QR Code Integration**
   - QR code printed on each certificate.
   - Scanning redirects to validation portal.

🔷 Admin Dashboard

- Event organizer login.
- Upload participants & generate certificates.
- View certificate issuance logs.
- Analytics → No. of certificates issued, validated, etc.

🔷 Optional Enhancements

- Multi-event management (different templates for different events).

- API integration for external verification (e.g., LinkedIn certificate sharing).
- SMS-based certificate delivery link.

---

4. Technology Stack

🔷 Certificate Generation & Distribution

- Python (reportlab / PIL / fpdf2) → Certificate creation.
- SMTP / Gmail API → Email distribution.
- SQLite/PostgreSQL → Participant & certificate records.

🔷 Blockchain Integration

- Web3.py → Interaction with Ethereum/Polygon blockchain.
- Ganache/Truffle (local blockchain test environment).
- Smart Contracts → Immutable certificate storage.

🔷 Web Validation Portal

- Flask/Django → Backend.
- HTML/CSS/JS (Bootstrap) → Frontend.
- QR Code → Python qrcode library for generation.

---

5. Workflow (4-Day Roadmap)

Day 1 – Certificate Generation

- Design certificate template with placeholders.
- Implement Python script for bulk certificate generation.

Day 2 – Email Distribution & Database

- Integrate SMTP for sending certificates via participant emails.
- Store issued certificate records in SQLite/PostgreSQL.

Day 3 – Blockchain Hashing & Smart Contract

- Generate certificate hash and deploy smart contract.
- Store issued certificate hashes on blockchain.

Day 4 – Validation System & QR Code

- Build Flask-based validation portal.
- Generate QR codes linked to validation page.
- Integrate certificate verification with blockchain.

✅ **Extra Credit:**

- Build organizer dashboard.
- Add participant self-check verification tool.

---

6. Expected Outcomes

- **Digital certificates** auto-generated for each participant.
- **Automated distribution** to participant emails.
- **Immutable blockchain record** for authenticity.
- **Validation portal** where anyone can check certificate validity via ID/QR code.
- Secure, scalable system for academic & professional events.

---

7. Possible Enhancements

- Multi-language certificate generation.
- Blockchain-based credential wallet for participants.
- Integration with university ERP system.
- AI-based certificate design suggestion tool.
- NFT-style unique certificate issuance.

(3) Project Definition: Driver Drowsiness Detection in Python (with GUI, Flask Extension)

1. Introduction

Driver drowsiness is one of the leading causes of road accidents. Detecting signs of fatigue such as prolonged eye closure, yawning, or head tilting can help prevent accidents. This project focuses on building a **Python-based drowsiness detection system with a GUI**, which can later be extended into a **Flask-based web/mobile app**.

2. Objectives

1. Develop a real-time drowsiness detection system in **Python**.
2. Provide a **GUI interface** for easy interaction.
3. Alert the driver when drowsiness is detected.
4. Extend the Python project using **Flask**, making it accessible via a **mobile browser or app**.

3. Features

Python Desktop Version (Main Requirement)

- **Face & Eye Detection:** Detect driver’s face and monitor eyes using OpenCV/MediaPipe.
- **Eye Blink & Closure Detection:** Use Eye Aspect Ratio (EAR) to identify drowsiness.
- **Yawning Detection (Optional):** Detect if the mouth is open too long.
- **Drowsiness Alert:** Play alarm sound + show warning in GUI.
- **GUI Panel (Tkinter/PyQt):**
    - Start Monitoring / Stop Monitoring
    - Show Live Camera Feed
    - Display Status → *"Awake"* or *"Drowsy"*

Flask Extension (For Web/Mobile Use)

- **Webcam Feed in Browser:** Use Flask + OpenCV to stream live camera detection.
- **API Endpoint:** Expose REST API that returns detection results (Awake/Drowsy).
- **Mobile Access:** Open the Flask app from mobile browser → works like a mobile app.
- **Optional PWA (Progressive Web App):** Make it installable like a mobile app.

