Helwan University

Faculty of Computers & Artificial Intelligence

Department of Computer Science

EduGuard: Smart Learning Management System with Cheating Detection in Quizzes

Prepared By:

Hassan Abdelhamed	**20210290**
Youssef Ahmed	**20211058**
Muhammed Naser	**20210840**
Omar Sayed	**20210604**
Doaa Magdy	**20210309**
Habiba Muhammed	**20210279**

Supervised By:

Prof. Dr. Islam Gamal

Lecturer at the Department of Computer Science
Faculty of Computers and Artificial Intelligence
Helwan University

Abstract

With the rapid advancement of digital education, especially post-pandemic, the need for secure and intelligent online learning platforms has become more crucial than ever. EduGuard is a smart e-learning monitoring and management system that aims to enhance academic integrity while offering a complete virtual learning environment for both instructors and students.

EduGuard combines AI-powered cheating detection with traditional e-learning features to provide a seamless and secure online education experience. Using facial recognition, object detection, and behavior tracking, the system monitors students in real time during exams and automatically notifies administrators of any suspicious activities. This ensures fairness and reliability in remote assessments.

Beyond proctoring, EduGuard allows instructors to upload and manage course materials, including PDFs and video lectures, while students can register for courses to access those materials and participate in quizzes. The platform supports role-based access control, interactive learning, and real-time monitoring—all built using modern technologies such as React.js, PHP (Laravel), FastAPI, MySQL, and AI frameworks like TensorFlow and OpenCV.

EduGuard is designed to be a scalable, secure, and user-friendly system that not only promotes academic honesty but also facilitates structured and accessible online learning. Through extensive testing and implementation, it proves to be a reliable solution for institutions seeking to digitize their learning processes while maintaining high educational standards.

EduGuard uses real-time facial recognition, gaze tracking, and object detection to flag cheating (e.g., unauthorized devices or multiple people). Pilot testing showed ~90% detection accuracy, reducing manual proctoring efforts. Instructors benefit from an automated alert system, while students enjoy a fair, structured learning environment.

Acknowledgements

We would like to express our sincere gratitude to everyone who supported and guided us throughout the development of our graduation project, EduGuard: Smart Learning Management System with Cheating Detection in Quizizz. This project was a collaborative effort, and each team member contributed significantly to its success.

Team Members and Their Roles:

- **Hassan Abdelhamed (Team Leader)**
  Hassan played a central role in the development of the system, contributing to both the frontend and backend. He was responsible for integrating machine learning models with the backend and frontend, designing user interfaces in Figma, writing documentation, and creating all required system diagrams.
- **Youssef Ahmed**
  Youssef focused on backend development and worked on integrating machine learning models into the system. He also participated in preparing technical diagrams.
- **Omar Sayed**
  Omar worked on the frontend development of the platform, collaborated on UI design using Figma, and contributed to creating system diagrams.
- **Muhammed Naser**
  Muhammed contributed to the backend development and participated in designing technical and system diagrams.
- **Doaa Magdy & Habiba Muhammed**
  Doaa and Habiba were fully responsible for the development of the machine learning models. They handled data processing, model training, evaluation, and worked closely with the rest of the team to ensure seamless integration into the system.

We also extend our gratitude to Prof. Dr. Islam Gamal their guidance, and to our beta testers for invaluable feedback during development.

Table of Contents

Table of Figures

Table of Tables

1. Introduction

1.1. Problem statement

With the rapid shift towards online education—accelerated by the COVID-19 pandemic—many educational institutions adopted digital platforms for learning and assessment. However, despite their widespread use, current e-learning systems present several critical challenges that undermine the quality and integrity of remote education.

1) **Limited Interactivity in Online Learning**

   Most traditional e-learning platforms focus primarily on delivering static content such as PDFs and pre-recorded videos, lacking real-time interaction and engagement tools. This limitation affects the learning experience, especially for university-level courses where dynamic interaction between instructors and students plays a vital role in knowledge retention and concept clarity.

2) **High Incidence of Cheating in Online Quizizz**

   Another major issue in digital education is the lack of proper monitoring mechanisms during online assessments. Many instructors rely on tools like *Google Forms* or simple quiz modules to conduct tests and assignments. However, these platforms provide no means to ensure that students are completing the assessments honestly. Without supervision, students may search the internet for answers, communicate with peers via messaging apps, or even use unauthorized devices.

   This vulnerability severely undermines the credibility of assessments and academic grading. In several cases, entire quizzes have been canceled due to widespread cheating, resulting in wasted time, administrative complications, and unfair outcomes for honest students. Moreover, such issues damage the reputation of the educational institution and cast doubt on the legitimacy of its online certifications and results.

1.2. Importance of the project

In the evolving landscape of education, especially with the rapid transition to digital learning, there is a critical need for platforms that not only facilitate online education but also uphold academic integrity and deliver a seamless user experience. The importance of *EduGuard* lies in its holistic approach to tackling the major pain points of current e-learning systems.

1) **Ensuring Academic Integrity**

   EduGuard provides intelligent, real-time monitoring using machine learning models such as facial recognition, object detection, and behavioral tracking. This helps detect and prevent cheating during online assessments, which is a growing concern in remote education. By automatically notifying instructors or administrators of suspicious activity, the system maintains fairness and credibility in the examination process.

2) **Combining Learning and Proctoring in One System**

   Unlike fragmented systems that require students and educators to use multiple tools for learning and examination, EduGuard offers an all-in-one platform. Course material uploads (PDFs, videos), student course registration, and quiz participation are integrated alongside cheating detection features. This not only simplifies the user experience but also increases productivity and learning outcomes.

3) **Promoting Accessibility and Flexibility**

   EduGuard is accessible from anywhere, allowing students and educators to participate in the learning process regardless of their location. This flexibility supports inclusive education and ensures continuity even during emergencies or pandemics.

4) **Improving Student Engagement and Instructor Efficiency**

With its user-friendly interface and structured design, EduGuard enhances engagement and reduces confusion among students. For instructors, the platform reduces the time and effort needed to manage materials, quizzes, and student behavior, thereby allowing them to focus more on teaching quality and student performance.

5) **Contribution to Modern Educational Practices**

EduGuard demonstrates how AI and modern web technologies can be responsibly and effectively integrated into education. It contributes to the broader academic and technological discourse by offering a working solution to common e-learning problems and can serve as a model for future educational platforms.

1.3. Objective

The primary objective of *EduGuard* is to develop a secure, intelligent, and user-friendly online learning and assessment platform that addresses the key challenges in remote education. The system is designed to combine educational content delivery with AI-based monitoring tools to ensure both high-quality learning and academic integrity. The specific objectives include:

1) **Enhance Interactivity**

   Create an engaging and interactive virtual learning environment that supports uploading and accessing various types of educational materials, such as video lectures, downloadable notes, and dynamic quizzes, to improve knowledge retention and student involvement.

2) **Ensure Integrity in Online Assessments**

   Integrate advanced machine learning models for:

   - **Face Detection:** Ensure the registered student is present and actively participating in the exam.
   - **Object Detection:** Identify unauthorized devices or materials within the camera frame.
   - **Eye Movement Tracking:** Monitor student focus and detect suspicious behavior, such as looking away from the screen frequently.

3) **Real-Time Monitoring and Notification**

   Provide live exam monitoring and automated alerts to instructors or admins when suspicious activity is detected, reducing manual effort and improving response time.

4) **Role-Based Access and Security**

   Implement a secure login system with session-based role control, allowing students, instructors, and administrators to access features based on their privileges, thus ensuring system integrity and data privacy.

5) **Centralized Learning Management**

   Offer a unified platform for managing course materials, exams, and student records to minimize fragmentation and make it easier for users to navigate and perform their tasks efficiently.

6) **Promote Scalability and Flexibility**

   Design the system to be easily adaptable to various institutional needs, with scalability in mind for future growth and support for multiple courses and users.

7) **Improve Learning Outcomes and Assessment Reliability**

   Provide tools and features that support continuous assessment, timely feedback, and reliable data to evaluate student performance effectively.

1.4. Scope

The scope of *EduGuard* is focused on designing and developing a smart e-learning and assessment platform tailored specifically for university educators and their students. The platform aims to bridge the gap between traditional classroom learning and online education by offering enhanced interactivity, academic integrity, and centralized content management.

Key areas covered within the scope of this project include:

1) **Target Users:**

   The primary users of the system are:

   - **University Educators** can upload lecture materials, manage quizzes, and monitor students during assessments.
   - **Students** who can access course content, participate in interactive quizzes, and complete assessments under intelligent surveillance.
   - **Administrators** who oversee user management, system performance, and academic data integrity.

2) **Learning Management System (LMS) Features:**
   - Uploading and organizing course content (videos, notes, slides, etc.)
   - Creating, editing, and scheduling online quizzes
   - Managing student profiles and access

3) **AI-Based Monitoring Tools:**
   - Face detection to verify student identity.
   - Object detection to identify cheating tools or unauthorized materials.
   - Eye movement tracking to monitor focus and attention during quizzes.

4) **Real-Time Monitoring and Reporting:**
   - Live video feed from student cameras during assessments
   - Automatic alerts and logs in case of suspicious behavior
   - Performance reports and quiz analytics for educators

5) **Security and Role-Based Access:**
   - Secure authentication with role-based permissions
   - Student access to only permitted courses and quizzes

2. Market Research & Analysis

2.1. Market Need

1) **Enhanced Interactivity:**

   Traditional e-learning platforms often lack engaging features, leading to student disengagement and lower retention rates.

2) **Academic Integrity Concerns:**

   - The shift to online education has led to a surge in academic dishonesty, with studies indicating a 300% increase in cheating between 2020 and 2021 during the pandemic's peak. Platforms like Google Forms, which lack built-in security mechanisms, are especially prone to impersonation, unauthorized collaboration, and search-engine use during exams.
   - Addressing these concerns requires real-time monitoring solutions that ensure students remain honest during online assessments without infringing on privacy. Key technologies like face detection, object detection, and eye-tracking help uphold integrity while minimizing false accusations. For example, AI-driven proctoring systems combined with human reviewers have proven effective at reducing incidents of cheating by 50-70%.

2.2. Target Audience

- **Primary Users:** University educators seeking effective online teaching tools.
- **Secondary Users:** Students enrolled in university courses who require a secure and engaging learning environment.

2.3.Competitors Overview

1) **ProctorU**

Overview: An online proctoring service that helps educational institutions maintain academic integrity during online assessments.

Features: Live proctoring, identity verification, and monitoring tools to prevent cheating.

ML Models and AI Capabilities: Uses automated systems to monitor activities like facial recognition, gaze tracking, and sound detection. Their AI tools flag behaviors such as looking off-screen or suspicious movement for human review.

Fees:

Table 1: ProctorU Fees

Proctoring Service	Flex Scheduling	Take it Soon	Take it Now
60 Minutes or Less	$17.50	$22.50	$26.25
120 Minutes or Less	$25.00	$30.00	$33.75
180 Minutes or Less	$33.75	$38.75	$42.50
240 Minutes or Less	$42.50	$47.50	$51.25

2) **Honorlock**

Overview: A remote proctoring solution that combines AI and live proctors to monitor students during online exams.

Features: Real-time monitoring, browser lockdown, and integration with various learning management systems (LMS).

ML Models and AI Capabilities: Incorporates AI-driven behavior monitoring alongside voice recognition to detect unauthorized use of voice assistants. If the system flags unusual behavior, a human proctor can step in immediately. It also has tools like “BrowserGuard” to block specific websites and prevent cheating through search engines.

Fees:

- Institutional Pre-Pay:
    - By Exam
        - $5 per exam - automation only
        - $6.50 per exam - automation with pop-in
        - $12.50 per exam - Proctor Record & Review - includes auto and pop-in
- Student Pay:
    - By Exam
        - $7.5 per exam - automation only
        - $9 per exam - automation with pop-in
        - $15.50 per exam - Proctor Record & Review - including auto and pop-in

3) **Examity**

Overview: Examity is a leading online proctoring service designed to help educational institutions ensure the integrity of online assessments. It provides a secure environment for students taking exams remotely.

Features: Live proctoring, identity verification, Automated Proctoring, and Integration with LMS.

ML Models and AI Capabilities: Offers both live and automated proctoring, using AI for identity verification and to flag suspicious actions during exams. The flagged incidents are compiled into reports for instructors, making the review process more efficient.

Fees:

The cost of Examity is dependent on the modality chosen. The price breakdown is as follows:

- Live Proctoring: $25.00
- Automated Examity Review: $10.00

4) **Moodle**

Overview: A widely used open-source learning platform.
Features: Course management, quiz creation, and basic activity tracking, though it lacks advanced monitoring tools.
Fees: free

2.4.Comparison between competitors

Table 2: Comparison between competitors

Platform	AI Features	Human Involvement	Unique Selling Point	Fees
ProctorU	Facial Recognition, Gaze Tracking	High	Live and automated proctoring blend	From $17.5 to $51.25 per test
Honorlock	Voice Detection, BrowserGuard	Medium	Blocks search engines & voice assistants	From $5 to $17.5 per test
Examity	AI Flagging Suspicious Actions	High	AI reports with integrated LMS dashboards	From $10 to $25 per test
Moodle	None	None	Open source but lacks academic integrity tools	Free

2.5.Cheating Detection in Existing Platforms

These platforms employ hybrid models that combine AI with human proctors. When the AI detects potential misconduct, such as unusual eye movements or use of unauthorized applications, it:

- **Alerts live proctors** in real-time to intervene and monitor the student directly (in platforms like Honorlock).
- **Generates detailed reports** summarizing flagged behaviors, which are sent to educators for review. The reports help instructors decide whether further investigation or disciplinary action is needed.

2.6.Challenges and Risks

- **Privacy Concerns:** Data security and student privacy will be critical considerations when implementing face detection and eye movement tracking. Clear policies and data handling practices will need to be established.
- **Technical Challenges:** Continuously improving the accuracy and reliability of machine learning models is crucial for the platform's effectiveness.

3. Scientific Review

3.1. Literature Review

3.1.1. E-Learning Platforms and Proctoring Tools

1) **ProctorU**

ProctorU provides online proctoring services that combine AI with human oversight. Key features include live proctoring, identity verification, and monitoring tools to flag suspicious activity. ProctorU leverages facial recognition and gaze tracking models to monitor student attention, alerting human proctors to any anomalies.

2) **Honorlock**

Honorlock utilizes AI to monitor students and integrates with popular LMS (Learning Management Systems). It offers browser lockdown, voice detection, and web search detection features. AI-driven tools track student behavior in real-time, flagging potential integrity breaches for further review by live proctors.

3) **Examity**

Examity uses a hybrid model with AI and live proctoring to verify identity and monitor online exams. The system logs instances of suspicious behavior, generating reports for instructors. Its integration with LMS and support for automated proctoring make it popular, though concerns about privacy and AI accuracy persist.

4) **Moodle**

Moodle, an open-source LMS, allows course management and quiz creation but lacks built-in proctoring features. Institutions often enhance Moodle with third-party plugins to monitor assessments, highlighting the need for secure, integrated proctoring options.

3.1.2. Machine Learning Models in Proctoring

ML models used in proctoring are varied, each designed to monitor specific aspects of student behavior. Commonly implemented models include:

- **Face Detection**: Uses datasets like WIDER FACE to detect faces on-screen and ensure the correct person is taking the assessment.
- **Eye Movement Tracking**: Datasets like Eye Gaze assist in tracking where students look during the quiz, flagging off-screen gazes.
- **Voice Recognition**: Tracks unauthorized conversation to prevent verbal collaboration.
- **Object Detection**: Helps identify unauthorized materials using datasets such as COCO to detect items like books or phones.

3.2. How ProctorU works

3.2.1. Faculty Process for Adding an Exam Iteration on ProctorU

1) Account Creation and Login:
   - Faculty, such as instructors and administrators, must have a ProctorU account. If they don't have one, they contact a ProctorU representative for account setup.
   - Once the account is set up, the faculty logs in to access exam management features.
2) Exam Creation:
   - To add a new exam, faculty members navigate to the "Add New" button in the top-right corner and select "Exam" from the drop-down menu.

Figure 1: Add New Exam Button

- This opens a form where faculty can input the exam details, including:
  - **Exam Title**: Include course code (e.g., STA 101).
  - **Department and Term**.
  - **Instructor Name and Duration**.
  - **Exam URL and Password** for access.
  - **Permitted Resources and Browsers**.
  - **Expected Number of Test-Takers** and other notes.
- Faculty specify allowed materials (e.g., calculators, notes) and set up the exam availability window by defining start and end dates and times, which can also support recurring exam sessions.

Figure 2: Exam Details Form

3) Review and Activation:
   - Once the exam form is completed, ProctorU's assessment services team reviews the submission and activates the exam within the system.
   - Faculty receive an email notification once the exam is active and ready for test-takers to schedule appointments.
4) Exam Modifications:
   - Faculty can edit exam details until 24 hours before the exam start time. For last-minute changes, they need to contact a ProctorU representative to update exam settings.

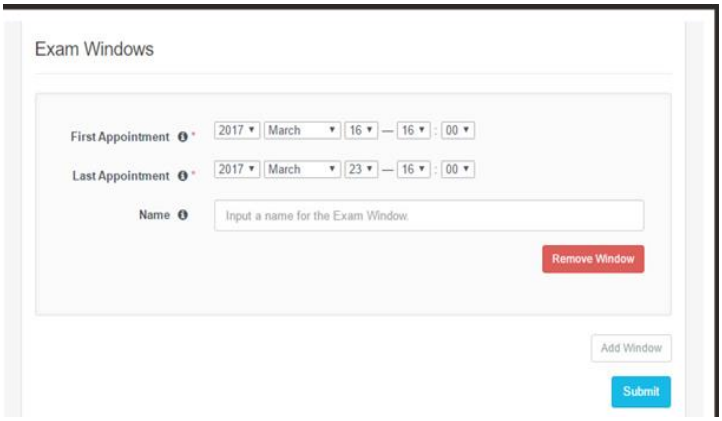


Figure 3: Exam Modifications Window

3.2.2. Reporting Capabilities

1) Activity Reporting

- Faculty can monitor exam sessions and appointments via the **Reports/Activity Report** section. This includes:
  - Filtering by test-taker name, start/end times, or reason for cancellation.
  - Detailed views of test-taker sessions, including exam name, instructor, start/end times, and duration

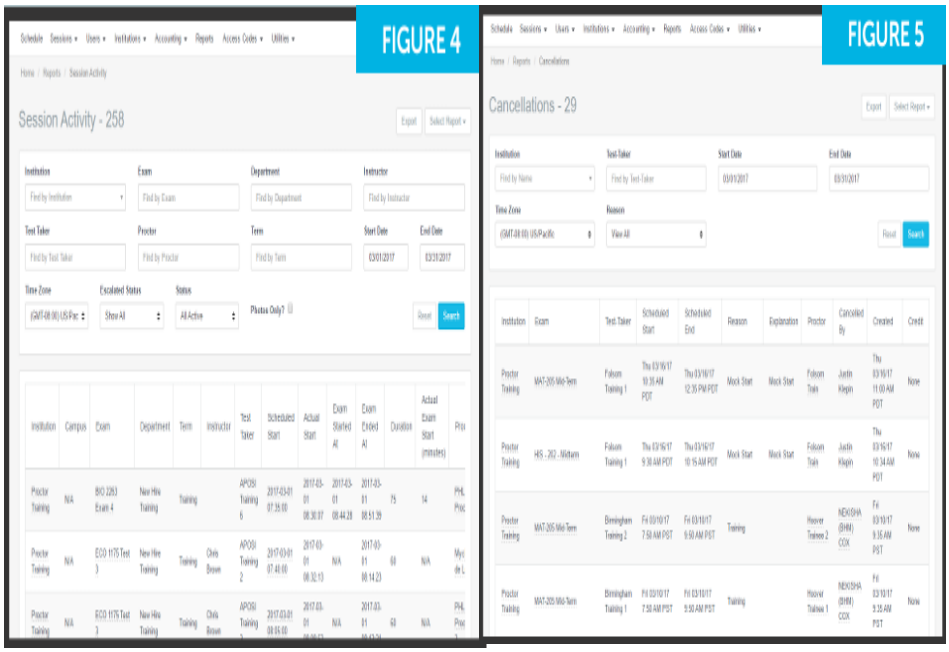


Figure 4: Activity Report Section

2) Test-taker appointment timeline

- The instructor can also view the events of a particular appointment in a detailed timeline (Figure 6), next page). This timeline displays all the events during an appointment in chronological order as well as any notes made by a proctor or manager. The timeline will also display the relevant test-taker and exam session information for the appointment.

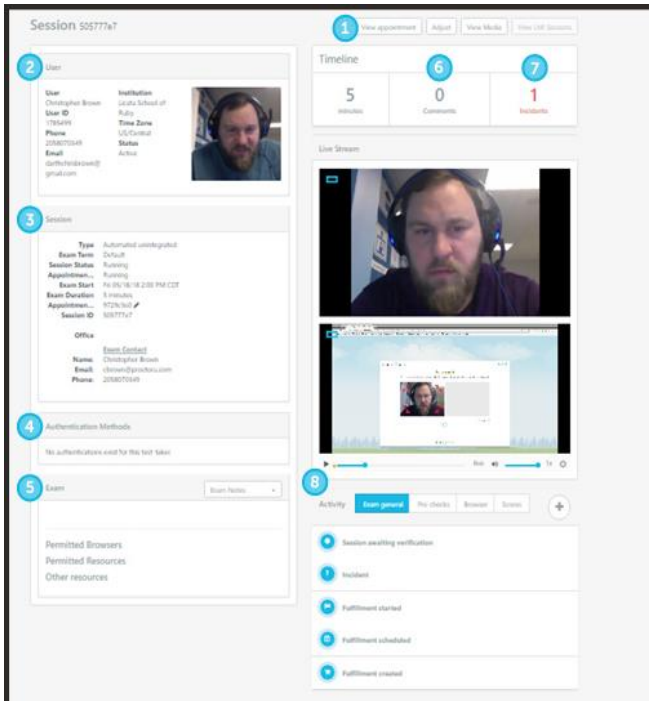


Figure 5: Appointment in a Detailed Timeline

3) Incident Reporting:

- ProctorU provides detailed **incident reports** for any suspicious activity detected during an exam, which faculty can access through the **Incident Report Center (IRC).**
- Reports include:
  - **Session Data**: Information like exam details, test-taker, and instructor.
  - **Incident Type**: Level of urgency (high, medium, low) with a description.
  - **Timeline of Events**: A chronological list of incidents or alerts triggered.

- **Media Playback**: Video playback options with flagged points of interest, allowing instructors to review alerts and verify behavior.

- Faculty receive an email alert when an incident report is generated and can also review the report's urgency breakdown and key statistics within the IRC.

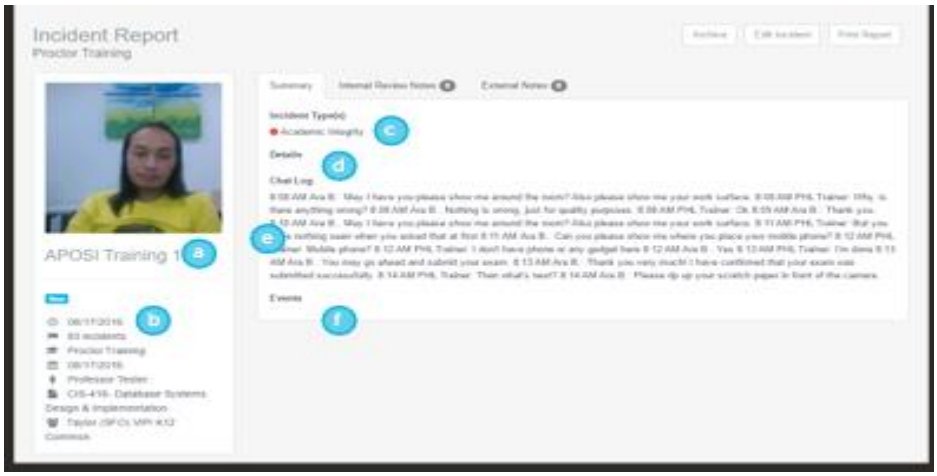


Figure 6: Exam Report

4) Statistical Breakdown:

- Faculty can view metrics on exam fulfillment rates, incident counts, and urgency levels of flagged events, helping instructors and administrators assess exam.

Figure 7: Exam Statistical

3.2.3. Test-Taker Experience on ProctorU

ProctorU provides a remote proctoring experience that involves several steps to ensure the integrity and security of online exams. The process begins with account setup, followed by exam scheduling, connecting to the proctoring room, and completing authentication and environment checks.

1) Account Creation and Profile Setup
   - Test-takers visit go.proctoru.com to create an account by selecting "New User? Sign up here."
   - During profile setup, they enter personal information and select their institution.
   - Accessibility options are available for those needing accommodations per the Americans with Disabilities Act (ADA), such as screen reader compatibility.
   - Once completed, the profile redirects users to the My Exams page for managing exams.
2) Exam Scheduling and Preferences
   - Test-takers set their **exam preferences**, including preferred appointment windows, which can be customized by selecting specific days and times.

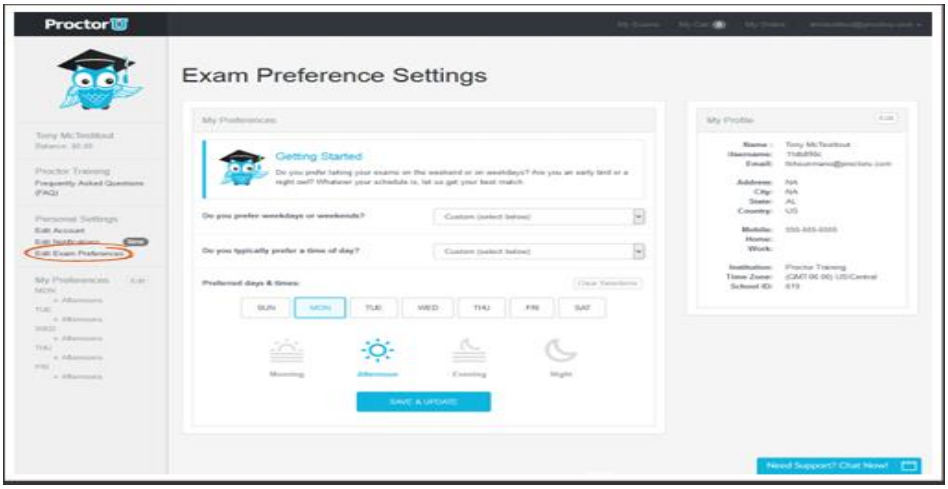


Figure 8: Exam Preference Settings

- From the **My Exams** page, test-takers can:
    - Use **Live Chat** for assistance.
    - **Test Equipment** to check for compatibility with ProctorU.
    - **Schedule New Exam** to book an appointment.
- The scheduling process involves confirming the institution and selecting an exam term. **Find Session** searches for available appointments, and test-takers can book their slot through **Book It**.
- Additional Fees: Late bookings within 72 hours incur a small fee, and on-demand appointments within an hour use the "Take It Now" feature for added flexibility, though at a premium.

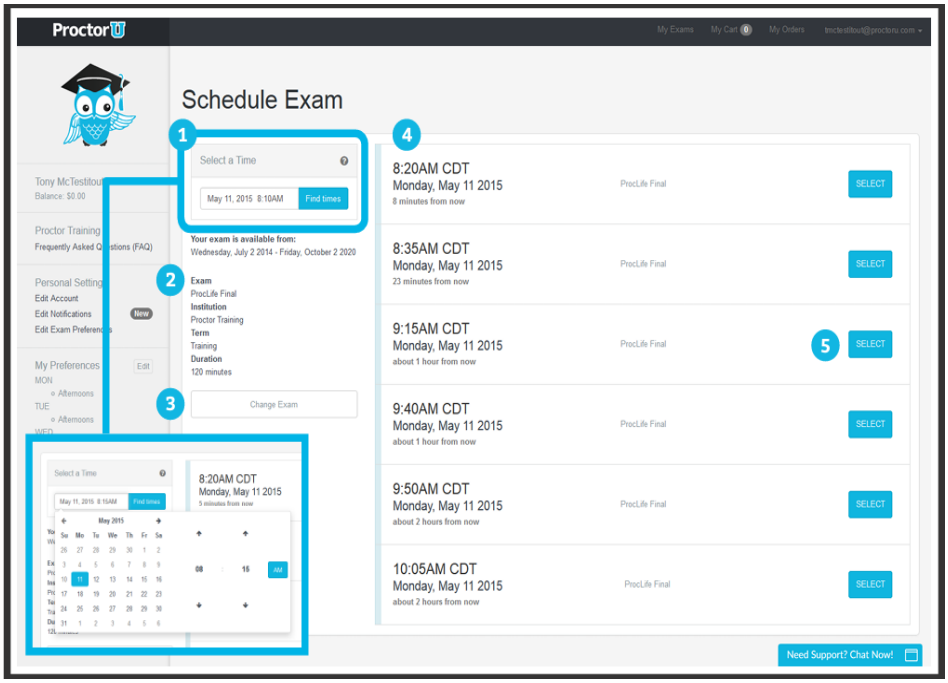


Figure 9: Schedule Exam Page

3) Connecting to the Proctor at Exam Time

- As the exam appointment nears, a countdown appears on the My Exams page. At the scheduled time, this changes to a start link directing the test-taker to the proctoring room.

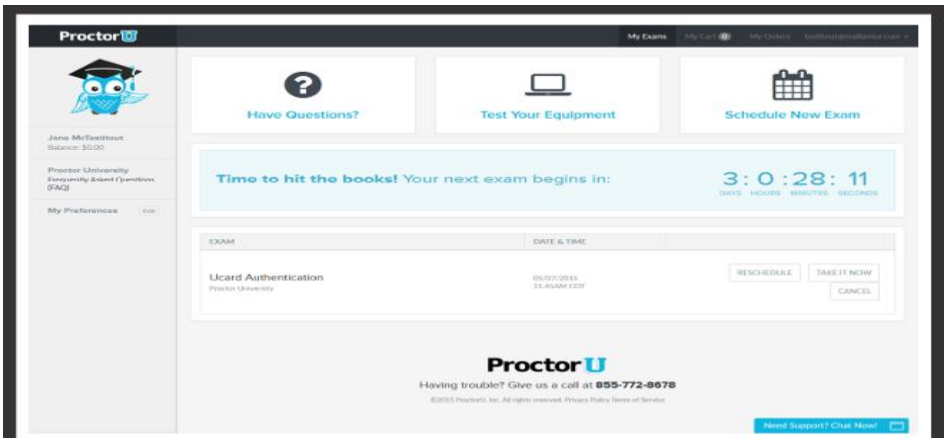


Figure 10: My Exams Page

- The connection process begins by downloading an applet, which enables screen-sharing and connects with the proctor via video and audio.

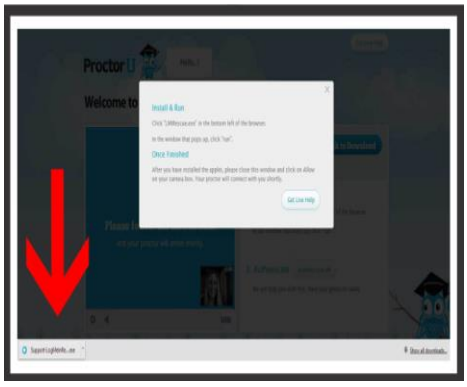


Figure 11: Download an applet

4) Authentication and Securing the Exam Environment

- After connection, the proctor initiates a two-step authentication and security check:

1) **Identity Verification:** The test-taker presents a valid ID for verification.

Figure 12: Identity Verification

2) **Challenge Questions**: Test-takers answer personal questions based on public records for additional identity confirmation.

Figure 13: Testing Session in ProctorU

3) **Environment Check**: The proctor instructs the test-taker to pan the camera around their workspace to ensure no unauthorized materials are present. If using an internal camera, the test-taker shows monitor edges using a reflective surface (e.g., mirror or CD) to check for attached materials.

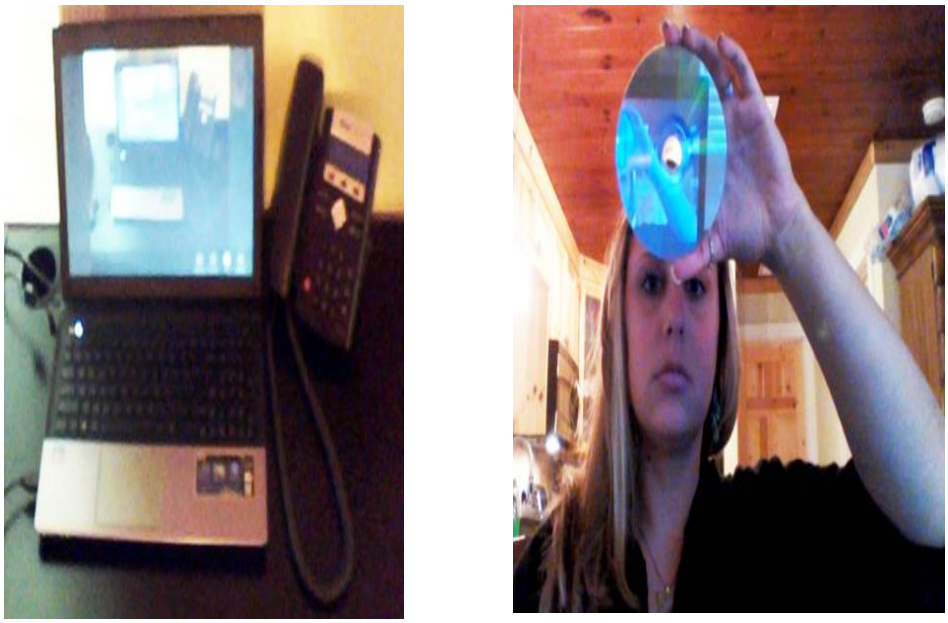


Figure 14: Environment Check

5) Proctoring and Monitoring
   - Throughout the exam, the proctor monitors the test-taker's screen and environment via live video. Any flagged behaviors are marked for review.
   - If any suspicious behavior is detected, a log entry is made, and relevant session segments are flagged, with video playback and timestamped alerts for further review by faculty.

3.3.How Honorlock Work

3.3.1. Instructor Process for Configuring Exams on Honorlock

Honorlock integrates directly with Brightspace, a popular Learning Management System (LMS), to facilitate proctored online exams. Instructors can configure exams, set proctoring options, and define student guidelines using their Honorlock dashboard.

1) Accessing Honorlock:

- Instructors navigate to Honorlock via the course's navbar under UA Tools in Brightspace. If the link is not visible, instructors may need to add Honorlock to the navbar.

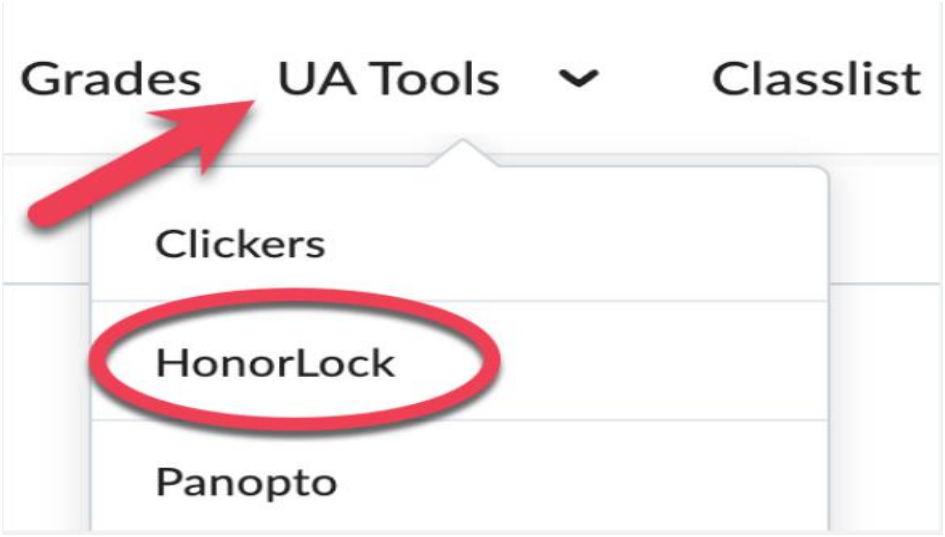


Figure 15: Navigate to Honorlock via the Course's Navbar

2) Enabling Exams for Proctoring:

- Once exams are created in Brightspace, instructors access the Honorlock dashboard, where all course exams are listed. By selecting an exam and clicking Enable, instructors initiate the proctoring setup process.

Figure 16: Enable Honorlock in Exam

3) Setting Up Proctoring Options:

- **Proctoring Features**: Instructors configure Honorlock's proctoring features to monitor test-taker behavior during exams. Options include:
    - **Record Webcam**: Captures HD video of the test-takers face and environment (enabled by default).
    - **Record Screen**: Records the computer screen.
    - **Record Web Traffic**: Logs browser activity.
    - **Student Photo and ID Verification**: Requires the test-taker to take a picture of themselves and their government-issued ID.
    - **Browser Guard**: Prevents access to external applications, triggering an alert if the test-taker attempts to navigate outside the browser.
    - **Disable Copy, Paste, and Print**: Limits the ability to copy or print exam content to prevent data leakage.

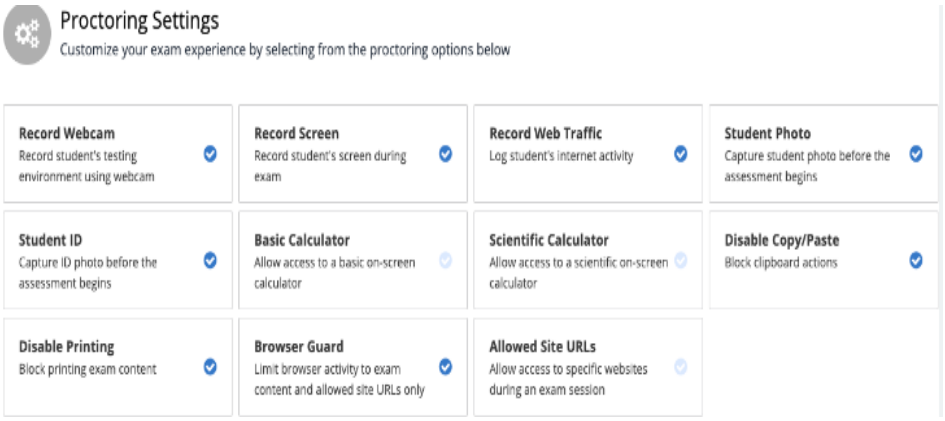


Figure 17: Setting Up Proctoring Options in Honorlock

4) Defining Student Guidelines:

- Instructors specify student allowances for certain resources during the exam, such as open-book access, notes, restroom breaks, scratch paper, or calculators. These settings are visible to both students and proctors.

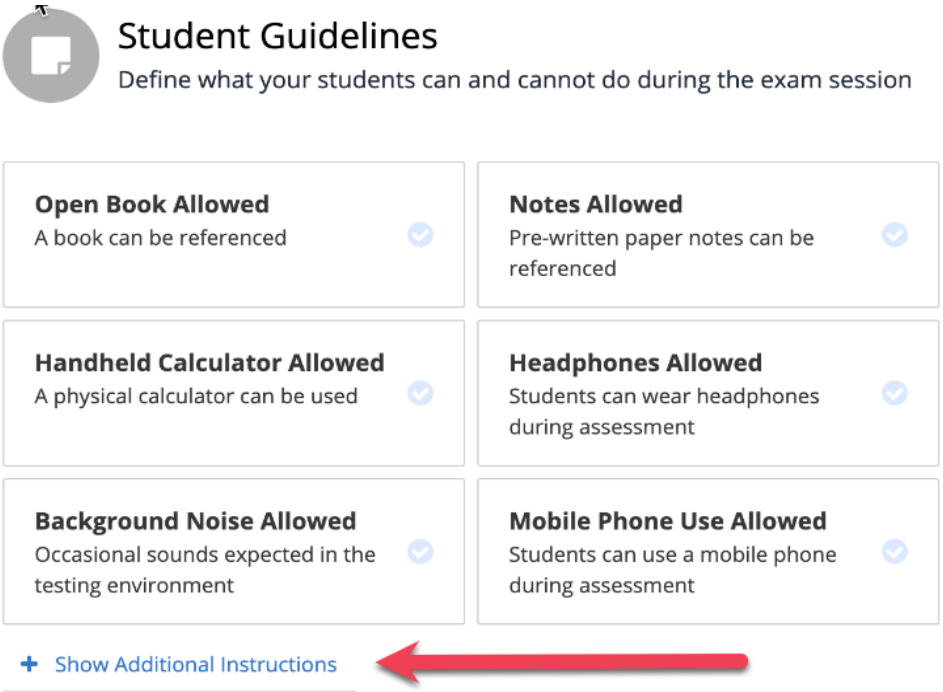


Figure 18: Defining Student Guidelines in Honorlock

- **Additional Instructions:** Instructors can add custom guidelines or course-specific instructions to clarify exam expectations. This section is visible to both students and proctors to ensure clear communication of rules.

5) Setting Accommodations:
   - Instructors can list any specific accommodation (e.g., extra time, special resources) in a designated section. This information is only accessible to proctors, ensuring that they are aware of any adjustments required for individual students.
6) Exam Visibility and Scheduling:
   - The Exam Visibility setting allows instructors to control whether the exam is always visible in the student view or only available during specific dates. Regardless of visibility settings, students cannot access the exam content until the designated testing window opens.

