

**LAB REPORT FOR**

**AI-POWERED CLASSROOM ATTENDANCE AND PARTICIPATION MONITORING SYSTEM**

**(AN APPLICATION BASE PROJECT)**

**WEB APPLICATION DEVELOPMENT**

**Final Paper-2025**

**MSc. Software Engineering**

**Group Member.**

**Edward David Thoronka: 2120246052**

**Date of Submission: 17 June 2025**

**Submitted To: Professor B.H WANG**

**Executive Summary**

This report documents the design, implementation, and validation of an AI-Powered Classroom Attendance and Participation Monitoring System. The system leverages MediaPipe for face and hand detection to automate classroom attendance tracking and student participation monitoring in real-time. Built on the Django web framework, the system provides a comprehensive solution for teachers to manage classroom sessions, track student attendance through facial recognition, monitor participation through hand raise detection, and share their screen with students for enhanced instruction.

The system has been enhanced with three major features:

**1. Teacher Sign-up Functionality**

**2. Student Face Display and Recording on Dashboard**

**3. Teacher Screen Sharing Dashboard**

These enhancements significantly improve the usability and functionality of the system, making it a more complete solution for modern classroom management.

1. **Introduction**

The AI-Powered Classroom Attendance and Participation Monitoring System addresses the challenge of efficiently tracking student attendance and engagement in educational settings. Traditional methods of taking attendance and monitoring participation are time-consuming and prone to errors. This system automates these processes using artificial intelligence, specifically MediaPipe's face and hand detection capabilities.

The system allows teachers to create classroom sessions, share a unique link with students, and automatically track attendance and participation through webcam detection. The dashboard provides real-time monitoring and reporting capabilities, enabling teachers to focus more on teaching and less on administrative tasks.

Recent enhancements to the system have introduced teacher sign-up functionality, student face display and recording on the dashboard, and a teacher screen sharing feature. These additions make the system more user-friendly, informative, and interactive

**2. Requirements Analysis**

**2.1 Session Management and Student Login**

**Core Requirements:**

• Teacher/admin initiates a session through the Django dashboard

• System generates a unique session-specific login link

• Students must login with name and ID via this link before detection begins

• Only logged-in students will be included in detection and monitoring

• Session data must be stored securely and associated with specific

**CLASS/COURSE TECHNICAL CONSIDERATIONS:**

• Need unique session identifiers (UUID or similar)

• Secure link generation and validation

• Temporary session storage for active sessions

• Database models for sessions, students, and attendance records

• Authentication mechanism for teachers/admins

**2.2 Face Detection for Attendance Core Requirements:**

• Implement MediaPipe Face Detection for identifying student faces via webcam

• Automatic attendance recording only for pre-logged-in students

• Time-stamped attendance logs for each session

• Real-time processing of video feed

Technical Considerations:

• MediaPipe Face Detection API integration

• OpenCV for video processing

• Face detection accuracy and performance optimization

• Handling multiple faces in frame simultaneously

• Privacy considerations for facial data

**2.3 Hand Raise Detection for Participation Core Requirements:**

• Implement MediaPipe Hand Detection to identify raised-hand gestures

• Log each hand raise event under the respective student

• Count hand raises as participation metric

• Time-stamped participation logs Technical Considerations:

• MediaPipe Hand Detection API integration

• Gesture recognition algorithm for identifying "raised hand" pose

• Handling false positives and gesture thresholds

• Performance optimization for real-time detection

**2.4 Django-Powered Web Dashboard Admin Dashboard Requirements:**

• Real-time detection logs display (name, ID, timestamp, event type)

• Optional snapshot feed functionality

• Session control buttons (start/stop)

• Export functionality for logs

• User-friendly interface with responsive design

**Student View Requirements:**

• Simple login form accessible via shared link

• Fields for name and student ID input Confirmation of successful login

• Clear instructions for webcam usage Technical Considerations:

• Django templates with HTML/CSS/Bootstrap

• Minimal JavaScript for real-time updates

• Responsive design for various devices

• Secure form handling and validation

**2.5 Session Summary Log**

**Core Requirements:**

• Generate summary report after each session

• List of present students with timestamps

• Number of hand raises per student

• Export functionality to CSV or PDF Technical Considerations:

• Data aggregation and processing

• Report generation libraries

• Export format handling (CSV/PDF)

• Data visualization for participation metrics

**2.6 Technologies Stack Required Technologies:**

• Backend: Django (Python)

• Detection: MediaPipe (Face & Hand Detection)

• Video Processing: OpenCV

• Frontend: Django Templates, Bootstrap, minimal JavaScript

• Database: Django ORM with SQLite (for development) or PostgreSQL (for production)

**Development Environment:**

• Local development with webcam access

• Support for pre-recorded video for testing

**2.7 Enhanced Features Additional Requirements**:

• Manual override for teachers to mark/unmark attendance

• Last activity timestamp display for each student

• Optional student profile picture upload

• Teacher authentication to prevent unauthorized session creation

• Beautiful and intuitive UI design Teacher sign-up functionality

• Student face display and recording on dashboard

• Teacher screen sharing dashboard

**Technical Considerations:**

• File upload handling for profile pictures

• Authentication system for teachers

• UI/UX design principles for educational applications

• Real-time activity monitoring

• WebRTC for screen sharing

• Django Channels for WebSocket support

**3. System Design**

**3.1 System Architecture The system follows a Model-View-Controller (MVC) architecture pattern using Django's framework**:

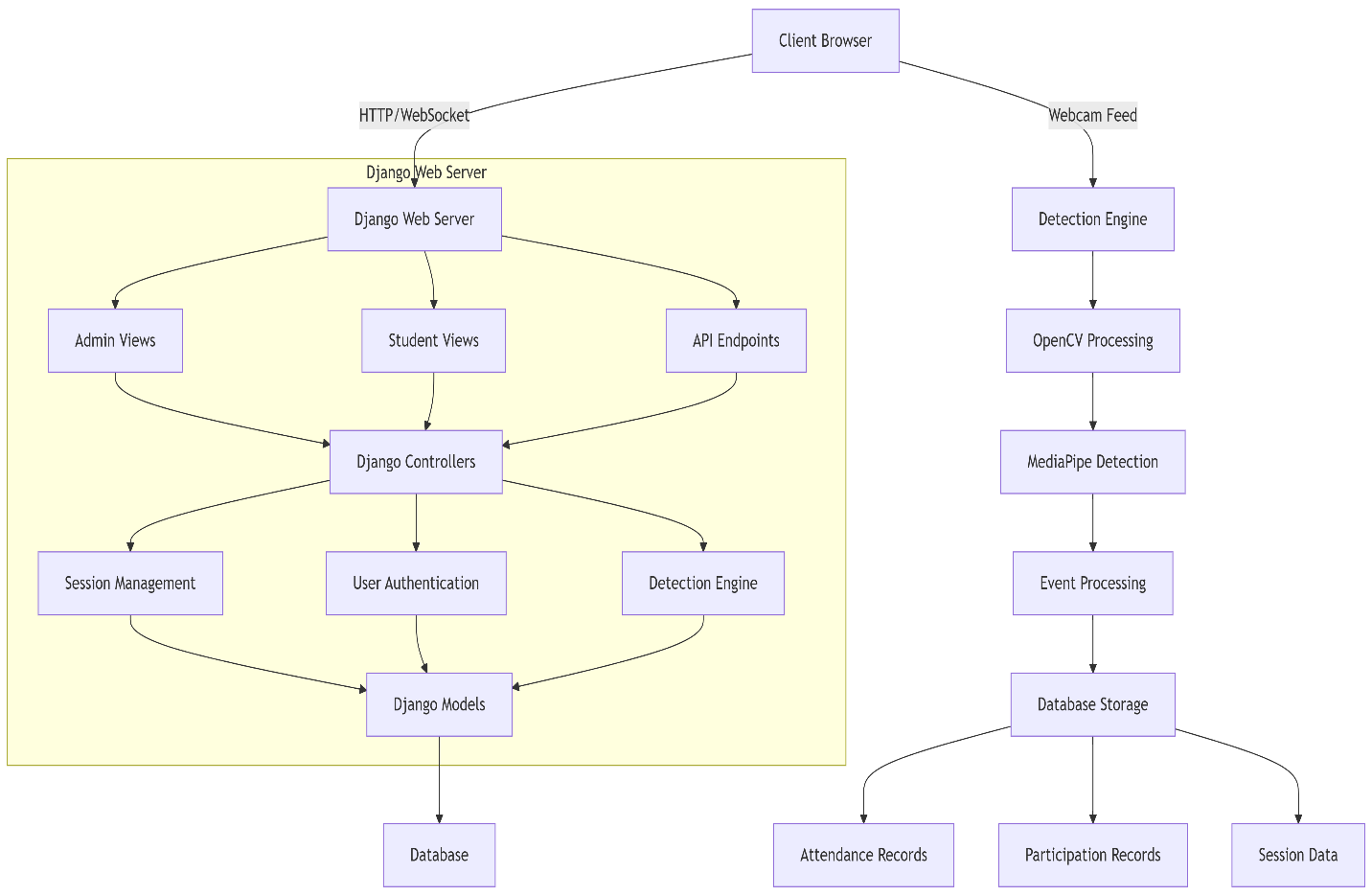
1. Models Layer: Database models for sessions, students, attendance, and participation