4. Technology Stack

- **Python 3.x**
- **Libraries:** OpenCV, dlib/mediapipe, numpy, playsound, tkinter/PyQt (for GUI)
- **Flask:** For web/mobile access
- **Frontend (Optional Flask UI):** HTML, CSS, JS for better mobile experience

5. Workflow

1. **Python GUI Phase (Competition Deliverable):**
   - Build detection logic (face, eyes, EAR).
   - Integrate with GUI (Tkinter or PyQt).
   - Add sound alerts.
2. **Flask Extension Phase (Mobile Ready):**
   - Wrap detection code into Flask app.
   - Stream video feed via Flask (cv2.VideoCapture + Flask Response).
   - Test app on local mobile device using http://<IP>:5000.

6. Expected Outcomes

- A **Python desktop app with GUI** that detects driver drowsiness in real-time.
- An extended **Flask version** that runs in a browser (mobile/desktop).
- A potential roadmap for converting it into a **full mobile app**.

(4) Project Definition: Background Removal from Video

---

1. Introduction

Background removal is widely used in **video conferencing, filmmaking, gaming, and AR/VR**. This project aims to build a **Python-based application** that removes or replaces the background of a video in real-time. The system can later be extended into a **mobile/web application** so users can try it on their phones.

---

2. Objectives

- Develop a **Python application** that removes/replaces video backgrounds.
- Implement **real-time processing** using webcam or pre-recorded videos.
- Provide a **GUI** for easy usage (upload video, choose background).
- Extend it into a **Flask/mobile app** for portability.

---

3. Features

Python Desktop Version (Main Requirement)

1. **Video Input:**
    - Webcam (real-time)
    - Uploaded video file
2. **Background Removal Methods:**
    - Basic: Color-based (green screen removal with OpenCV)
    - Intermediate: Background subtraction (cv2.createBackgroundSubtractorMOG2)
    - Advanced: AI-based segmentation (MediaPipe Selfie Segmentation or U^2-Net model)
3. **Background Options:**
    - Blur the background
    - Replace with an image (e.g., office background)
    - Replace with a video
4. **GUI (Tkinter/PyQt):**
    - Buttons → Upload Video / Start Webcam / Select Background / Save Output
    - Preview Window for processed video

Flask/Mobile App Extension

1. **Flask Backend:**
   - Upload video through browser/mobile
   - Run background removal using the same Python model
   - Stream processed video back in real-time
2. **Mobile App (Options):**
   - Use Flask app in mobile browser (acts like an app)
   - OR convert Flask app into **PWA (Progressive Web App)**
   - OR use **Flutter + TensorFlow Lite/MediaPipe** for fully native mobile background removal
3. **Mobile Features:**
   - Choose live camera feed or upload video
   - Select replacement background
   - Save processed video to gallery

4. Technology Stack

🔷 Python

- **Libraries:** OpenCV, MediaPipe, NumPy, Tkinter/PyQt (GUI), MoviePy (video saving)
- **Optional ML Models:** U^2-Net (PyTorch-based), DeepLabV3

🔷 Flask/Mobile

- **Flask:** For web service + API
- **Frontend:** HTML/CSS/JS for UI
- **Mobile:** Flutter/React Native OR Flask PWA
- **AI:** TensorFlow Lite / MediaPipe segmentation for mobile efficiency

5. Workflow

1. **Python Phase (Competition):**
   - Implement background removal using OpenCV/MediaPipe
   - Build GUI for video upload, processing, and saving
   - Test on different videos
2. **Flask Extension:**
   - Create API to handle video upload & processing
   - Return processed video stream to browser
   - Test on mobile by accessing Flask server (http://<IP>:5000)
3. **Mobile App (Optional):**
   - Integrate model into Flutter/React Native
   - Provide real-time background replacement

6. Expected Outcomes

- A working **Python desktop app** with GUI for background removal.
- A **Flask web app** accessible from desktop/mobile browsers.
- Optional: **Mobile app version** with real-time background replacement.

(5) Project Definition: Automatic Number Plate Recognition (ANPR)

---

1. Introduction

Automatic Number Plate Recognition (ANPR) is a computer vision-based technology that detects vehicles, localizes license plates, and extracts plate numbers using Optical Character Recognition (OCR). It is widely used in **traffic monitoring, toll collection, parking management, and law enforcement**.

This project aims to build a **Python-based ANPR system with a GUI** that can detect and recognize vehicle license plates from images or videos. Students will also extend it into a **Flask-based web application**, allowing mobile devices to upload vehicle images and get recognized plate numbers.