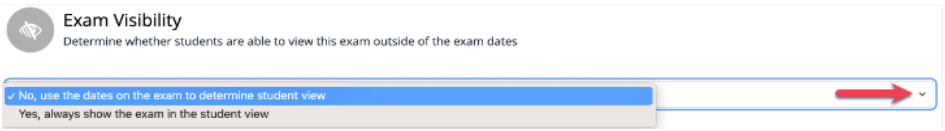


Figure 19: Exam Visibility Setting in Honorlock

7) Activating the Exam:
    - Once all settings are configured, instructors enable the exam, making it live on the Honorlock system and allowing students to schedule their proctored sessions.

3.3.2. Honorlock Test-Taker Experience

Honorlock provides a streamlined proctoring process that ensures exam integrity by guiding test-takers through setup, environment checks, and exam monitoring. The process requires students to use Google Chrome and the Honorlock Chrome extension, which enables monitoring through screen recording, webcam, and other integrated proctoring features.

1) Setting Up the Honorlock Chrome Extension
    - **Download Extension**: Before starting their first Honorlock-proctored exam, test-takers must download the Honorlock extension for Chrome. This process is done by selecting Honorlock from within their LMS course site (typically under “UA Tools” in Brightspace).
    - **Installation**: Once the extension is added, the browser confirms installation, allowing the test-taker to access Honorlock for all subsequent exams on the device.
2) Preparing for the Exam
    - **Required Resources**: Test-takers are advised to have a government-issued photo ID, a webcam and microphone, and a stable internet connection, along with any instructor-allowed materials such as scratch paper or calculators.
    - **Exam Environment**: Honorlock sets specific environment guidelines:
        - A well-lit, quiet room with a clean desk free from unauthorized materials.
        - A full 360-degree room scan, showing the workspace and testing area, is required before the exam begins.

- Prohibited items include headphones, smartwatches, phones, and dual monitors. Additionally, no other person is allowed in the room, and the test-taker must remain in the same location throughout the exam period.

3) Starting the Exam

- **Exam Access**: Within the exam's start window, test-takers open their exam on the LMS and select "Take the Quiz."
- **Launching Proctoring**: After initiating the exam, a "Launch Proctoring" button activates the proctoring session. Honorlock prompts the test-taker through several authentication steps:
  - **Face and ID Verification**: A photo of the test-taker and a scan of their ID are taken to confirm identity.
  - **Room and Desk Scan**: Using the webcam, test-takers perform a 360-degree view of their environment, showing that no unauthorized materials are present.
  - **Screen Recording:** Honorlock requests access to record the test-taker's screen, which monitors all activity within the testing browser

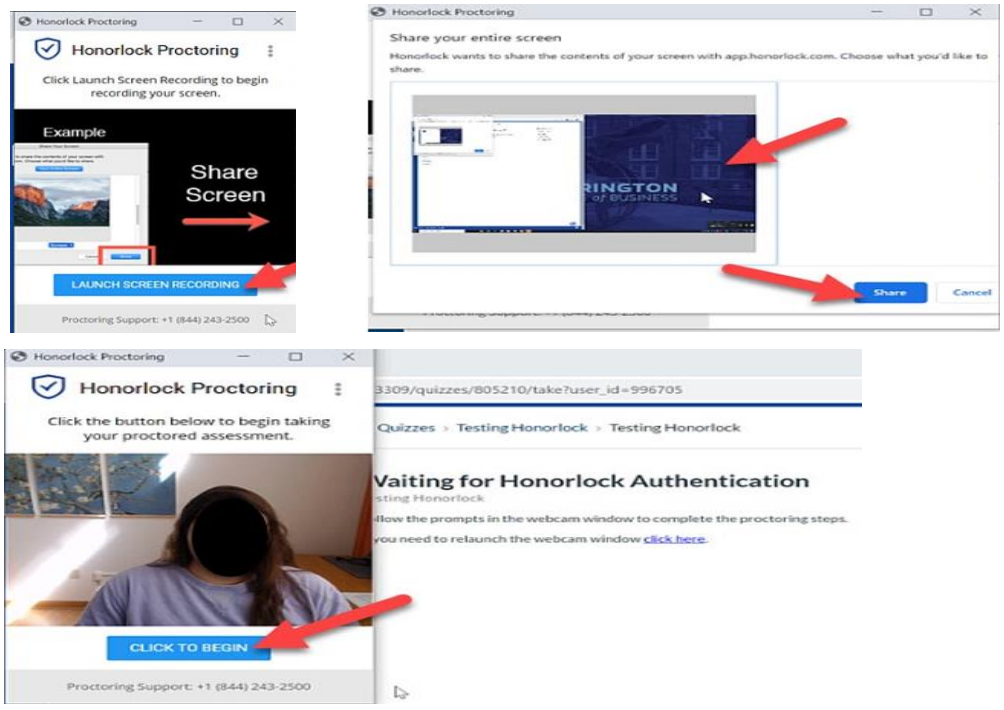


Figure 20: Starting the Exam in Honorlock

- **Support Access**: Throughout the exam, test-takers can access Honorlock's 24/7 support by selecting the support option from the toolbar.

4) Completing the Exam
   - **Submitting the Exam**: After completing the exam, test-takers submit it through the LMS. Honorlock then ends the proctoring session, confirming the session's closure through a pop-up notification.
   - **Recording and Review**: All interactions, including screen activity, webcam footage, and any flagged behaviors, are recorded and stored for review by instructors, providing a comprehensive record to ensure academic integrity.

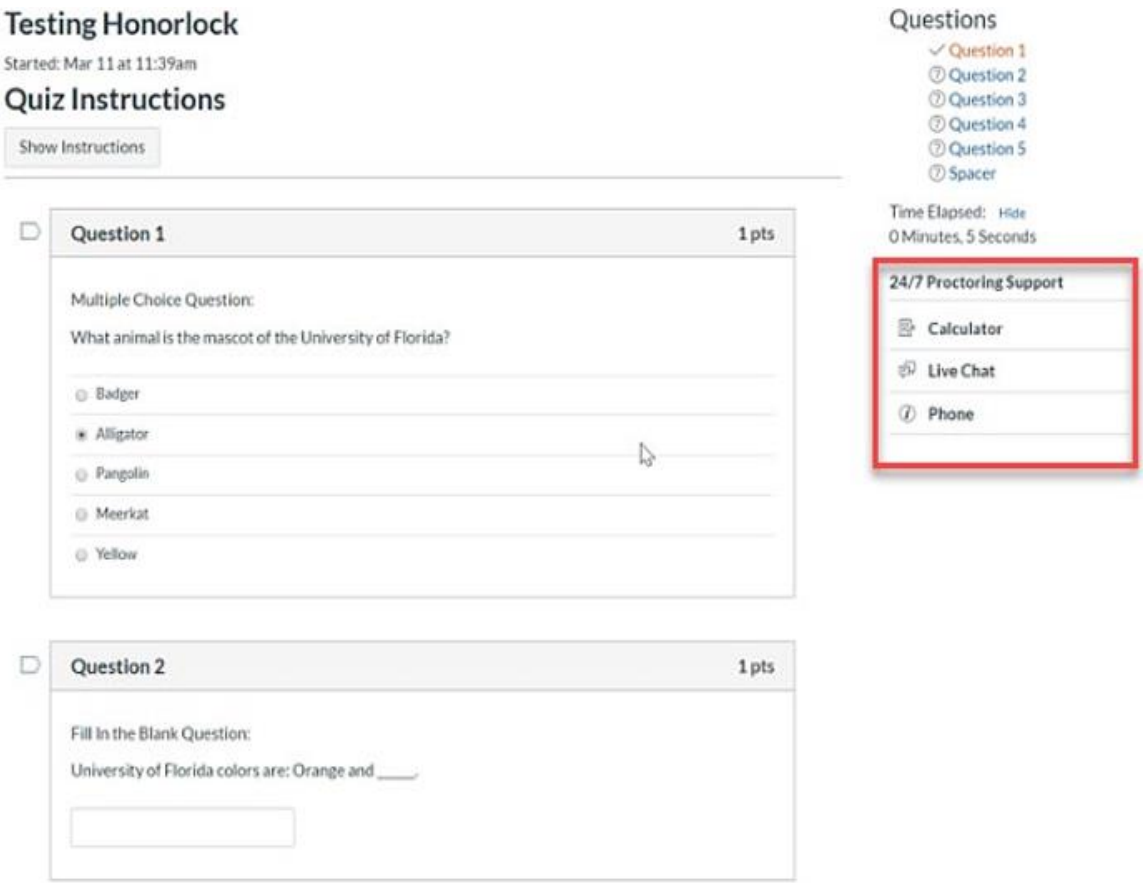


Figure 21: Quiz Interface in Honorlock

3.4.How Examity Work

3.4.2. From a Instructor's Perspective

Examity serves as a comprehensive proctoring platform that integrates with educational platforms (like onQ at Queen's University) to streamline the process of managing online exams. Here's an overview of how it works from the instructor's point of view:

1) Setting Up Your Exam In onQ:

- Include the word Examity in the Quiz name (e.g., Examity Midterm) – this will ensure the Quiz/exam gets imported to the Examity dashboard.

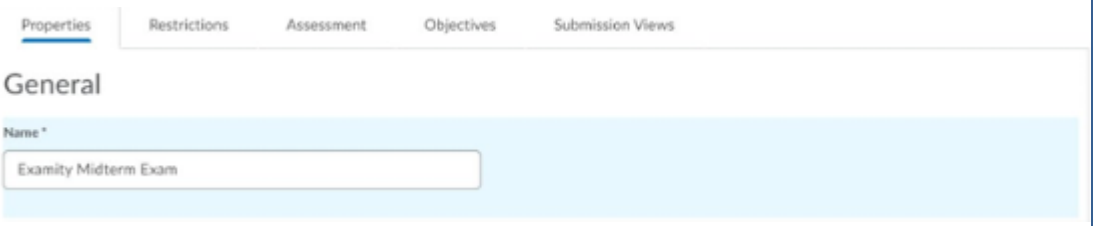


Figure 22: Enter Quiz Name in Examity

- In the Restrictions tab:
  - Ensure the Hide from Users checkbox is not checked.
  - Check Has Start Date and enter a Start Date and Time.
  - Check Has End Date and enter an End Date and Time.
  - Enter a Password into the Password field. (Note: students will not see this password – it will be entered automatically at the outset of students' exam sessions in Examity)

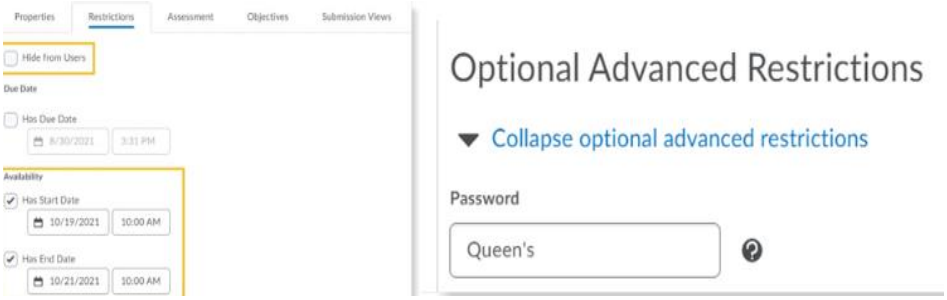


Figure 23: Restrictions Tab in Examity

- When you finish, click on Save and Close.
- Similarly, any changes you make to the above exam (Quiz) settings in onQ (i.e., name, start/end date, and password) after it is initially imported will only be reflected in Examity the following day.

2) Adding The Examity Link to Your onQ Course

You and your students will access the Examity Dashboard via a link in your onQ course. To add the Examity link to your course:

1. In Content, from the Existing Activities dropdown list, choose External Learning Tools.
2. Next, click on the Examity link.
3. Open the context menu (down arrow) next to the newly created Examity module, then select Edit Properties In-place.
4. Check the Open as External Resource checkbox. This will ensure the Examity Dashboard will open in a new browser tab instead of being constrained to a smaller iframe within your course. Note that this 'open as external resource' setting is recommended to optimize the student experience.

3) Dashboard Navigation

Instructors can access the Examity Dashboard, which has four main tabs:

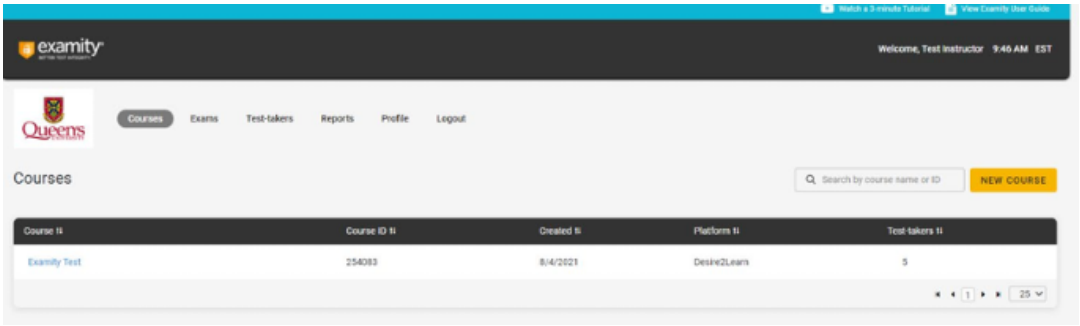


Figure 24: Examity Dashboard

- **Courses:** Displays of all linked courses, showing details like course ID, creation date, platform (e.g., D2L/onQ), and the number of enrolled students.

- **Exams:** Lists all scheduled exams across linked courses, with information on each exam's end date/time and the number of students who have scheduled the exam.
- **Test-takers:** Provides a list of students in all linked courses, indicating if they've completed their Examity profiles and if any accommodation or scheduling exceptions are set.
- **Reports:** Offers various reports, such as student scheduling data, and an overview of all exam sessions for effective monitoring.

4) Configuring Exams
   - **Basic Settings:** Exam details (name, start/end dates) are auto-imported from onQ, but instructors can configure security levels, rules, and instructions.
   - **Security Levels:** Instructors choose from five levels, ranging from fully automated proctoring (Automated Standard) to fully live proctoring (Live Premium), impacting how sessions are monitored and reviewed.

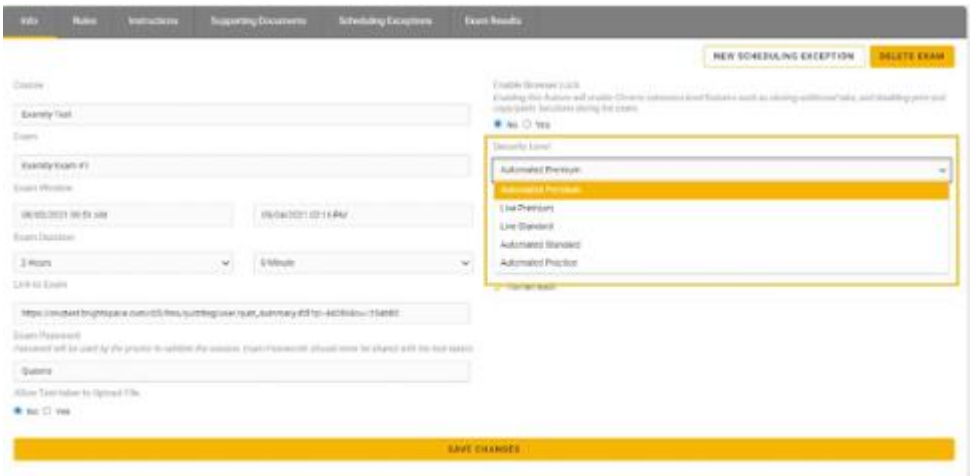


Figure 25: Security Configuration Page in Examity

   - **File Uploads:** They can allow students to upload files post-exam by enabling a specific setting.

- **Rules and Instructions**: Examity provides a set of standard rules and optional ones, which instructors can apply to ensure a secure testing environment. Additional instructions or rules can be added to guide students and proctors.

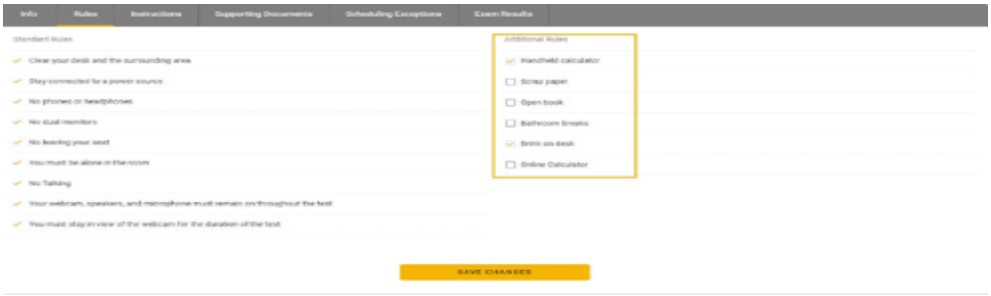


Figure 26: Rules and Instructions Page in Examity

- **Exam Instructions:**
  - **Adding a New Rule/Instruction:**
    To add a new rule or instruction, select the Add Another Set of Instructions button – in the pop-up window that appears, enter the rule/instruction, and check the necessary boxes to select whether the rule/instruction will appear for the reviewer/proctor, the test-taker, or both. Then, click Add Instructions.

Figure 27: Adding a New Rule/Instruction

  - **Importing Instructions:**
    To import a set of instructions from another exam, click Import Special Instructions from an Existing Exam, select the relevant course and exam from the dropdown menus, then click Import.

Figure 28: Importing Instructions

    Once imported, review the instructions, make any necessary edits, then click Save Changes.

- **Supporting Documents:**
  This tab allows you to add any document(s) that students will need to complete the exam – note that Examity will automatically block a student from starting their exam if they have not yet downloaded the supporting document(s).
  To add supporting documents, click the Upload New link, and select the file(s) from your computer – once they have uploaded, click Save Changes.

Figure 29: Supporting Documents

5) Managing Accommodations and Exceptions
   - **Accommodation:** Student accommodation is set at the course level, ensuring tailored support for students with specific needs.

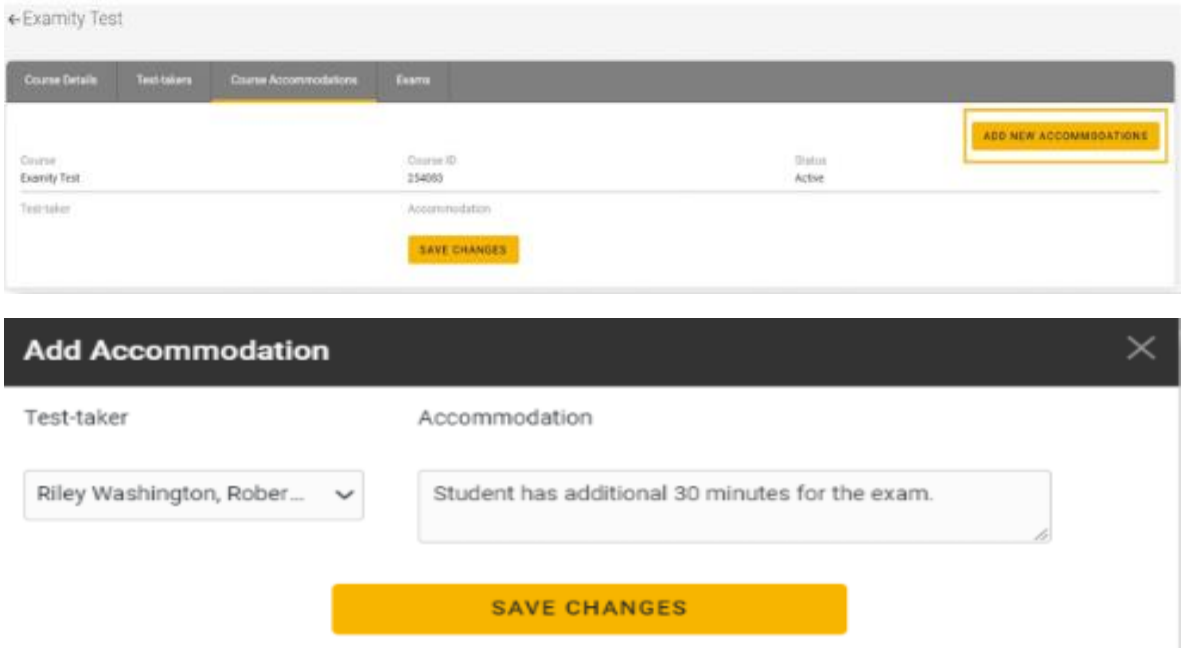


Figure 30: Add Accommodation Page

- Scheduling Exceptions: Instructors can modify exam windows for individual students by setting exceptions, allowing for flexible exam scheduling if needed.

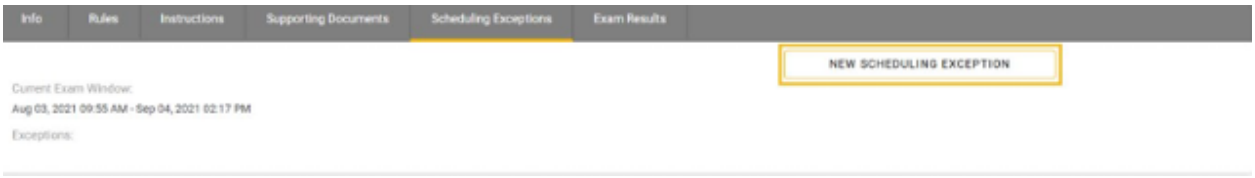


Figure 31: Add New Scheduling Exceptions

6) Monitoring and Reminding Students

- **Tracking Exam Status:** Instructors can track how many students have scheduled an exam, and view details of those who haven't through the Exam Results tab. They can send email reminders to prompt students to schedule their exams.

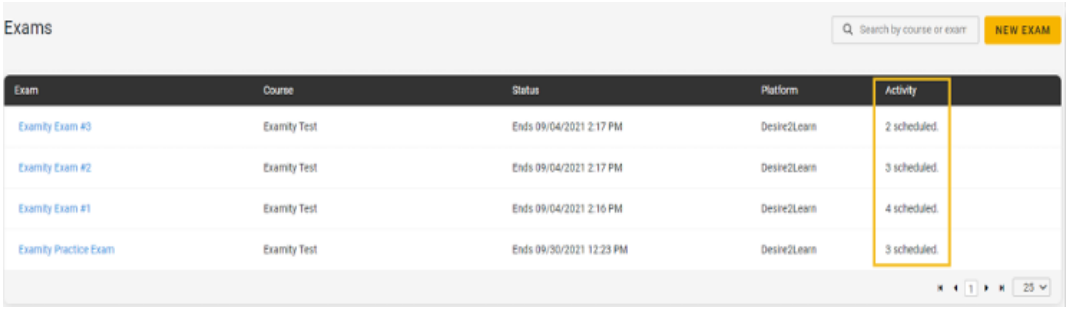


Figure 32: Track Exams Page

- **Reviewing Exam Results**: Completed exams are reviewed based on the chosen security level, and the **Flag System** provides an overview of any issues or violations, categorized as:
  - **Green**: No violation (e.g., identity verification success).
  - **Yellow**: Minor rule violations not likely affecting academic integrity.
  - **Red**: Breaches of academic integrity, such as unauthorized internet use or collaboration.

- **Blue**: Technical issues that may have affected the exam.

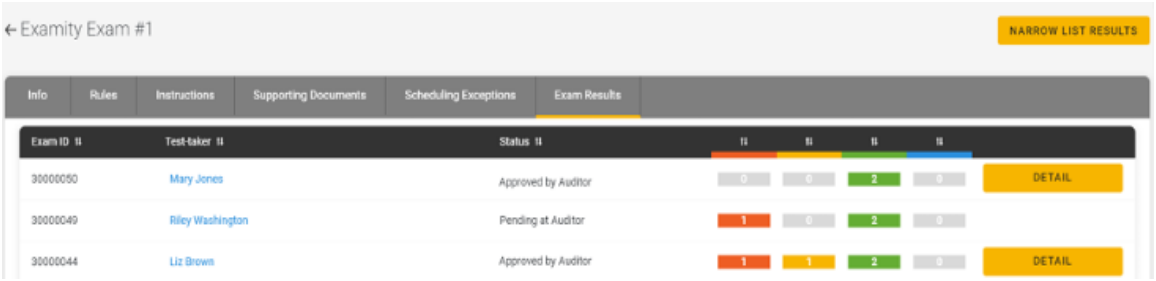


Figure 33: Reviewing Exam Results Page

7) Flagged Events and Additional Context

- Instructors can review flagged events in detail by accessing the Flags tab. Time-stamped flags allow direct navigation to specific moments in the exam session, with accompanying comments and captured images for context.

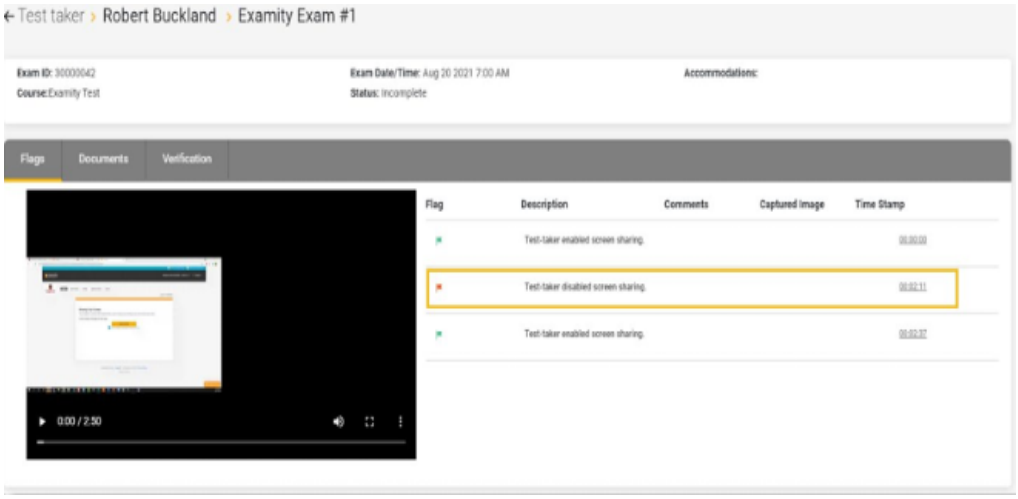


Figure 34: Review Flagged Events in Detail

8) Data Retention

- **Privacy Compliance:** According to Queen's agreement with Examity, recordings are retained for 30 days for general exams and one year for red-flagged exams, ensuring student privacy.

3.4.2. From a Student's Perspective

Examity functions as a secure online proctoring platform that facilitates exam scheduling, verification, and monitoring to maintain academic integrity. Here's a breakdown of how it works for students:

1) Access and Registration:
   - Go to prod.examity.com/DbtLabs.
   - Enter your information and select your institution from the drop-down menu.
   - Use your work email for registration to access your Examity dashboard.
2) Complete your Profile:
   - On the dashboard, select "My Profile" to begin profile setup.

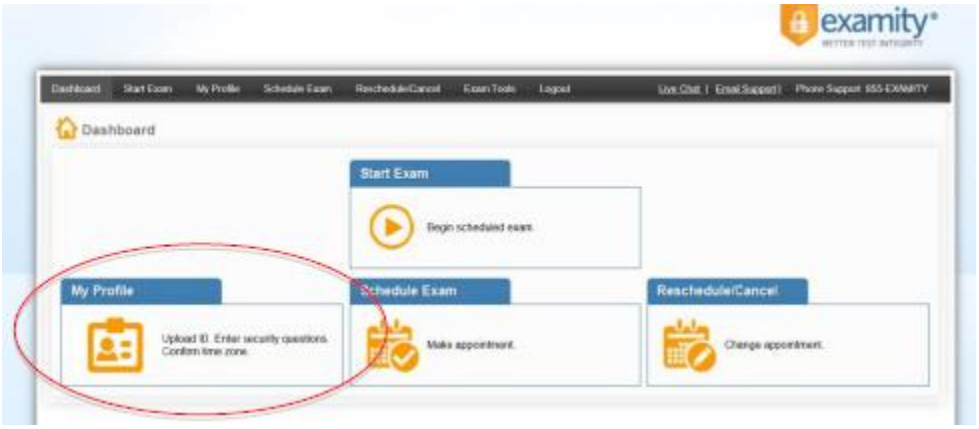


Figure 35: Examity Dashboard

   - Choose your time zone.
   - Upload a photo of your ID, which will be needed for identity verification every time you take an exam.
   - Answer three unique security questions.
   - Enter a keystroke biometric signature, which will be used for future authentication.

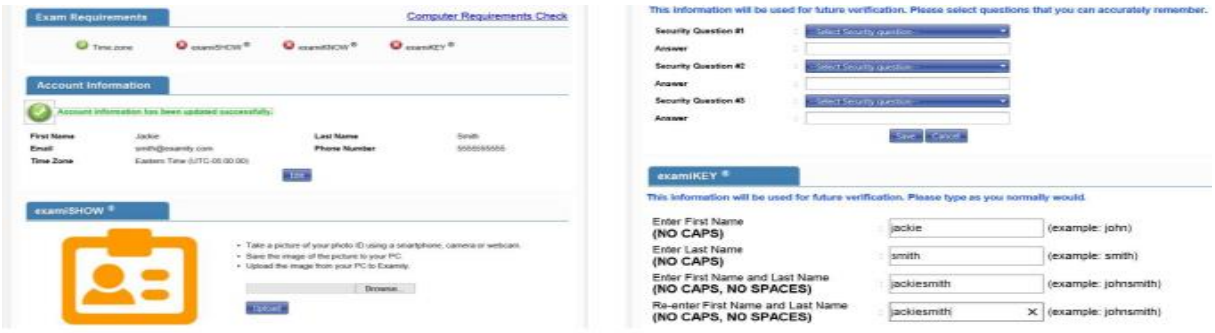


Figure 36: Enter Your Details in Examity

3) Run Computer Requirements Check
   - On the "My Profile" page, click the computer requirements check link in the upper right corner.
   - Complete the check on the same computer you will use for the exam to ensure compatibility with Examity.

Figure 37: Computer Requirements Check

4) Schedule Your Exam
   - When you're ready, click "Schedule Exam" on the dashboard.
   - Select your instructor, course, and exam name.
   - Choose a date and time for the exam using the calendar.
   - If scheduling less than 24 hours before the exam, select the "On-Demand" option (additional fees apply).

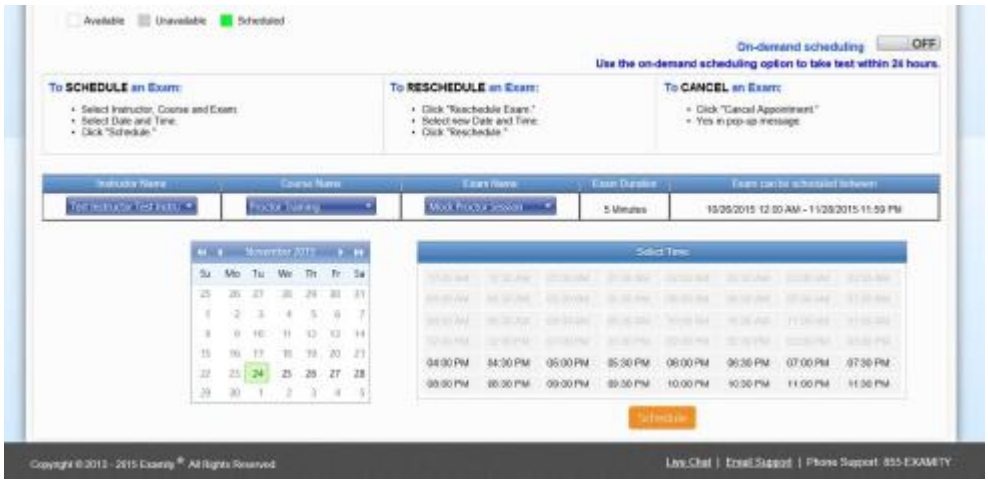


Figure 38: Schedule Exam Page

5) Reschedule or Cancel (if needed)
   - Go to the “Reschedule/Cancel” tab on the dashboard.
   - Canceling over 24 hours before the exam provides a full refund; canceling within 24 hours incurs a $5 fee.
6) Start the Exam
   - At the scheduled time, log in to your Examity dashboard.
   - Click “Start Exam” and connect to a proctor.

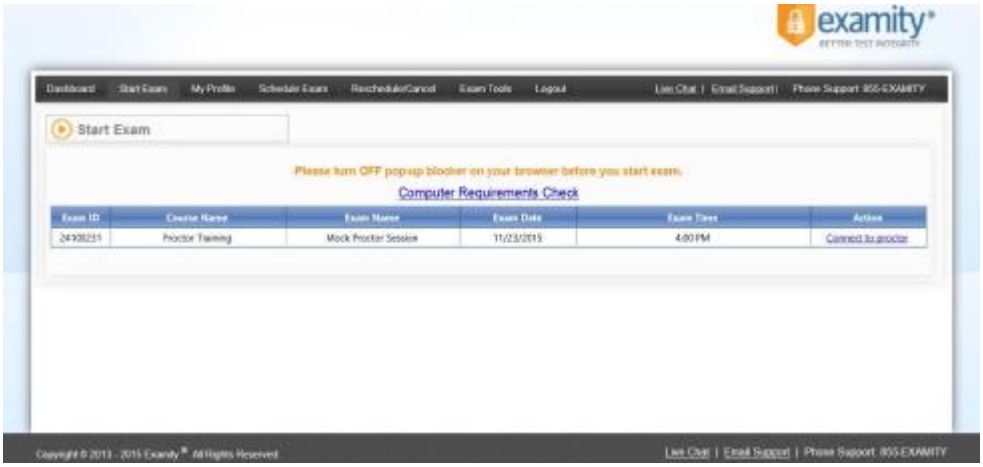


Figure 39: Start Exam Page in Examity

7) Complete the Authentication Process
    - Follow these steps with the proctor:
        - Verify your identity by showing your photo ID to the webcam.
        - Review exam rules provided by the proctor.
        - Show your workspace: Perform a 360˚ room and desk sweep with the webcam to confirm a clear workspace.
        - Answer a security question as an additional security measure.
        - Enter your biometric signature.
        - Agree to the user agreement and exam rules.
8) Beginning the Exam
    - After completing authentication, click “Begin Exam” to start.
    - The proctor will enter the exam password, granting access to the exam.

3.5.Moodle Functions

3.5.1. Introduction to Moodle

Moodle is a free, open-source Learning Management System (LMS) designed to provide educators, students, and administrators with a dynamic online learning environment. It allows educational institutions and organizations to create customized virtual learning spaces, fostering an environment that supports continuous, anytime, anywhere learning.

3.5.2. Core Functionality and Features

- **Responsive User Interface:** Moodle offers a modern, user-friendly interface that is accessible on both desktop and mobile devices. This responsive design ensures that users can access course materials, submit assignments, and engage with content from any device.
- **Personalized Dashboard:** Each user has a customizable dashboard that displays their current, past, and upcoming courses, along with a timeline for tracking deadlines and calendar events.

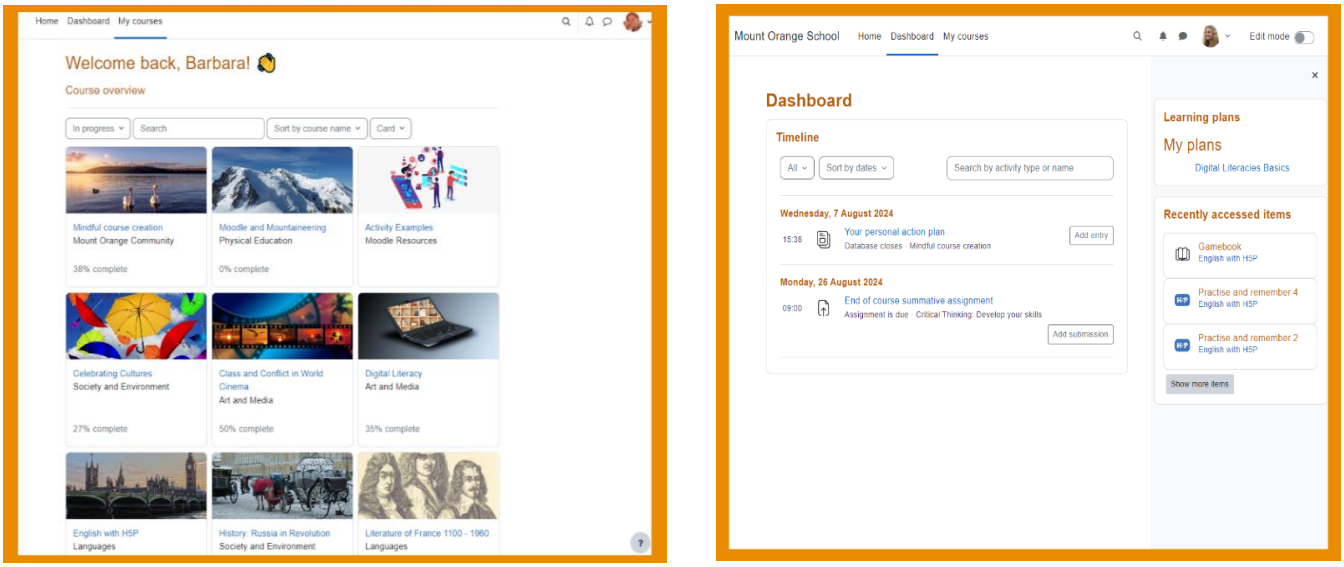


Figure 40: Customizable Moodle Dashboard

- **Collaborative Learning Tools:** Moodle supports interactive learning through forums, wikis, glossaries, and databases, allowing users to

collaborate and engage in discussions. These tools foster community-driven learning.

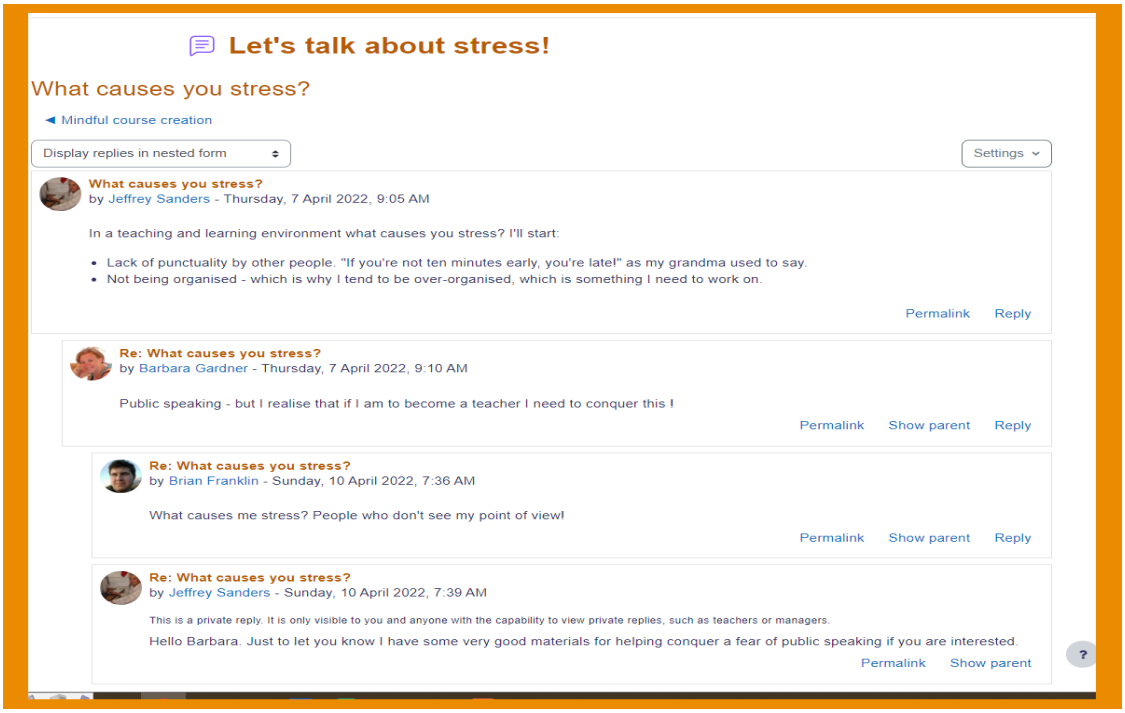


Figure 41: Discussions in Moodle

- Educational Tools for Course Management:
  - **Content Management:** Educators can upload files from various sources, including cloud storage platforms, and organize them into structured courses. Moodle also supports multimedia integration, enabling the inclusion of videos and audio for more engaging content.