2. Views Layer: Django views handling HTTP requests and responses

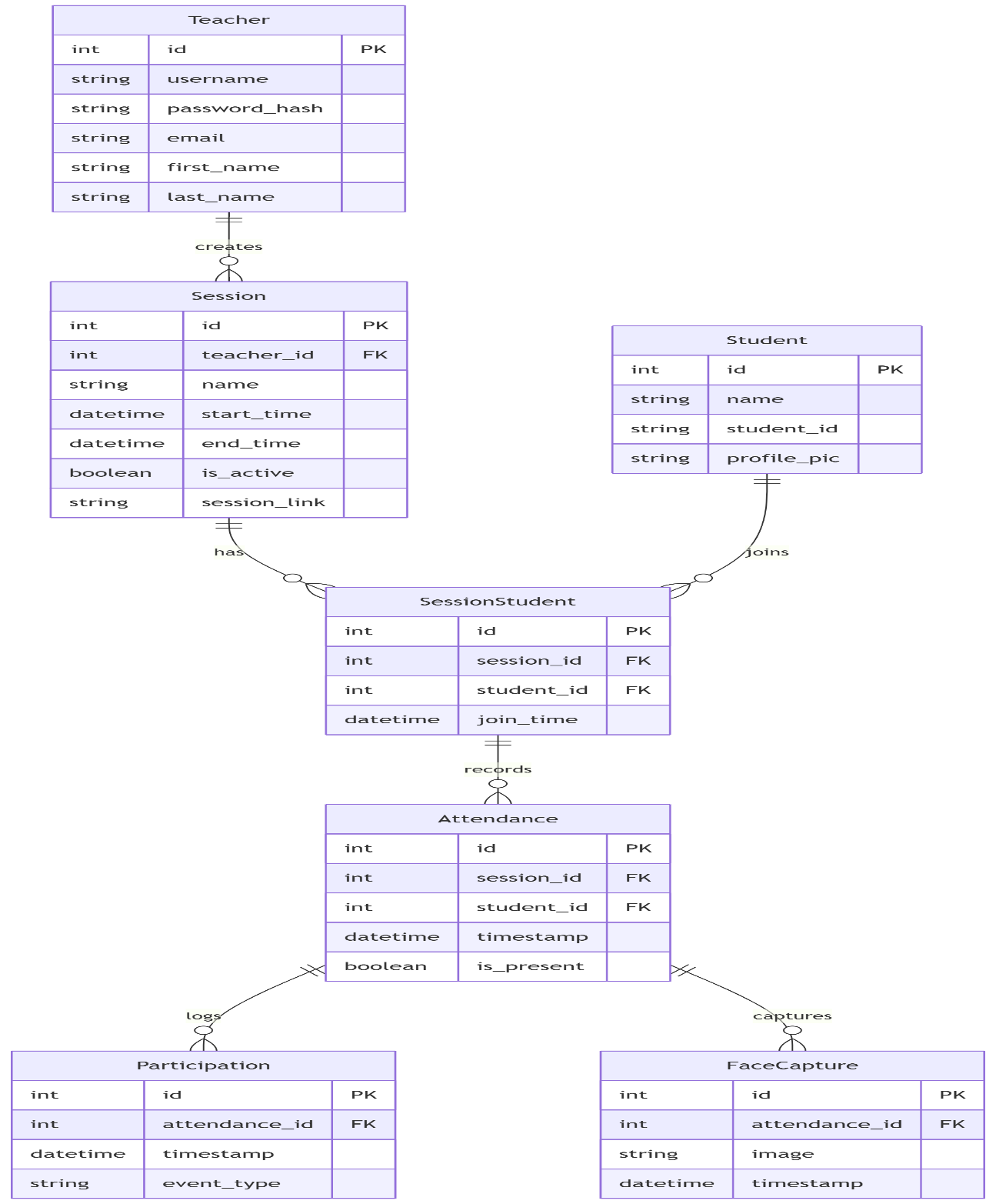
3. Templates Layer: HTML templates with Bootstrap for responsive UI