---

2. Objectives

- Detect vehicles and their license plates in real-time or from uploaded media.
- Apply OCR (Optical Character Recognition) to extract alphanumeric characters.
- Provide a **user-friendly GUI** for uploading images/videos and viewing results.
- Store recognized license numbers in a database for record-keeping.
- Extend the system using **Flask** to make it accessible from mobile devices.

---

3. Features

🔷 Python Desktop Version (Main Requirement)

1. **Input Options:**
   - Upload an image of a car
   - Upload a video (CCTV footage, dashcam, etc.)
   - Real-time detection using webcam
2. **Vehicle & Plate Detection:**
   - Use **OpenCV Haar cascades** or **YOLO-lite** for plate localization.
   - Draw bounding boxes around detected plates.
3. **OCR Extraction:**
   - Apply **Tesseract OCR** to recognize characters from extracted plate region.

    - Preprocess plate image (grayscale, thresholding, contour detection) to improve accuracy.
4. **GUI (Tkinter or PyQt):**
    - Buttons → Upload Image, Upload Video, Start Webcam, Save Output
    - Display → Original image/video with detected plate + extracted number
    - Output → Recognized license number displayed in text box
5. **Database Integration:**
    - Save recognized license numbers with timestamp in **SQLite or CSV**
    - Option to export results for reporting

🔷 Flask/Mobile App Extension

1. **Web-based API:**
    - Upload vehicle image from browser/mobile
    - Server processes image → detects license plate → extracts number
    - Return recognized number as JSON or display in UI
2. **Mobile Accessibility:**
    - Access Flask app on mobile (http://<IP>:5000)
    - Upload photos directly from phone gallery/camera
    - Get recognized number instantly
3. **Optional Features:**
    - Multiple language plate recognition (if dataset available)
    - Integration with GPS/location data

---

4. Technology Stack

🔷 Python (Desktop Version)

- **Language:** Python 3.x
- **Libraries:**
    - OpenCV → Vehicle & plate detection
    - PyTesseract → OCR
    - NumPy, Pillow → Image preprocessing
    - Tkinter/PyQt → GUI
    - SQLite / Pandas → Database/CSV storage

🔷 Flask (Mobile/Web Version)

- Flask (for backend API)
- HTML/CSS/JS (for frontend interface)
- REST API (for mobile interaction)

5. Workflow (4-Day Competition Plan)

Day 1: Vehicle + License Plate Detection

- Implement OpenCV Haar cascade or YOLO-lite for detecting license plates.
- Test with sample car images and videos.

Day 2: OCR Extraction

- Crop plate region and preprocess (grayscale, threshold, edge detection).
- Apply PyTesseract OCR to extract text.
- Validate results (A–Z, 0–9 characters).

Day 3: GUI Integration

- Create GUI with Tkinter/PyQt.
- Add buttons: Upload Image, Upload Video, Start Webcam.
- Display detected plate and recognized number.

Day 4: Database + Reporting

- Save recognized numbers with timestamp in SQLite/CSV.
- Export results option.
- Polish GUI (status messages, error handling).

✅ **Extra Credit (Optional):** Extend with Flask so mobile users can upload a photo of a car and get plate number recognition in the browser.

---

6. Expected Outcomes

- A **Python GUI application** that detects license plates and extracts numbers.
- A **database system** storing recognized plates for later analysis.
- A **Flask-based extension** for web/mobile access.
- A prototype that can be used for **parking systems, traffic monitoring, and surveillance**.

---

7. Possible Enhancements

- **YOLOv5/Deep Learning models** for more accurate detection.
- **Multi-plate detection** in a single frame (highway traffic).
- **Integration with vehicle details database** (e.g., check if stolen car).
- **Edge deployment** for Raspberry Pi-based traffic monitoring.

(6) Project Definition: Smart People Density Monitoring & Analysis in Shopping Malls

---

1. Introduction

Managing customer flow in large shopping malls is essential for **sales optimization, crowd management, and safety**. By analyzing the density of people in different sections of a mall and tracking customer traffic over time, mall managers can:

- Deploy more sales staff during peak hours.
- Identify low-traffic areas for marketing offers.
- Improve customer experience and safety.