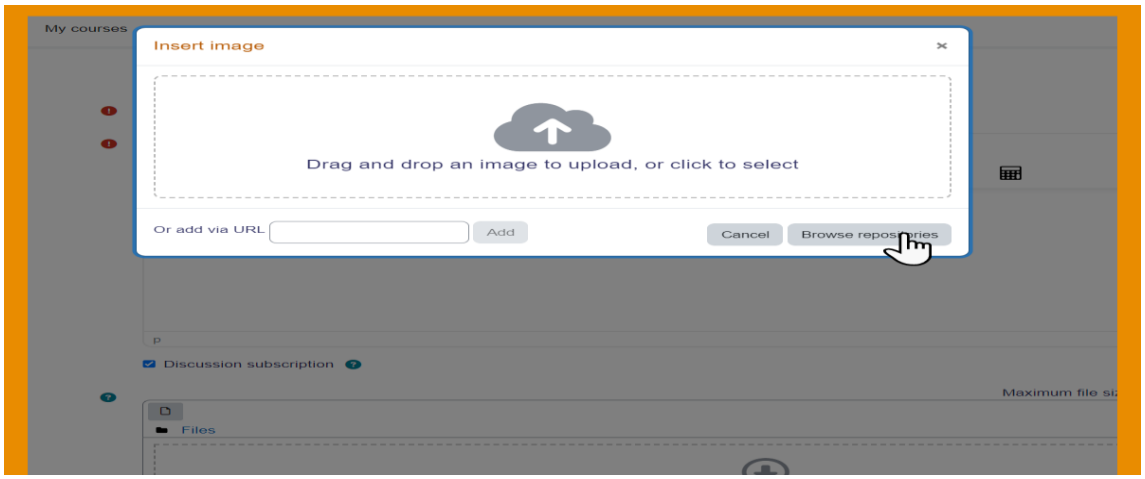


Figure 42: Educators can upload different files in Moodle

- **Activity and Progress Tracking:** Educators can monitor students' progress and completion of activities, providing tools to track engagement at both the individual and course levels.

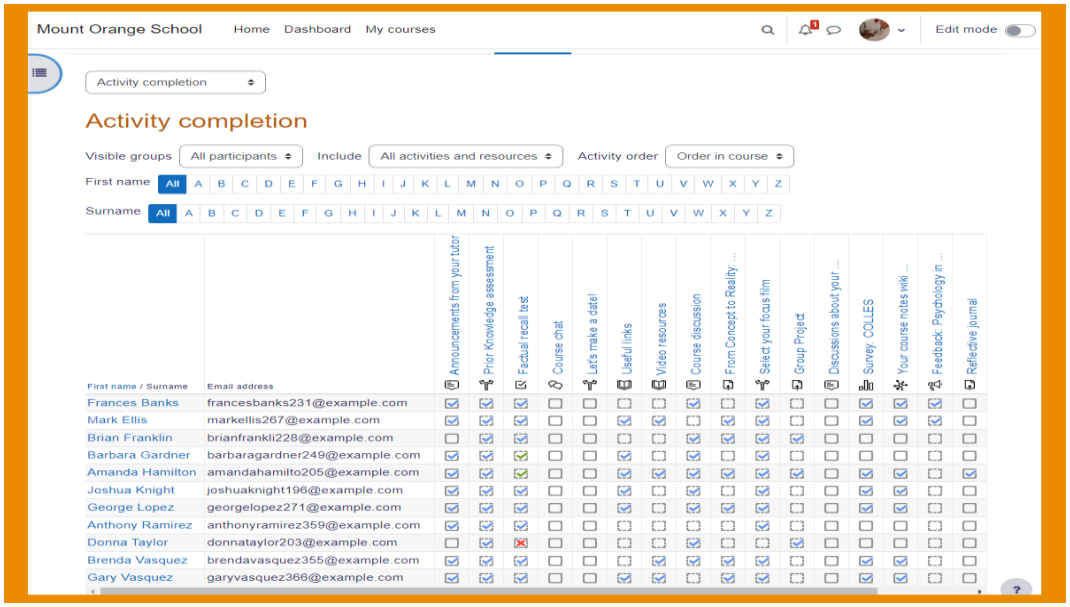


Figure 43: Activity and Progress Tracking in Moodle

- **Grading and Assessment:** Moodle offers a flexible grading system with advanced options, including competency-based marking and peer/self-assessment activities. These tools support diverse assessment methods to evaluate students effectively**.**

- Administrative and Security Features:
  - **Customizable Themes and Layout:** Moodle allows administrators to tailor the site's appearance to match institutional branding. It includes a wide range of theme options and the ability to design unique layouts.
  - **Secure Authentication and Access Control**: With over 50 authentication options, Moodle provides a secure system for enrolling users. Administrators can define roles to manage permissions and access across the platform.
  - **Data Privacy and Security:** Moodle is committed to data privacy, enabling institutions to create secure, private learning spaces. Regular security updates help protect the system from vulnerabilities.

- Interoperability and Integration:
  - **Open Standards Support:** Moodle is compatible with several educational standards, such as IMS-LTI and SCORM, which allows for the easy integration of external content and tools.
  - **Plugin System:** Moodle's modular design supports a variety of plugins for customization. This extensibility enables institutions to adapt Moodle to their specific needs by adding functionalities like advanced analytics or integrations with other software.

3.5.3. Impact on Educational Delivery

- Moodle has transformed online education by providing a robust platform that supports traditional, blended, and fully online learning environments. Its extensive toolset allows educators to create tailored learning experiences that support diverse educational strategies, from self-paced learning to collaborative group projects.

3.6.Findings and Discussion

3.6.1. Comparison of Platform Features

Table 3: Comparison of Platform Features

Feature	ProctorU	Honorlock	Examity	Moodle
Face Detection	Yes	Yes	Yes	No
Gaze Tracking	Yes	Limited	No	No
Voice Recognition	Limited	Yes	No	No
Object Detection	No	Yes	Limited	No
Browser Lockdown	Yes	Yes	Yes	No
LMS Integration	Yes	Yes	Yes	Yes
Pricing	Variable	Variable	Variable	Free

3.6.2. Technical and Privacy Analysis

The privacy concerns associated with proctoring tools are significant, with students often uncomfortable about continuous monitoring. Platforms like ProctorU and Honorlock have tried to address this by providing transparent data use policies. However, the possibility of false positives (e.g., gaze tracking flagging innocent off-screen glances) can negatively impact students.

3.6.3. Gaps in Current Solutions

Current solutions still rely heavily on human intervention to verify flagged incidents, suggesting that AI alone is insufficient for full automation. Additionally, open-source LMS platforms like Moodle lack built-in proctoring features, making it harder for institutions to implement integrity solutions without third-party tools.

3.7. Proposed Model for an Intelligent E-Learning Platform

To address the identified gaps, we propose an intelligent e-learning platform with integrated AI features to monitor exams and enhance engagement. Our proposed model includes:

- **Face and Identity Verification**: Using the CASIA WebFace dataset, this feature ensures the right individual is taking the test.
- **Eye Tracking**: Real-time eye tracking flags gaze anomalies and alerts the proctor.
- **Object Detection**: With COCO dataset support, this feature flags unauthorized items visible in the video feed.
- **Privacy-Centric Design**: We incorporate privacy-preserving machine learning to ensure compliance with data regulations, encrypting all sensitive information.

Our model is built on a tech stack that includes **Laravel** for backend services, **FastAPI** for ML model integration, and **React** for the front end.

3.8. Academic Integrity in Online Assessments

3.8.1. The Role of AI Technologies

Recent research highlights the transformative role of AI in online assessments. AI-driven systems can monitor student behavior in real-time, flagging suspicious activities that may indicate cheating. By employing machine learning algorithms, platforms can analyze patterns in student behavior, providing educators with actionable insights to uphold integrity.

3.8.2. Real-Time Monitoring and Proctoring

Real-time proctoring features are becoming standard in e-learning platforms. These systems utilize AI to analyze video feeds from surveillance cameras, detecting behaviors indicative of cheating. For instance, the integration of biometric authentication ensures that the individual taking the exam is indeed the registered student, significantly reducing the risk of identity fraud.

3.9. Cheating Detection Techniques

3.9.1. Deep Learning Approaches

A comparative study on deep transfer learning algorithms for cheating detection demonstrates the effectiveness of using advanced architectures like ResNet50V2 combined with Long Short-Term Memory (LSTM) networks. These models achieve high accuracy in identifying various forms of cheating during examinations, such as using unauthorized materials and communicating with others.

3.9.2. Behavioral Analysis

Cheating detection frameworks not only rely on technology but also incorporate behavioral analysis. By monitoring student interactions and movements during exams, platforms can identify potential cheating behaviors, such as consulting hidden notes or communicating with peers.

3.10. Trends in AI and Online Exams

3.10.1. Adoption of AI

The integration of AI in online examinations is a growing trend, offering significant advantages in efficiency and security. AI tools assist with grading, monitoring, and providing feedback, thus streamlining the assessment process. This trend is critical in the wake of the COVID-19 pandemic, which has accelerated the shift towards online learning and assessment.

3.10.2. Biometric Authentication

Biometric authentication technologies, as discussed in the "Secure Online Examination" framework, provide an additional layer of security. By employing fingerprint or facial recognition systems, educational institutions can ensure that only authorized individuals access examination materials.

3.11. Frameworks for Secure Online Examinations

Implementing a blockchain-based framework for online examinations presents a promising avenue for enhancing security. By ensuring data integrity and transparency, blockchain technology can help mitigate risks associated with data breaches and unauthorized access to examination systems.

4. System Requirements

4.1. Functional requirements

1) **Admin Role:**

Admin oversees the platform's management, user control, and global settings:

Function Name	Admin Login
Description	Admin logs into the platform using valid credentials to access system features.
Pre-Condition	Admin must have a valid account.
Priority	High

Function Name	Admin Signup
Description	A function allows user to register -have an account- on the website.
Pre-Condition	Must enter valid data.
Priority	High

Function Name	Logout
Description	Admin can log out from their accounts safely.
Pre-Condition	User must be logged in
Priority	Medium

Function Name	Password Reset
Description	Allows admin to reset their password if forgotten.
Pre-Condition	Admin must initiate a reset request.
Priority	High

Function Name	Create User Account
Description	Admin creates new user accounts with roles (e.g., Professor, Admin, or Student).
Pre-Condition	Admin must be authenticated.
Priority	High

Function Name	Edit User Information
Description	Admin modifies details like email, name, or assigned role of existing users.
Pre-Condition	User account must exist in the system.
Priority	Medium

Function Name	Delete User Account
Description	Admin deletes inactive or unauthorized user accounts.
Pre-Condition	User account must exist in the system.
Priority	Medium

Function Name	Assign Roles
Description	Admin assigns specific roles with permissions (e.g., Professor, Student).
Pre-Condition	User account must exist in the system.
Priority	High

Function Name	Modify User Access
Description	Admin enables or disables access to specific platform areas for individual users.
Pre-Condition	User account must exist in the system.
Priority	Medium

Function Name	Reset User Password
Description	Admin resets passwords for any user and sends a new password via email.
Pre-Condition	User account must exist in the system.
Priority	High

Function Name	View User Activities
Description	Admin monitors user activities such as logins, quiz attempts, and course participation.
Pre-Condition	User must be registered in the system.
Priority	Medium

Function Name	Configure APIs
Description	Admin manages APIs for communication with ML models (e.g., Face Detection, Eye Tracking).
Pre-Condition	Backend APIs must be properly configured.
Priority	High

Function Name	View Cheating Detection Logs
Description	Admin reviews logs of flagged cheating incidents to ensure data privacy and compliance.
Pre-Condition	Monitoring system must be active.
Priority	High

Function Name	Backend Service Setup
Description	Admin configures and monitors backend services (e.g., Laravel, Fast API, MySQL) to ensure smooth operation.
Pre-Condition	Backend services must be accessible.
Priority	High

Function Name	Control Content Access
Description	Admin disables content or suspends accounts based on behavior or violations.
Pre-Condition	Admin must be authenticated.
Priority	High

Function Name	View Registered Courses
Description	Admin can access a user's profile to view registered courses and quiz marks.
Pre-Condition	User must have registered for courses.
Priority	Medium

Function Name	Suspend Accounts
Description	Admin can temporarily suspend user accounts if they violate platform rules.
Pre-Condition	User account must exist.
Priority	Medium

2) Professor Role:

Function Name	Professor Login
Description	The professor logs into the platform using credentials to access course management and quiz setup.
Pre-Condition	Must have a valid professor account.
Priority	High

Function Name	Professor Signup
Description	A function allows professor to register -have an account- on the website.
Pre-Condition	Must enter valid data.
Priority	High

Function Name	Logout
Description	Professor can log out from their accounts safely.
Pre-Condition	User must be logged in
Priority	Medium

Function Name	Password Reset
Description	Allow the professor to reset the password if forgotten.
Pre-Condition	Must have a valid professor account.
Priority	High

Function Name	Block Student
Description	Professor can block specific students from accessing their course if necessary.
Pre-Condition	Student must be registered in the course.
Priority	Medium

Function Name	View Registered Students
Description	Shows the number and details of students registered for a specific course.
Pre-Condition	Course must have at least one student.
Priority	High

Function Name	Upload Video Lectures
Description	Uploads lecture videos to the course module for student access.
Pre-Condition	Video files must be under size limits.
Priority	High

Function Name	Upload Course Materials
Description	Uploads PDFs and supplementary notes for students to download.
Pre-Condition	Must have access to upload content.
Priority	High

Function Name	Organize Course Modules
Description	Structures course materials into modules for better navigation and tracking.
Pre-Condition	At least one course must be active.
Priority	Medium

Function Name	Create Quiz
Description	Creates quizzes with various question types (e.g., MCQ, short answers).
Pre-Condition	Quiz module must be enabled.
Priority	High

Function Name	Edit and Delete Quiz
Description	Allows modification or deletion of quizzes.
Pre-Condition	Quiz must already exist.
Priority	Medium

Function Name	Set Quiz Duration
Description	Defines the duration for quiz completion.
Pre-Condition	Quiz must be created.
Priority	High

Function Name	Activate Browser Lockdown
Description	Enables lockdown mode to prevent unauthorized browsing during exams.
Pre-Condition	Quiz must be active.
Priority	High

Function Name	Apply ML Models to Quiz
Description	Selects ML models (e.g., face detection, eye tracking) for quiz monitoring.
Pre-Condition	ML models must be configured in the system.
Priority	High

Function Name	Identity Verification for Quiz
Description	Ensures students pass face recognition before starting quizzes.
Pre-Condition	Student must initiate the quiz.
Priority	High

Function Name	Receive Cheating Alerts
Description	Receives real-time alerts for suspicious behavior (e.g., off-screen gaze).
Pre-Condition	AI monitoring tools must be active.
Priority	High

Function Name	Review Flagged Sessions
Description	Manually reviews quiz sessions flagged by AI and overrides results if necessary.
Pre-Condition	Quizzes must be completed.
Priority	Medium

Function Name	Access Cheating Reports
Description	Views reports summarizing incidents detected by the AI tools.
Pre-Condition	Reports must be generated.
Priority	High

Function Name	View Exam Participation
Description	Displays the number of students who took the exam.
Pre-Condition	At least one student must take the quiz.
Priority	Medium

Function Name	View Quiz Performance
Description	Shows quiz scores and statistics for individual students.
Pre-Condition	Quiz must be completed by students.
Priority	Medium

3) Student Role

Students engage with the platform by accessing course material, taking quizzes, and interacting with instructors:

Function Name	Student Login
Description	The student logs into the platform using credentials to access course management and quiz setup.
Pre-Condition	Must have a valid professor account.
Priority	High

Function Name	Student Signup
Description	A function allows student to register -have an account- on the website.
Pre-Condition	Must enter valid data.
Priority	High

Function Name	Logout
Description	Student can log out from their accounts safely.
Pre-Condition	User must be logged in
Priority	Medium

Function Name	Password Reset
Description	Allow the student to reset the password if forgotten.
Pre-Condition	Must have a valid professor account.
Priority	High

Function Name	Manage Profile Information
Description	Students can update personal details such as name, email, and password.
Pre-Condition	User must be logged in
Priority	Medium

Function Name	Course Registration
Description	Students select courses they want to enroll in.
Pre-Condition	Course must be available for registration.
Priority	High

Function Name	Access Course Materials
Description	Students can view and download videos, notes, and assignments uploaded by professors.
Pre-Condition	Must be registered for the course.
Priority	High

Function Name	Track Course Progress
Description	Allow students to monitor their progress through course modules.
Pre-Condition	Course modules must be available.
Priority	Medium

Function Name	Quiz Reminder Notifications
Description	Sends alerts to students about upcoming quizzes and deadlines.
Pre-Condition	Quiz must be scheduled.
Priority	Medium

Function Name	Identity Verification
Description	Students complete a face recognition check before accessing a quiz.
Pre-Condition	Face recognition system must be active.
Priority	High

Function Name	Open camera
Description	Students must open camera before take the quiz.
Pre-Condition	Quiz is exist.
Priority	High

Function Name	Start Quiz
Description	Students participate in quizzes monitored by AI models (e.g., face tracking).
Pre-Condition	Student must complete identity verification.
Priority	High

Function Name	Receive Suspicious Behavior Alerts
Description	Alerts students if they engage in suspicious behavior during a quiz (e.g., looking off-screen).
Pre-Condition	Quiz monitoring must be active.
Priority	High

Function Name	Submit Quiz Answers
Description	Students submit quiz responses within the allotted time limit.
Pre-Condition	Quiz must be in progress.
Priority	High

Function Name	View Quiz Results
Description	Students can view their graded results after submission.
Pre-Condition	Quiz grading must be completed.
Priority	Medium

4.2.Non-Functional Requirements

1) **Scalability**
    - The system should handle up to 500 concurrent users, with 300 simultaneous quizzes and 50 live monitoring sessions, without performance degradation.
    - The platform must support future growth with an increase to 10,000 students and 1,000 courses.
2) **Performance**
    - The system should load pages in under 2 seconds for 90% of interactions, even under high traffic.
    - Any content updates made by admins or professors (e.g., quiz modifications) should reflect instantly across all user sessions.
3) **Security**
    - Password reset and authentication processes must comply with industry security standards.
    - Passwords must be hashed with algorithms like bcrypt and stored securely.
    - Sensitive data (e.g., user credentials and quiz results) must be encrypted.
    - Admins, professors, and students should have specific permissions, with no unauthorized cross-access between roles.
    - Put strong validation on inputs.
4) **Availability**
    - The system should guarantee 99.9% uptime, ensuring access during critical exam sessions.
5) **Reliability**
    - Quiz submissions and updates must be processed with ACID compliance to avoid inconsistencies or loss of responses.

6) **Usability and User Experience**
    - Use clear layouts and labels to simplify access to course content, quizzes, and reports.
    - Ensure mobile responsiveness to offer a seamless experience across smartphones, tablets, and desktops.
    - Provide a dark mode option to enhance readability and reduce eye strain during long usage sessions.
7) **Monitoring and Reporting**
    - Real-time alerts and detailed logs must be generated for suspicious activities (e.g., face detection failures or gaze shifts).
    - Track metrics such as course views, quiz participation, and student engagement to help professors assess their content.
8) **Maintainability and Support**
    - Ensure the platform follows a modular design to simplify future updates or feature enhancements.
    - Integrate external tools (e.g., ML models) via RESTful APIs to ensure interoperability.

4.3.Hardware & Software Requirements

1) Hardware Requirements

Table 4: Hardware Requirements

Component	*Minimum Requirement*	*Recommended Requirement*
Processor (CPU)	Intel Core i5 / AMD Ryzen 5	Intel Core i7+ / AMD Ryzen 7+
RAM	8 GB	16 GB or higher
Storage	256 GB SSD	512 GB SSD or higher
Camera	720p integrated or external webcam	1080p HD webcam
Internet	Stable connection (5 Mbps or higher)	High-speed broadband (10+ Mbps)
GPU *(Optional)*	Integrated GPU	Dedicated GPU (for ML processing)

2) Software Requirements

Table 5: Software Requirements

Category	Technology Used
Frontend	React.js (JavaScript)
Backend	Laravel (PHP)
Machine Learning API	FastAPI (Python)
Database	MySQL
Server OS	Ubuntu 20.04+ / Windows 10+
Browser	Google Chrome / Firefox
Camera Access	WebRTC-compatible browser
Version Control	Git & GitHub
Package Managers	npm, pip, Composer
Other Tools	Postman, VS Code, Docker, Google Drive, Colab

5. System Design

5.1.System architecture

The system architecture consists of three elements:

1. Controller: handles request flow and never handle data logic
2. model: handles data logic and interacts with the database
3. view: handles the data presentation and is what the user sees

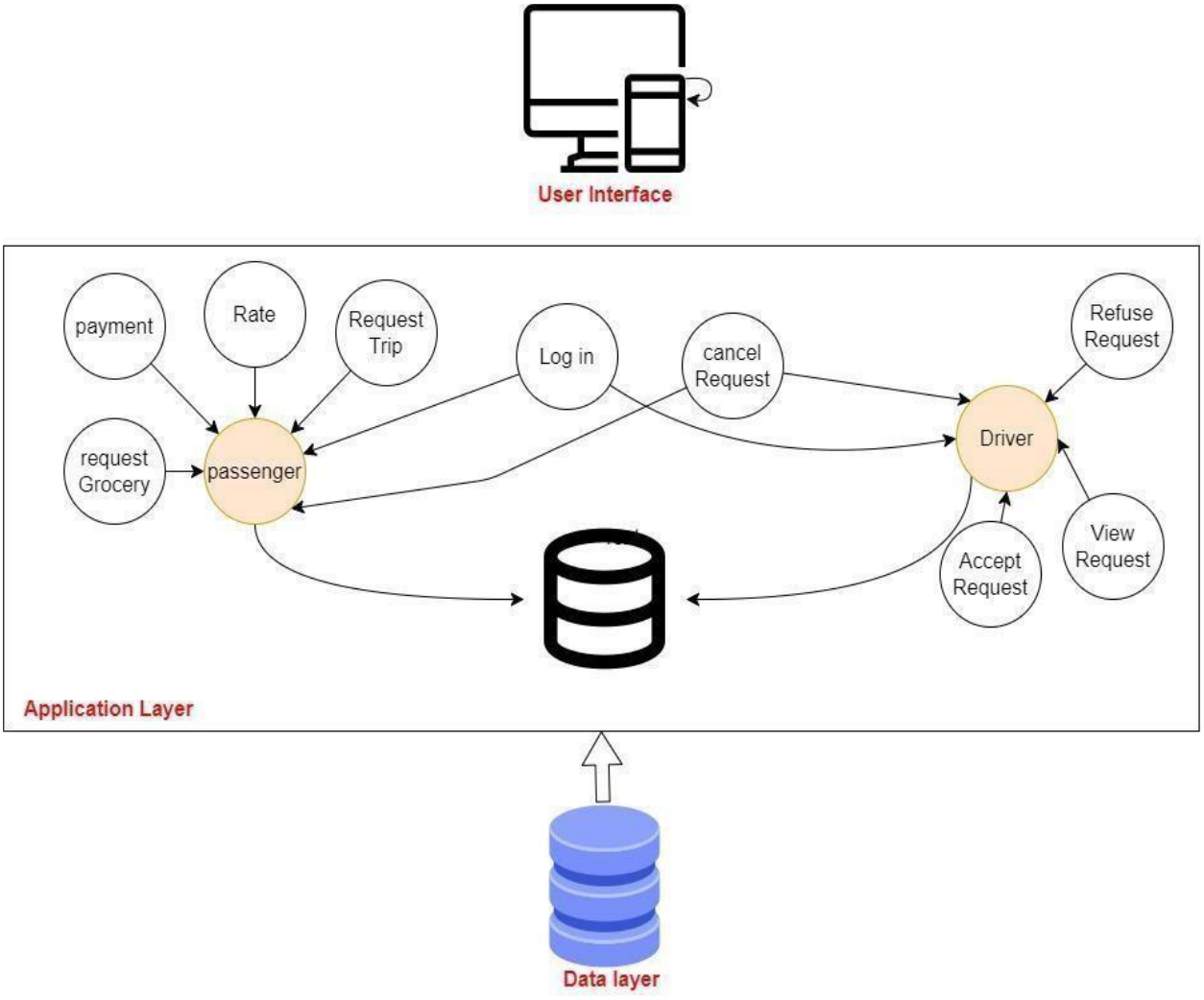


Figure 44: MVC Pattern

The description of how the system will work:

1. User interacts with the system in any way like (retweeting, replying or blocking ...etc.)
2. The controller will take the request and ask the model to get the data that the user wants
3. The model will then request the data from the database and handle it and return it to the controller
4. The controller will take data and forward it with a view to get a presentation of the data
5. The view will take the data from the controller and return a presentation of the data
6. The controller will finally take the presentation of the data from the view and return it to the user

Benefits of using MVC pattern:

1. 1Easy code maintenance
2. 2 It avoids complexity by dividing the system into three parts
3. 3 Modification does not affect the whole system

5.2. Use-Case Diagram

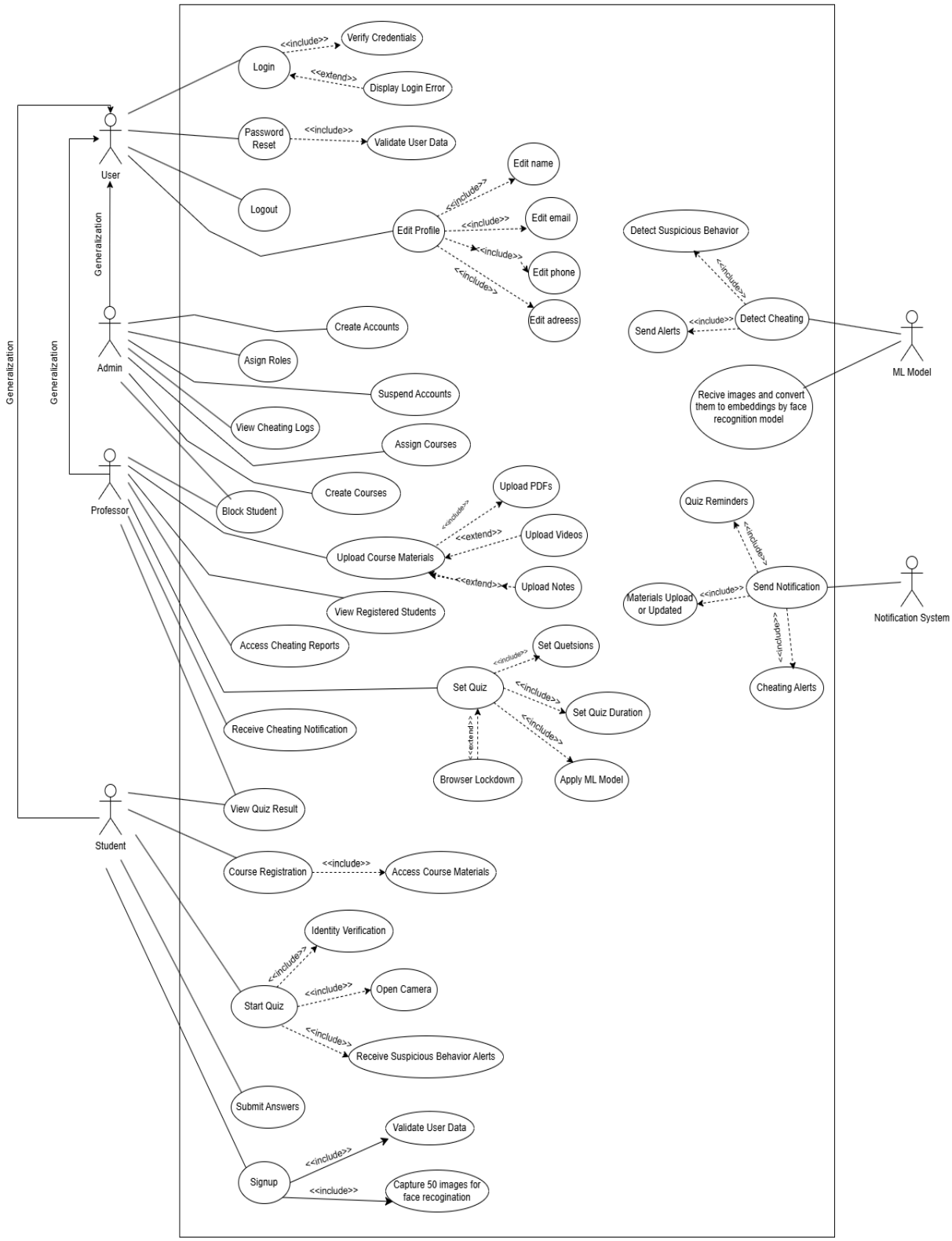


Figure 45: Use-Case Diagram

Detailed Use-Cases Description:

Name	Login
Goal	Allow users to authenticate into the system using valid credentials.
Actors	User (Admin, Professor, Student)
Preconditions	• The user must already have a registered account in the system. • The system database must be active and reachable.
Postconditions	• **If successful:** The user gains access to the system. • **If unsuccessful:** An error message is displayed, and the user remains on the login page.
Error Situations	• Invalid credentials (wrong username/password). • System downtime or connectivity issues.
Standard Process	1. User inputs their email/username and password. 2. The system verifies the credentials. 3. If credentials are valid, the user is redirected to the relevant dashboard. 4. If invalid, the system displays an error message (e.g., "Invalid Username or Password").

Name	Password Reset
Goal	Allow users to reset their password if they forget it.
Actors	User (Admin, Professor, Student)
Preconditions	• The user must have a registered account in the system. • The user must provide a valid email or username associated with their account.
Postconditions	• The user's password is reset, and they receive a • confirmation email or SMS with a reset link or code.
Error Situations	• Invalid email or username entered. • Reset link expired or already used. • System issues prevent sending the reset link.
Standard Process	1. User clicks the "Forgot Password" link on the login page. 2. User enters their registered email or username. 3. System sends a password reset link or code to the user's email. 4. User enters the code that sends in email. 5. User sets a new password and confirms it. 6. If the reset is successful, the system displays a 7. success message, and the user can log in with the new password.

Name	Logout
Goal	Allow users to safely end their session and log out from the system.
Actors	User (Admin, Professor, Student)
Preconditions	• The user must be logged in.
Postconditions	• The user session is ended, and the system redirects to the login or homepage.
Error Situations	• The system failed to end the session due to connectivity issues. • Incomplete logout may leave the session active in another browser tab.
Standard Process	1. Incomplete logout may leave the session active in another browser tab. 2. The system clears the user session and cookies. 3. User is redirected to the login or home page. 4. If any error occurs, the system displays an appropriate error message.

Name	Edit Profile
Goal	Allow users to update their personal information.
Actors	User (Admin, Professor, Student)
Preconditions	• The user must be logged in. • The user should have access to their profile page.
Postconditions	• The updated profile information is saved successfully in the database.
Error Situations	• Required fields are missing or incorrectly formatted (e.g., invalid email). • Network or database issues prevent the update.
Standard Process	5. User navigates to their profile page. 6. User updates fields such as name, email, phone, or address. 7. The system validates the changes. 8. If valid, the system saves the changes and displays a success message. 9. If invalid, the system prompts the user to correct the errors.

Name	Create Accounts
Goal	Admin creates new accounts for users (professors or students).
Actors	Admin
Preconditions	• Admin must be logged into the system. • Admin must have appropriate privileges to create accounts.
Postconditions	• A new user account has been added to the system.
Error Situations	• Missing required information (e.g., email). • Database connectivity issues preventing account creation.
Standard Process	1. Admin selects "Create Account" option. 2. Admin fills out required details (name, role, email, etc.). 3. System checks for duplicate entries (e.g., existing email). 4. If valid, the new account is created. 5. If invalid, an error message is displayed (e.g., "Email already exists").

Name	Assign Roles
Goal	Allow the admin to assign specific roles (e.g., student, professor, or other admin) to registered users.
Actors	Admin
Preconditions	• Admin must be logged into the system. • The user to be assigned a role must exist in the system. • Admin has the necessary permissions to modify user roles.
Postconditions	• The selected user is assigned the new role. • The system logs the role assignment activity.
Error Situations	• User not found in the database. • Invalid role assignment (e.g., assigning a role that does not exist). • System error during the role update process.
Standard Process	1. Admin navigates to the user management page. 2. Admin selects a user and chooses the role to assign. 3. The system validates the role and the user information.

Name	Suspend Accounts
Goal	Temporarily disable a user's access to the system
Actors	Admin, Professor
Preconditions	• Admin must be logged into the system. • The user to be suspended must exist in the system.
Postconditions	• The selected user is suspended, and the account is marked as inactive. • A notification or alert is sent to the user (if applicable). • The system logs the suspension action.
Error Situations	• User not found in the system. • System error during the account suspension process. • Admin attempts to suspend a system-critical account (e.g., another admin account without proper authority).
Standard Process	1. Admin navigates to the user management page. 2. Admin selects a user and chooses the "Suspend Account" option. 3. The system validates the user information and confirms the action. 4. If confirmed, the user's account status is set to "Suspended.". 5. If an error occurs, the admin is notified, and the action is logged.

Name	Receive Cheating Alerts
Goal	Notify professor about suspicious behavior during exams.
Actors	Professor
Preconditions	• A quiz must be in progress, and ML monitoring must be active.
Postconditions	• Cheating alert is sent to the relevant actor (admin or professor).
Error Situations	• Notification system fails to send the alert due to downtime. • Delayed alerts due to network issues.
Standard Process	1. ML model detects suspicious behavior. 2. Notification system sends an alert to the professor. 3. The professor reviews the flagged session and takes appropriate action.

Name	View Cheating Logs
Goal	Provide the admin with access to logs of detected cheating incidents.
Actors	Admin
Preconditions	• Admin must be logged into the system.
	• Cheating incidents must have been detected and recorded in the system by the ML model or monitoring tools.
Postconditions	• The admin can view detailed cheating logs, including the time, user, and type of cheating incident. • Admin may take further action (e.g., suspending user or send warnings).
Error Situations	• No cheating logs were found (if no incidents were recorded). • System error while retrieving logs from the database.
Standard Process	1. Admin navigates to the "Cheating Logs" section. 2. System retrieves and displays the cheating logs. 3. Admin can filter or search for specific logs (e.g., by date or user). 4. Admin reviews the details of each incident and acts if necessary. 5. If an error occurs (e.g., "No logs found" or "Database error"), the system displays an appropriate message.

Name	Upload Course Materials
Goal	Professors upload videos, notes, or PDFs for students to access.
Actors	Professor
Preconditions	• The professor must be assigned to a course.
Postconditions	• Materials are accessible to students enrolled in the course.
Error Situations	• File format is not supported. • Upload fails due to network issues.
Standard Process	1. The professor selects a course and navigates to the "Upload Materials" section. 2. The professor uploads files (PDFs, videos, or notes). 3. The system validates and stores the files. 4. Materials become available for students to access.

Name	Organize Course Modules
Goal	Enable professors to create, manage, and organize course modules (e.g., by topic or week).
Actors	Professor
Preconditions	• The professor must be logged in. • The course must be active and assigned to the professor.
Postconditions	• The course modules are organized and saved in the system. • Students can view the modules when accessing the course.
Error Situations	• Invalid data entry (e.g., missing module titles or descriptions). • System error during module creation or update.
Standard Process	1. The professor selects the course and navigates the course module section. 2. The professor adds, updates, or reorganizes modules by topic or week. 3. The system validates the input and saves the changes. 4. If successful, a confirmation message is displayed. If there is an error, the system displays an appropriate message.

Name	Access Cheating Reports
Goal	Allow professors to view detailed reports on detected cheating incidents.
Actors	Professor
Preconditions	• The professor must be logged in. • Cheating incidents must be detected and reported by the system.
Postconditions	• The professor can view cheating reports and take actions such as blocking students or notifying them. • The system logs the professor's access to the report.
Error Situations	• No cheating incidents are reported for the course. • System error while retrieving the reports.
Standard Process	1. The professor logs in and selects the course from the dashboard and navigates the cheating reports. 2. The system retrieves the cheating reports and displays them. 3. The professor can review the reports and take appropriate actions. 4. If no reports are found, the system displays a message (e.g., "No cheating incidents detected"). 5. In case of an error, the system displays an error message.

Name	Set Quiz
Goal	Allow professors to create quizzes for students.
Actors	Professor
Preconditions	• The professor must be logged in and assigned to at least one course.
Postconditions	• The quiz is linked to the course and visible to students
Error Situations	• Missing quiz questions or other mandatory fields. • Incorrect quiz settings (e.g., invalid duration).
Standard Process	1. The professor selects a course and chooses the "Create Quiz" option. 2. The professor sets quiz title, duration, and questions. 3. Quiz is saved and linked to the course.

Name	View Registered Students
Goal	Allow professors to view the list of students registered for a specific course.
Actors	Professor
Preconditions	• The professor must be logged in. • The course must be assigned to the professor. • Students must have registered for the course.
Postconditions	• The system displays a list of students along with their relevant details (e.g., name, email, student ID). • The professor can take further actions such as filtering the list or contacting students.
Error Situations	• No students are registered for the course. • The system fails to retrieve the list due to a server or database error.

Name	View Quiz Results
Goal	Allow students and professors to view quiz results and performance details.
Actors	Professor, Student
Preconditions	• Student or professor must be logged in.
	• The quiz must be completed, and results must be available.
Postconditions	• Students can view their own quiz results. • The system logs the professor's access to the report.
Error Situations	• Quiz results are not yet published. • System error while fetching the quiz results.
Standard Process	1. The actor logs in and navigates to the quiz section. 2. Students select their quiz to view personal results, while professors can view results for the whole class. 3. The system retrieves and displays the results. 4. Students can view their scores and feedback (if provided). 5. Professors can analyze class performance and download reports (if needed). 6. If results are not available, the system displays a message (e.g., "Results not published yet"). 7. If there is an error, the system provides an error message (e.g., "Unable to retrieve quiz results").

Name	Course Registration
Goal	Allow students to register for available courses.
Actors	Student
Preconditions	• Student must be logged in. • The course must be available for registration. • There should be no registration conflicts (e.g., schedule overlaps).
Postconditions	• The student is registered for the course. • The system updates the course's participant list. • The student gains access to course materials.
Error Situations	• The course is full or closed for registration. • Registration conflicts (e.g., schedule overlap). • System error during registration.
Standard Process	1. The student logs in and navigates the course catalog. 2. The student selects a course and clicks on the "Register" button. 3. The system checks for available seats and potential conflicts. 4. If all conditions are met, the student is enrolled. 5. A confirmation message is displayed, and course materials become accessible. 6. If an error occurs, an appropriate message is displayed(e.g.,"Course registration failed. Pleasetry again.").

Name	Start Quiz
Goal	Allow students to begin a quiz within a scheduled time
Actors	Student
Preconditions	• The student must be logged in. • The quiz must be available and scheduled. • The quiz may require the student to open the camera (if enabled).
Postconditions	• The quiz timer starts. • The student can proceed to answer the questions. • Any suspicious behavior is tracked during the quiz.
Error Situations	• Quiz not available or outside of scheduled time. • Camera or browser issues prevent quiz initiation. • System error during the quiz launch.
Standard Process	1. The student logs in and selects the quiz from the course dashboard. 2. If required, the system prompts the student to enable the camera. 3. The system verifies quiz availability and starts with the timer. 4. The student begins answering questions. 5. If an issue arises (e.g., browser crashes), the student can rejoin within the allowed time.

Name	Submit Answers
Goal	Enable students to submit their answers for evaluation at the end of the quiz.
Actors	Student
Preconditions	• The student must have started the quiz • The quiz timer must be running (or must have ended). • All required questions must be answered.
Postconditions	• The answers are saved and submitted for evaluation. • The system confirms the submission. • The professor can access the student'ssubmission.
Error Situations	• Internet or system error during submission. • Incomplete answers (if required). • Quiz time expired before submission.
Standard Process	1. The student completes the quiz and clicks the "Submit" button. 2. The system checks for unanswered mandatory questions. 3. If all the conditions are met, the answers are saved and submitted. 4. A confirmation message is displayed. 5. In case of submission failure, the system retrieves or shows an error message (e.g., "Submission failed. Please try again.").

Name	Sign-up
Goal	Allow new students to create an account in the system.
Actors	Student
Preconditions	• The user must not have an existing account in the system. • All required registration details (e.g., email, password, name) must be provided.
Postconditions	• A new user account is created, and data is sent to database. • Face embeddings saved in database • Password does not meet security requirements. • Network or server issues prevent registration.
Error Situations	• Email or username already exists. • System downtime or connectivity issues.

Name	Send Notification
Goal	Notify students, professors, or admins about important events (e.g., quiz reminders, cheating alerts).
Actors	Notification System
Preconditions	• The event triggering the notification must occur (e.g., quiz reminder time, cheating detection). • The recipient must be registered in the system.
Postconditions	• The recipient receives the notification. • The system logs the notification delivery status.
Error Situations	• Notification delivery failure (e.g., network issues). • Incorrect or missing recipient information. • Duplicate notifications due to system error.
Standard Process	1. An event triggers the notification system (e.g., quiz start time approaches). 2. The system composes the notification with relevant details. 3. The notification is sent to the intended recipient via email, SMS, or app alert. 4. The system logs the delivery status (e.g., success, failure). 5. If delivery fails, the system retrieves or alerts the admin.

Name	Detect Cheating
Goal	Use AI/ML models to detect suspicious behavior during a quiz to prevent cheating.
Actors	AI Model
Preconditions	• The quiz session must be active. • The student's camera must be enabled (if required). • The AI/ML model must be operational and integrated with the system.
Postconditions	• Suspicious behavior is detected and logged. • The professor is notified of detected cheating. • The student's quiz session might be flagged or interrupted.
Error Situations	• AI model failure or system error during behavior analysis. • False positives or negatives in behavior detection. • Camera malfunction during the quiz.
Standard Process	1. The quiz begins, and the AI model starts analyzing the student's behavior. 2. The model monitors camera feed (if applicable) for signs of cheating (e.g., multiple faces, eye movement). 3. If suspicious behavior is detected, the session is flagged. 4. The professor receives a notification with a report. 5. If a system or model error occurs, an appropriate message is logged.