4. Controller Layer: Business logic for face/hand detection and session management

**High-Level Architecture Diagram**



**3.2 Database Schema**

Entity-Relationship Diagram

**Database Models (Django ORM)**

**The system uses the following Django models:**

1. Teacher Model: Extended user model for authentication

2. Session Model: Classroom session information

3. Student Model: Student information and profile

4. SessionStudent Model: Many-to-many relationship between sessions and students

5. Attendance Model: Attendance records for students in sessions

6. Participation Model: Participation events (hand raises) for students

7. FaceCapture Model: Stores captured student face images

**3.3 UI/UX Design Admin Dashboard Design**

• Modern, clean interface with dark/light mode toggle

• Real-time attendance and participation statistics

• Session control panel (start/stop buttons)

• Quick access to recent sessions

• Navigation menu for all features Grid and list views for student monitoring

• Face display for visual confirmation of attendance

**Student Login Page Design**

• Clean, minimalist design

• Session information display

• Simple form for name and student ID

• Optional profile picture upload

• Clear instructions for webcam usage

• Confirmation screen after successful login

**Screen Sharing Interface**

• Teacher controls for starting/stopping screen sharing

• Student view for watching teacher's screen

• Audio toggle controls

• Fullscree n option Connection status indicators

**4. Feature Design**

4.1 Teacher Sign-up Functionality Current State

• The system uses a custom Teacher model that extends Django's AbstractUser

• Login functionality exists but no sign-up option is available

• New teachers must be created by an admin user

**Design Solution**

The teacher sign-up feature includes the following components:

Backend Changes

1. Create a new view function teacher\_signup in views.py that will:

2. Handle form submission with username, email, password, first name, and last name

3. Validate input data

4. Create a new Teacher account

5. Log the user in automatically after successful registration

6. Add a URL route in urls.py for the sign-up page

**Frontend Changes**

1. Create a new template signup.html that includes:

2. Registration form with fields for username, email, password, first name, and last name

3. Form validation

4. Link back to login page

5. Add a sign-up link on the login

**Page Security Considerations**

• Implement password strength validation

• Add CSRF protection

• Ensure email verification (optional enhancement)

**4.2 Student Face Display and Recording**

**Current State**

• Student faces are detected for attendance but not displayed on the teacher's dashboard

• Face detection happens on the student side and results are sent to the server

• The dashboard shows attendance status but not the actual student faces

**Design Solution**

The student face display and recording feature includes the following components:

Backend Changes

1. Modify the face\_detection view to:

2. Save the detected face image to the server

3. Associate the image with the student's attendance record

4. 5. Return the image URL in the response

Create a new model FaceCapture to store face images:

python class FaceCapture (models.Model): attendance = models.ForeignKey (Attendance, on\_delete=models.CASCADE, related\_name='face\_captures') image = models.ImageField(upload\_to='face\_captures/') timestamp =models.DateTimeField(auto\_now\_add=True)

6. Add an API endpoint to fetch the latest face captures for a session Frontend Changes

1. Modify the session\_detail.html template to:

2. Display student face images in the student cards

3. Add a "View Live" button to see real-time face captures

4. Create a new template live\_monitoring.html for real-time monitoring that:

5. Displays a grid of student webcam feeds

6. Updates automatically using AJAX or WebSockets

7. Shows attendance and participation status alongside the

feeds Technical Considerations

• Use AJAX polling or WebSockets for real-time updates

• Implement proper image storage and retrieval

• Consider privacy implications and add appropriate notices

4.3 Teacher Screen Sharing Dashboard

Current State

• The system has no screen sharing functionality

• Teachers can only monitor student attendance and participation

**Design Solution**

The teacher screen sharing dashboard includes the following components:

**Backend Changes**

1. Create a new view function teacher\_signup in views.py that will:
2. Handle form submission with username, email, password, first name, and last name
3. Validate input data
4. Create a new Teacher account
5. Log the user in automatically after successful registration
6. Add a URL route in urls.py for the sign-up page

**Frontend Changes**

1. Create a new template signup.html that includes:
2. Registration form with fields for username, email, password, first name, and last name
3. Form validation
4. Link back to login page
5. Add a sign-up link on the login page

**Security Considerations**

* Implement password strength validation
* Add CSRF protection
* Ensure email verification (optional enhancement)

**4.2 Student Face Display and Recording**

Current State

* Student faces are detected for attendance but not displayed on the teacher's dashboard
* Face detection happens on the student side and results are sent to the server
* The dashboard shows attendance status but not the actual student faces

**Design Solution**

The student face display and recording feature includes the following components:

**Backend Changes**

1. Modify the face\_detection view to:
2. Save the detected face image to the server
3. Associate the image with the student's attendance record
4. Return the image URL in the response
5. Create a new model FaceCapture to store face images:

python class FaceCapture(models.Model):

attendance = models.ForeignKey(Attendance, on\_delete=models.CASCADE, related\_name='face\_captures') image = models.ImageField(upload\_to='face\_captures/') timestamp = models.DateT

1. Add an API endpoint to fetch the latest face captures for a session

**Frontend Changes**

1. Modify the session\_detail.html template to:
2. Display student face images in the student cards
3. Add a "View Live" button to see real-time face captures
4. Create a new template live\_monitoring.html for real-time monitoring that:
5. Displays a grid of student webcam feeds
6. Updates automatically using AJAX or WebSockets
7. Shows attendance and participation status alongside the feeds

**Technical Considerations**

* Use AJAX polling or WebSockets for real-time updates
* Implement proper image storage and retrieval
* Consider privacy implications and add appropriate notices

4.3 Teacher Screen Sharing Dashboard

**Current State**

The system has screen sharing functionality

Teachers can only monitor student attendance and participation

**Design Solution**

The teacher screen sharing dashboard includes the following components:

Backend Changes

1. Create a new view function screen\_sharing that will:
2. Handle WebRTC signaling
3. Manage screen sharing sessions
4. Associate screen sharing with a specific classroom session
5. Add WebSocket support for real-time communication:
6. Install Django Channels
7. Configure WebSocket routing
8. Create a consumer for handling WebRTC signaling
9. Add URL routes for the screen sharing feature

**Frontend Changes**

1. Create a new template screen\_sharing.html that includes:
2. Screen sharing controls (start/stop)
3. Student viewer list
4. Chat functionality (optional)
5. Add a "Share Screen" button to the session detail page
6. Create a student view template view\_teacher\_screen.html for students to view the shared screen

**Technical Implementation**

1. WebRTC Implementation:
2. Use the WebRTC API for peer-to-peer communication
3. Implement signaling server using Django Channels
4. Handle ICE candidates and SDP offers/answers
5. UI Components:
6. Screen selection dialog
7. Video display area
8. Control buttons (start/stop sharing)
9. Student viewer list