This project aims to build a **Python-based application** that uses cameras to **detect, count, and analyze people density** in different mall sections, with a **GUI dashboard** for reports. It can also be extended into a **Flask-based mobile/web app** for real-time monitoring.

---

2. Objectives

- Detect and count number of people in video feeds (CCTV cameras).
- Track density in different **sections/zones of the mall**.
- Analyze traffic patterns by **time of day (hourly/daily/weekly)**.
- Provide a GUI for monitoring and generating reports.
- Extend via Flask to allow **real-time dashboard on mobile devices**.

---

3. Features

🔷 **Python Desktop Version (Main Requirement)**

1. **Input Sources:**
    - Live CCTV feed (Webcam/IP camera)
    - Pre-recorded mall videos
2. **People Detection & Counting:**
    - Use **YOLO / MediaPipe / OpenCV HOG + SVM** for human detection.
    - Count number of people per frame.
3. **Zone-based Density:**

    - Divide camera view into sections (e.g., Zone A, B, C).
    - Count people in each zone separately.
4. **Time-based Analytics:**
    - Store counts with timestamps in SQLite/CSV.
    - Generate hourly/daily traffic patterns.
5. **GUI Dashboard (Tkinter/PyQt):**
    - Live video with bounding boxes for detected people.
    - Current count in each section.
    - Graphs → Density vs. Time (Matplotlib).
    - Reports export to CSV/PDF.

🔷 Flask/Mobile App Extension (Optional)

1. **Web Dashboard:**
    - Real-time people count & density viewable in browser/mobile.
    - Graphs & analytics updated live.
2. **Mobile Features:**
    - See current mall occupancy.
    - View section-wise density heatmap.
    - Daily/weekly peak time reports.
3. **Optional AI Features:**
    - Predict peak hours based on past data.
    - Send mobile alerts when a section exceeds safe crowd limit.

4. Technology Stack

🔷 Python (Desktop Version)

- **Libraries:**
    - OpenCV → Video capture & processing
    - YOLO / MediaPipe → Person detection
    - NumPy, Pandas → Data processing
    - Matplotlib/Seaborn → Data visualization
    - Tkinter/PyQt → GUI dashboard
    - SQLite → Database for counts

🔷 Flask/Mobile Version

- Flask → Web backend
- HTML/CSS/JS → Dashboard frontend
- Chart.js / D3.js → Interactive graphs
- SQLite/PostgreSQL → Data storage

5. Workflow (4-Day Roadmap)

Day 1 – People Detection & Counting

- Implement people detection in video feed using OpenCV + YOLO/MediaPipe.
- Test detection accuracy on mall-like datasets.

Day 2 – Zone-based Density & Data Logging

- Divide video into sections (Zone A, B, C).
- Count people per section.
- Store counts with timestamp in SQLite/CSV.

Day 3 – GUI Dashboard

- Build GUI with:
    - Live video stream with bounding boxes
    - Current people count per section
    - Total occupancy count
- Add graph plotting (Density vs. Time).

Day 4 – Reporting & Analytics

- Generate reports (Peak hours, low-traffic times).
- Export daily/weekly CSV/PDF reports.
- Polish GUI with alerts if density crosses threshold.

✅ **Extra Credit (Optional Flask Extension):**

- Build web dashboard with Flask.
- Show live counts, graphs, and reports accessible from mobile.

6. Expected Outcomes

- A **Python desktop application with GUI** that can:
    - Detect people in live video
    - Count people per zone
    - Track traffic density with time
    - Provide reports & visualizations
- An extended **Flask-based mobile dashboard** for mall management.

7. Possible Enhancements

- Multi-camera integration for entire mall monitoring.
- Heatmaps of crowd density over mall floor plan.
- Predictive analytics (using ML) to forecast peak hours.
- Integration with **IoT sensors** (for smart mall automation).

(7) Automated Faculty & Class Management System with Muster Generation

---

1. Introduction

Managing faculty workload, class scheduling, and lab allocations in academic institutions is often a manual and error-prone task. Faculty members need well-structured timetables, departments require workload summaries, and attendance (muster) must be properly maintained for lectures and labs.

This project proposes a **Python-based Faculty & Class Management System** that generates:

- **Faculty Master** with workload details,
- **Individual faculty timetables**,
- **Class/Lab occupancy charts**,
- **Faculty workload reports**, and
- **Automated muster (attendance registers)** for lectures and labs, excluding government holidays.