5.3. Class Diagram

1) V1

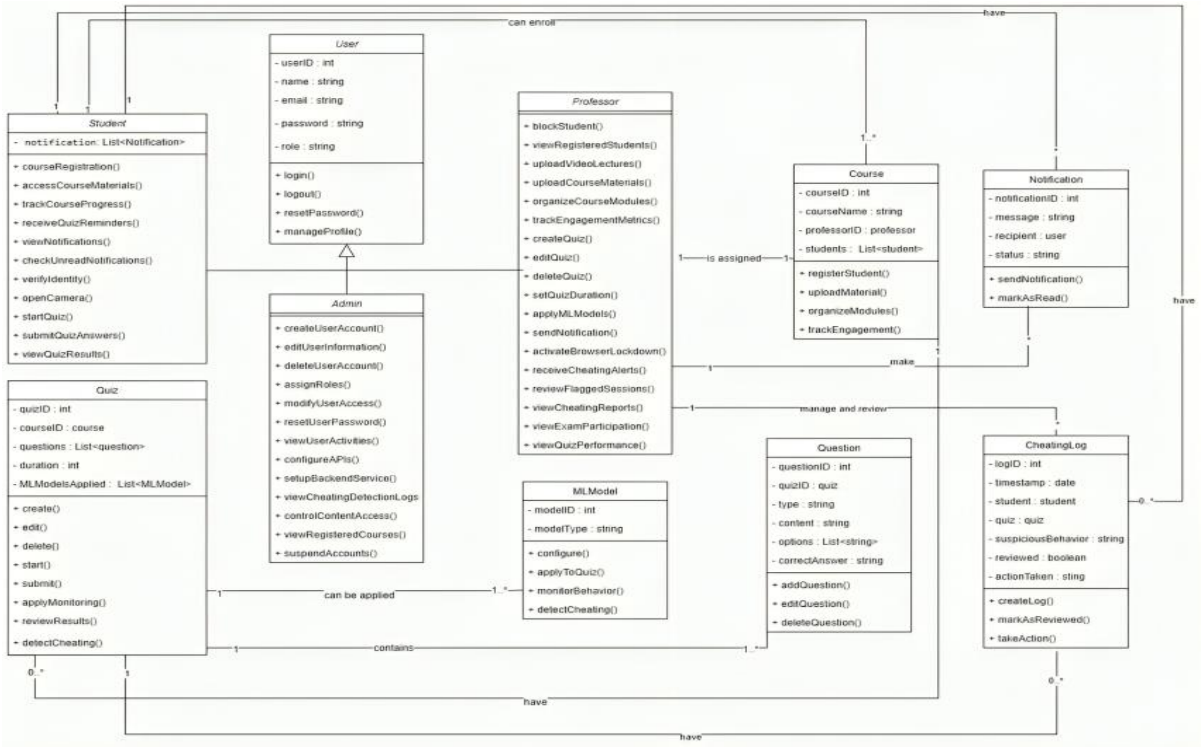


Figure 46: Class Diagram V1

2) V2

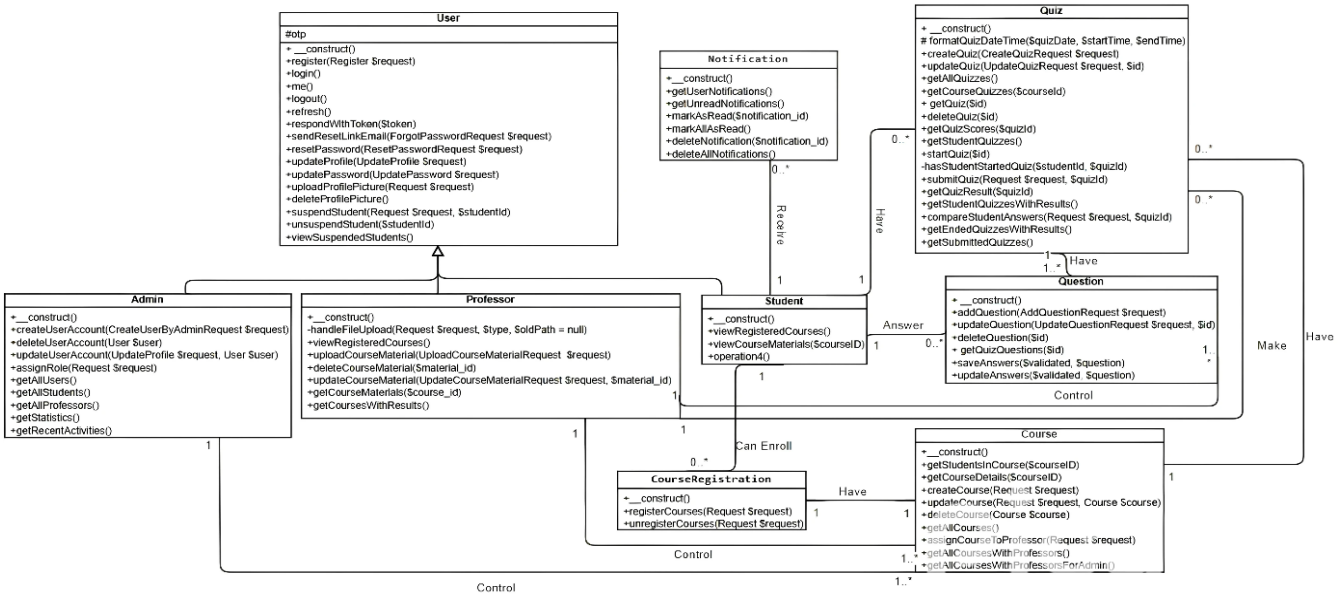


Figure 47: Class Diagram V2

3) V3

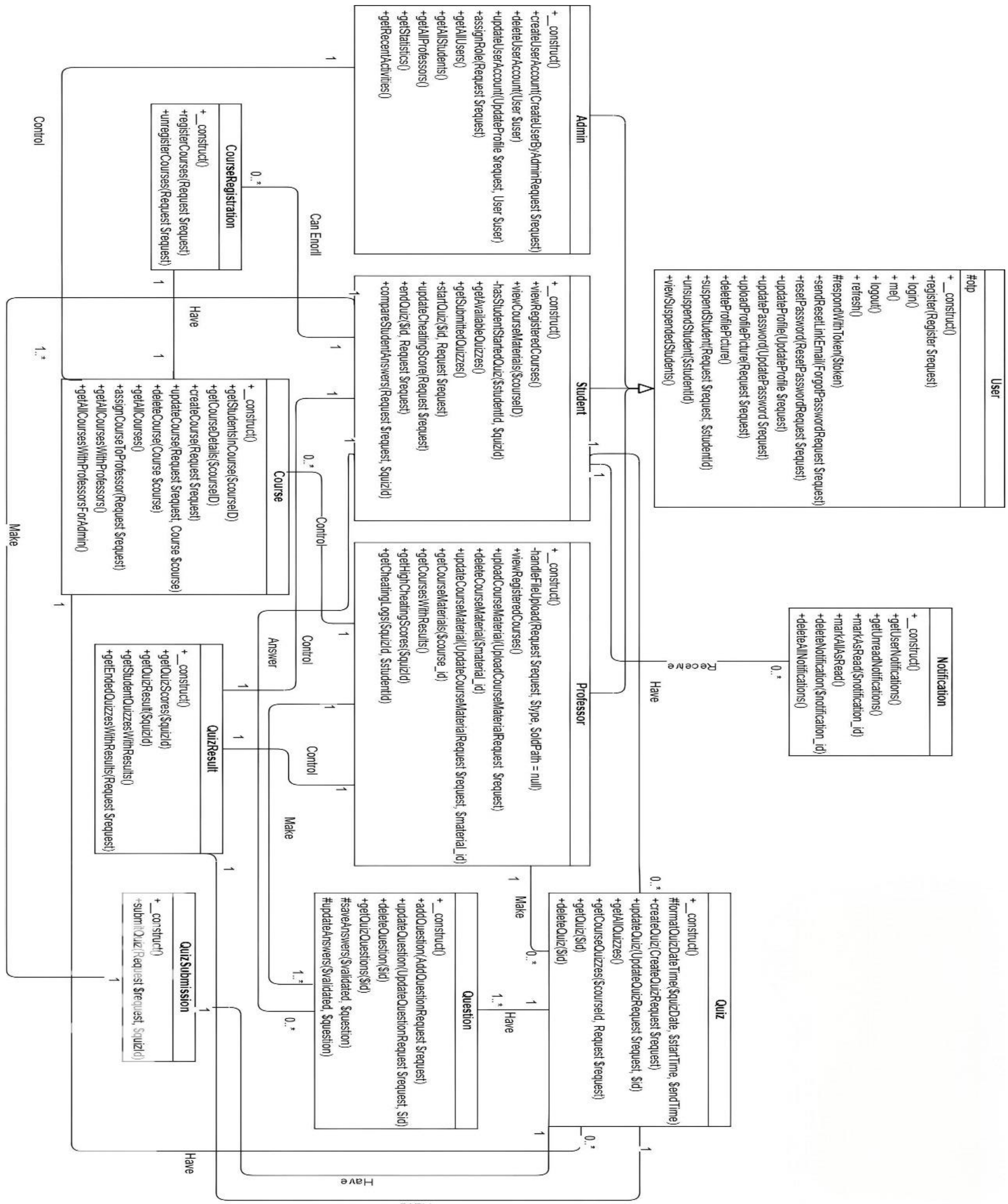


Figure 48: Class Diagram V3

4) Class Diagram for ML Service

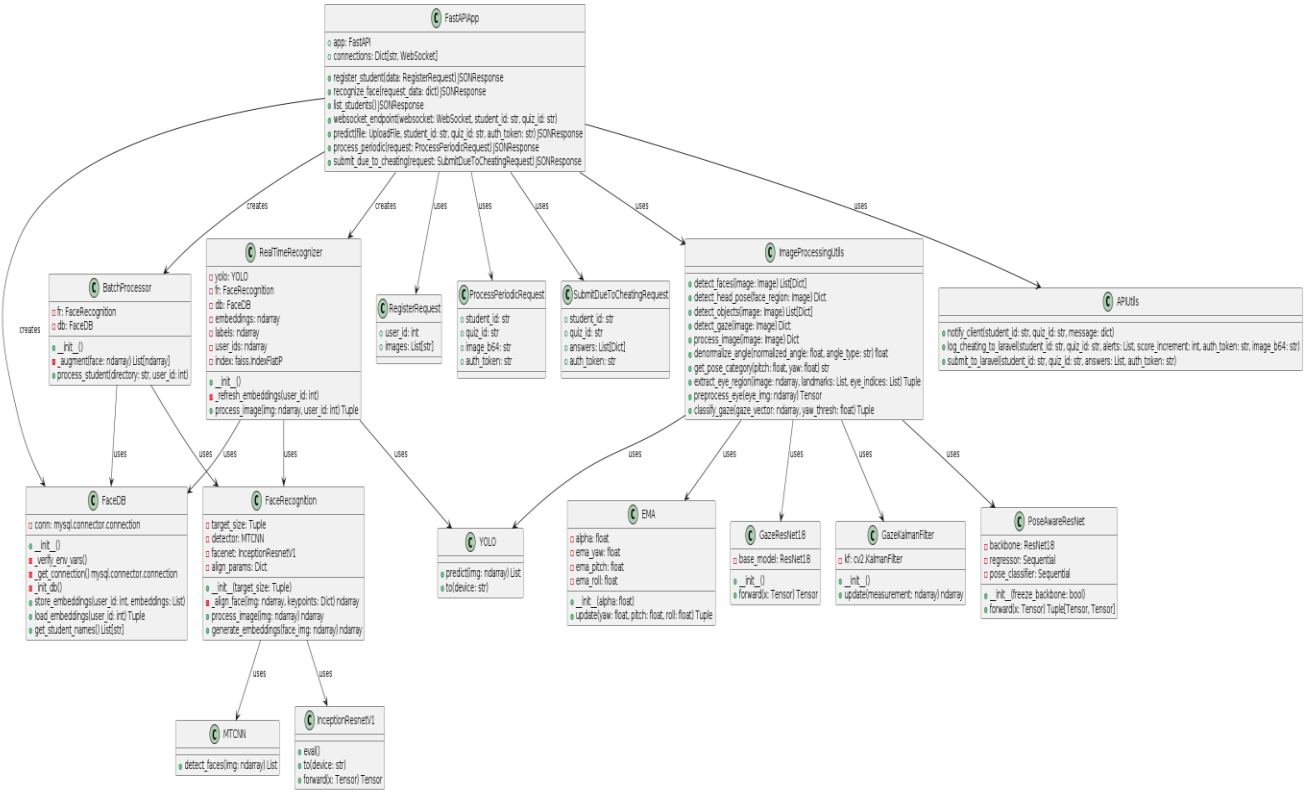


Figure 49: Class Diagram for ML Service

5.4.ERD

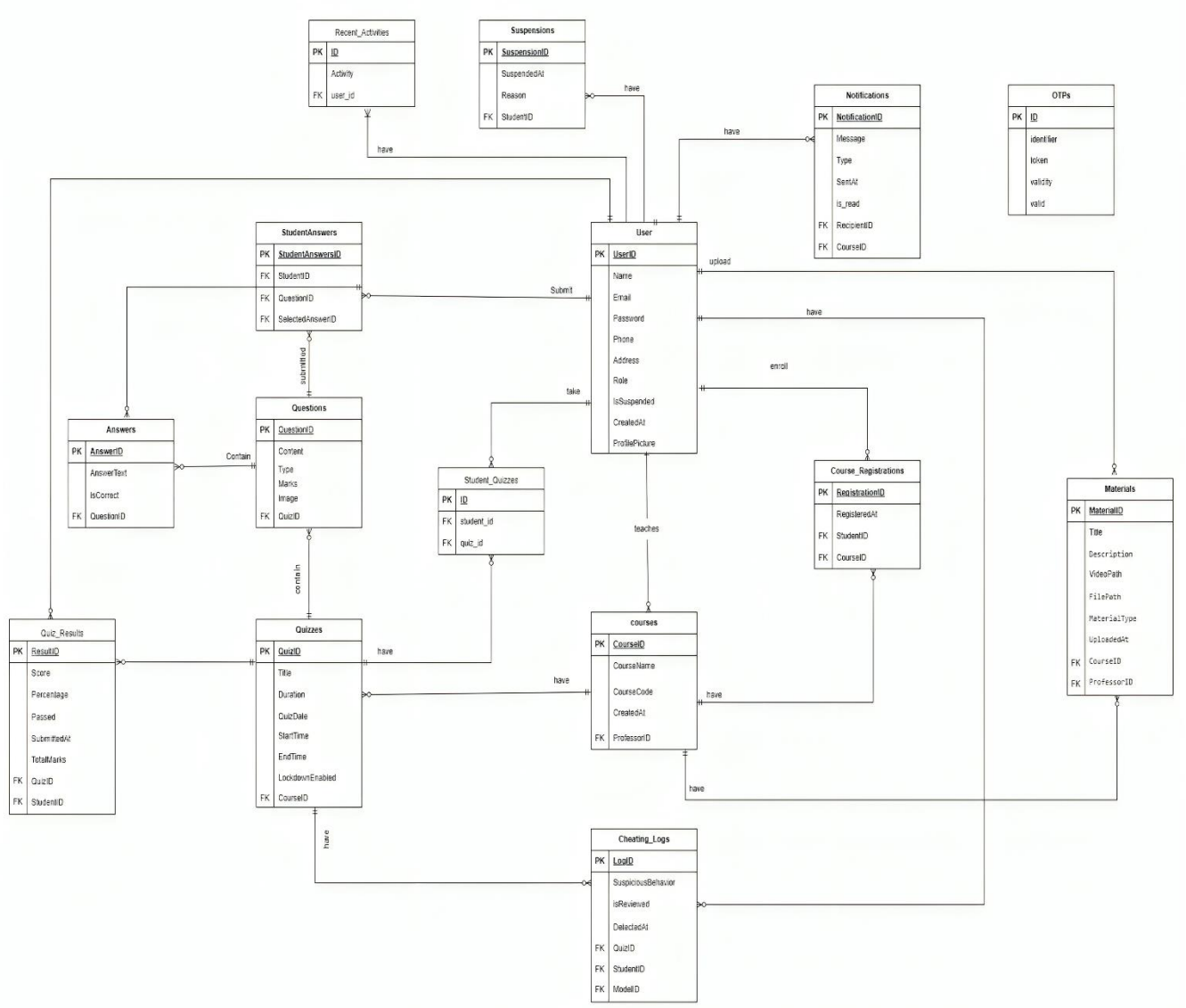


Figure 50: ERD Diagram

5.5.Object Diagram

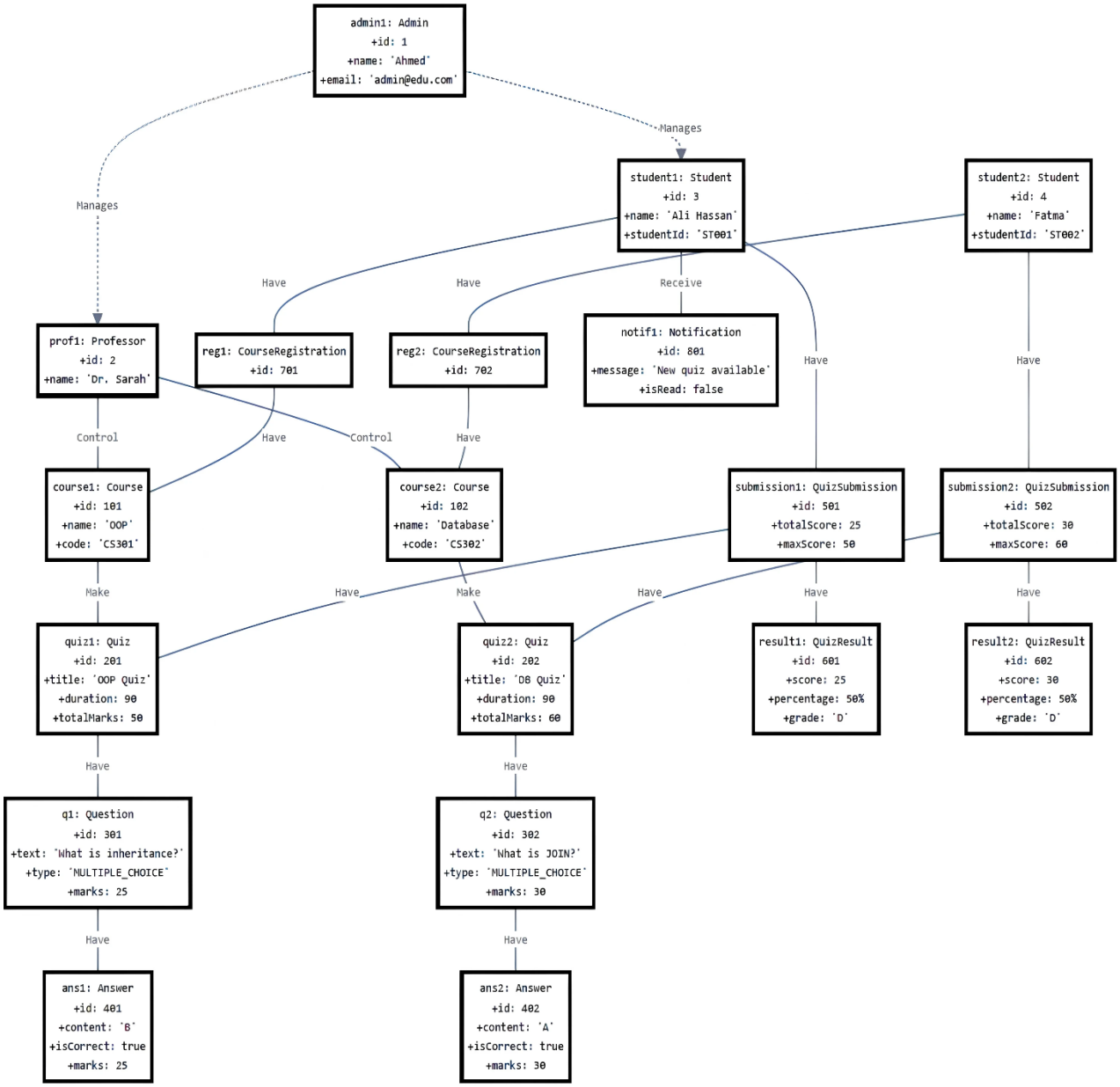


Figure 51: Object Diagram

5.6.Package Diagram

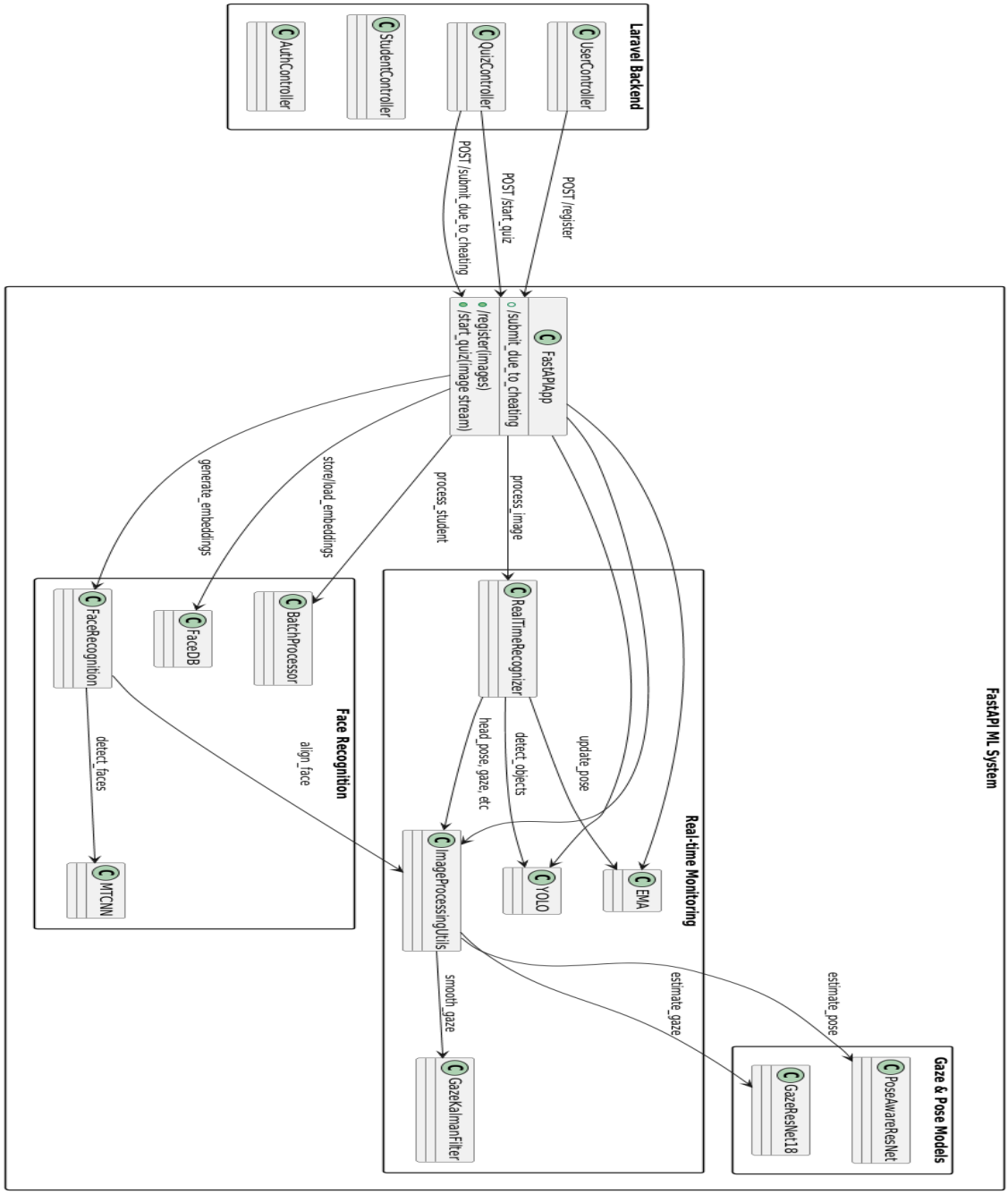
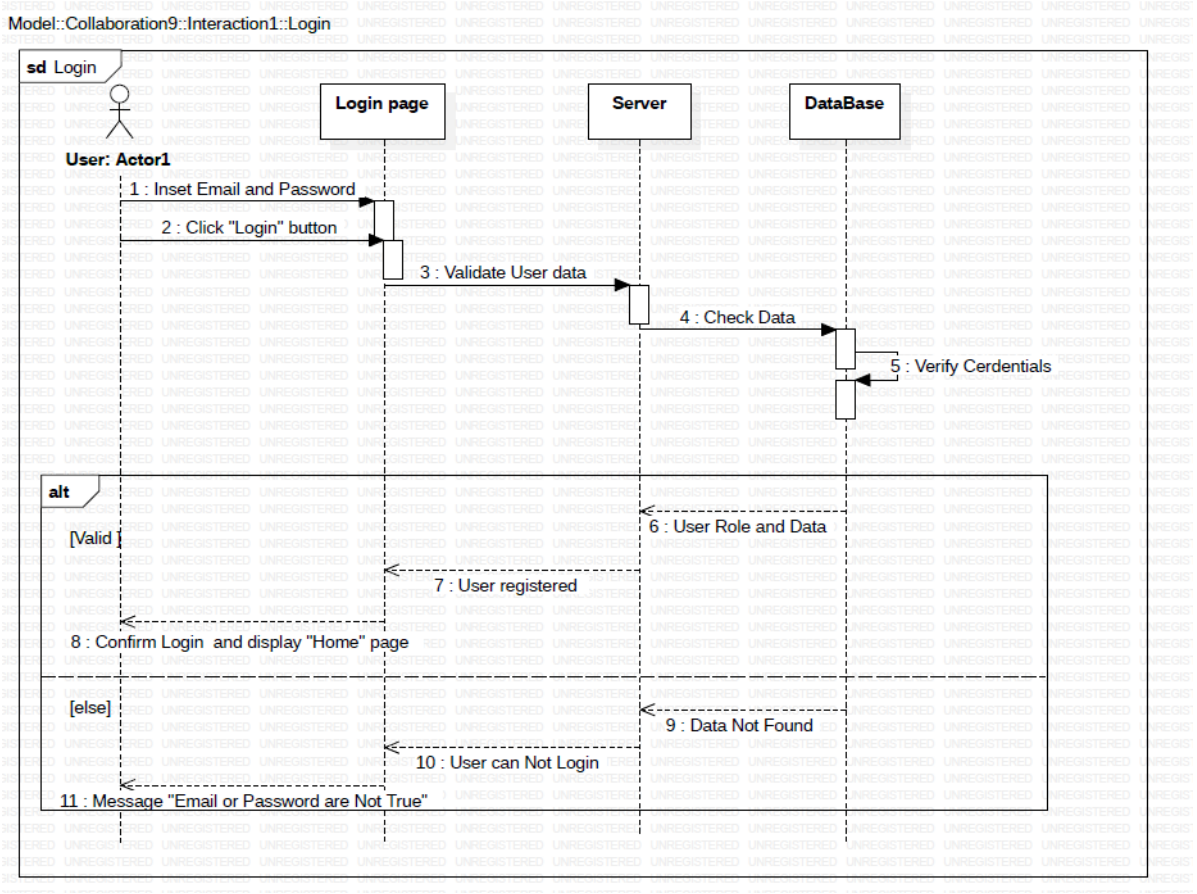


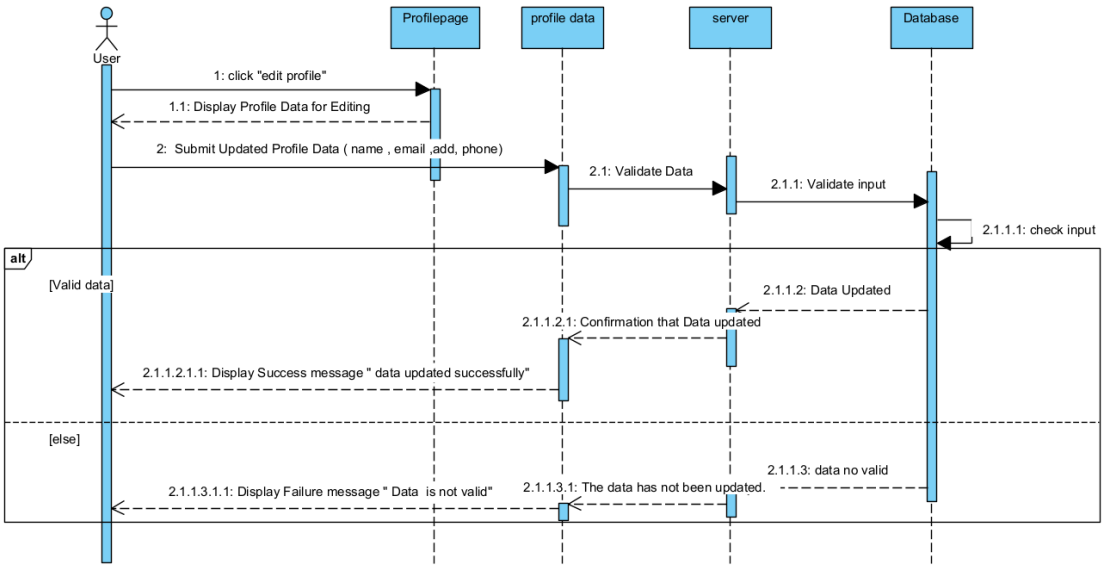
Figure 52: Package Diagram

5.7.Sequence Diagram

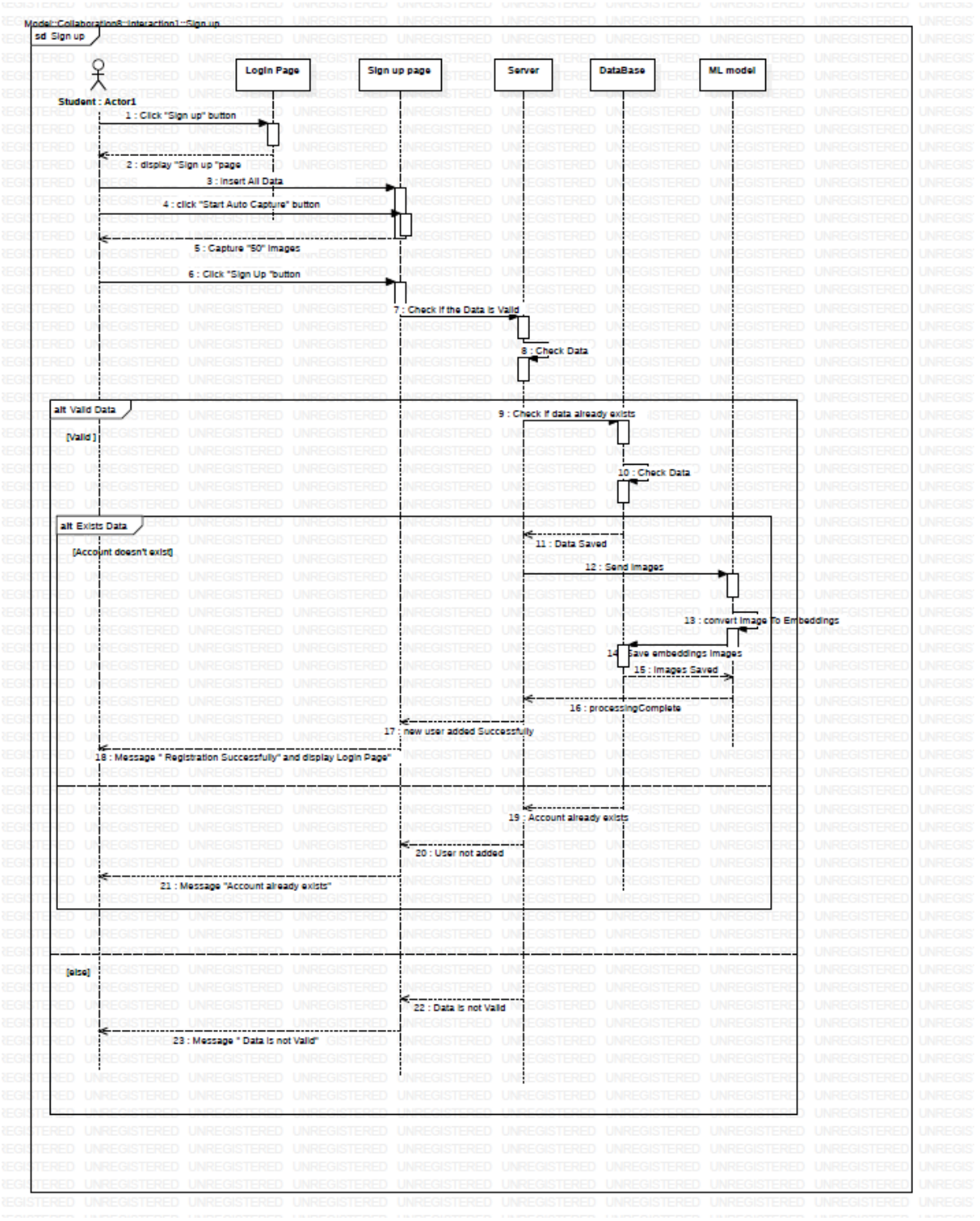
1) Login



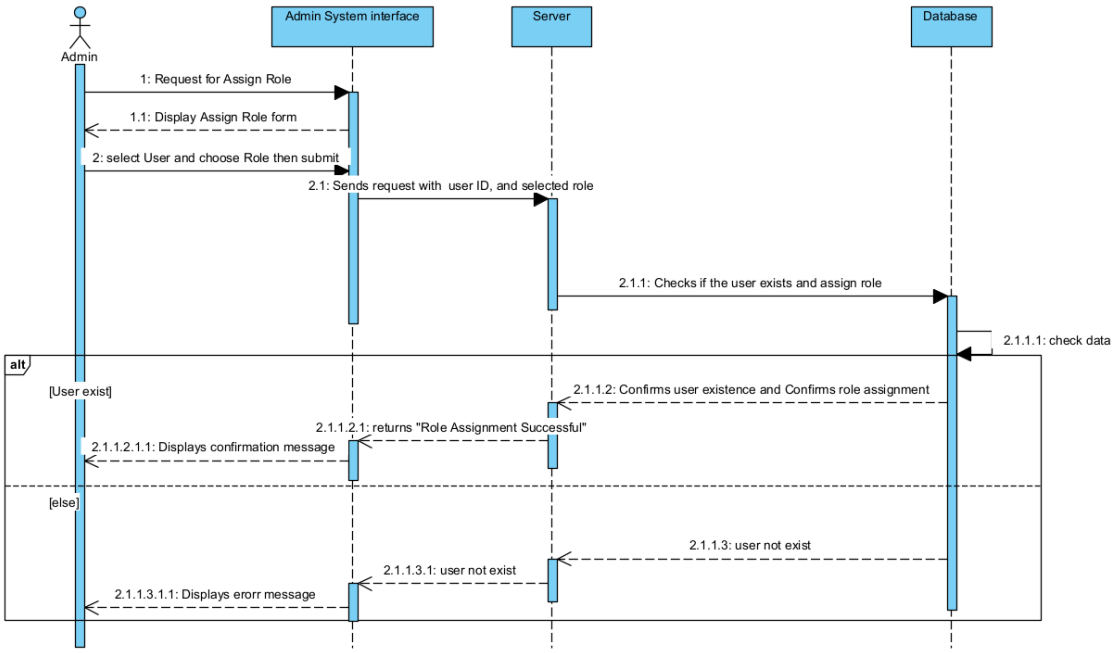
2) Edit Profile



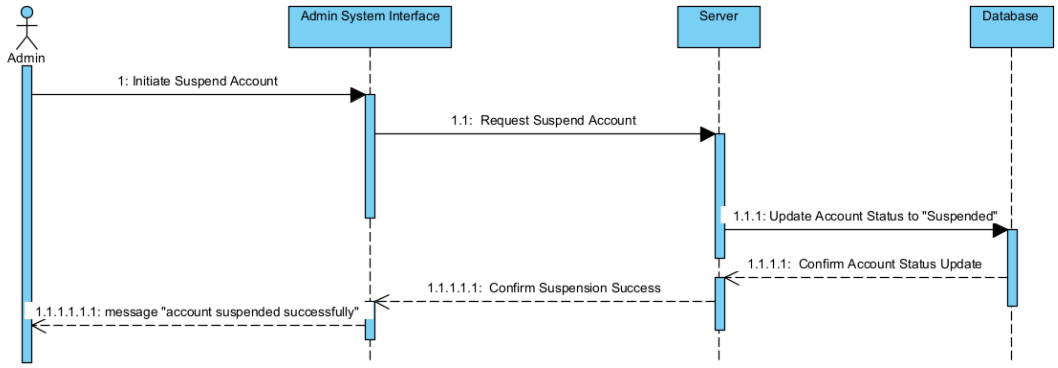
3) Sign Up


Model::Collaboration8::Interaction1::Sign up
sd Sign up
Login Page
Sign up page
Server
DataBase
ML model
Student : Actor1
1 : Click "Sign up" button
2 : display "Sign up "page
3 : Insert All Data
4 : click "Start Auto Capture" button
5 : Capture "50" Images
6 : Click "Sign Up "button
7 : Check if the Data is Valid
8 : Check Data
alt Valid Data
[Valid]
9 : Check if data already exists
10 : Check Data
alt Exists Data
[Account doesn't exist]
11 : Data Saved
12 : Send Images
13 : convert Image To Embeddings
14 : Save embeddings Images
15 : Images Saved
16 : processingComplete
17 : new user added Successfully
18 : Message " Registration Successfully" and display Login Page"
19 : Account already exists
20 : User not added
21 : Message "Account already exists"
[else]
22 : Data is not Valid
23 : Message " Data is not Valid"