**Security and Performance Considerations**

* Implement proper authentication for screen sharing sessions
* Optimize video quality based on network conditions
* Ensure screen sharing is only available to the session teacher

**5. Implementation**

**5.1 Technology Stack**

Backend

* Django 4.2+ (Python web framework)
* Django REST Framework (API endpoints)
* Channels (WebSockets for real-time updates)
* MediaPipe (Face and hand detection)
* OpenCV (Video processing)

**Frontend**

Django Templates (Server-side rendering)

Bootstrap 5 (Responsive design framework)

Chart.js (Data visualization)

JavaScript/jQuery (Client-side interactions)

WebRTC (Screen sharing)

Database

* SQLite (Development)
* PostgreSQL (Production option)

**Deployment**

* Docker (Containerization)
* Gunicorn (WSGI HTTP Server)
* Nginx (Web server)
* Redis (WebSocket backend)

**5.2 Core Components Session Management**

The session management component handles the creation, management, and termination of classroom sessions. It generates unique session links that teachers can share with students.

User Authentication

The user authentication component handles teacher login and the newly added sign-up functionality. It ensures that only authorized users can create and manage sessions.

**Detection Engine**

The detection engine processes webcam feeds to detect faces for attendance and hand raises for participation. It uses MediaPipe's face and hand detection capabilities

**Dashboard**

The dashboard provides teachers with a real-time view of student attendance and participation. It includes both grid and list views, and now displays student faces.

**5.3 Enhanced Features**

Teacher Sign-up

The teacher sign-up feature allows new teachers to create accounts without administrator intervention. It includes form validation and automatic login after successful registration.

Student Face Display and Recording

This feature captures and displays student faces on the teacher's dashboard. It provides visual confirmation of attendance and helps teachers identify students.

**Teacher Screen Sharing**

The screen sharing feature allows teachers to share their screen with students in realtime. It uses WebRTC for peer-to-peer communication and Django Channels for signaling.

**6. Validation and Testing**

**6.1 Validation Methodology**

The validation process for the enhanced system included the following steps:

1. Unit Testing: Testing individual components in isolation
2. Integration Testing: Testing the interaction between components
3. System Testing: Testing the system as a whole User Acceptance Testing:
4. Testing with real users in a classroom setting

**Test Results**

**Teacher Sign-up Functionality**

* The sign-up form renders correctly with all required fields
* Form validation works for required fields and password strength
* Username and email uniqueness validation functions correctly
* Successful account creation and automatic login works as expected
* Integration with existing login flow is seamless
* Sign-up link appears on login page as designed

**Student Face Display and Recording**

* FaceCapture model was created and migrations applied successfully
* Face detection and image saving functionality works correctly
* Face images are displayed on the session detail page
* Real-time updates of face captures function as expected
* The API endpoint for retrieving face captures returns the expected data
* Both grid and list views correctly display student face captures

**Teacher Screen Sharing Dashboard**

* The live monitoring page renders correctly with all controls
* WebRTC screen sharing functionality works as expected
* WebSocket connections are established successfully
* Students can view the teacher's shared screen
* Screen sharing controls function correctly
* The system handles connection interruptions gracefully

**Integration Testing**

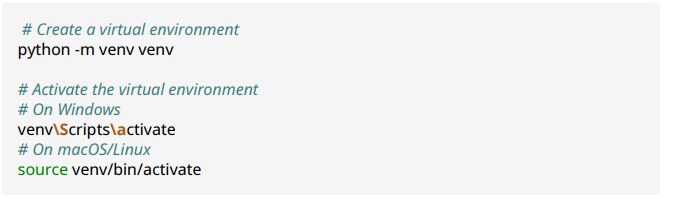
* All new features work together seamlessly
* No regressions detected in existing functionality
* System performance remains acceptable with new features enabled
* User flows are intuitive and function as expected