The system integrates class master data (semester-wise subject allocation, student lists, and labs) to provide a centralized, automated solution for academic planning and monitoring.

---

2. Objectives

- Generate **Faculty Master** from class master of all semesters.
- Prepare **individual faculty timetables** (lecture + lab).
- Calculate **faculty workload** (weekly hours).
- Track **classroom and lab occupancy**.
- Create **lecture/lab muster sheets** for each faculty.
- Exclude **government holidays** automatically while creating muster.
- Provide reports for faculty, HoD, and department-level analytics.

---

3. Features

◆ **Faculty & Class Management**

1. **Faculty Master Creation**
   - Import class master (semester-wise subject allocation).
   - Auto-generate Faculty Master → Faculty ID, Subjects, Hours, Labs assigned.
2. **Timetable Generation**
   - Individual timetable per faculty (lecture + lab hours).
   - Weekly structure with time slots (Mon–Sat).
   - Export to PDF/Excel.
3. **Workload Calculation**
   - Auto-calculate weekly workload hours per faculty.
   - Department workload summary (total lecture + lab hours).
   - Highlight underload/overload faculty.
4. **Class & Lab Occupancy**
   - Track classroom and lab usage by time/day.
   - Generate occupancy reports to avoid clashes.
   - Heatmap-style visualization for peak usage.

◆ **Muster (Attendance Register) Generation**

1. **Lecture Muster**
   - Faculty-wise attendance sheet for lectures.
   - Student list (from semester data) with dates pre-filled.
2. **Lab Muster**
   - Lab-wise attendance sheet (batch-wise if applicable).
   - Separate columns for weekly lab sessions.

3. **Holiday Exclusion**
    - Preload government holiday list (CSV/Excel).
    - Exclude holidays automatically from muster.
4. **Export Options**
    - Save muster as Excel/PDF.
    - Digital attendance entry option (optional).

🔷 Admin Dashboard (Optional GUI/Web Extension)

- Upload **class master** and **student lists**.
- View faculty timetables, workload charts, and occupancy graphs.
- Generate and download muster per faculty/semester.

---

4. Technology Stack

🔷 Python (Core System)

- **Pandas** → Data processing (class master, student lists, holiday lists).
- **NumPy** → Workload & timetable calculations.
- **Matplotlib/Seaborn** → Occupancy visualization.
- **Tkinter/PyQt** → GUI for non-technical users.
- **ReportLab / openpyxl** → Muster & report generation (PDF/Excel).

🔷 Database

- SQLite / PostgreSQL → Store faculty, subjects, students, timetable data.

🔷 (Optional Web Extension)

- Flask/Django → Web dashboard.
- Chart.js/D3.js → Interactive workload/occupancy graphs.

---

5. Workflow (4-Day Roadmap)

Day 1 – Faculty Master & Timetable

- Import class master data.
- Generate Faculty Master (faculty–subject mapping).
- Create individual timetables per faculty.

Day 2 – Workload & Occupancy

- Calculate workload per faculty (lecture + lab hours).
- Generate department workload summary.
- Build classroom/lab occupancy reports.

Day 3 – Muster Generation

- Import student list for each semester.
- Create faculty-wise muster (lecture + lab).
- Integrate holiday exclusion in muster.

Day 4 – Reporting & Visualization

- Export timetables, workload reports, and muster in PDF/Excel.
- Add occupancy visualization (heatmap/graph).
- Build optional GUI dashboard for admin usage.

✅ **Extra Credit (Optional Web Version):**

- Deploy Flask dashboard for HoD/Faculty.
- Online muster with attendance marking.

---

6. Expected Outcomes

- **Faculty Master** automatically generated from class master.
- **Individual faculty timetables** ready in PDF/Excel format.
- **Workload reports** for each faculty & department.

- **Classroom/lab occupancy** report to optimize usage.
- **Automated muster sheets** (faculty-wise, lab + lecture) excluding holidays.

---

7. Possible Enhancements

- AI-based **automatic timetable generator** (clash-free).
- Multi-department integration.
- Cloud-based validation (faculty login for reports/muster).
- Student-side digital attendance app (linked with muster).
- Integration with **Google Calendar/Outlook** for timetables.