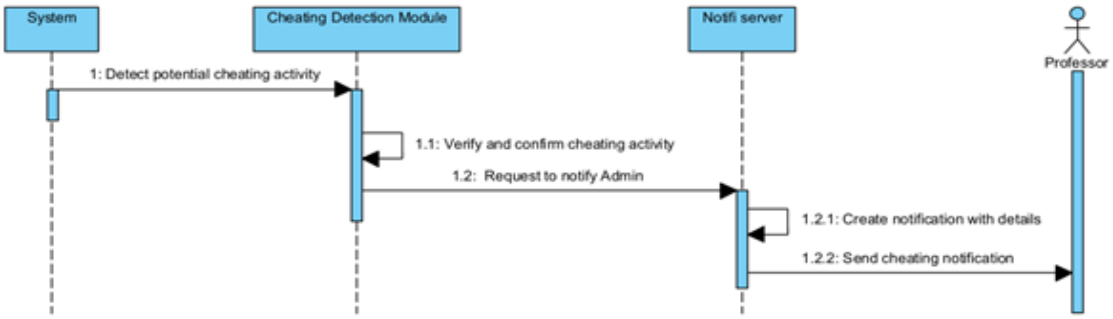
4) Assign Role



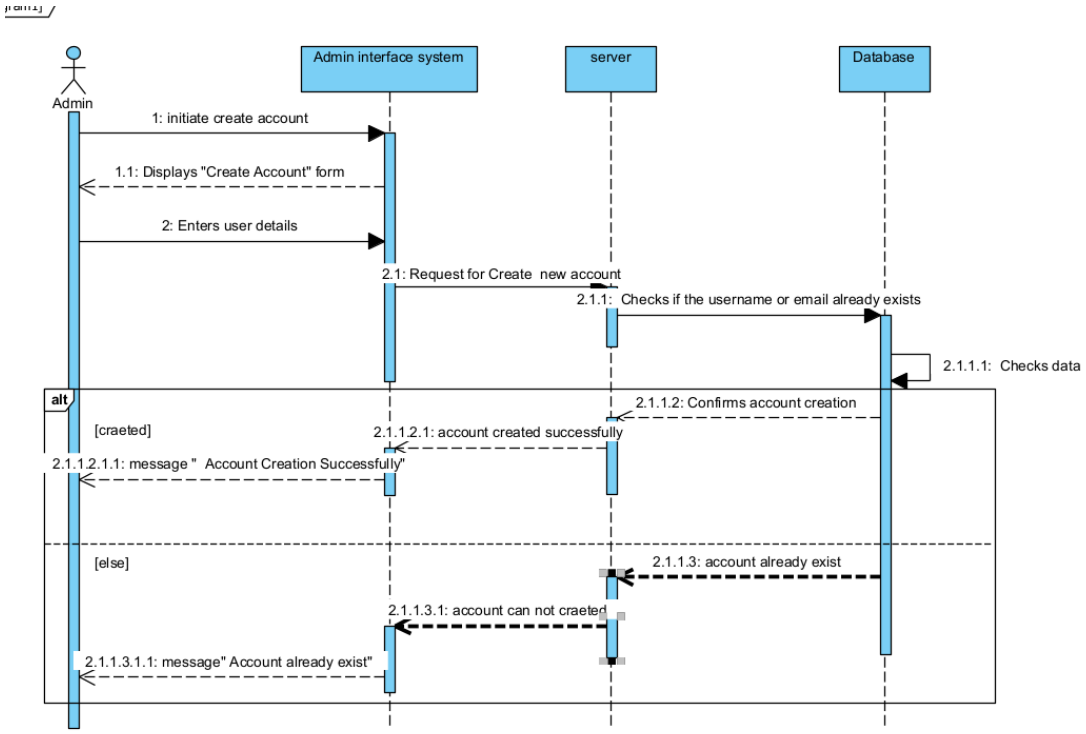
5) Suspended Account



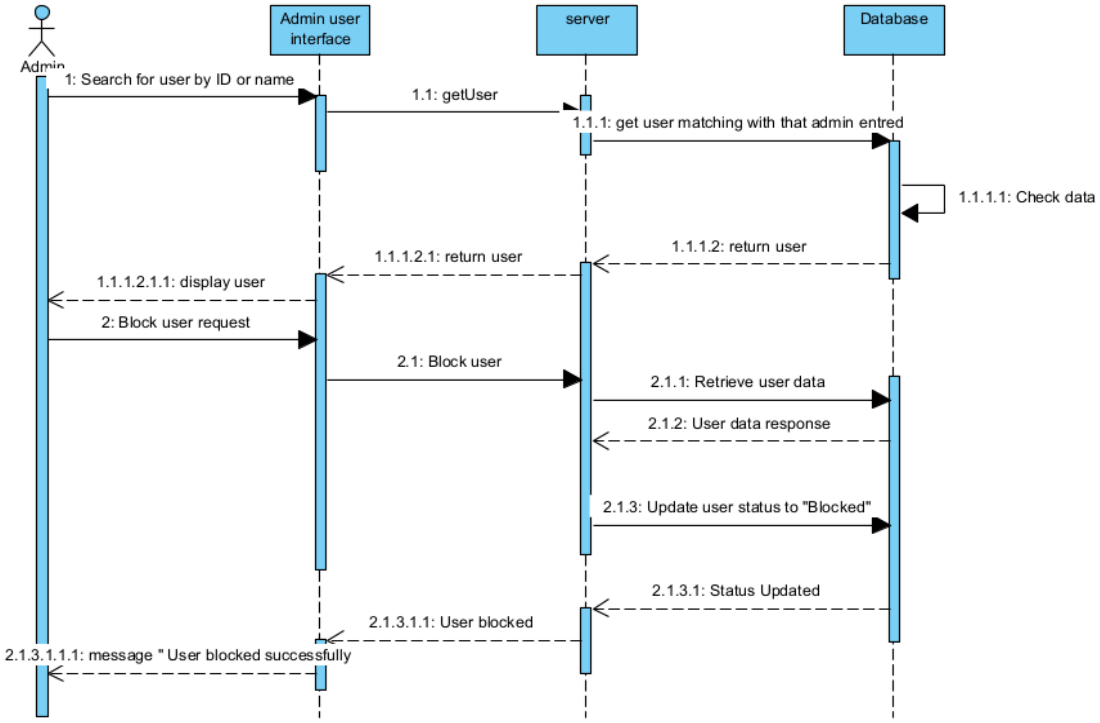
6) Receive cheating Notification



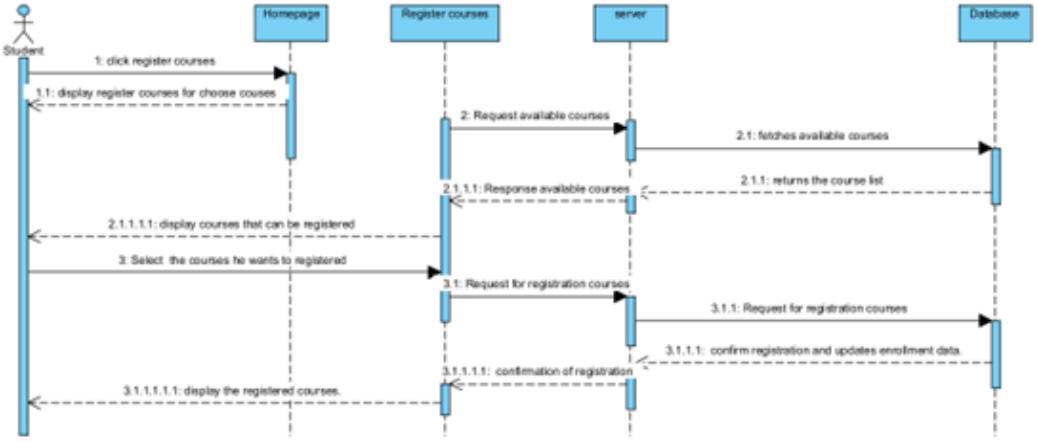
7) Create Account



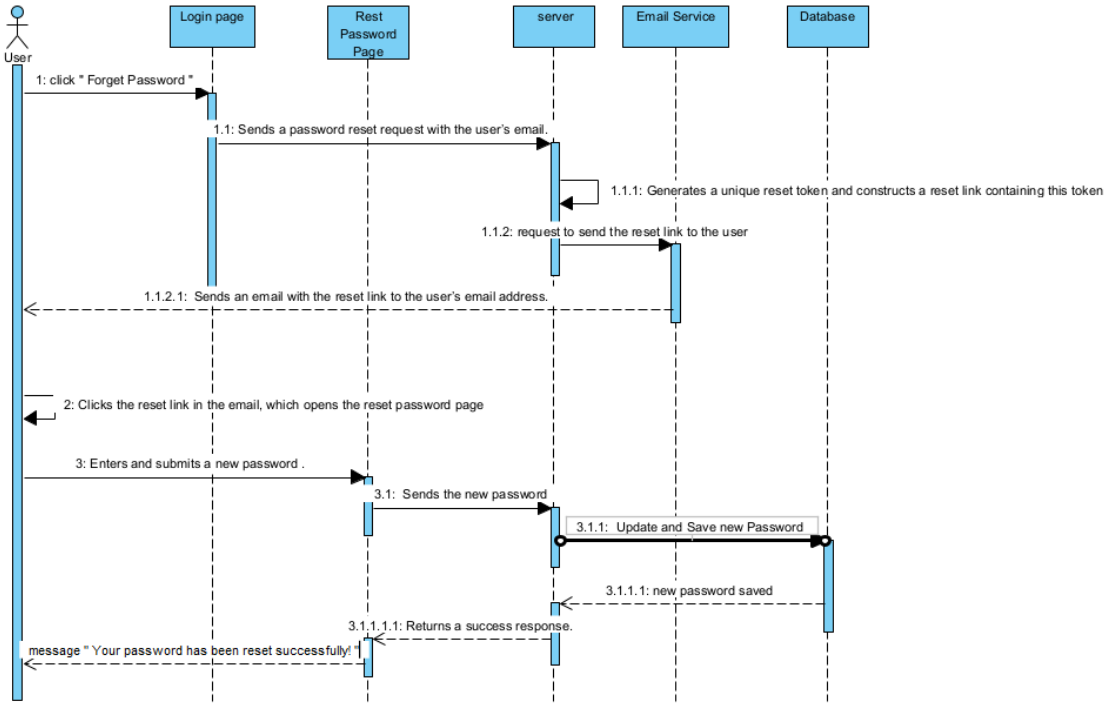
8) Blok Account



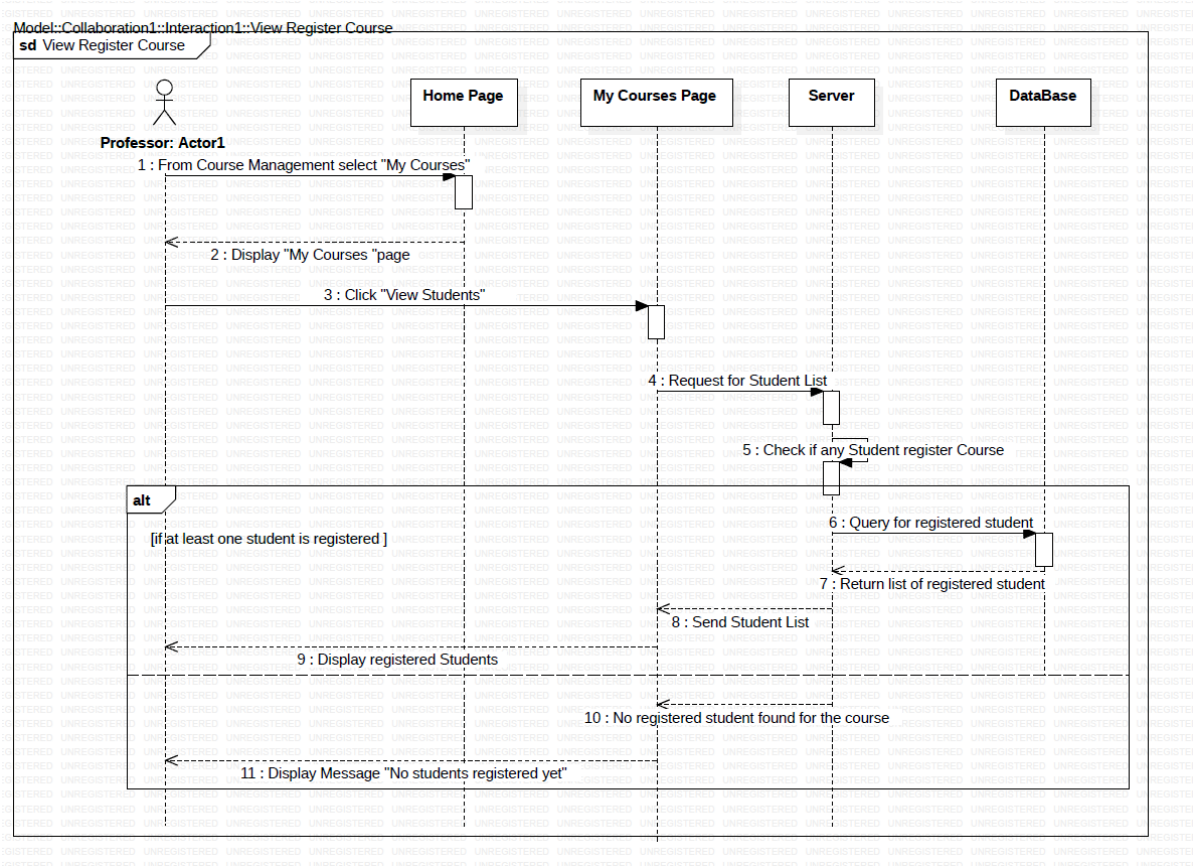
9) Course Registration



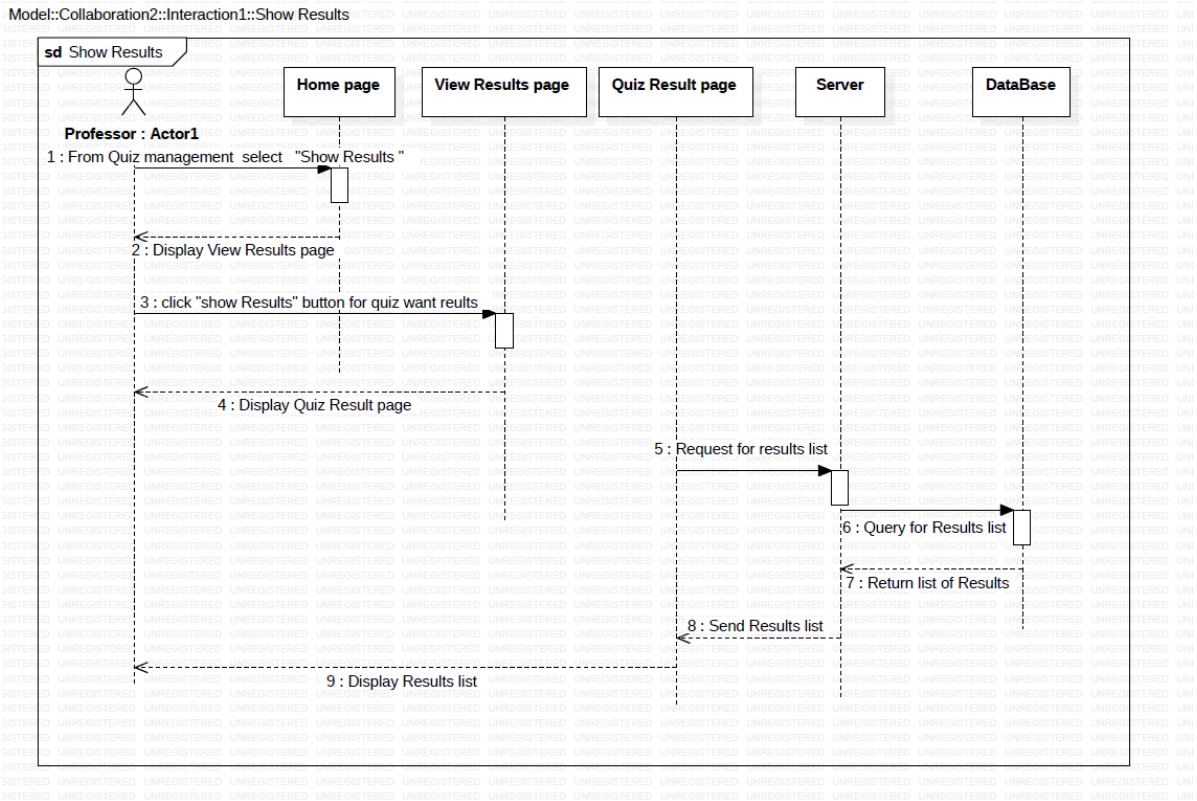
10) Rest Password



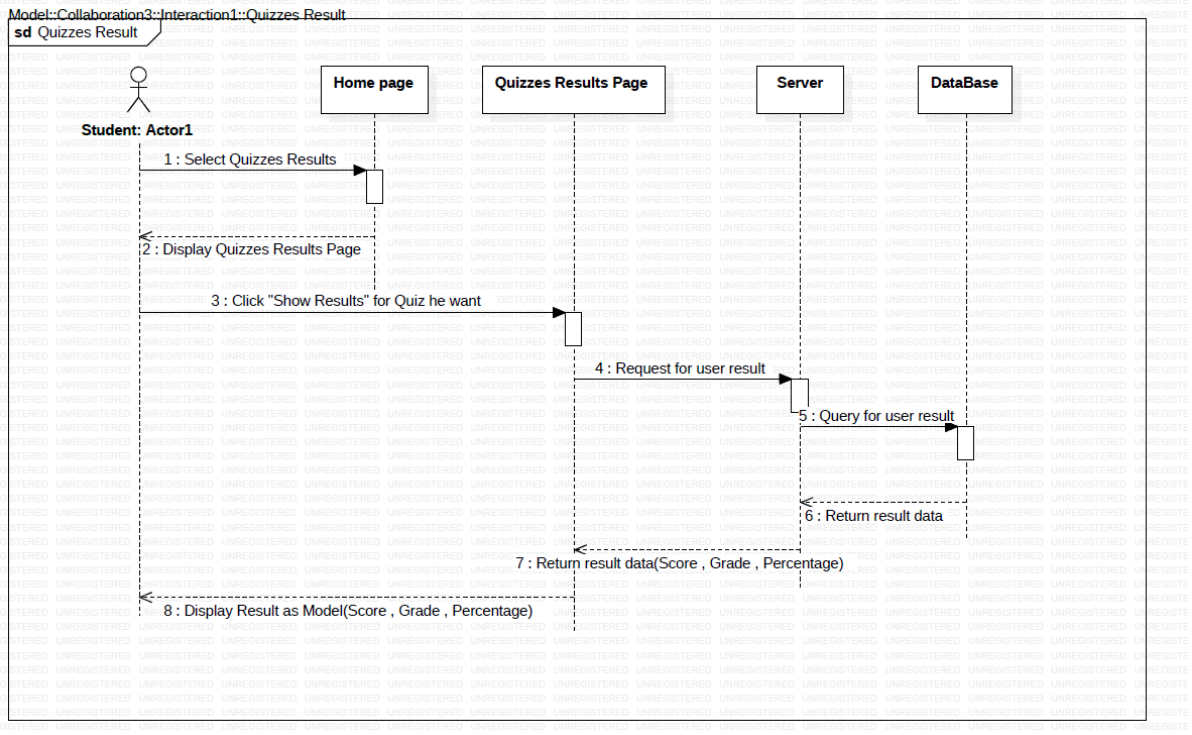
11) View Register Course



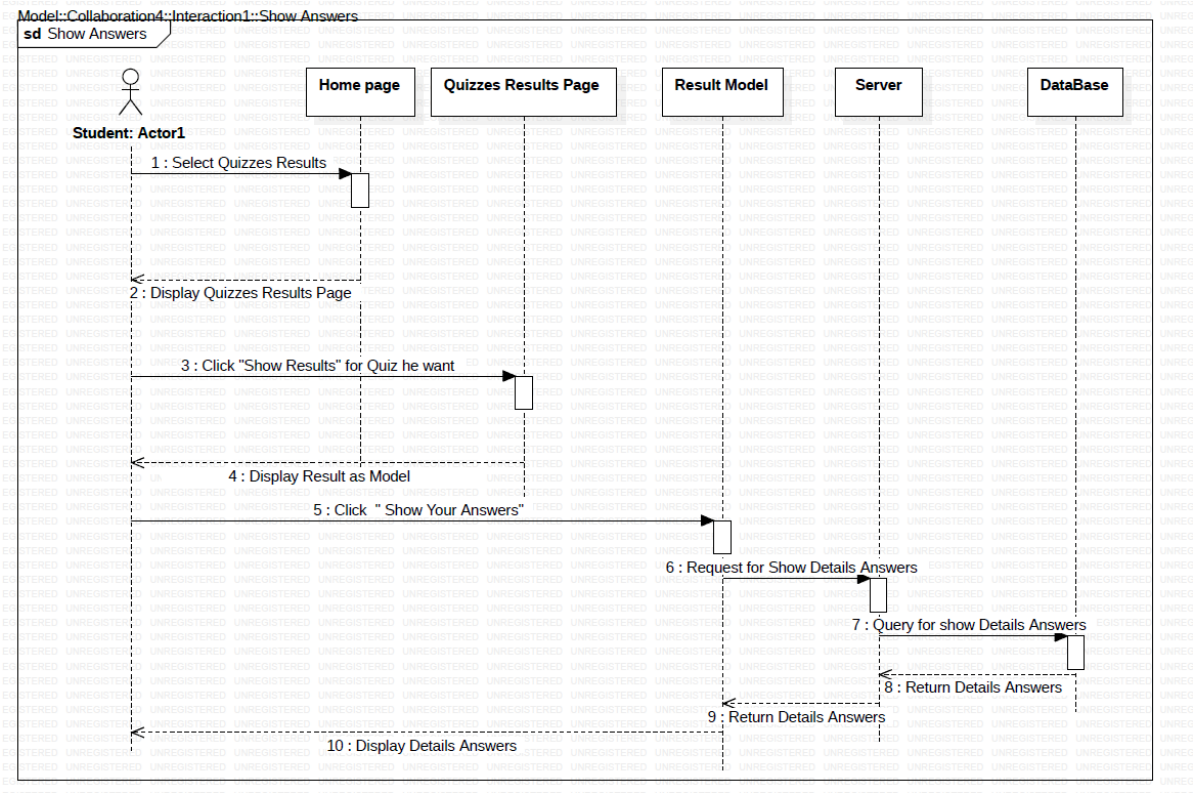
12) Show Results



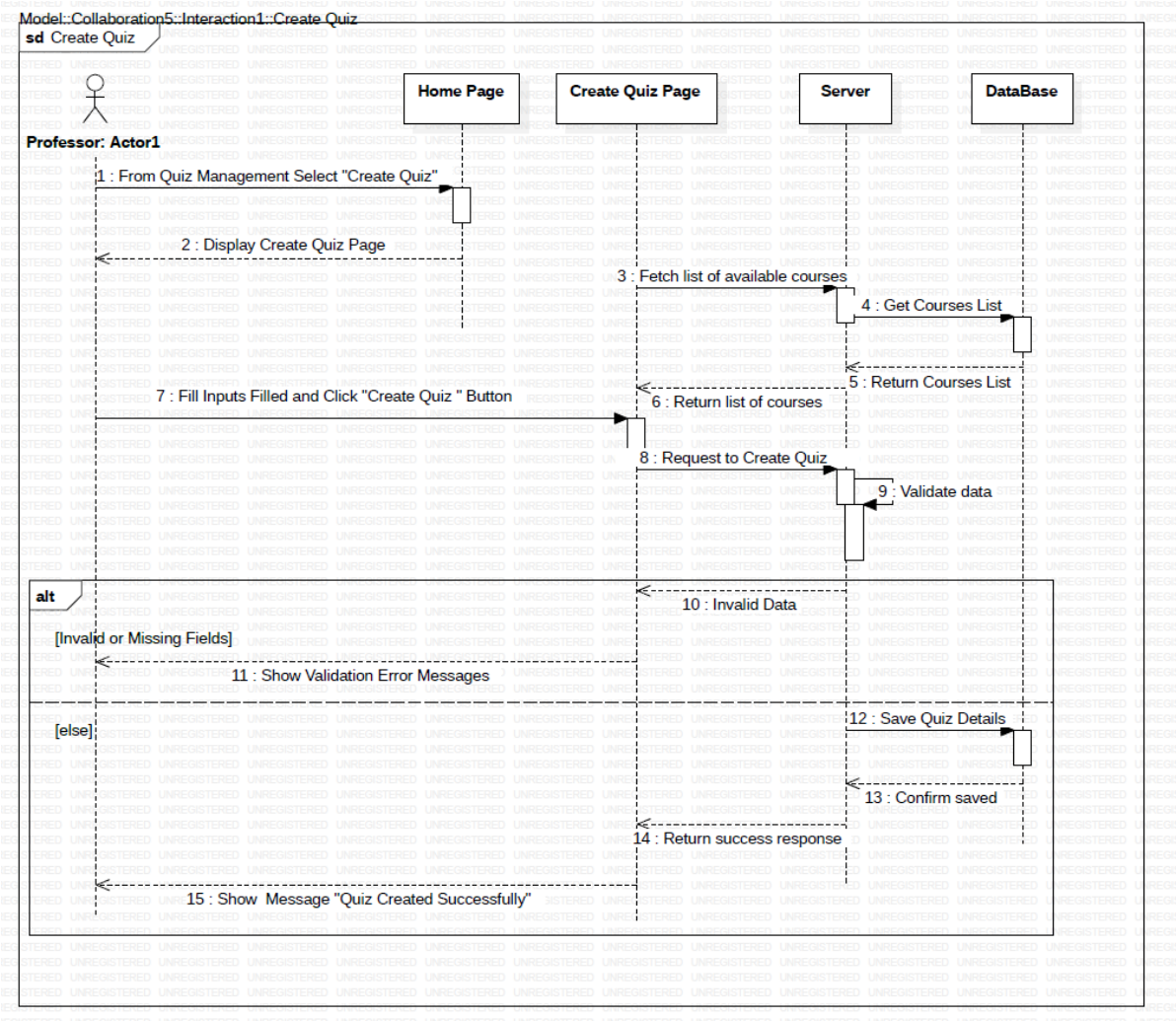
13) Quizzes Result



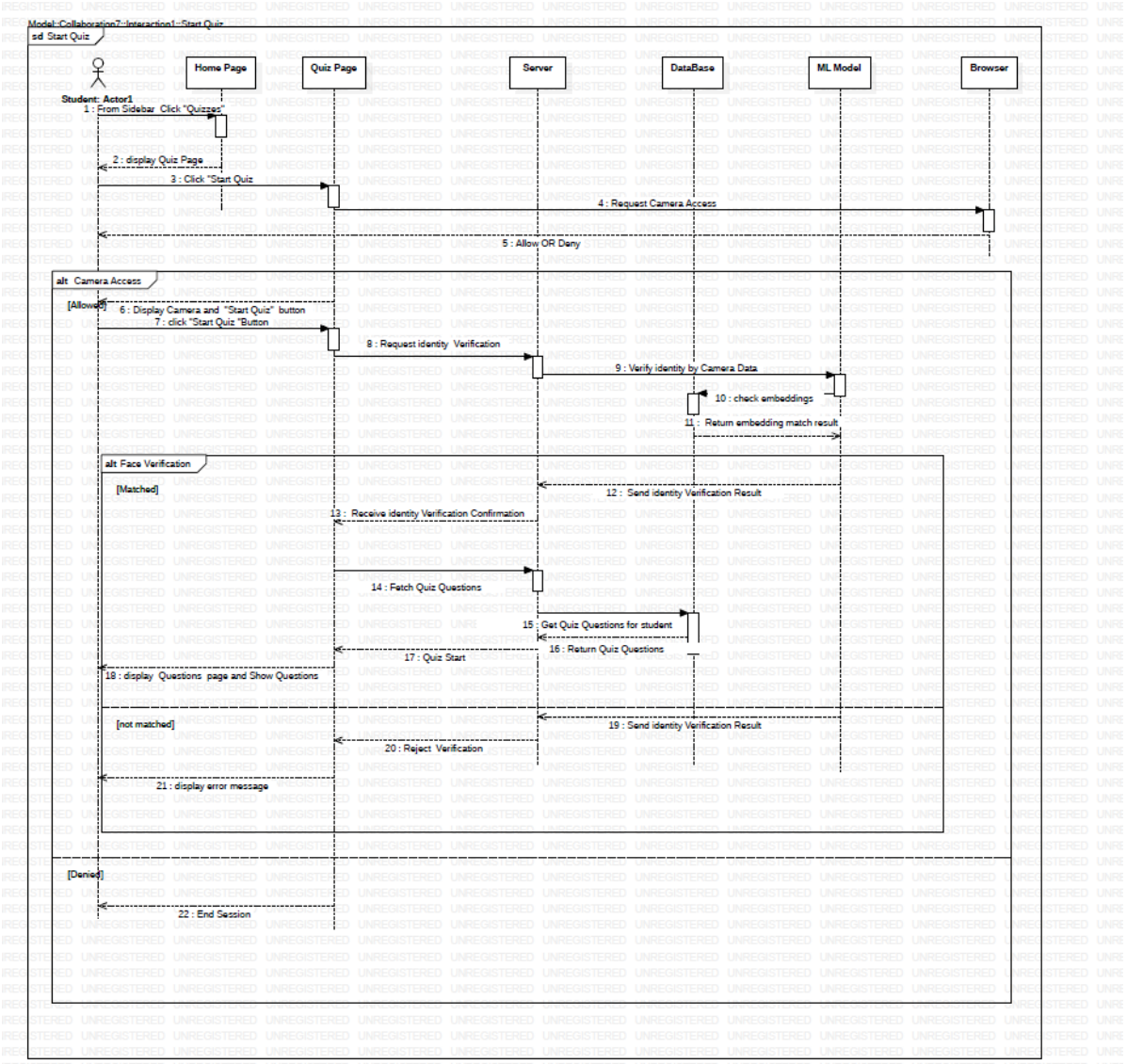
14) Show Answers



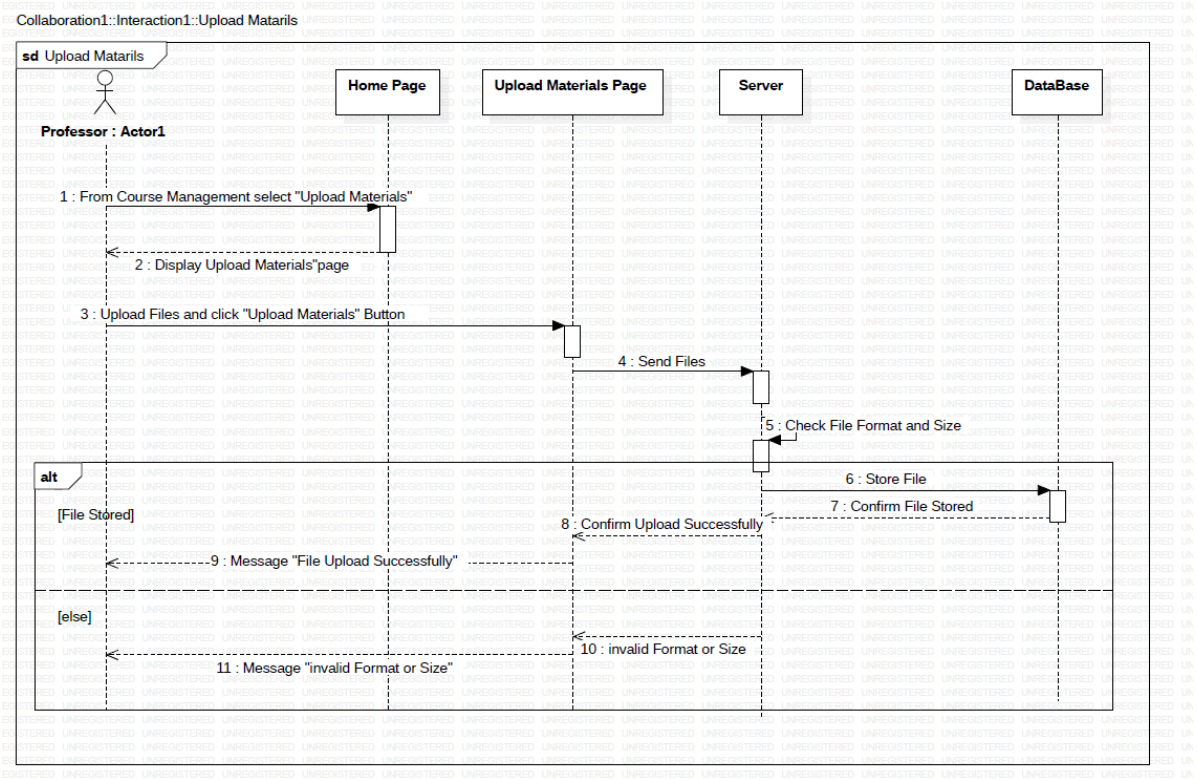
15) Create Quiz



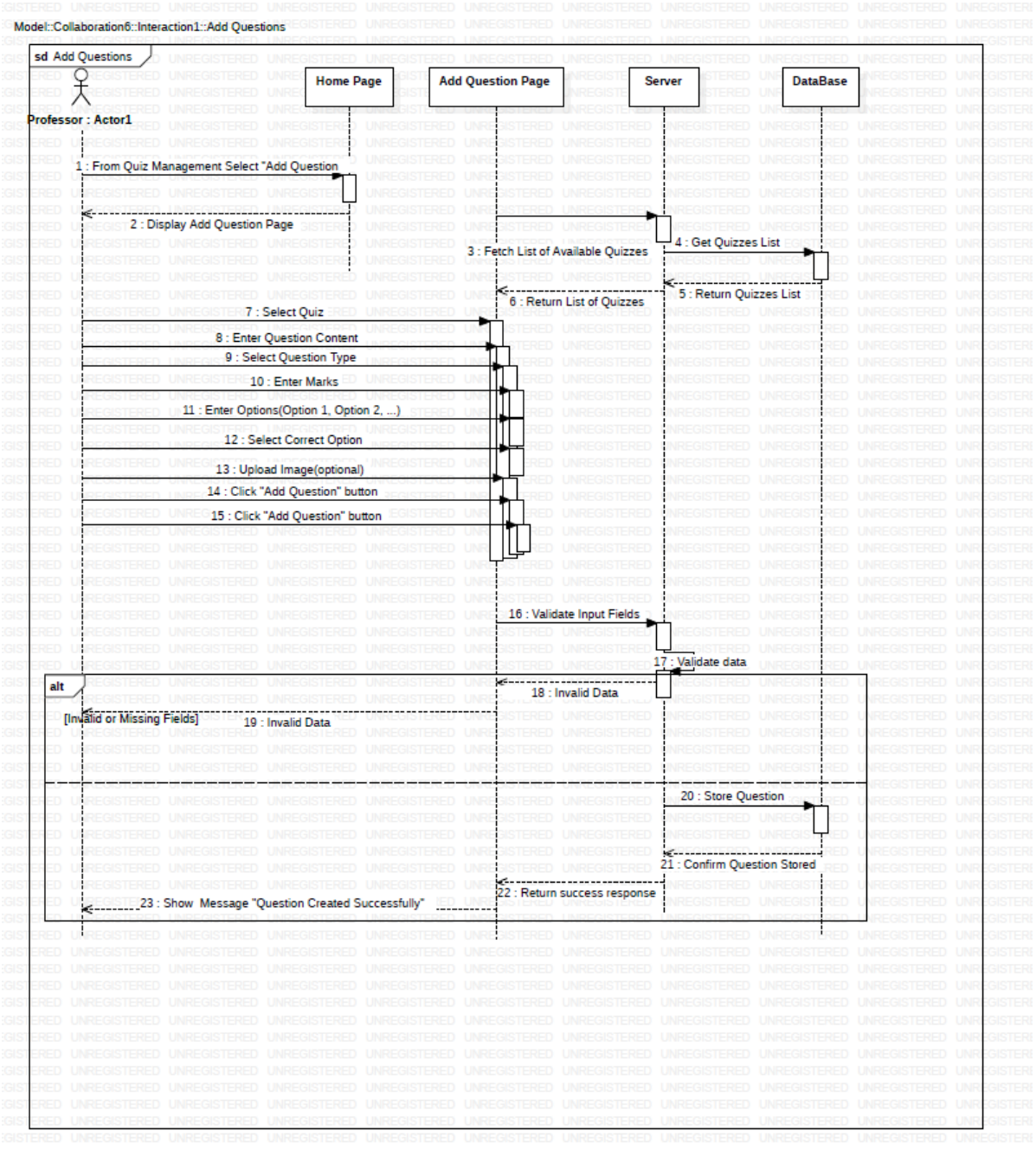
16) Start Quiz



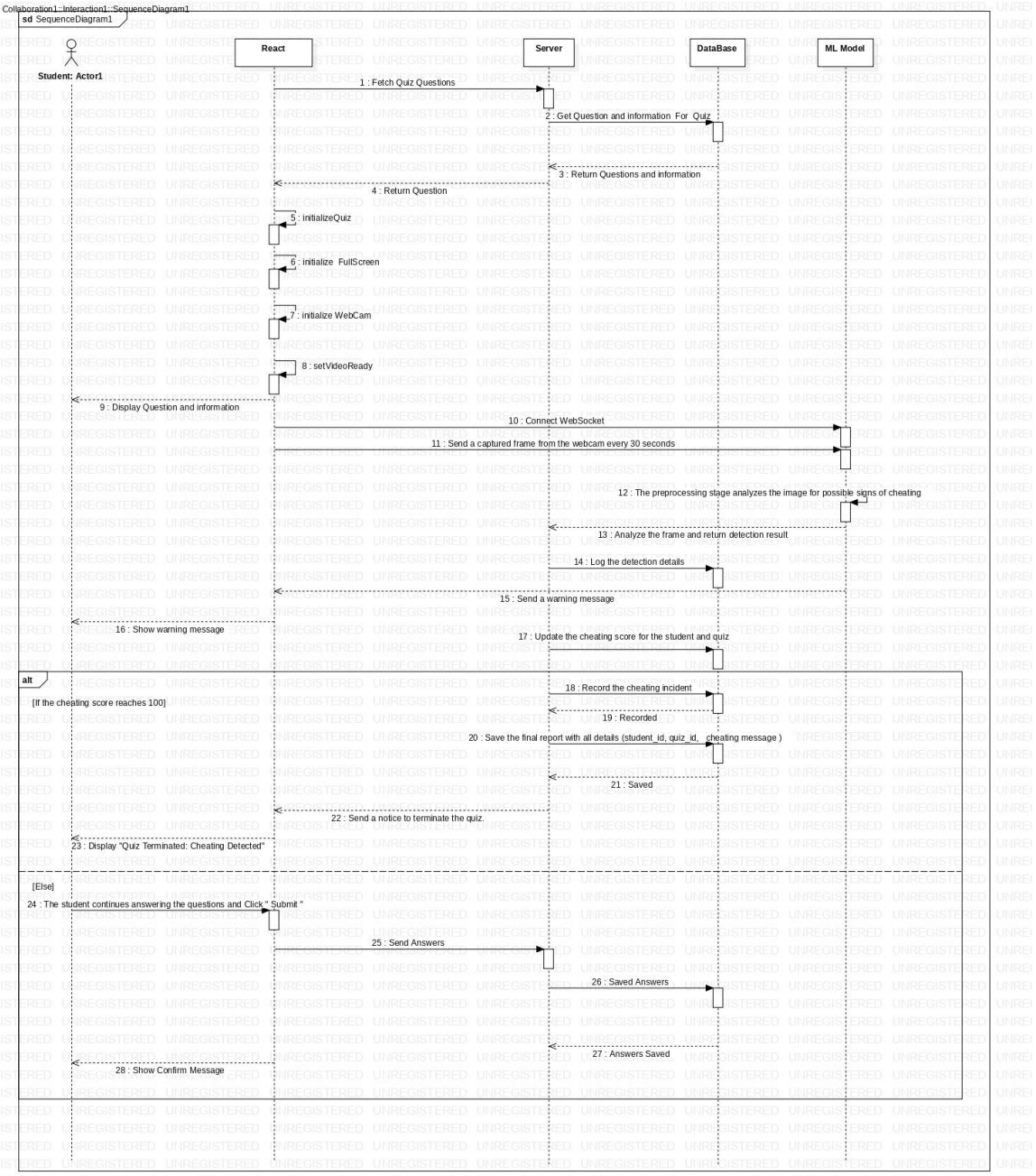
17) Upload Materials



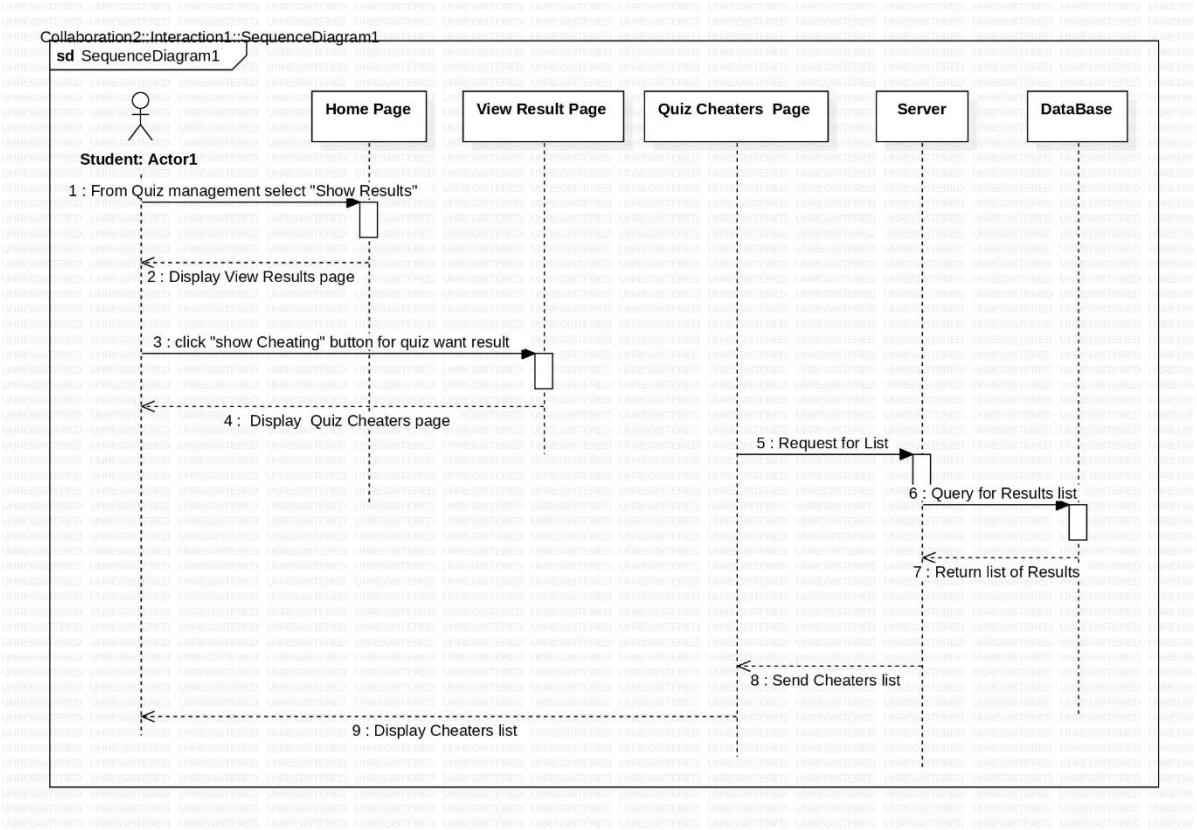
18) Add Question



19) During The Quiz

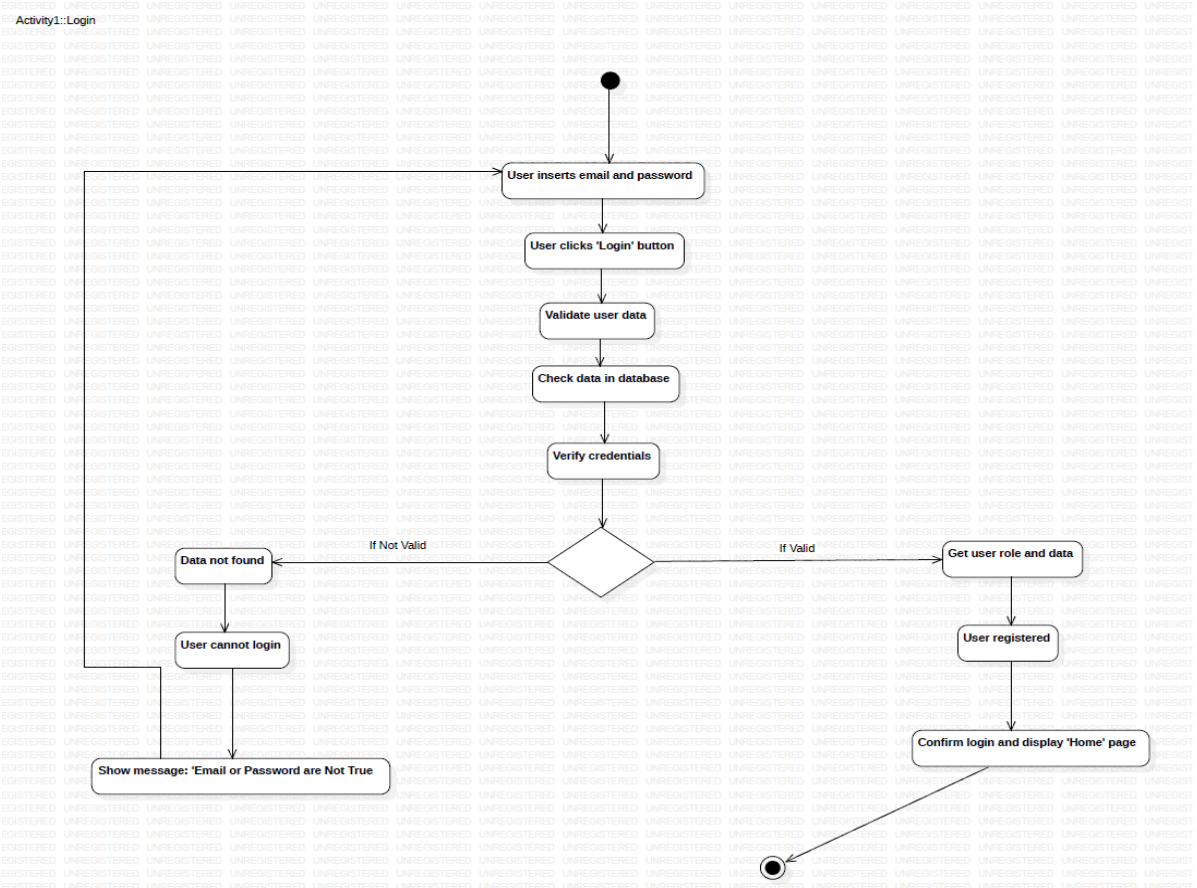


20) Show Results for Cheaters

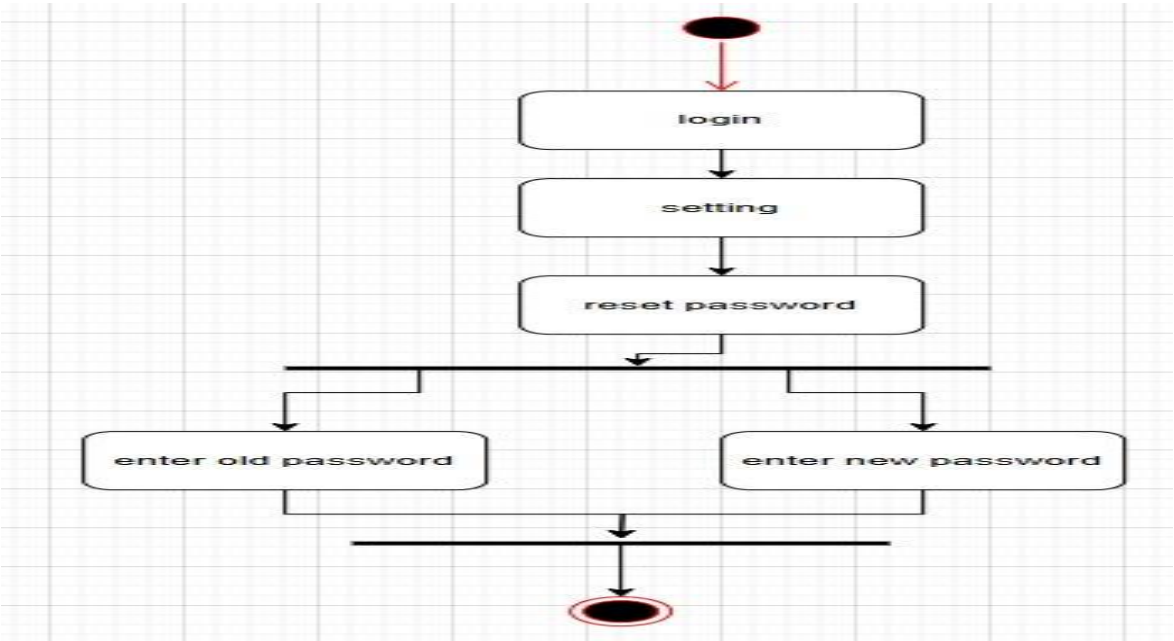