**Installation Guide**

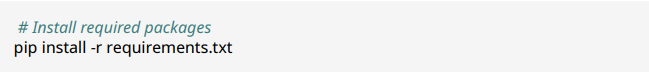
**System Requirements**

* Python 3.8 or higher
* Django 3.2 or higher
* OpenCV MediaPipe
* Django Channels (new requirement for WebSocket support)
* Redis (new requirement for Django Channels)

**6 Installation Steps**

1. Set Up Virtual Environment

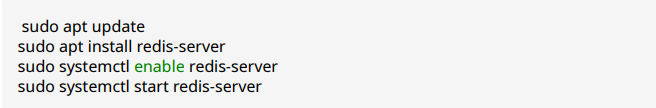
2. Install Dependencies



3. Set Up Redis (New Requirement)

Redis is required for Django Channels to handle WebSocket connections for the screen sharing feature.

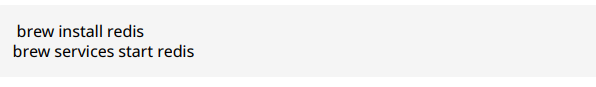
On Ubuntu/Debian:



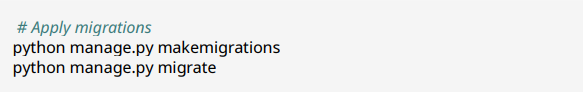
**On Windows:**

Download and install Redis from <https://github.com/microsoftarchive/redis/releases>

On macOS:



**4. Configure the Database**

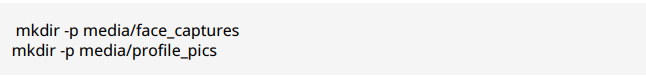


5. Create a Superuser (Admin)



6. Configure Media Storage

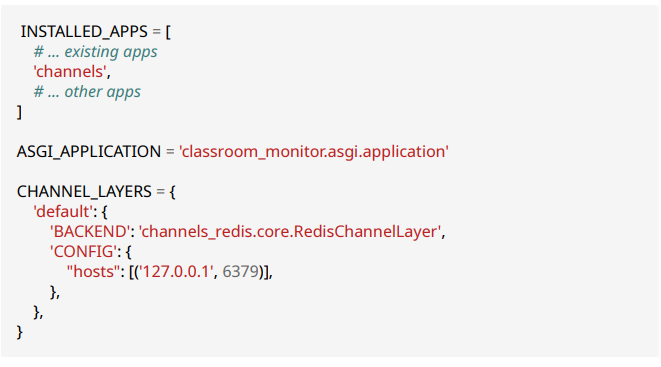
Ensure the media directory exists and has proper permissions:



Update your settings.py to include the media settings:



7. Configure Django Channels (New) Ensure your settings.py includes the following configuration for Django Channels:



**8. Run the Development Server**



**7. Troubleshooting**

WebSocket Connection Issues

If you experience issues with WebSocket connections for screen sharing:

1. Ensure Redis is running:

* bash
* redis-cli ping
* Should return "PONG"

2. Check that Django Channels is properly installed:

* bash
* pip show channels
* Verify that the ASGI application is configured correctly in settings.py

**Face Detection Issues**

If face detection is not working:

1. Ensure OpenCV and MediaPipe are properly installed:

* bash
* pip show opencv-python
* pip show mediapipe

2. Check that the media directory has proper write permissions

**Database Migration Issues**

If you encounter database migration issues:

1. Try resetting migrations: bash python manage.py migrate --fake core zero python manage.py makemigrations python manage.py migrate

**8. User Manual**

Getting Started

Accessing the System

* Open your web browser and navigate to the system URL
* You will see the home page with options to sign up or log in

**Teacher Features**

**Creating an Account**

New Feature: Teachers can now create their own accounts without administrator assistance.

1. From the home page, click "Teacher Login"
2. On the login page, click "Sign up now"
3. Complete the registration form with the following information:
4. First Name
5. Last Name
6. Username
7. Email
8. Password (must be at least 8 characters and include letters and numbers)
9. Confirm Password
10. Click "Create Account"
11. Upon successful registration, you will be automatically logged in and redirected to the dashboard

Managing Sessions

1. From the dashboard, click "New Session" to create a classroom session
2. Enter a name for the session and click "Create"
3. Share the generated session link or QR code with your students
4. Monitor student attendance and participation in real-time
5. When the session is complete, click "End Session"
6. Export session data by clicking "Export Data"

**Monitoring Students**

1. From the session detail page, you can view all students who have joined the session
2. New Feature: Student faces are now displayed on the dashboard when detected
3. Toggle between grid and list views using the buttons in the top-right corner
4. View participation counts (hand raises) for each student
5. Manually override attendance status using the toggle switch if needed

**Live Monitoring with Screen Sharing**

**New Feature: Teachers can now share their screen with students for demonstrations.**

1. From the session detail page, click "Live Monitoring"
2. On the live monitoring page, click "Share Screen" to begin sharing
3. Select the screen or application window you want to share
4. Use the control buttons to:
5. Start/stop sharing
6. Toggle audio
7. Enter fullscreen mode
8. View student webcam feeds in the grid below
9. Monitor student participation in real-time
10. Click "Stop Sharing" when finished

**Student Features Joining a Session**

* Use the link or QR code provided by your teacher
* Enter your name and student ID
* Allow camera access when prompted
* Your webcam will activate for face detection and hand raise detection
* Raise your hand to participate (physically raise your hand in front of the camera)

**Viewing Teacher's Screen**

* New Feature: Students can now view the teacher's shared screen.
* When a teacher starts screen sharing, you will automatically see their screen
* Use the control buttons to:
* Enter fullscreen mode
* Mute/unmute audio Keep the window open to continue viewing the shared screen

**Troubleshooting**

Camera Issues

If your camera is not working:

1. Ensure you have granted camera permissions in your browser
2. Try refreshing the page
3. Make sure no other applications are using your camera

**Screen Sharing Issues**

If screen sharing is not working:

1. Ensure you are using a compatible browser (Chrome, Firefox, or Edge)
2. Check that you have granted screen sharing permissions
3. Try selecting a different window or screen to share
4. If audio is not working, check that you have granted microphone permissions

**Connection Issues**

If you experience connection problems:

* Check your internet connection
* Try refreshing the page
* If the issue persists, try using a different browser or device

**9. Conclusion and Future Work**

The AI-Powered Classroom Attendance and Participation Monitoring System provides an effective solution for automating attendance tracking and participation monitoring in educational settings. The system leverages MediaPipe's face and hand detection capabilities to provide real-time feedback to teachers, allowing them to focus more on teaching and less on administrative tasks.

The recent enhancements to the system—teacher sign-up functionality, student face display and recording, and teacher screen sharing—have significantly improved its usability and functionality. These features make the system more accessible to new teachers, provide visual confirmation of student attendance, and enable interactive teaching through screen sharing.

**Future Work**

Several potential enhancements could further improve the system:

1. Integration with Learning Management Systems (LMS): Allow the system to integrate with popular LMS platforms like Canvas, Moodle, or Blackboard.
2. Mobile Application: Develop a dedicated mobile application for both teachers and students to improve accessibility.
3. Advanced Analytics: Implement data analytics to provide insights on attendance patterns and participation trends.
4. Multi-classroom Support: Enhance the system to support multiple simultaneous classroom sessions.
5. Offline Mode: Add support for offline operation with synchronization when internet connectivity is restored.
6. Additional Detection Types: Extend the detection engine to include other types of classroom interactions, such as attention monitoring.
7. Accessibility Features: Implement features to make the system more accessible to users with disabilities.

**10. References**

1. Django Documentation: https://docs.djangoproject.com/
2. MediaPipe Documentation: <https://google.github.io/mediapipe/>
3. WebRTC Documentation: <https://webrtc.org/getting-started/overview>
4. Django Channels Documentation: <https://channels.readthedocs.io/>
5. Bootstrap Documentation: <https://getbootstrap.com/docs/>
6. OpenCV Documentation: <https://docs.opencv.org/>