5.8. Activity Diagram

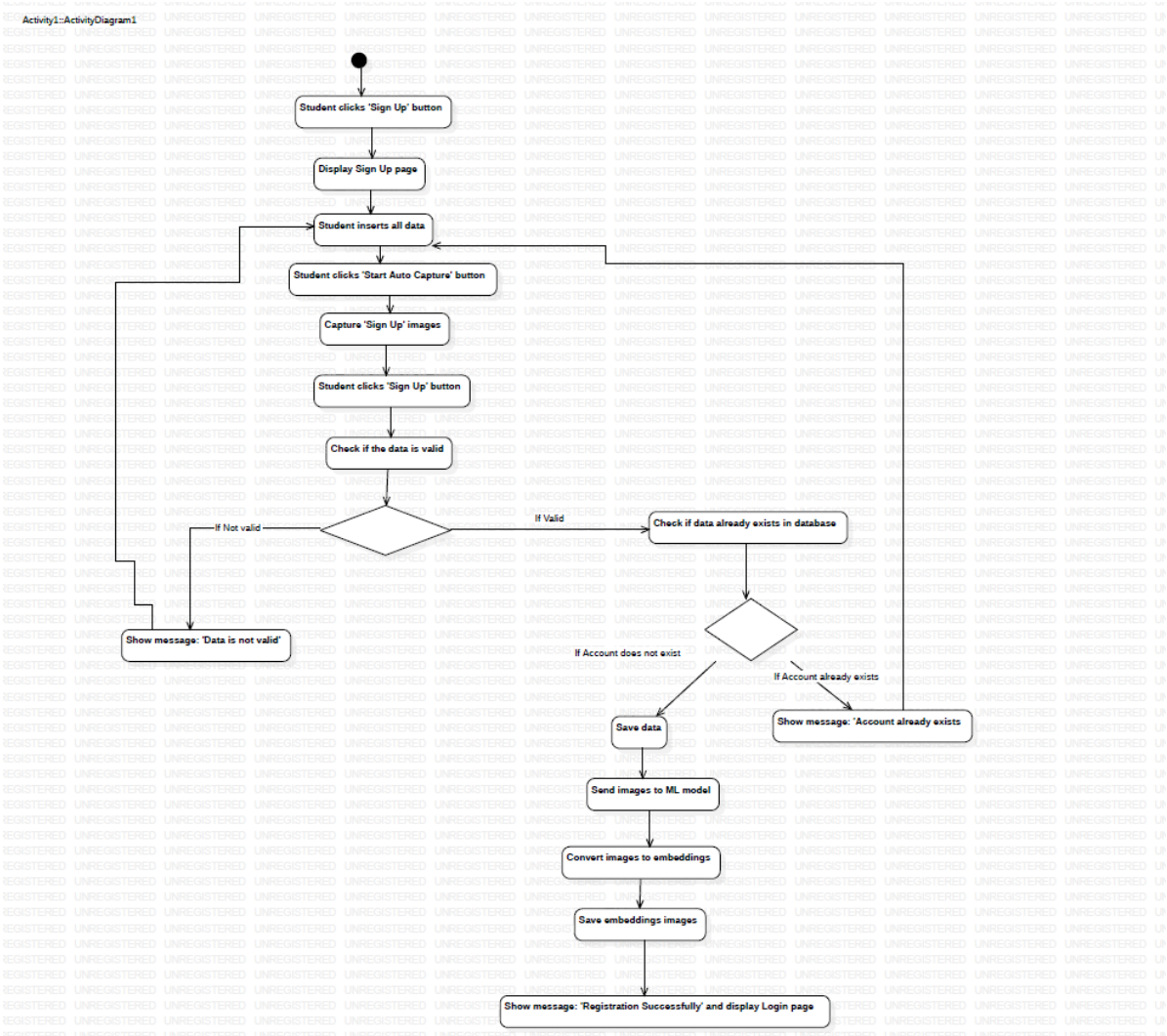
1) Login



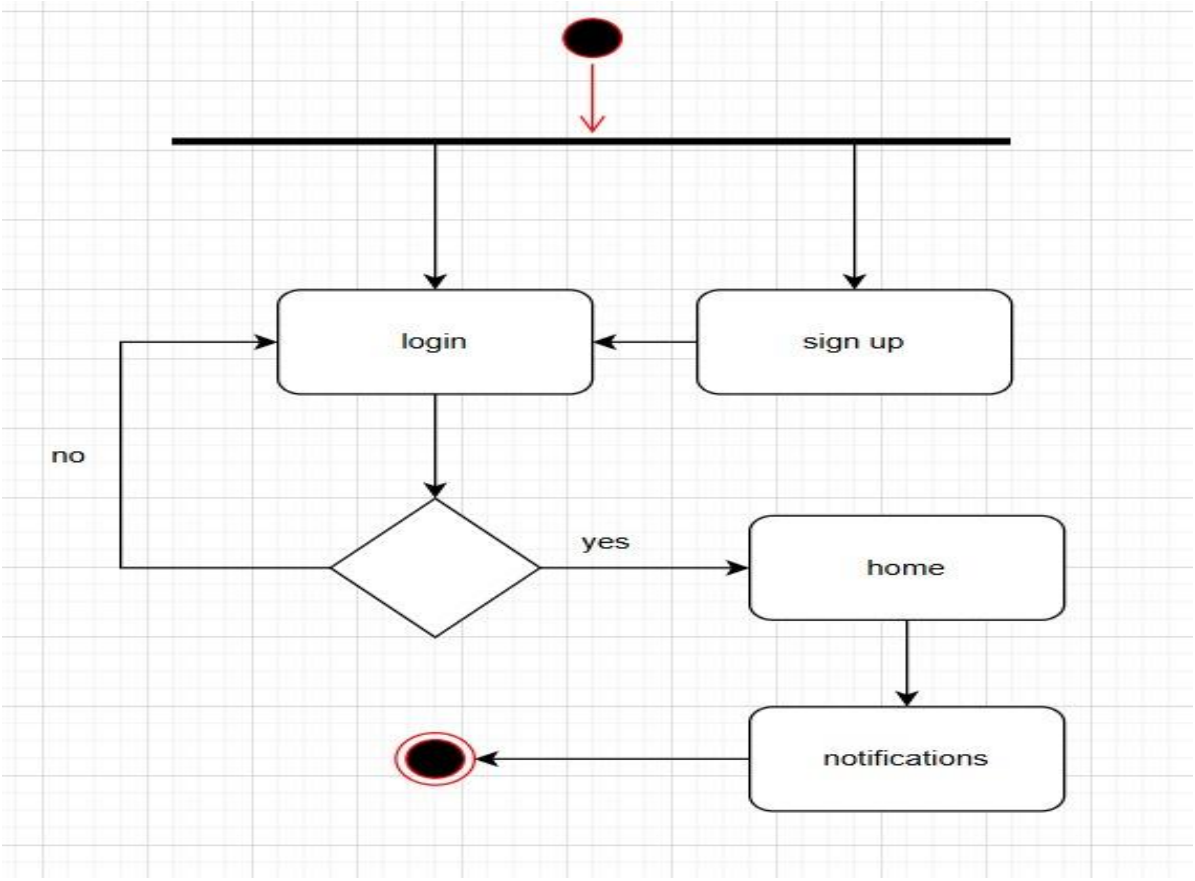
2) Reset Password



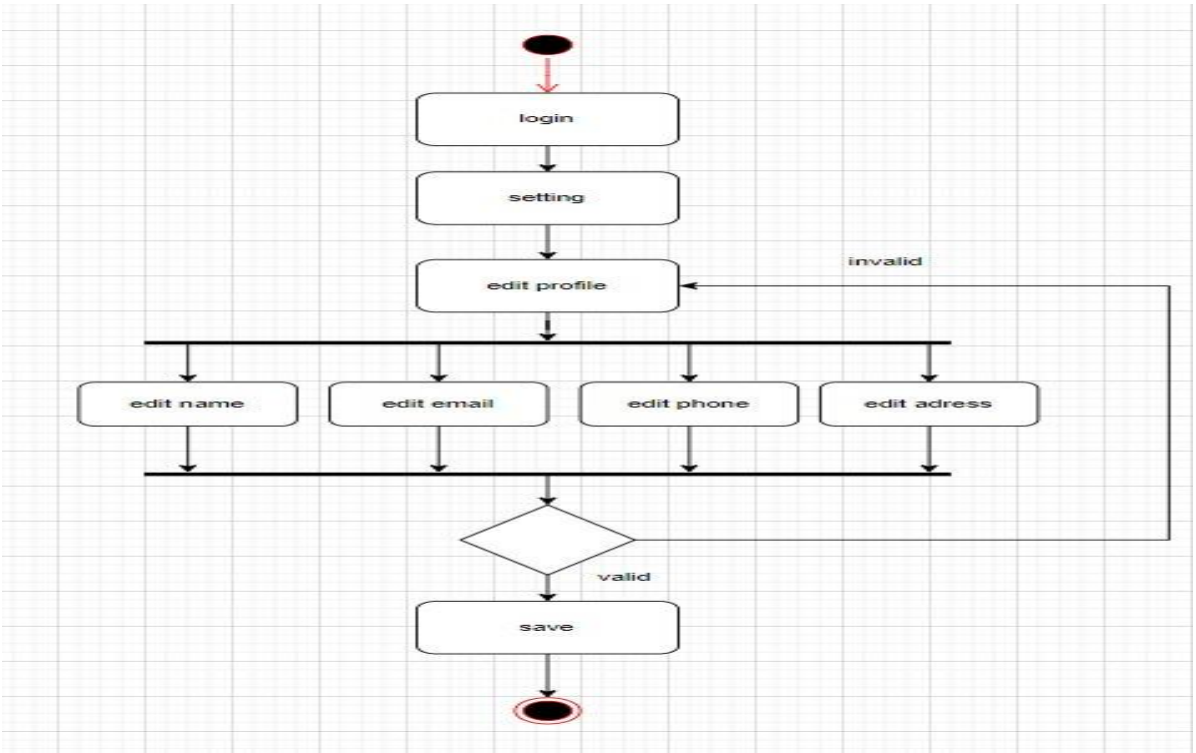
3) Sign Up


Activity1::ActivityDiagram1
Student clicks 'Sign Up' button
Display Sign Up page
Student inserts all data
Student clicks 'Start Auto Capture' button
Capture 'Sign Up' images
Student clicks 'Sign Up' button
Check if the data is valid
If Not valid
If Valid
Check if data already exists in database
Show message: 'Data is not valid'
If Account does not exist
If Account already exists
Save data
Show message: 'Account already exists
Send images to ML model
Convert images to embeddings
Save embeddings images
Show message: 'Registration Successfully' and display Login page

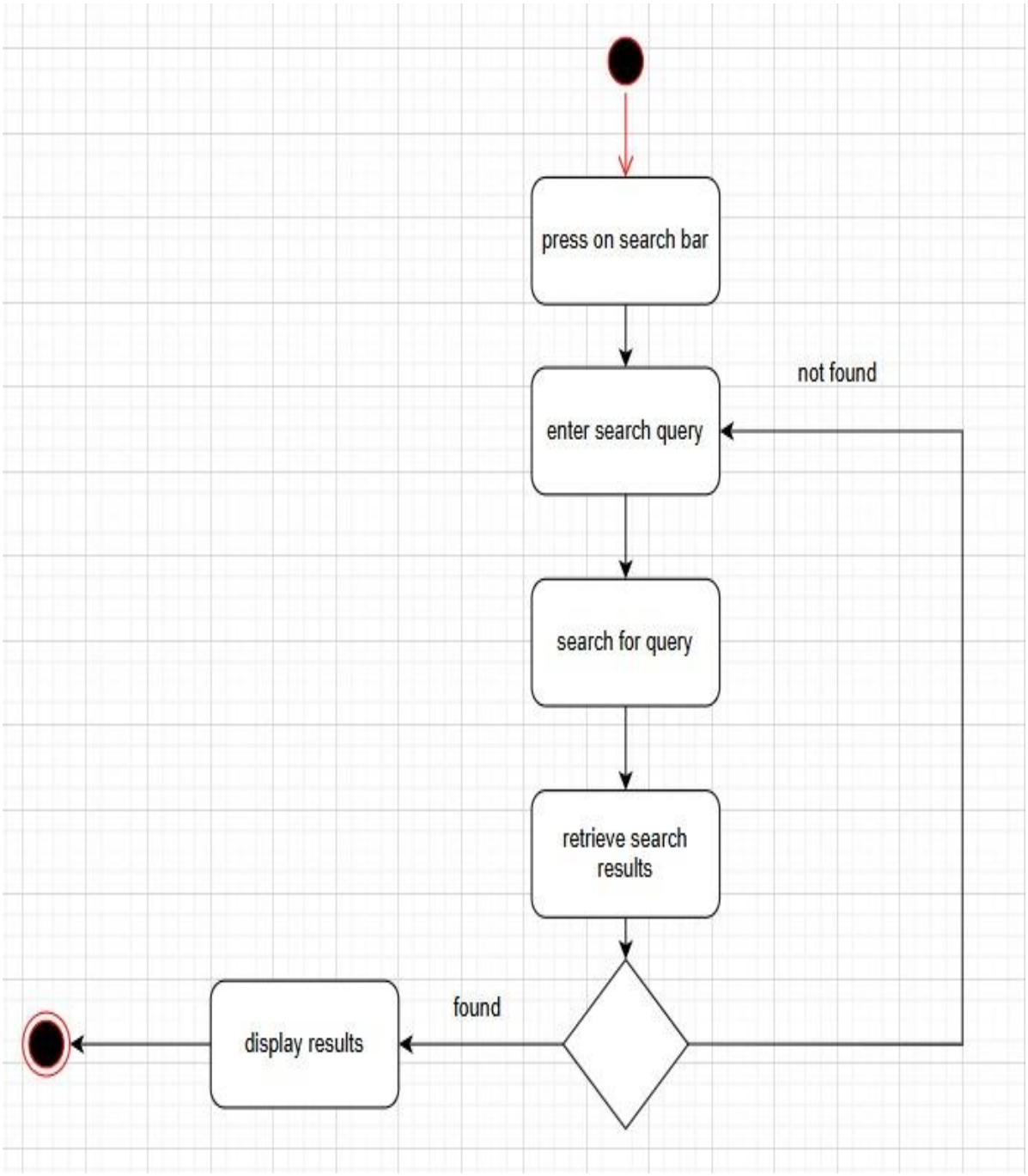
4) Check Notification



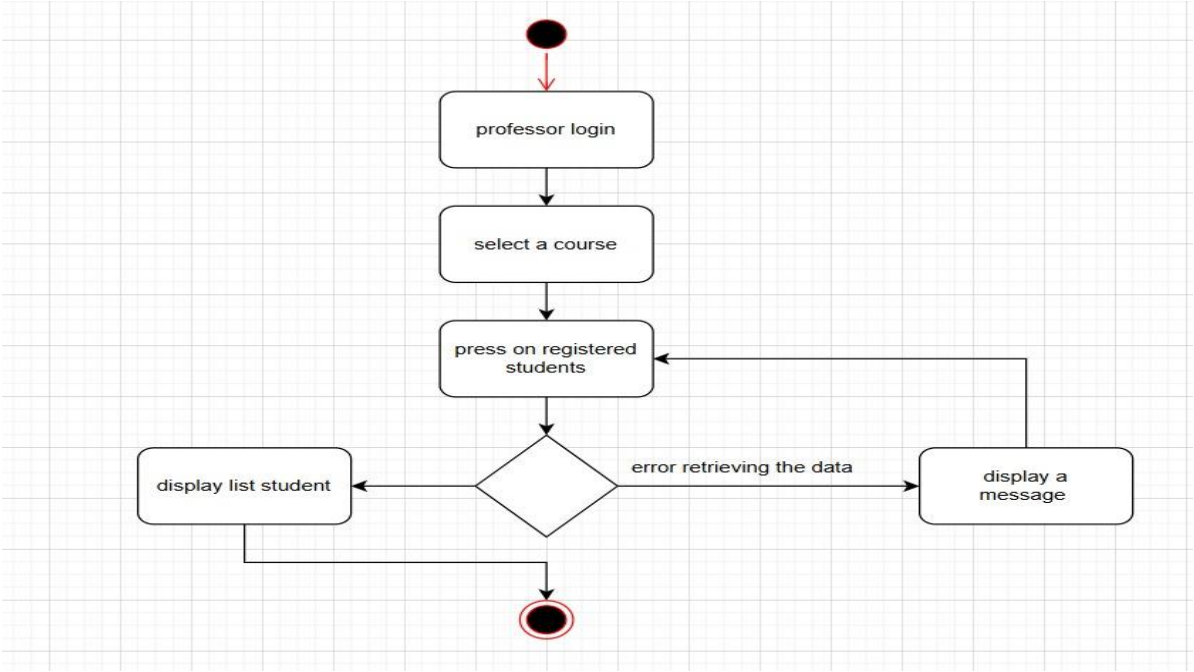
5) Edit Profile



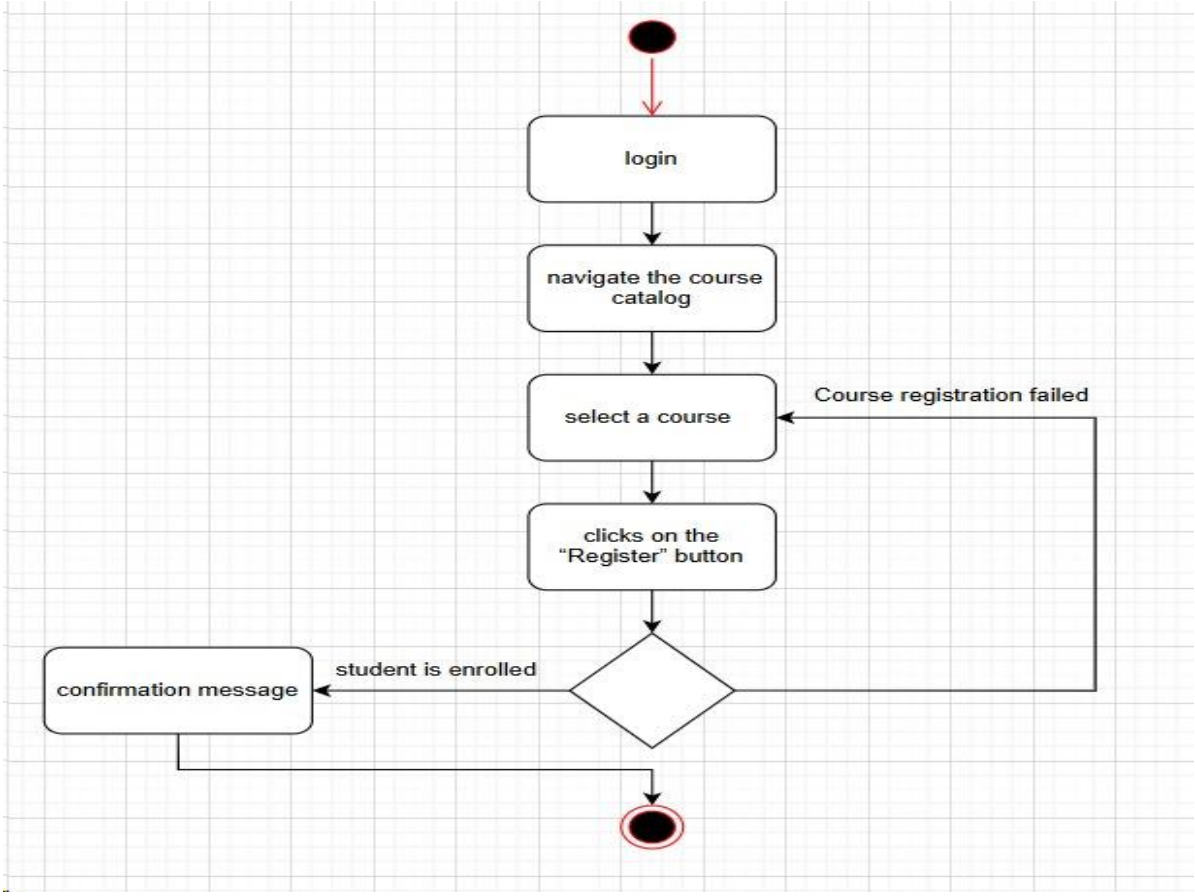
6) Search


press on search bar
enter search query
not found
search for query
retrieve search results
found
display results

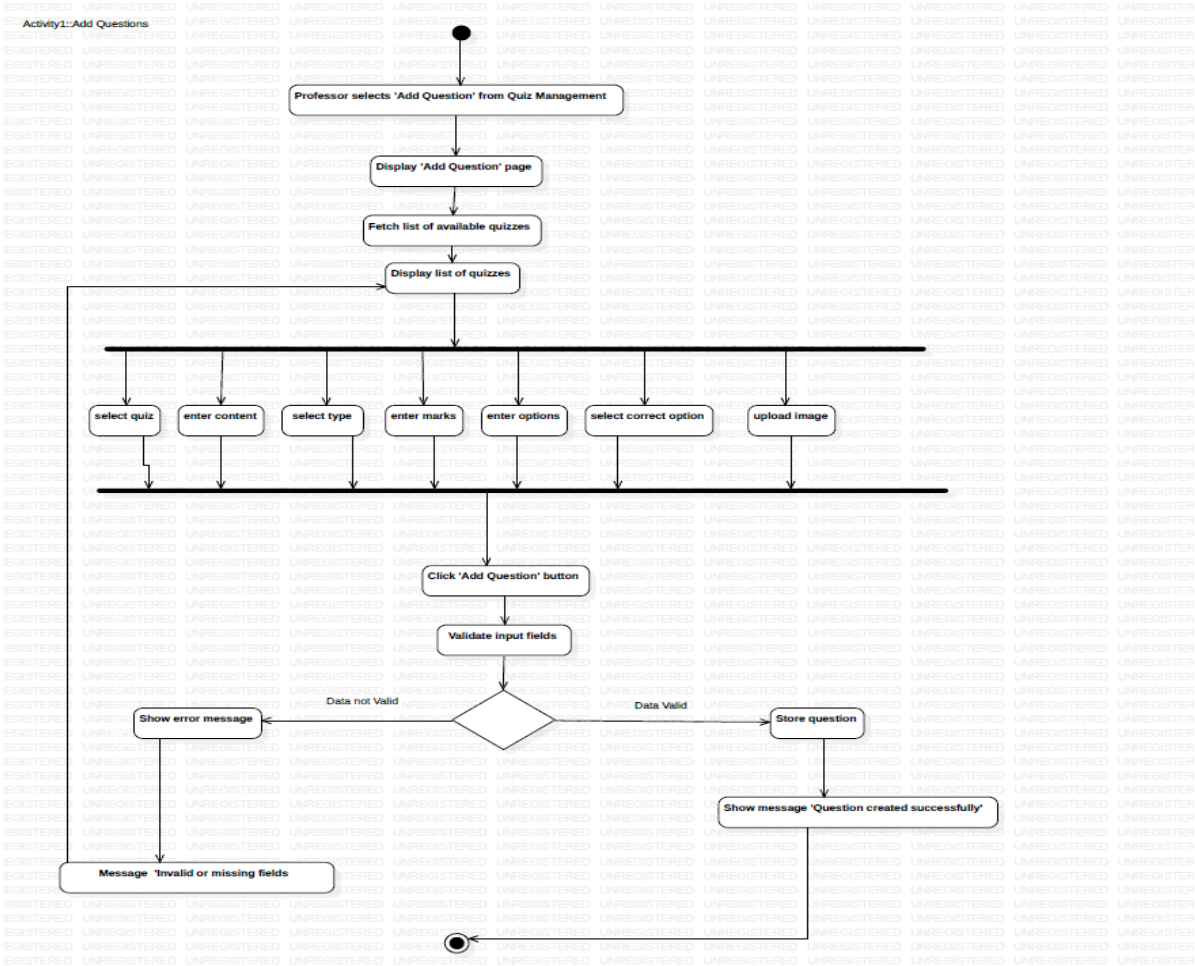
7) View registered students



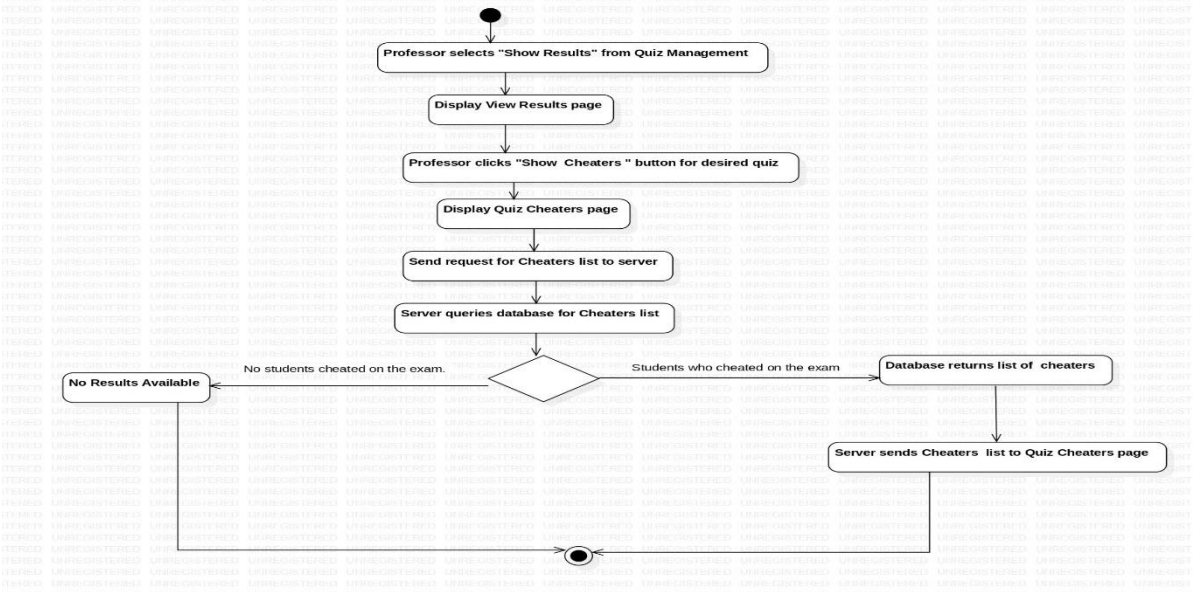
8) Course Registration



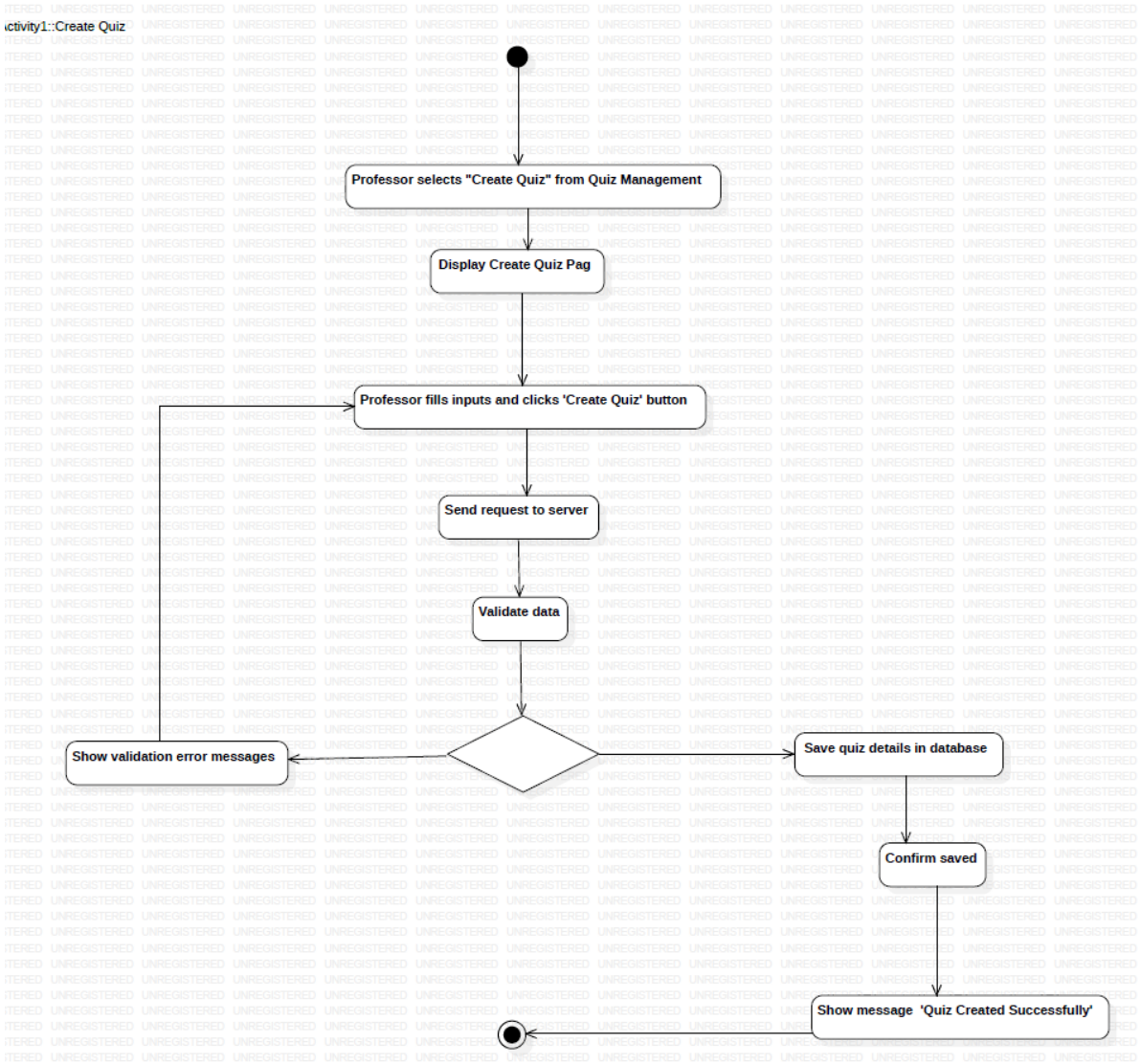
9) Add Questions



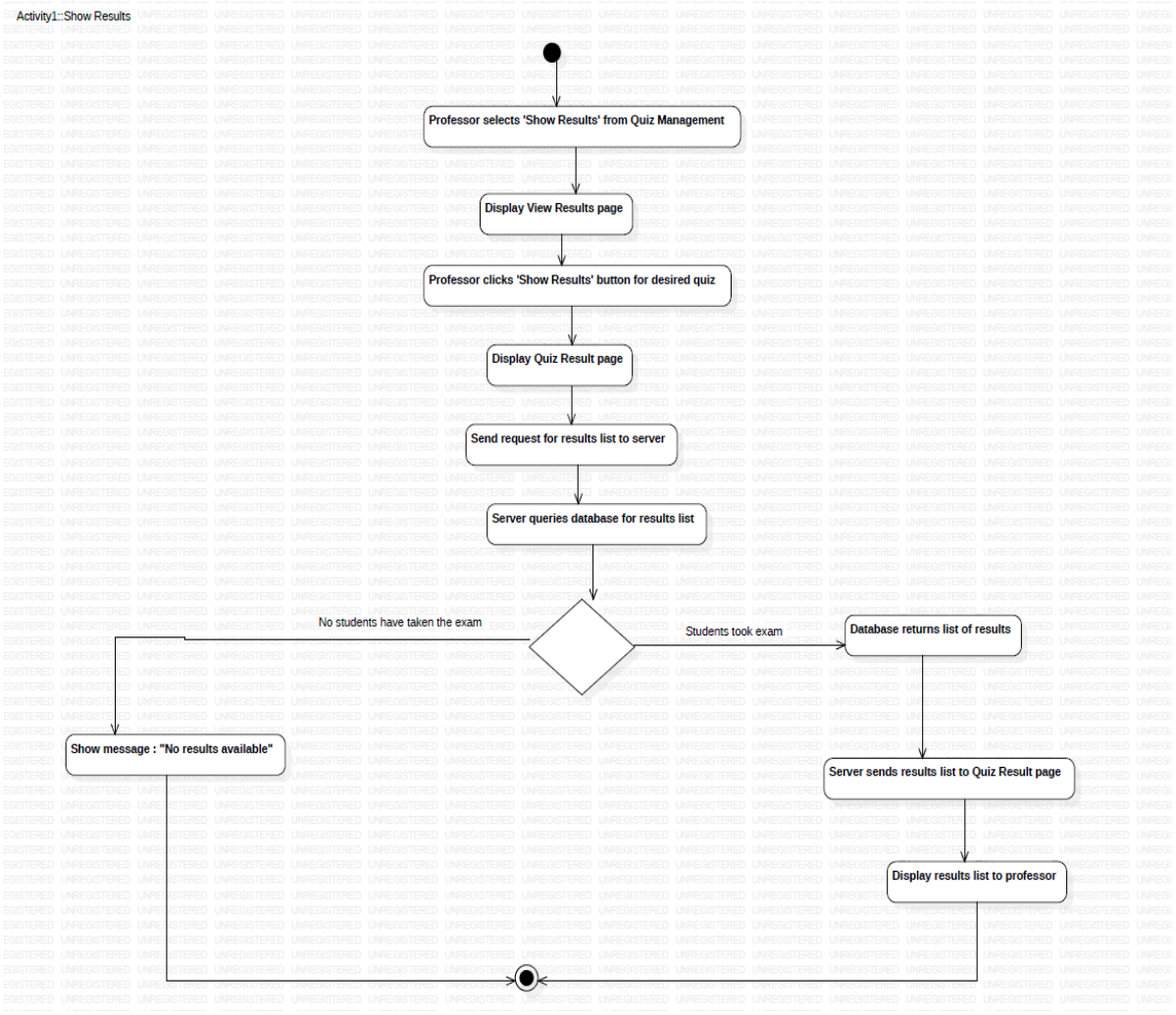
10) Show Results for Cheaters



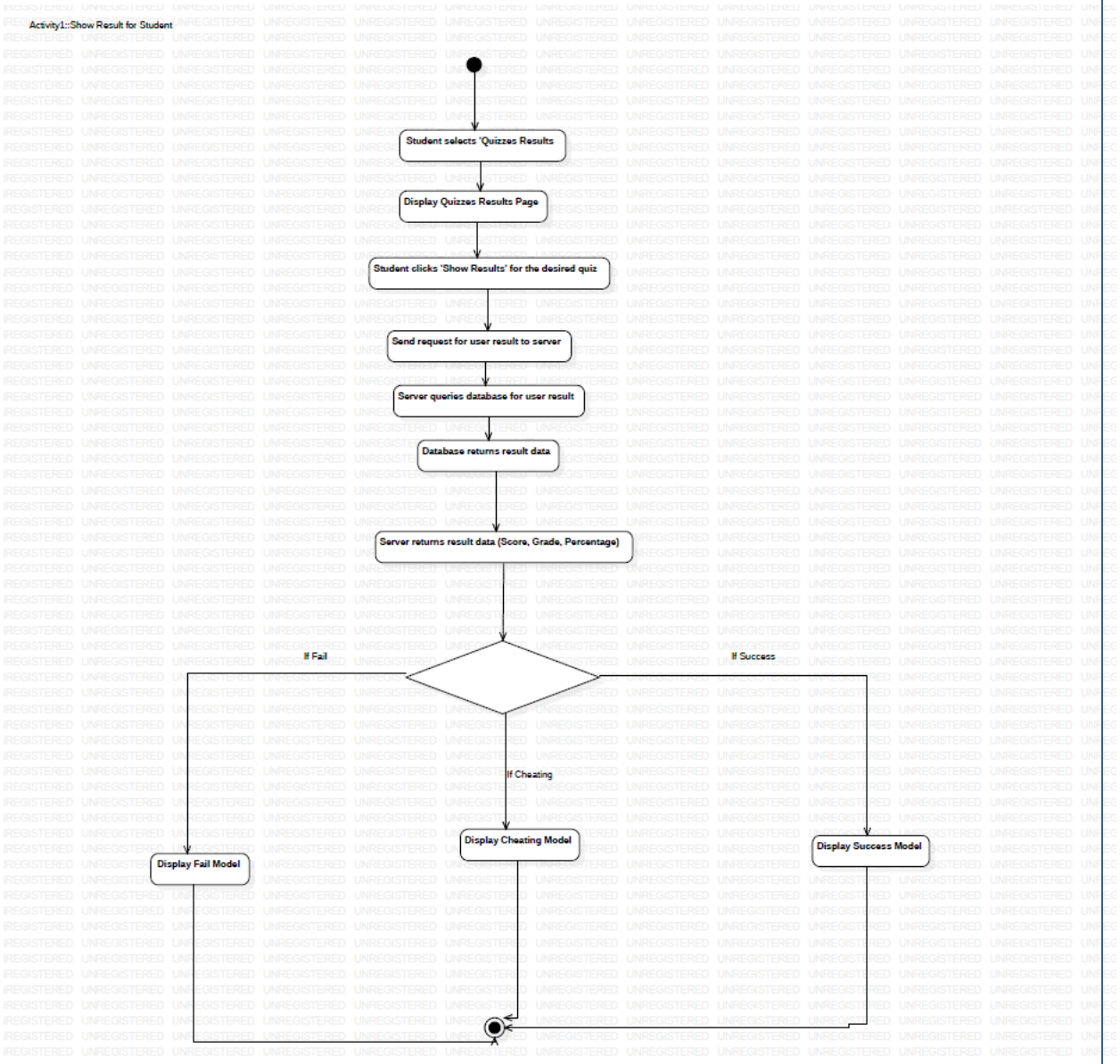
11) Create Quiz



12) Show Results for Prof

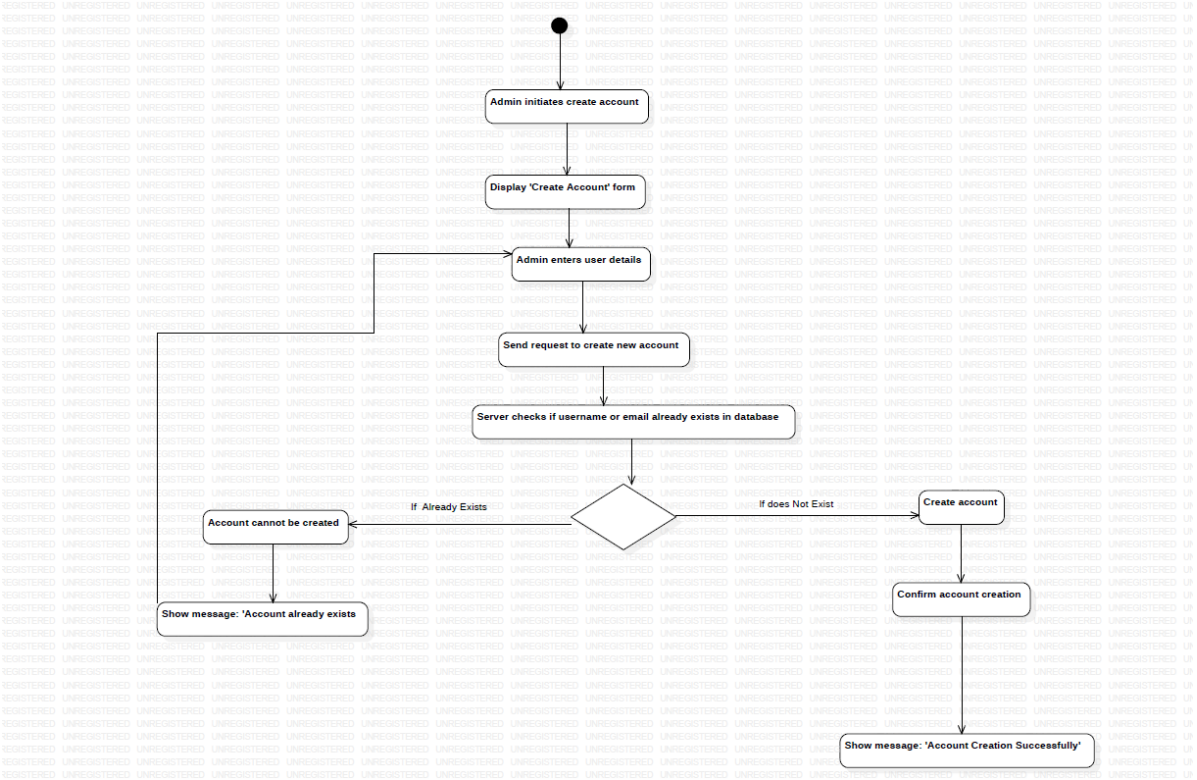


13) Show Results for Student

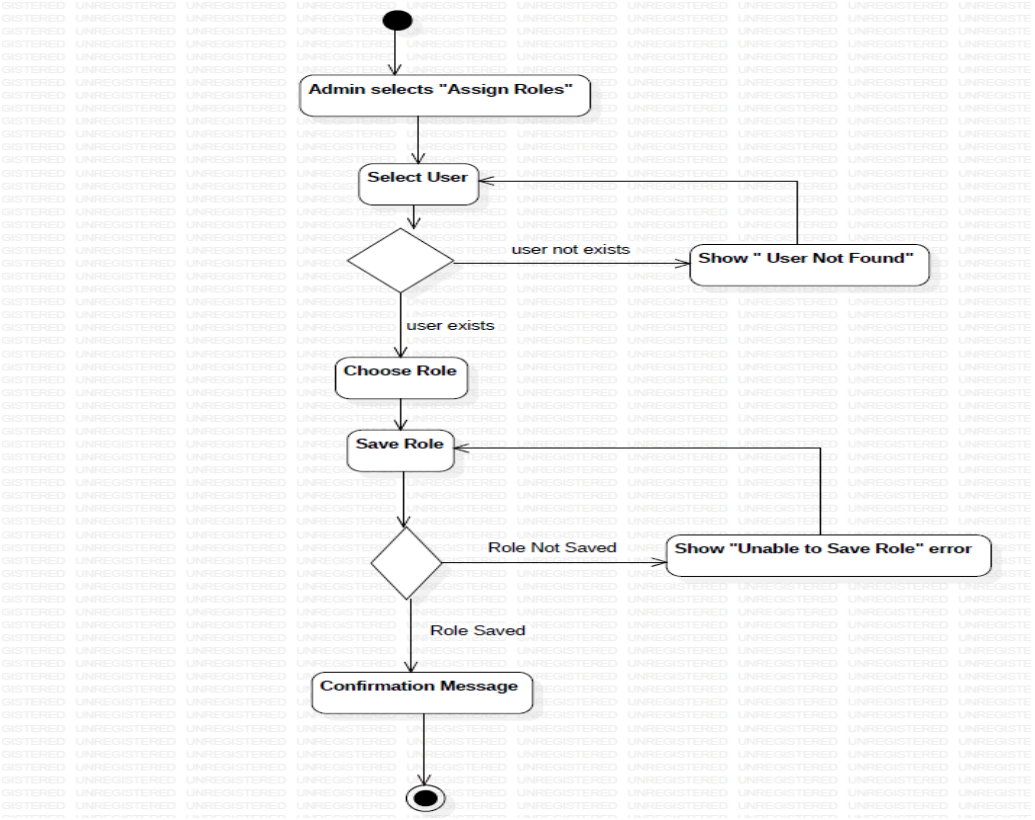


14) Show Answers

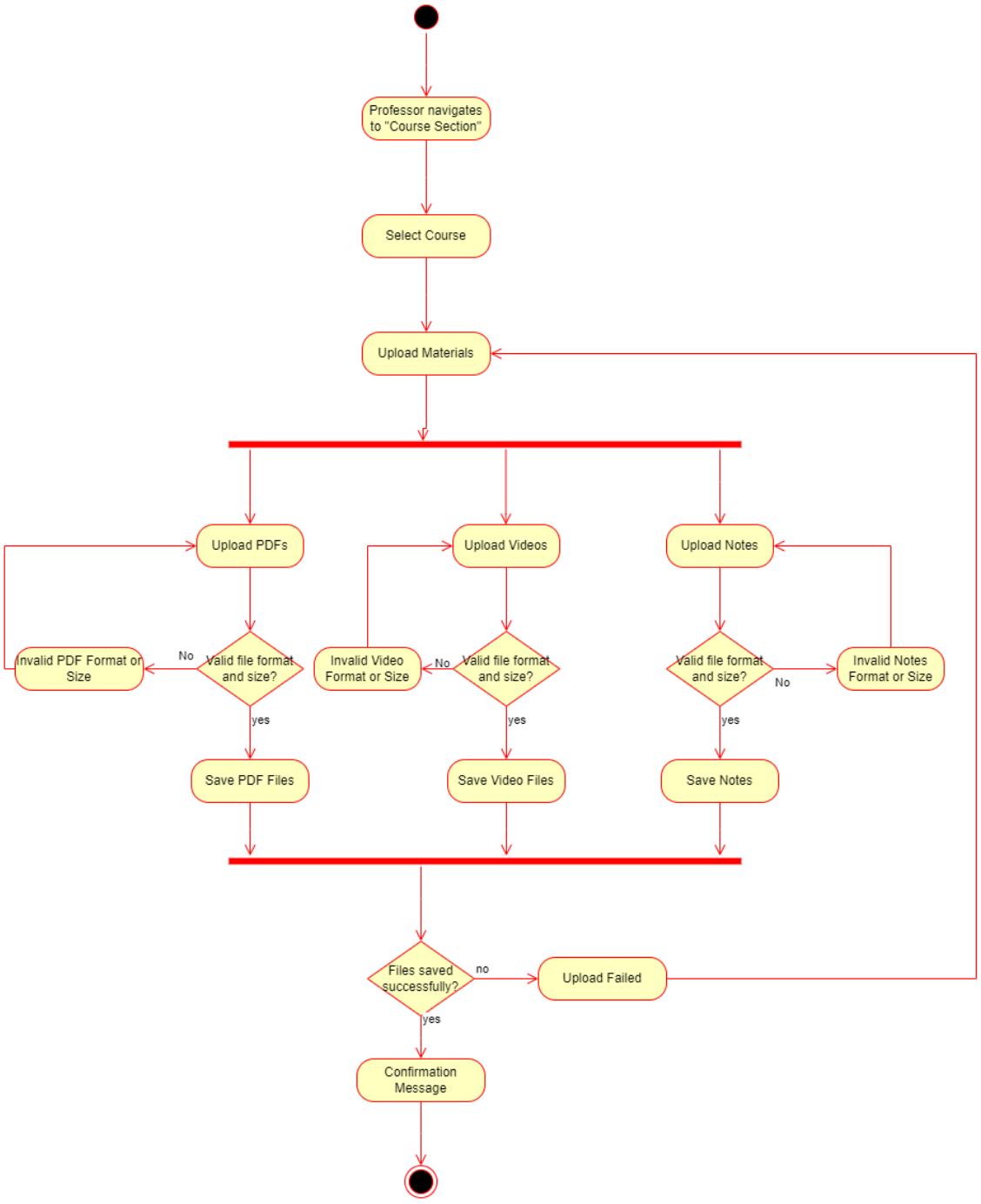
15) Create Account



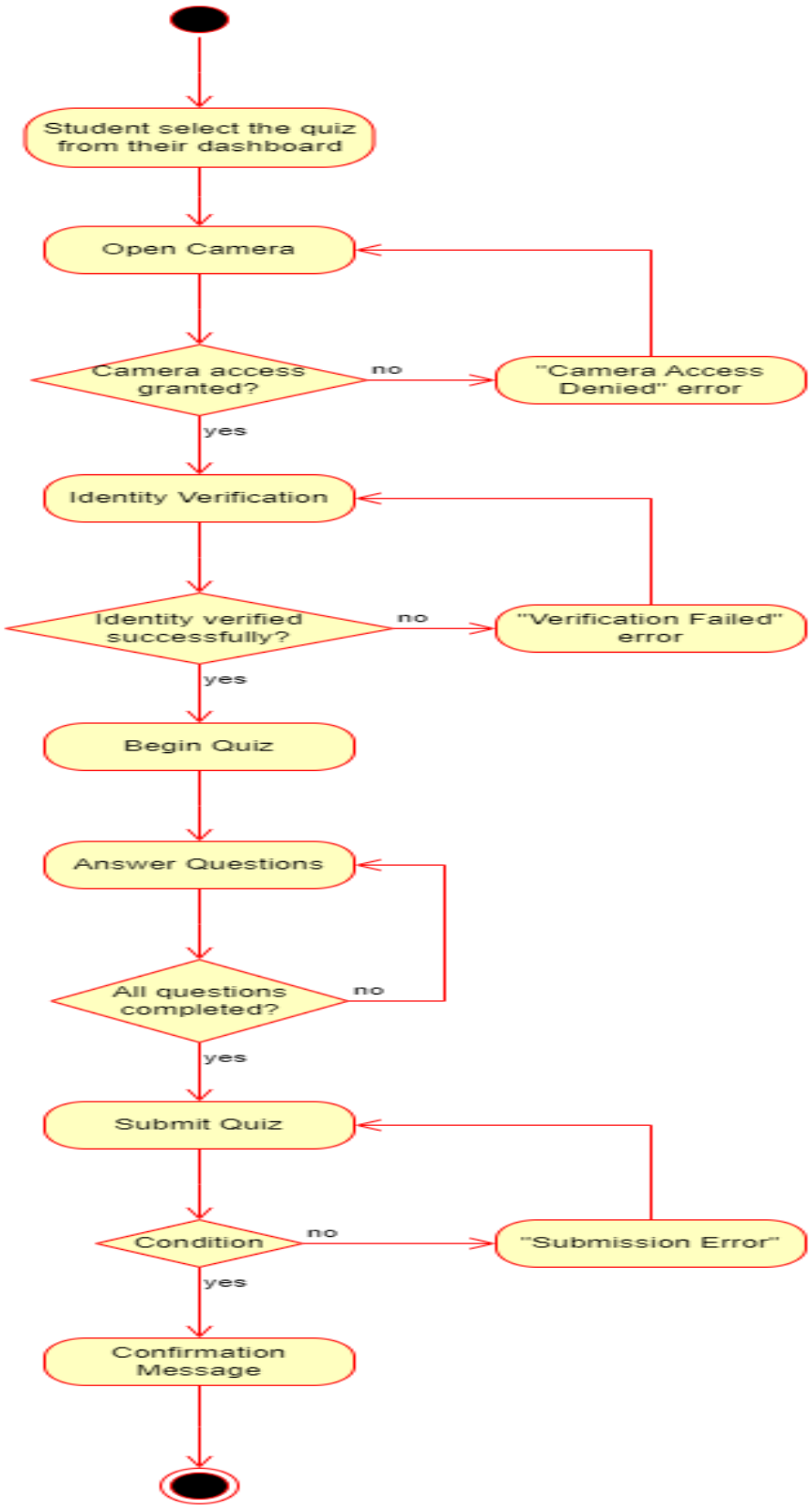
16) Assign Role



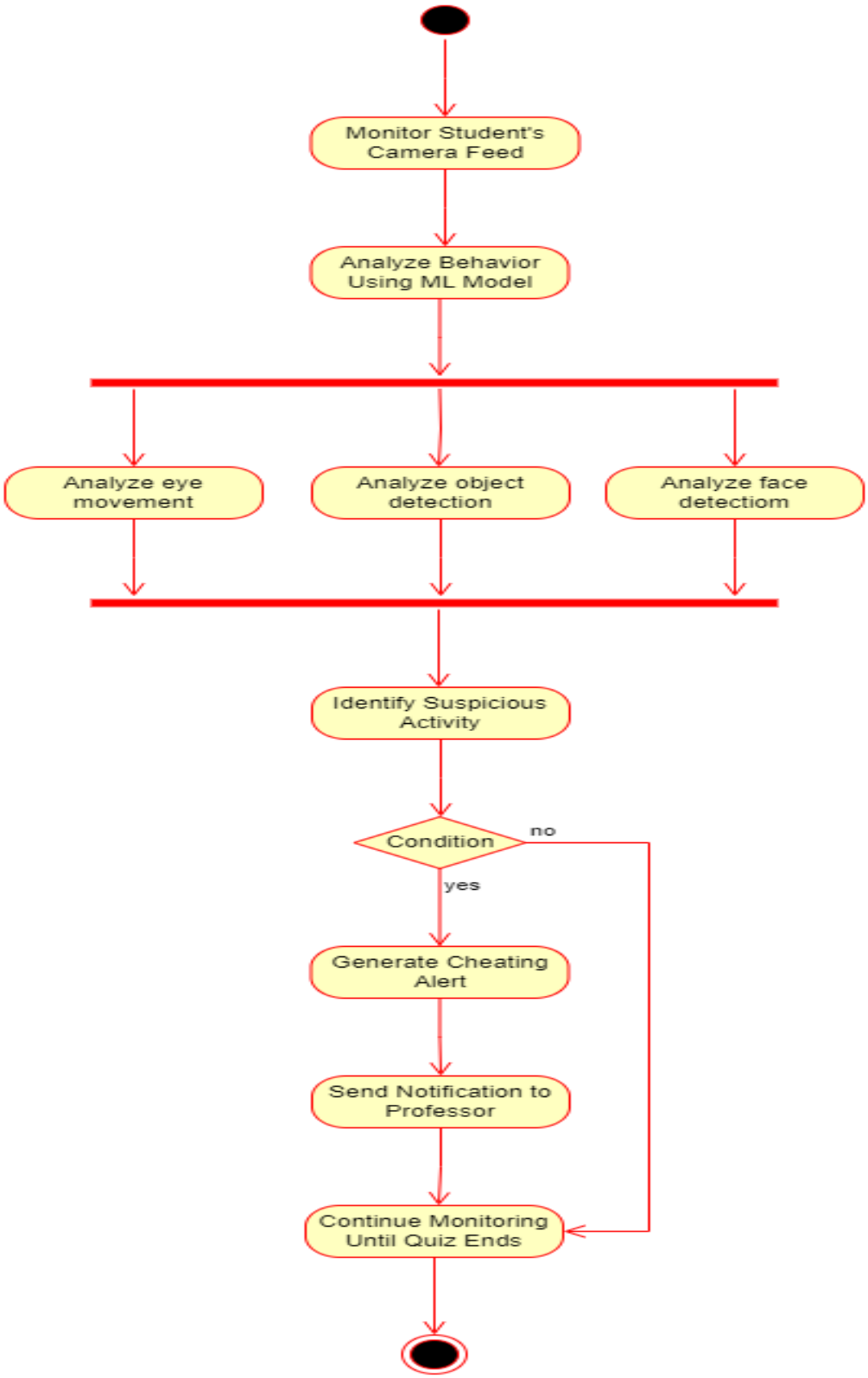
17) Upload Course Materials



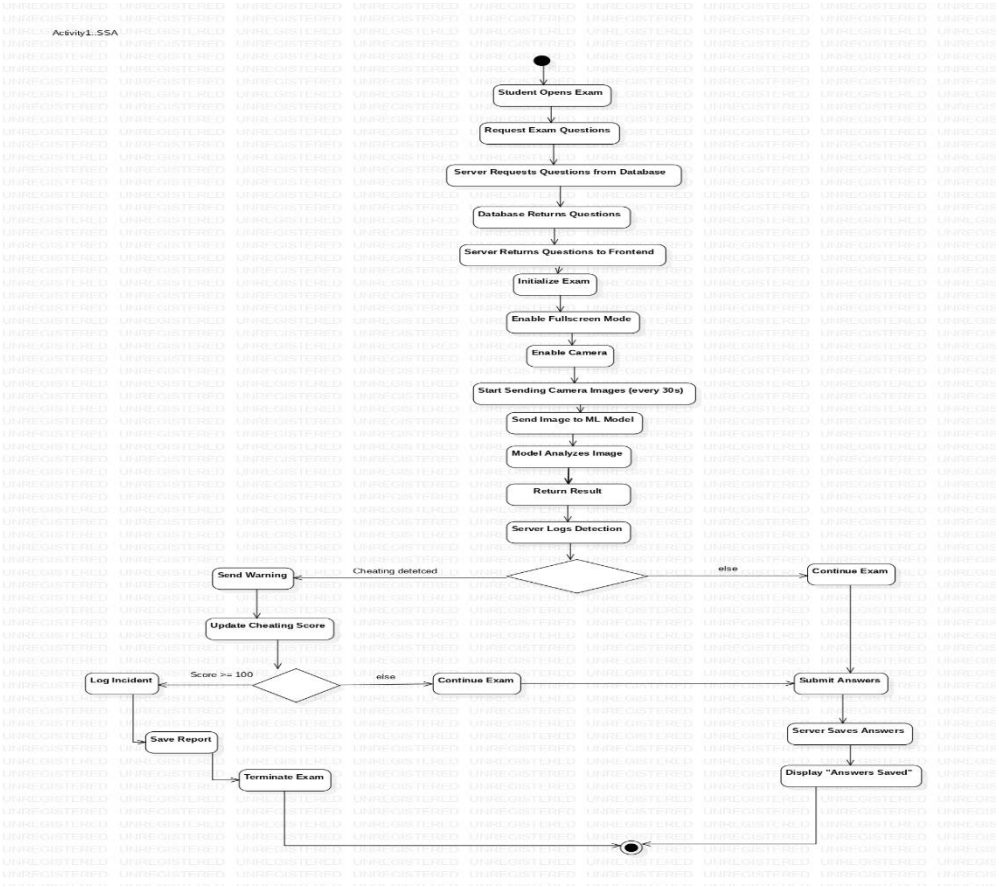
18) Start Quiz



19) Detecting Suspicious Behavior



20) During the quiz



6. Technology Stack

- **Frontend:**

  Developed using React.js to create a dynamic, responsive, and user-friendly interface that ensures seamless interaction across different devices.

- **Backend:**
  - **Laravel (PHP):** Handles core backend functionalities, including user authentication, API development, and database management.
  - **FastAPI (Python):** Used for deploying and serving machine learning models efficiently with high performance.

- **Machine Learning Models:**

  Utilized frameworks such as TensorFlow, OpenCV and Yolo for training, inference, and real-time image processing and object detection tasks.

- **Databases:**

  MySQL was used for structured data storage, user information, and historical logs, providing reliable and scalable data management.

- **WebSocket:**

  Integrated WebSocket for real-time communication between the client and server, ensuring low-latency updates, live notifications, and interactive features.

7. ML Models

7.1. Face Detection Model

7.1.1 Introduction

Face detection serves as a fundamental component in ensuring the integrity and fairness of online examinations. In remote assessment environments, traditional physical invigilation is absent, increasing the risk of impersonation and unauthorized collaboration. Implementing real-time face detection enables continuous verification of examinee presence and the identification of multiple individuals in the camera frame. This proactive monitoring mechanism addresses critical security concerns in digital examination systems and supports academic institutions in maintaining examination standards.

Practical uses:

- Validate a student's presence.
- Detect additional unauthorized faces.
- Enable other models like gaze and head pose estimators for behavioral monitoring.

7.1.2 Model Overview

Architecture

YOLOv8 (You Only Look Once version 8) is an anchor-free, one-stage object detector. It uses a single neural network for both object localization and classification within a single pass over the image.

Key features

- Backbone: CSPDarknet-based feature extractor.
- Neck: PANet for enhanced multi-scale feature aggregation.
- Head: Anchor-free detection head predicting bounding boxes, objectness scores, and class probabilities.

- Export options: Supports ONNX, TensorRT, and OpenVINO for deployment flexibility.

Why YOLOv8

Selection based on the following criteria:

- Outperformed YOLOv5, Faster R-CNN, and SSD in latency and accuracy trade-offs for small, fast-moving targets like faces.
- Native support for fine-tuning and transfer learning.
- Real-time inference (50+ FPS on GPU for 640×640 images).
- Active community, maintained documentation, and regular updates.

7.1.3 Dataset Description

Source: Public face detection datasets (WIDER FACE dataset).

Features:

- 32,203 images
- 393,703 annotated faces
- Diverse settings: lighting, pose, occlusion, expressions, crowd density

Dataset Split

Table 6: Dataset Split for Face Detection Model

Set	Images	Purpose
Train	22,400	Model training
Val	3,646	Hyperparameter tuning
Test	5,957	Final evaluation

Label Format

- YOLO format: class_id x_center y_center width height
- Normalized values (range: 0–1)

Image Processing:

- Resized to 640x640 pixels for model compatibility.
- Color space retained as RGB (no grayscale conversion).
- Verified data integrity for missing or corrupt images.

Label Generation:

- Converted ground-truth XML annotations to YOLO format using YOLOv8 (yolov8s.pt pretrained weights).
- Coordinates normalized and rounded to 6 decimal places for precision.

Augmentations:

- Integrated via data.yaml (flips, HSV adjustments, scaling, rotation).

Table 7: Augmentations for Face Detection Model

Technique	Purpose
Horizontal flip	Handle lateral variation (fliplr = 0.15)
Vertical flip	Handle vertical variation (flipud = 0.15)
HSV Saturation	Address color saturation changes (hsv_s = 0.2)
HSV Value	Address lighting inconsistencies (hsv_v = 0.2)
Random scaling	Improve size robustness (scale = 0.3)
Random rotation	Improve robustness to orientation (rotate = ±5°)

7.1.4 Training Process

Base model: yolov8n-face-lindevs.pt

Model type: Fine-tuned version of YOLOv8n adapted for face detection

The training process employed supervised learning with the following parameters:

- Framework: Ultralytics YOLOv8 implementation.
- Optimizer: SGD.

- Learning rate: 0.0001.
- Batch size: 16.
- Number of epochs: 40.
- Image size: 640x640 pixels.
- Loss function: Combination of bounding box regression loss, confidence loss, and class probability loss.

Strong transfer learning from pre-trained weights enabled rapid convergence by epoch 2.

7.1.5 Evaluation

Evaluation metrics included:

- Evaluation metrics included:
- Precision: 95.1%
- Recall: 95.1%
- mAP (IoU=0.5) (mean Average Precision at Intersection-over-Union threshold of 0.5): 98.7%
- mAP@0.5–0.95 (mean Average Precision averaged over IoU thresholds from 0.5 to 0.95 in 0.05 increments): 88.9%
- Inference speed: ~0.9 ms per image.

Results indicated robust face detection performance in varied environmental conditions typical of home exam settings.

Figure 53: Training vs. Validation Loss for Face Detection Model

7.1.6 Challenges and Solutions

Key challenges:

- Variable lighting and webcam quality.
- Detecting partially occluded faces.

- Maintaining low latency on mid-range hardware.

Solutions applied:

- Extensive data augmentation to simulate different lighting and angles.
- Fine-tuning YOLOv8 anchor box sizes for faces.
- Implementing real-time frame resizing and adaptive detection thresholds.

7.1.7 Conclusion and Future Work

The YOLOv8-based face detection model demonstrated high accuracy and reliable real-time performance for online exam proctoring. It successfully identified multiple faces and supported continuous candidate verification throughout exam sessions.

Future improvements:

- Integrate a face recognition module for identity confirmation.
- Expand training data with more diverse environments and camera types.
- Implement server-based detection for centralized exam monitoring.
- Enhance model robustness against occlusions and extreme lighting variations.

This implementation forms a critical layer in the broader anti-cheating framework, contributing to secure and equitable remote examination processes.

7.2.Head Pose Estimation Model

7.2.1 Introduction

Real-time face detection serves as a fundamental component in addressing these security concerns by enabling continuous verification of examinee presence and identifying multiple individuals in the camera frame. Beyond mere presence verification, head pose estimation plays a critical role in detecting suspicious behaviors that may indicate cheating. For instance, significant deviations in yaw, pitch, or roll can signal attempts to consult external materials, communicate with others, or otherwise violate exam rules. The model leverages state-of-the-art object detection and regression techniques to accurately estimate head poses in real-time, thereby contributing to the proactive monitoring of examinees' behavior during exams.

7.2.2 Dataset Description

Source and Size:

The dataset used for training the head pose estimation model was sourced from two primary datasets:

1. **BIWI Dataset**: A well-known benchmark for head pose estimation, containing images of faces annotated with ground-truth yaw, pitch, and roll angles.
2. **300W-LP Dataset**: A large-scale dataset for facial landmark detection, which includes diverse head poses and challenging lighting conditions.

The combined dataset consisted of:

- Total Images: 31,903
- Annotated Faces: 393,703
- Diverse Settings: Captured under varying lighting conditions, poses, occlusions, expressions, and crowd densities

Preprocessing Steps: To prepare the dataset for training, the following preprocessing steps were applied:

1. **Image Resizing:**
    - All images were resized to 640x640 pixels to ensure compatibility with the YOLOv8 model architecture.
    - Aspect ratio preservation was maintained using padding where necessary.
2. **Face Detection and Cropping:**
    - A pre-trained YOLOv8n-face model (yolov8n-face-lindevs.pt) was used to detect faces in each image.
    - Detected faces were cropped and saved as separate images for head pose estimation.
3. **Label Generation:**
    - Ground-truth yaw, pitch, and roll values were extracted from annotation files
    - Labels were normalized to the range [-1, 1] based on predefined angle ranges:
        - Yaw: [−75∘,75∘]
        - Pitch: [−60∘,80∘]
        - Pitch: [−60∘,80∘]
4. **Data Splitting:**
    - The dataset was split into training (70%), validation (15%), and test (15%) sets.

Data Augmentation

To improve generalization and robustness, the following augmentations were applied during training:

- **Geometric Transformations:**
    - Horizontal flip (f 15% probability)
    - Vertical flip (flipud: 15% probability)

    - Random scaling (scale: up to ±30%)
    - Random rotation (rotate: ±5°)
- **Color Space Adjustments:**
    - HSV saturation variation (hsv_s: ±20%)
    - HSV value variation (hsv_v: ±20%)
- **Geometric Transformations:**
    - Random horizontal flipping (p=0.5).
    - Random rotation (±10°).
    - Random gaussian noise with a standard deviation of (0.01).
- **Color Space Adjustments:**
    - Randomly adjusts brightness between 90% and 110%.
    - Randomly adjusts contrast between 90% and 110%.

7.2.3 Statistical Characteristics

The distribution of yaw, pitch, and roll angles in the dataset is illustrated in the provided histograms. Key observations include:

- Yaw Distribution: Peaks around 0∘, indicating frontal-facing poses dominate the dataset.
- Pitch Distribution: Concentrated near 0∘, with some samples showing upward and downward tilts.
- Roll Distribution: Skewed toward negative values, reflecting slight leftward tilts.

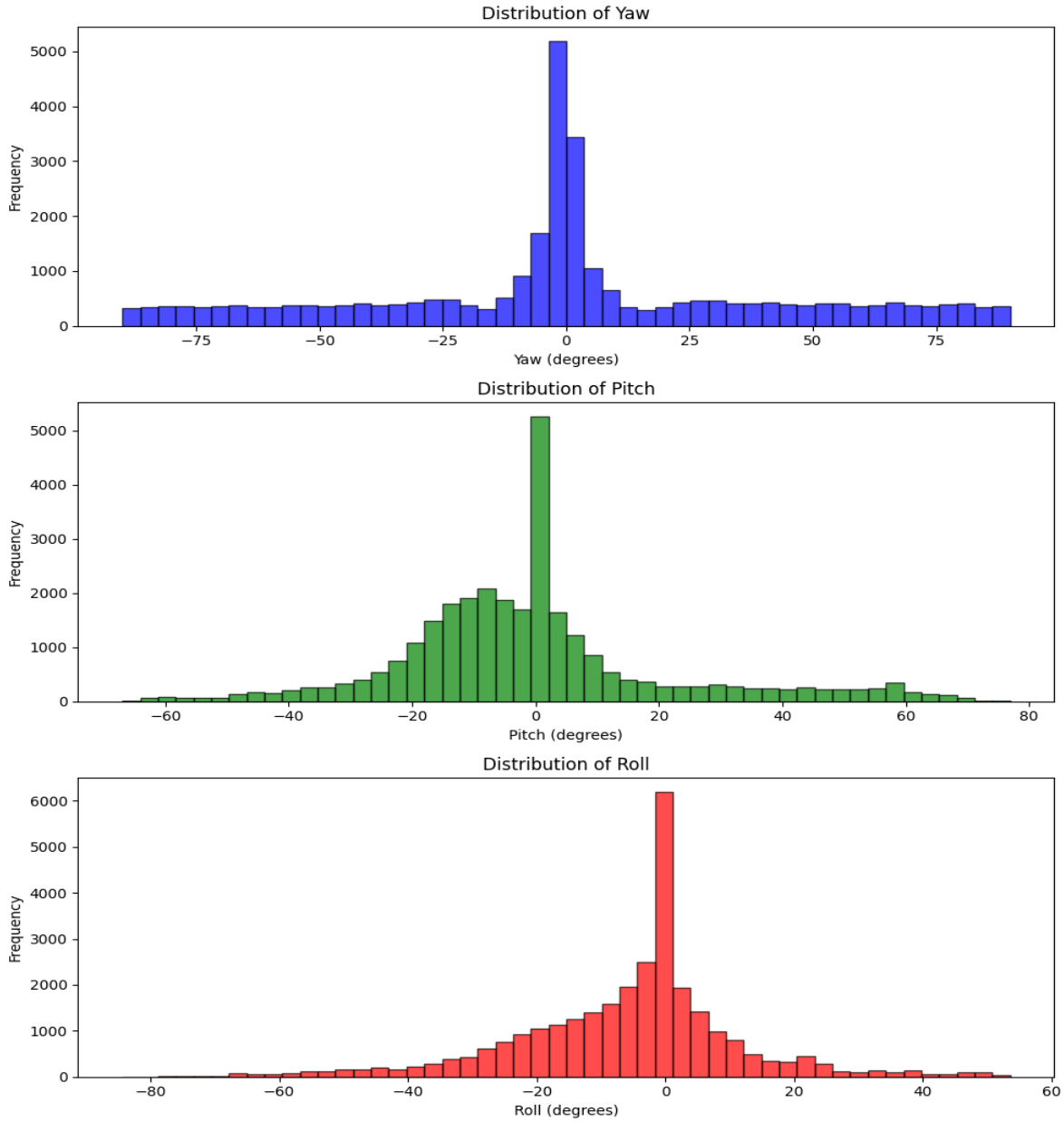


Figure 54: The distribution of yaw, pitch, and roll angles in the dataset for Head Pose Estimation Model

7.2.4 Potential Biases and Representativeness

While the combined BIWI + 300W-LP dataset is large and varied, it initially showed a strong neutral-pose bias:

- **Frontal Bias:** A higher frequency of frontal-facing poses may lead to overfitting if not addressed.

- **Lighting Variability:** While diverse lighting conditions are present, extreme scenarios (e.g., very low light) are underrepresented.
- **Up/Down Bias**: Insufficient data on up and down poses caused residual bias in the initial model.

To mitigate these biases:

- **Under sampling:** Neutral poses (frontal, minimal yaw/pitch/roll) were under sampled to balance the dataset.
- **Data Augmentation:** Extensive use of geometric and color transformations ensured robustness to variations.
- **Fine-Tuning:** Retrain the model using a filtered dataset to mitigate bias.

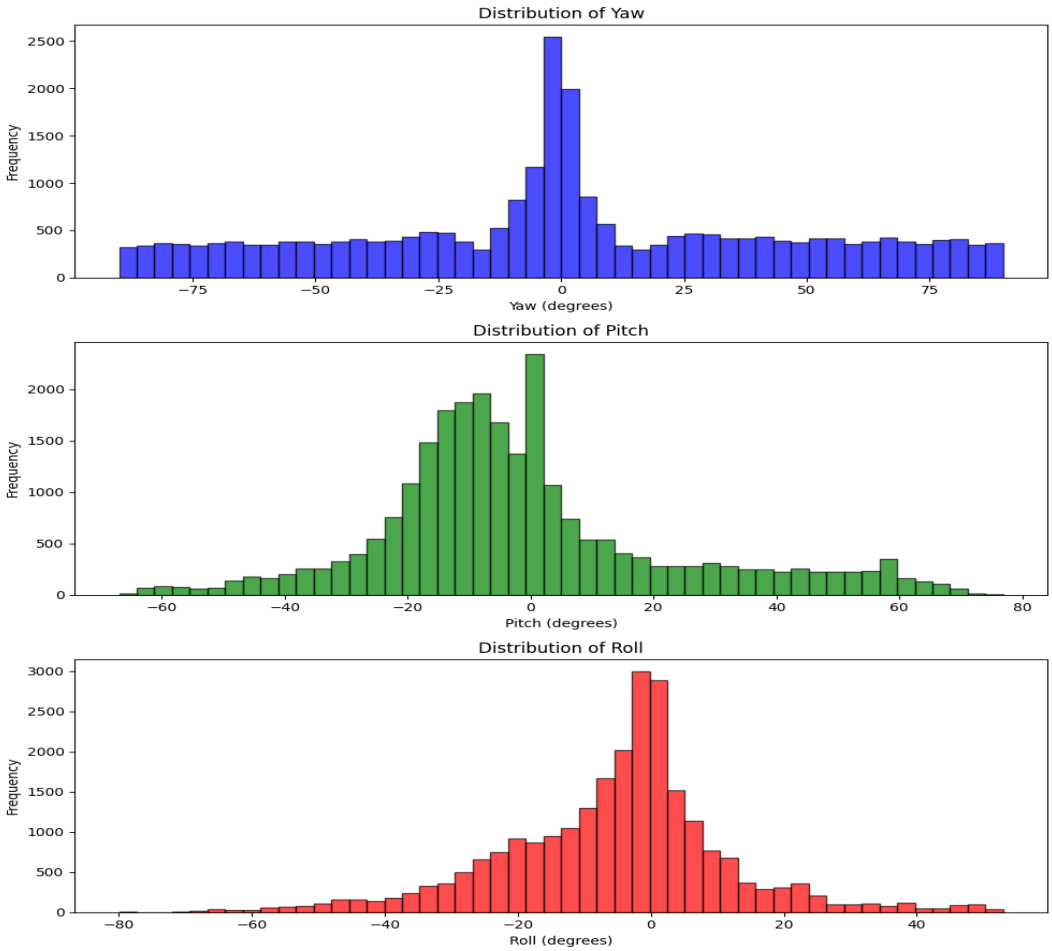


Figure 55: The distribution of yaw, pitch, and roll angles in the dataset for Head Pose Estimation Model

7.2.5 Model Overview

Architecture

The head pose estimation model is based on a modified version of **ResNet-18**, fine-tuned for regression tasks. The architecture comprises the following components:

1. **Backbone:**
    a. **ResNet-18:** Pre-trained on ImageNet for feature extraction.
    b. **Frozen Layers:** Lower layers were frozen to retain pre-trained features while fine-tuning higher layers.
2. **Regression Head:**
    a. A fully connected network regresses yaw, pitch, and roll angles.
    b. **Layers**:
        i. Input: 512-dimensional features from ResNet-18.
        ii. Hidden Layers: Two sequential layers with 512 and 256 neurons, respectively.
        iii. Output: Three regression outputs (yaw, pitch, roll).
    c. **Activation Functions:**
        i. ReLU used between hidden layers
        ii. Dropout applied for regularization (rates: 0.5 and 0.3)
3. **Pose Classification Head**:
    a. A parallel branch classifies poses into five categories: frontal, left, right, up, and down.
    b. **Layers**:
        i. Input: 512-dimensional features from ResNet-18.
        ii. Hidden Layer: 256 neurons.
        iii. Output: Five classification outputs.

Advantages

- Efficiency: ResNet-18 is lightweight and computationally efficient, suitable for real-time applications.

- Transfer Learning: Pre-trained weights on ImageNet provide strong initial features.
- Dual Tasking: Simultaneous regression and classification enable both precise angle estimation and pose categorization.

Why This Model Over Others

- Multi-task Learning: Combining regression and classification improves overall performance by leveraging complementary signals.
- Robustness: Fine-tuning on diverse datasets ensures adaptability to varied head poses and lighting conditions.
- Scalability: The modular design allows easy integration with existing face detection pipelines.

Fine-Tuning Strategy to Address Up/Down Bias

To correct the bias toward frontal poses and improve detection of extreme poses (up and down), the model underwent an additional phase of fine-tuning using the AFLW2000-3D dataset, which contains more diverse and extreme head orientations.

Reason for Choosing AFLW2000-3D:

- Rich annotation of extreme head poses
- Includes ground-truth 3D face shapes and Euler angles
- Better representation of upward and downward gazes compared to BIWI and 300W_LP

Key Enhancements for Up/Down Detection

- Dynamic Weighting in Focal Loss
    - Higher weights assigned to "up" and "down" pose classifications to emphasize them during training.
- Manual Balancing of Extreme Poses
    - Under sampled frontal, left, and right pose images to 500 each
- Custom Thresholds for Pose Categorization
    - Defined thresholds for pose categorization

```
self.POSE_THRESHOLDS = {
    'frontal': (-15, 15),
    'left': (15, float('inf')),
    'right': (-float('inf'), -15),
    'up': (15, float('inf')),
    'down': (-float('inf'), -15)
}
```

Figure 56: Thresholds for Pose Categorization

- Data Augmentation During Fine-Tuning
    - Applied advanced augmentations including:
        - RandomAffine: **degrees=20**, **translate= (0.15, 0.15)**
        - ColorJitter: brightness/contrast/saturation ±0.3
        - Additional perspective warping to simulate extreme viewing angles

7.2.6 Training Process

Methodology

The training process involved the following steps:

1. **Dataset Loading:**
    a. Images and labels were loaded using PyTorch's DataLoader with weighted random sampling to address class imbalance.
2. **Model Initialization:**
    a. ResNet-18 was initialized with pre-trained weights.
    b. Regression and classification heads were randomly initialized.
3. **Loss Function:**
    a. A custom loss function combined Mean Squared Error (MSE) for regression and Cross-Entropy Loss for classification.
    b. Class weights were dynamically adjusted based on pose category frequencies.
4. **Optimization:**
    a. AdamW optimizer with differential learning rates for backbone and task-specific layers.

b. Learning rate scheduling to ensure convergence.

Hyperparameters

- Batch Size: 32
- Epochs: 25
- Learning Rate (Backbone): 5e-6
- Learning Rate (Regressor): 5e-5
- Learning Rate (Classifier): 5e-5
- Early Stopping Patience: 5 epochs

7.2.7 Evaluation

Metrics

The following metrics were used to evaluate model performance:

- **Mean Absolute Error (MAE):** Measures the average absolute difference between predicted and ground-truth angles.
- **Classification Accuracy:** Evaluates the accuracy of pose categorization.
- **Runtime Performance:** Inference speed measured in milliseconds per image.

Results

Table 8: Evaluation for Head Pose Estimation Model

Metric	Initial Model	Fine-Tuned Model
MAE (Yaw)	2.3°	1.7°
MAE (Pitch)	1.8°	1.5°
MAE (Roll)	2.1°	1.9°
Classification Accuracy	92.5%	94.3%
Inference Speed	25 ms/image	24 ms/image
Best Validation Loss	0.1651	1.8610

Validation Loss Behavior:

- The initial model showed lower validation loss due to simpler task distribution and better convergence on neutral poses.
- The fine-tuned model had higher loss values but was trained on more challenging data (extreme angles), leading to better real-world performance.

7.2.8 Challenges and Solutions

Challenges

1. **Class Imbalance:**
    a. Frontal poses were overrepresented, leading to biased predictions.
2. **Pose Ambiguity:**
    a. Similar poses could result in ambiguous classifications.
3. **Real-Time Constraints:**
    a. Ensuring sub-second inference times was crucial for practical deployment.

Solutions

1. **Under sampling:**
    a. Neutral poses were under sampled to balance the dataset.
2. **Weighted Sampling:**
    a. During training, samples were sampled proportionally to their inverse frequency.
3. **Hybrid Loss Function:**
    a. Combined regression and classification losses to improve robustness.
4. **Optimization Techniques:**
    a. Mixed precision training and gradient clipping ensured efficient computation.

7.2.9 Conclusion and Future Work

Summary

The proposed head pose estimation model effectively detects cheating behaviors in online exams by accurately estimating yaw, pitch, and roll angles in real-time. By integrating face detection and pose estimation, the system provides a comprehensive solution for maintaining academic integrity in remote assessment environments.

Future Work

1. **Improved Generalization:**
    a. Expand training data to include more extreme lighting conditions and diverse demographics.
2. **Behavioral Analysis:**
    a. Deploy the model on centralized servers for enhanced scalability and security.
3. **Adversarial Robustness:**
    a. Test the model against adversarial attacks and develop countermeasures.

7.3.Gaze Pose Detection

7.3.1 Introduction

Online examinations have become increasingly prevalent, necessitating robust proctoring solutions to maintain academic integrity. A critical component of such systems is gaze pose detection, which identifies deviations in a test-taker's eye movements that may indicate cheating behavior (e.g., looking at secondary devices or notes). This project develops a deep learning model to estimate gaze direction from eye images, enabling real-time monitoring during exams.

The significance of this work lies in its potential to:

- Deter cheating by providing automated, non-intrusive surveillance.
- Reduce reliance on human proctors, lowering costs and scalability limitations.
- Improve fairness by standardizing detection across all test-takers**.**

7.3.2 Dataset Description

Data Source and Composition

The model was trained on the **SynthEyes dataset**, comprising synthetic eye images with annotated 3D gaze vectors. Key features include:

- **Size:** 11,382 samples (split into 70% train, 15% validation, 15% test).
- **Diversity:**
    - Yaw range: ±180° (horizontal gaze).
    - Pitch range: ±50° (vertical gaze).
    - Balanced distribution: 26.5% Center, 35.7% right, 31.2% left, 24.1% up, and 32.0% down gazes.

Preprocessing Steps

To prepare the dataset for training, the following preprocessing steps were applied:

1. **Image Resizing:**
    - All images were resized to 224x224 pixels to ensure compatibility with the ResNet-18 backbone
2. **Normalization:**
    - Pixel values were normalized using ImageNet statistics (**mean= [0.485, 0.456, 0.406]**, **std= [0.229, 0.224, 0.225]**).
3. **Data Splitting:**
    - The dataset was split into training (70%), validation (15%), and test (15%) sets.

Data Augmentation

To improve generalization and robustness, the following augmentations were applied during training:

- **Geometric Transformations:**
    - Random horizontal flipping (p=0.5).
    - Random rotation (±10°).
    - Random gaussian noise with a standard deviation of (0.01).
- **Color Space Adjustments:**
    - Randomly adjusts brightness between 90% and 110%.
    - Randomly adjusts contrast between 90% and 110%.

7.3.3 Statistical Characteristics

The dataset exhibits the following statistical properties:

- Approximately equal distribution of gaze vectors across horizontal and vertical angles

- Imbalanced participant representation, with some contributing more samples

Potential Biases

- Vertical pitch distribution peaks near 0° (long tail to negatives), but negative skew hinders upward gaze (e.g., top monitor usability).
- Yaw bias of 7.3° to the right (35.7% right gazes vs. 31.2% left) may hinder detection of leftward glances.
- Poor lighting performance due to underrepresented extreme lighting in training data

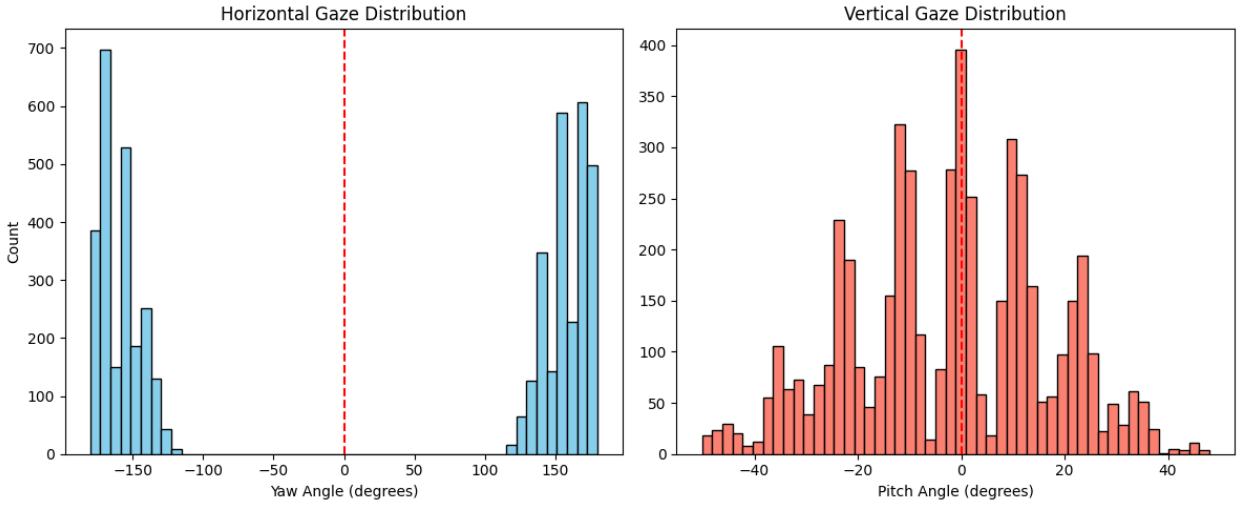


Figure 57: Horizontal and Vertical Gaze Distribution for Gaze Model

Mitigation Strategies

1. **Vertical Pitch Imbalance**
    - **Dynamic Loss Weighting:** (dynamic_weighted_mse_loss, adaptive_loss) increases the weight of upward gaze samples (pitch > 15° or upward_mask). This improves sensitivity to infrequent upward poses.
    - **Rotation Augmentation:** Random rotations (±10°) simulate head tilts, indirectly diversifying vertical pitch.

2. **Yaw Bias**
    - **Horizontal flip augmentation:** (50% probability in second model) mirrors images and inverts the x-component of the gaze vector, balancing left-right gaze distribution during training.
    - **Dynamic Peripheral Weighting (adaptive_loss):** Upweights samples with |yaw| > 30° to improve peripheral gaze detection and compensate for under-representation of left-gaze.
3. **Lighting Variability**
    - **Color Jitter Augmentation:** simulates lighting shifts
    - **Gaussian Noise Injection:** introduces mild visual degradation and improves resilience to poor lighting

7.3.4 Model Overview

Architecture

The selected model was based on the ResNet-18 architecture, fine-tuned for regression tasks (Baseline).

1. **Backbone**
    - Based on ResNet-18 pretrained on ImageNet
    - Lower layers frozen to retain generic features
    - The last two residual blocks unfrozen for task-specific fine-tuning
    - Final classification head replaced with regression head for 3D gaze prediction
2. **Regression Head**
    - Input: 512-dimensional features
    - Fully connected layer: 256 units, ReLU, Dropout 0.5
    - Output layer: 3 units (gaze vector x, y, z)

Training Details

- Loss: Dynamic weighted MSE
- Up-weights upward gazes (pitches > 0) by factor 1.5
- Optimizer: Adam, learning rate 0.001
- Scheduler: ReduceLROnPlateau, patience 2
- Early stopping patience: 5 epochs
- Batch size: 64
- Data split: 70% train, 15% val, 15% test

Evaluation

- Angular error computed via cosine similarity
- Best model checkpointed based on validation loss

Improved Model

Gaze Estimation with Data Augmentation and Adaptive Loss.

Architecture

- Also based on ResNet-18 pretrained on ImageNet
- First 30 parameters (4 layers approx.) frozen
- Regression head modified:
    - Fully connected layer: 256 units, ReLU, Dropout 0.3
    - Output layer: 3 units with Tanh activation
- Implement data augmentation to simulate real-time environmental variations (e.g., lighting).

Training Details

- Loss: Adaptive cosine similarity loss
- Peripheral gazes (yaw > 30°) upweighted by 1.2
- Upward gazes (pitch > 15°) upweighted by 1.3
- Optimizer: Adam, learning rate 0.0005
- Early stopping patience: 7 epochs

- Batch size: 64
- Data split: 70% train, 15% Val, 15% test

Evaluation

- Angular error computed via cosine similarity
- Best model checkpointed based on validation loss

Table 9: Evaluation for Gaze Pose Detection

Aspect	Model 1	Model 2
Loss Function	Dynamic Weighted MSE	Adaptive Cosine Loss with dynamic yaw/pitch weights
Data Augmentation	None	Random flips, jitter, noise, rotation
Learning Rate	0.001	0.0005
Dropout Rates	0.5	0.3
Frozen Layers	All except last 2 residual blocks	First 4 layers (30 parameters)
Activation (Output)	None	Tanh
Early Stopping	Patience 5	Patience 7

Advantages over alternatives:

- **Efficiency:** ResNet-18 balances accuracy and computational cost, crucial for real-time deployment.
- **Transfer learning:** Pretrained weights (ImageNet) mitigate data scarcity.
- **Robustness:** Fine-tuning on diverse datasets ensures adaptability to varied lighting and poses.

7.3.5 Results

Table 10: Results for Gaze Pose Detection

Metric	Model A	Model B
Initial Angular Error	26.53° (Epoch 1)	20.42° (Epoch 1)
Final Train Loss	0.0478 (Epoch 26)	0.0345 (Epoch 10)
Final Val Loss	0.0466 (Epoch 26)	0.0065 (Epoch 10
Best Val Angular Error	16.97° (Epoch 26)	5.06° (Epoch 10)
Best Val Loss	0.0466 (Epoch 26)	0.0065 (Epoch 10)

Model B reduced angular error by 70% compared to Model A

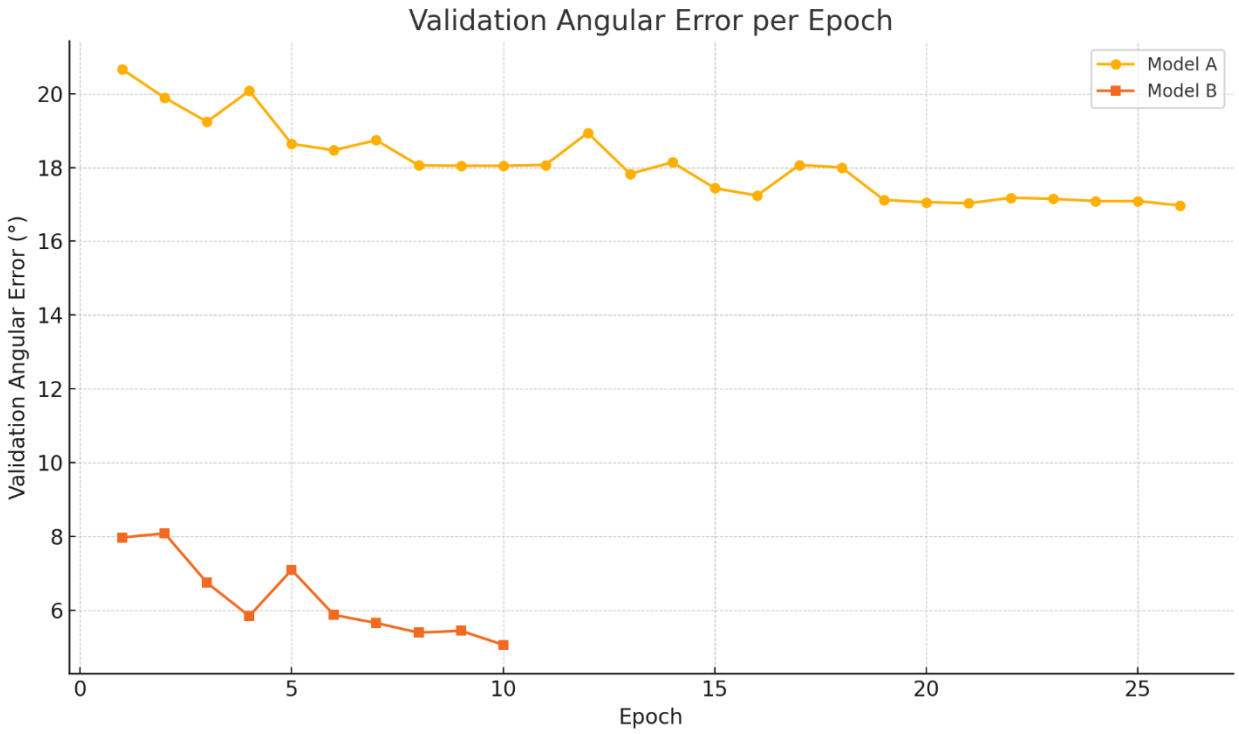


Figure 58: Validation Angular Error per Epoch for Gaze Estimation Model

7.3.6 Challenges and Solutions

Challenges

- Overfitting due to limited participant diversity
- Poor performance under extreme head tilts
- Inconsistent webcam resolutions during live testing

Solutions

- Implemented aggressive data augmentation strategies
- Applied weighted sampling to balance gaze angle bins during training
- Normalized gaze vectors relative to detected head pose angles
- Added a face detection fallback mechanism for frame drops or face absence

7.3.7 Conclusion and Future Work

Summary

This project demonstrated the feasibility of using a lightweight gaze pose estimation model for detecting cheating behaviors in online exams. The ResNet-18 based model achieved competitive accuracy with real-time performance on standard hardware.

Future Work

- Extend the system to analyze facial expressions, and eye blink rates
- Develop an anomaly detection module using statistical thresholds on gaze patterns

7.4.Object Detection Model

7.4.1 Introduction

Online exams face growing risks of cheating, especially with unauthorized devices and materials. To address this, our system uses **YOLOv8s**, a fast and accurate object detection model designed to monitor exams in real time. YOLOv8s detects suspicious activity instantly, ensuring fair testing without slowing down the exam process.

Built for efficiency, YOLOv8s works on standard computers and combines speed with precision. It identifies banned items like phones, notes, or hidden devices, making it ideal for proctoring remote quizzes.

Practical Uses

The YOLOv8s model prevents cheating by detecting:

- **Unauthorized Devices:** Flags smartphones or secondary laptops near the student.
- **Notes/Books:** Alerts when written or printed materials appear on screen.
- **Suspicious Hand** Movements: Tracks gestures like writing off-screen or handling hidden objects.
- **Headphones:** Spots audio devices used for illegal communication.

7.4.2 Model Overview

Architecture

YOLOv8s (You Only Look Once version 8 small) is an anchor-free, one-stage object detector. It uses a single neural network to perform object localization (detecting bounding boxes) and classification (identifying object categories) in a single pass over the input image or video frame. Designed for real-time efficiency, it maintains high accuracy while prioritizing speed and

low computational demands, making it ideal for applications like live exam proctoring.

Key Features

- Anchor-Free Design: Predicts objects directly (no predefined anchor boxes), simplifying training.
- Single-Stage Detection: Detects and classifies objects in one fast pass through the network.
- CSPDarknet Backbone: Efficiently extracts features for quick, accurate predictions.
- PANet Neck: Combines features from different scales (better for small objects like phones).
- Real-Time Speed: Processes 50+ FPS on GPUs (ideal for live video monitoring).
- Lightweight: Runs on low-cost hardware (e.g., basic laptops or edge devices).
- Easy Deployment: Supports ONNX/TensorRT for integration with apps and cloud systems.

Why YOLOv8

Selection based on the following criteria:

- Outperformed YOLOv5, Faster R-CNN, and SSD in latency and accuracy trade-offs for small, fast-moving targets like faces.
- Native support for fine-tuning and transfer learning.
- Real-time inference (50+ FPS on GPU for 640×640 images).
- Active community, maintained documentation, and regular updates.

7.4.3 Dataset Description

Data Sources & Labeling

a) Data Collection

1. Roboflow Datasets

- Pre-labeled datasets from Roboflow Universe were used as a foundational training corpus.
- Included annotated images of cheating-related objects (e.g., phones, headphones, notes).

2. Web Images

- Publicly available images of cheating-associated objects (electronic devices, written materials, hands) were downloaded from the internet.

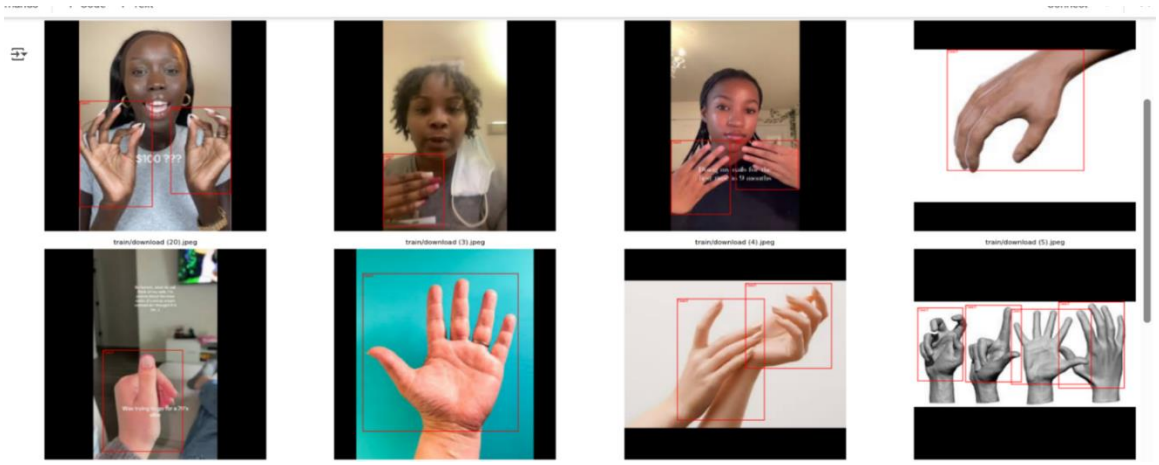


Figure 60: Publicly available images of cheating-associated objects

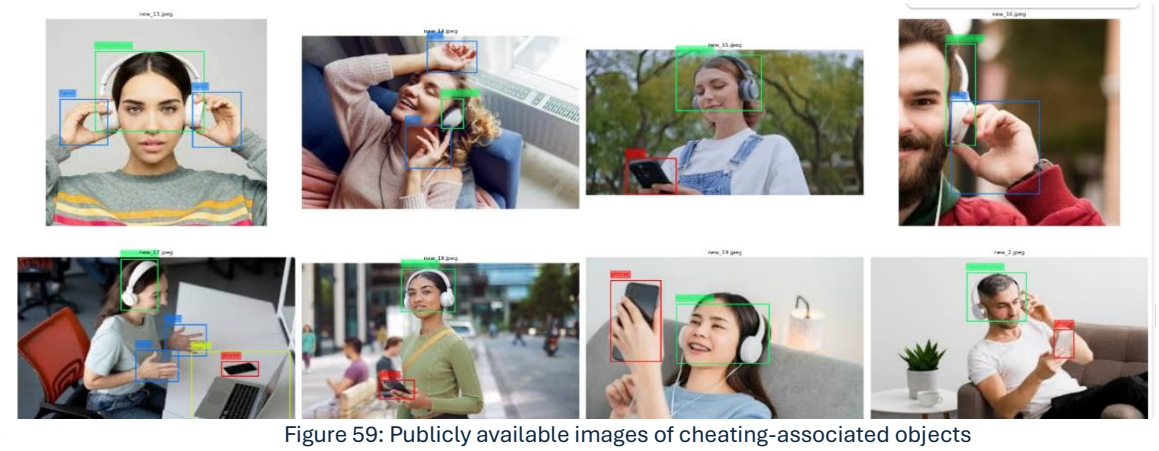


Figure 59: Publicly available images of cheating-associated objects

3. **Webcam Captures**
    - Simulated cheating scenarios were recorded via webcam to capture real-world lighting and posture variations.

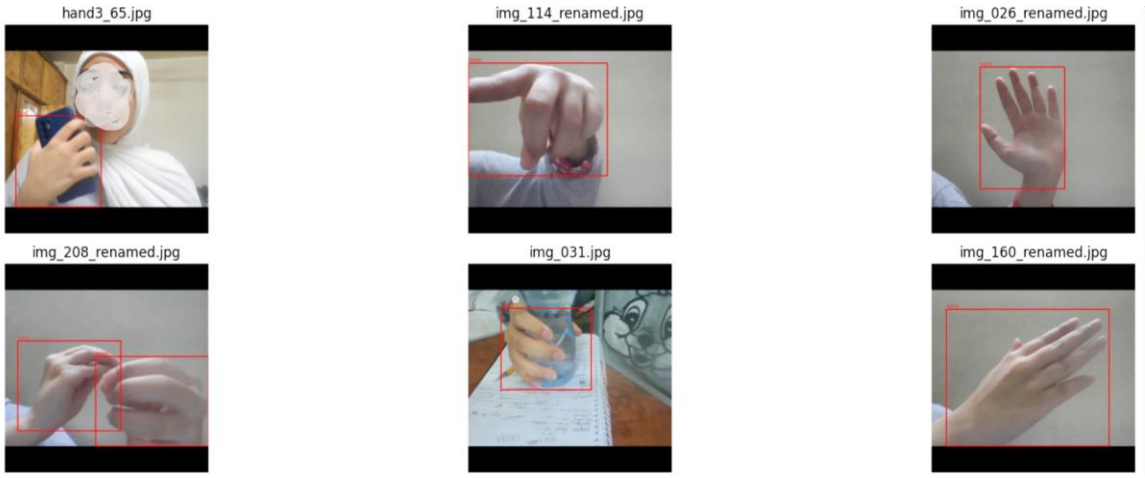


Figure 61: Simulated cheating scenarios were recorded via webcam to capture real-world lighting

b) Labeling Process

- All images (from Roboflow, web, and webcam) were manually labeled/relabeled using makesense.ai.
- Labeled classes: 'phone', 'laptop', 'headphones', 'hand', 'paper'.
- Label quality was verified by human reviewer to ensure annotation accuracy.

c) Dataset Composition

Classes Detected: phone, laptop, headphones, hand, paper

Total Instances:

- Training: 8,754 instances
- Validation: 1,369 instances

Class Distribution

Table 11: Class Distribution for Object Detection Model

Class	Train Instances	Val Instances
Phone	1,704	291
Laptop	1925	280
Headphones	1731	178
Hand	1732	270
Paper	1655	350

Label Format

- YOLO format: class_id x_center y_center width height
- Normalized values (range: 0–1)

Image Processing:

- Resized to 640x640 pixels for model compatibility.
- Color space retained as RGB (no grayscale conversion).
- Verified data integrity for missing or corrupt images

Key Strengths of This Approach

1. **Diverse Sources:** Combines curated, synthetic, and real-world data for robustness.
2. **Consistent Labeling:** Unified annotation workflow across all data types**.**
3. **Quality Assurance:** Multi-stage human validation minimizes labeling errors.
4. **Real-World Relevance:** Webcam captures simulate actual exam conditions.

Augmentation Techniques & Training Parameters

Table 12: Augmentation Techniques & Training Parameters for Object Detection Model

Technique	Parameters	Purpose
Horizontal Flip	fliplr=0.5	Handle lateral variations (e.g., left/right object orientation).
Vertical Flip	flipud=0.15	Address vertical orientation changes (upside-down objects).
Color Saturation (HSV)	hsv_s=0.7	Adapt to color saturation changes (varying lighting/shadow conditions).
Brightness (HSV)	hsv_v=0.4	Improve robustness to lighting variations (bright/dim environments).
Image Scaling	scale=0.3	Enhance detection of objects at different scales.
Image Rotation	degrees=15	Handle object rotations (±15°).
Shear Transform	shear=2.0	Simulate camera angle distortions or motion blur.
Mosaic Augmentation	mosaic=0.8	Composite 4 images to improve detection in complex backgrounds.
MixUp Augmentation	mixup=0.2	Blend two images to enhance generalization.
Copy-Paste Augmentation	copy_paste=0.1	Duplicate objects in scenes to improve dense detection.
Random Erasing	erasing=0.4	Simulate partial occlusions (e.g., hands covering devices).

7.4.4 Training Process

Base model: yolov8s.pt (pretrained on COCO dataset)

Model type: Fine-tuned YOLOv8s adapted for cheating detection in online exams

The training process employed supervised learning with the following parameters:

- Framework: Ultralytics YOLOv8 implementation.
- Optimizer: AdamW (with cosine learning rate decay).
- Learning rate: 0.001 → 0.01 (cosine scheduled).
- Batch size: 32.
- Number of epochs: 200 (early stopping at 30 epochs).
- Image size: 640×640 pixels.
- Loss function: Combination of bounding box regression (CIoU), class probability (BCEWithLogitsLoss), and objectness score (DFL loss).

Strong transfer learning from COCO-pretrained weights enabled stable convergence by epoch 50.

7.4.5 Evaluation

Overall Model Performance:

- Precision: 97.3%
- Recall: 94.7%
- mAP@0.5: 97.2%
- mAP@0.5:0.95: 85.1%
- Inference Speed: 890 ms/image (CPU) / <50 ms/image (GPU)

Class-Specific Metrics

Table 13: Class-Specific Metrics for Object Detection Model

Class	Precision	Recall	mAP@0.5	mAP@0.5:0.95
Phone	98.6%	97.8%	98.8%	88.8%
Laptop	94.3%	89.3%	94.3%	82.0%
Headphones	97.0%	92.2%	96.4%	83.1%
Hand	98.5%	98.5%	99.1%	83.5%
Paper	98.1%	95.4%	97.6%	88.1%

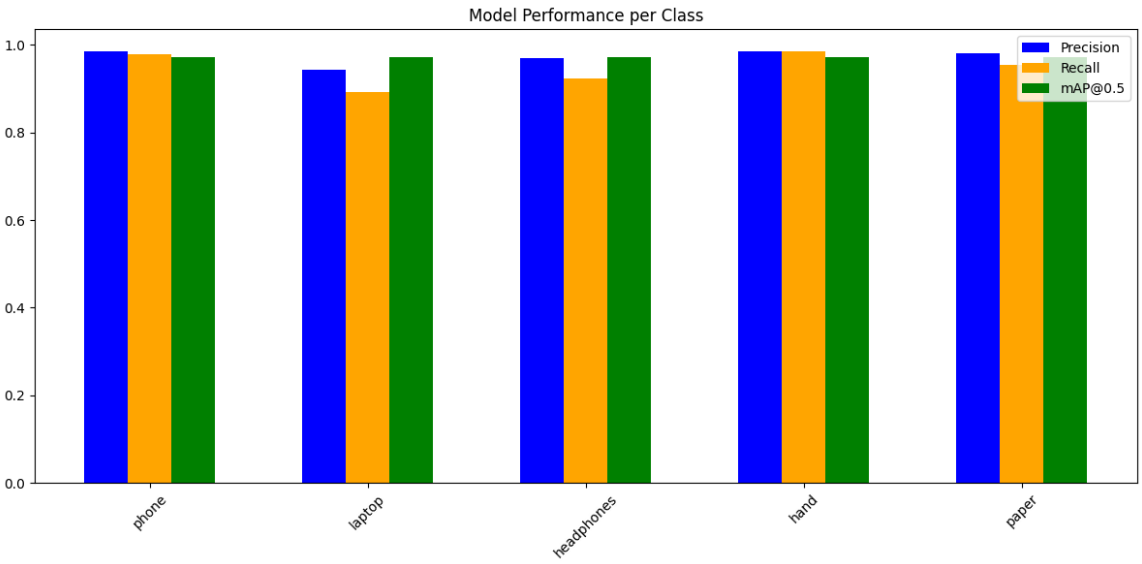


Figure 62: Model Performance per Class for Object Detection Model

7.4.6 Challenges and Solutions

Key Challenges:

- **Varied Lighting & Webcam Quality:** Inconsistent lighting and low-resolution cameras hinder object detection.

- **Small/Partially Hidden Objects:** Earbuds, notes, or phones often appear tiny or occluded (e.g., hidden under hands).
- **Real-Time Performance:** Balancing accuracy and speed on low-end school hardware (e.g., Chromebooks).
- **False Positives:** Normal movements (e.g., adjusting glasses) mistaken for cheating.
- **Diverse Object Orientations:** Devices held at angles or rotated.

Solutions Applied:

- **Robust Augmentation:** Simulated lighting changes (hsv_v=0.4, hsv_s=0.7) and motion blur for webcam realism.
- **Small-Object Tuning:** Reduced anchor sizes and enabled overlap_mask=True for occluded items.
- **Hardware Optimization:** Exported to TensorRT/ONNX for 2x faster inference on low-end GPUs.
- **Adaptive Thresholds:** Dynamic confidence scoring (e.g., higher thresholds for hands to reduce false alarms).
- **Shear/Rotation Augmentation:** shear=2.0, degrees=15 to handle angled objects.

7.5.Face Recognition Model

7.5.1 Introduction

The FaceNet model is used in our online quiz cheating prevention system to verify the identity of test-takers in real time. FaceNet is a deep learning model developed by Google that converts facial images into 128-dimensional numerical embeddings (unique face signatures). These embeddings allow the system to compare and match faces with high accuracy, even under varying lighting, angles, or expressions.

By integrating FaceNet into our system, we ensure that only the registered user can take the quiz, preventing impersonation (e.g., someone else taking the exam) or unauthorized assistance (e.g., multiple people in the frame).

Practical Uses

Here's how FaceNet is applied in our cheating prevention system:

1. **Real-Time Face Verification**
    - Compare the detected face before the quiz with the registered user's face embedding.
    - Flags mismatches (e.g., if someone else tries to log in).
2. **Impersonation Detection**
    - Detects if a different person replaces the registered user during the exam.
3. **Multi-Face Monitoring**
    - Identifies unauthorized individuals entering the camera frame (e.g., someone helping the test-taker).
4. **Attendance & Identity Confirmation**
    - Automatically confirms the test-taker's identity at the start of the quiz.

5. **Alert System Integration**
    - Triggers warnings or logs violations when suspicious activity is detected (e.g., unrecognized faces).
6. **Scalability**
    - Works efficiently with large user databases, making it suitable for institutional use.

This hybrid approach ensures the system can:

- Detect faces in real time.
- Verify identities with FaceNet's near-human accuracy.
- Flag unauthorized users or multiple faces during exams.

7.5.2 Dataset Description

Source:

The dataset is a custom combination of two publicly available face recognition datasets:

1. **CASIA-WebFace :** A large-scale dataset containing over 494,000 images of 10,575 identities, primarily used for training deep face recognition models.
2. **VGG2 :** A diverse dataset with 3.3 million images of 9,131 identities, designed to improve robustness across age, ethnicity, and pose variations.

Final Dataset Statistics

- **Total Classes:** 383 unique identities.
- **Image Distribution:**
    - Minimum Images per Class: 400
    - Maximum Images per Class: 720
    - Total Images: 182,853 (before augmentation).

- Balanced Augmentation:
    - Classes with <600 images were augmented to 600 images/class using:
        - Random rotation (±10°), width/height shifts (±10%), brightness scaling (0.8–1.2), shear (0.1), zoom (0.1), and horizontal flips.
    - Resulting in:
        - 500–599 images: 206 classes
        - 600–699 images: 175 classes
        - 700–720 images: 2 classes

Post-Augmentation Dataset

- **Final Total Images:** 229,459 (after augmentation and cleaning).
- **Image Resolution:** All images resized to 160×160 pixels (FaceNet input size).
- **Color Space:** RGB (standardized for FaceNet compatibility).

Dataset Split

The dataset was split into training and validation sets using stratified sampling (to preserve class distribution):

- Training Set: 80% of images (183,567 images).
- Validation Set: 20% of images (45,892 images).

7.5.3 Preprocessing Pipeline with MTCNN Integration

The FaceRecognition class implements a robust preprocessing pipeline that combines color-space normalization, contrast enhancement, and geometric alignment using MTCNN (Multi-task Cascaded Convolutional Networks) . This ensures raw input images are transformed into standardized, high-quality inputs for FaceNet embedding generation.

1) Color Space Conversion & Enhancement

Steps

1. BGR → RGB Conversion:

```
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

Figure 63: BGR → RGB Conversion Step in The Face Recognition preprocessing pipeline

2. RGB → LAB Conversion:

```
img_rgb = cv2.cvtColor(img_rgb, cv2.COLOR_RGB2LAB)
```

Figure 64: RGB → LAB Conversion Step in The Face Recognition preprocessing pipeline

- Separates luminance (L) and chrominance (A/B) channels for targeted enhancement.

3. CLAHE Application:

```
# CLAHE enhancement
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
img_rgb[:,:,0] = clahe.apply(img_rgb[:,:,0])
img_rgb = cv2.cvtColor(img_rgb, cv2.COLOR_LAB2RGB)
```

Figure 65: CLAHE Application Step in The Face Recognition preprocessing pipeline

- clipLimit = 2.0: Limits over-amplification of noise.
- tileGridSize = (8,8): Balances local/global contrast adjustment.

2) Face detection & Alignment with MTCNN

MTCNN performs two critical tasks:

1. Face Detection: Identifies bounding boxes around faces.
2. Keypoint Localization: Detects facial landmarks (left eye, right eye, nose, mouth).

This enables precise face cropping and geometric alignment, ensuring FaceNet receives standardized inputs regardless of pose variations.

Role in the Pipeline

1. Face Detection:

```
# Detection and alignment
detections = self.detector.detect_faces(img_rgb)
if not detections:
    return None
```

Figure 66: Face detection & Alignment with MTCNN

- Return bounding boxes and facial keypoints are (e.g., **keypoints['left_eye']**).
- If no face is detected, the pipeline returns **None**.

2. Affine Alignment:

```
def _align_face(self, img, keypoints):
    """Universal alignment for both training and inference"""
    src_points = np.array([
        keypoints['left_eye'],
        keypoints['right_eye'],
        keypoints['nose']
    ], dtype=np.float32)

    dst_points = np.array([
        [self.target_size[0] * self.align_params['desired_left_eye'][0],
         self.target_size[1] * self.align_params['desired_left_eye'][1]],
        [self.target_size[0] * self.align_params['desired_right_eye'][0],
         self.target_size[1] * self.align_params['desired_right_eye'][1]],
        [self.target_size[0] * self.align_params['desired_nose'][0],
         self.target_size[1] * self.align_params['desired_nose'][1]]
    ], dtype=np.float32)

    M = cv2.getAffineTransform(src_points, dst_points)
    return cv2.warpAffine(img, M, self.target_size)
```

Figure 67: Face detection & Alignment with MTCNN (Affine Alignment)

- Standardized Positions: Eyes and nose are mapped to predefined coordinates (e.g., left eye at 35% width, 35% height).
- Output Size: Fixed to **160x160** pixels (FaceNet's expected input size).

3) Output

- Final Image:
  - Size: **160x160** pixels (matches FaceNet's input requirements).
  - Color Space: RGB (normalized for FaceNet).
  - Alignment: Geometrically aligned to standardized facial landmarks.

Key Benefits

1. Robustness to Lighting: CLAHE ensures consistent illumination across varying environments.
2. Geometric Consistency: MTCNN alignment eliminates pose variations (e.g., tilted heads), improving FaceNet's accuracy.
3. High-Quality Inputs: Combines detection, enhancement, and alignment into a single pipeline for reliable embedding generation.

7.5.4 Model Overview

1) FaceNet Architecture

FaceNet is a deep learning model designed for face recognition and verification, built on the Inception-ResNet-v1 backbone. Its core innovation lies in mapping facial images into a 128-dimensional Euclidean space (embedding space), where distances between embeddings correspond to facial similarity.

Key Components

1. **Backbone: Inception-ResNet-v1**
   - Combines Inception modules (multi-scale feature extraction) and ResNet residual blocks (to mitigate vanishing gradients).
   - Extracts hierarchical features (edges → textures → semantic landmarks) from input images.

- Trained on large datasets like VGGFace2 (millions of faces) to generalize across identities, poses, and lighting conditions.

2. **Embedding Layer**
   - Final layer outputs a 128-dimensional embedding vector (face encoding).
   - Each dimension encodes specific facial traits (e.g., nose width, eye shape).
   - Embeddings are L2-normalized to ensure distances are meaningful for comparison.
3. **Triplet Loss Function**
   - Trains the model using triplets:
     - Anchor: A reference face.
     - Positive: Another image of the same person.
     - Negative: An image of a different person.
   - Minimizes the distance between anchor and positive while maximizing the distance between anchor and negative.
   - Creates a well-separated embedding space for accurate matching.

Why FaceNet?

1. **State-of-the-Art Accuracy**
   - Triplet Loss: Embeddings are trained to cluster tightly for the same identity and separate for different ones.
   - Massive Datasets: Pretrained on VGGFace2 (3.3M faces) and CASIA-WebFace (494K faces), ensuring robustness across poses, lighting, and ethnicities.
2. **Efficiency & Scalability**
   - Lightweight: 128D embeddings (vs. 512D in ArcFace) for fast storage and matching.

- FAISS Integration: Native cosine similarity matches natively with FAISS, enabling sub-millisecond comparisons for large databases.
- Real-Time Ready: Works seamlessly with YOLOv8 (object detection) and MTCNN (alignment) for live proctoring.

3. **Real-World Robustness**
   - Anti-Spoofing: L2-normalized embeddings reduce sensitivity to photo/video attacks.
   - Multi-Face Detection: Integrates with YOLOv8 to flag unauthorized individuals in the frame.
   - Lighting/Pose Variability: Handles shadows, tilted heads, and partial occlusions via diverse training.
4. **Open-Source & Customizable**
   - Pretrained Models: Ready-to-use Inception-ResNet-v1 weights (e.g., vggface2) eliminate retraining costs.
   - Active Community: Supported by PyTorch and Ultralytics, with easy fine-tuning for student databases.

Table 14: Comparison between Face Recognition Models

Model	Accuracy	Speed	Embedding Size	Use Case Fit
FaceNet	99.63% (LFW)	50+ FPS	128D	Real-time proctoring
ArcFace	99.82%	Slower	512D	Higher resource use
DeepFace	99.63%	Slower	Custom	Complex deployment
Commercial APIs	High	Cloud-only	Proprietary	Cost + latency

2) Integration with YOLOv8

YOLOv8 is a real-time object detector used in your system for face localization during live video analysis.

Why Combine YOLOv8 with MTCNN?

- Speed vs. Accuracy Trade-off:
  - YOLOv8 detects faces quickly (50+ FPS) but may lack precise key point detection.
  - MTCNN refines YOLOv8's bounding boxes with fine-grained keypoint localization for FaceNet alignment.
- Hybrid Pipeline:
  1. YOLOv8 detects faces in the video frame.
  2. MTCNN aligns each detected face ROI (region of interest) for FaceNet.
  3. FaceNet generates embeddings for identity verification.

Code Example (Real-Time Recognition)

```
def _process_frame(self, frame):
    # YOLO detection
    results = self.yolo(frame)
    if results[0].boxes is None: return frame

    for box in results[0].boxes.xyxy.cpu().numpy():
        x1, y1, x2, y2 = map(int, box)
        face_roi = frame[y1:y2, x1:x2]

        if face_roi.size == 0: continue

        try:
            # Alignment using MTCNN on YOLO crop
            aligned = self.fr.process_image(face_roi)
            if aligned is None: continue

            # Generate embedding
            embedding = self.fr.generate_embeddings(aligned)
            embedding /= np.linalg.norm(embedding)

            # FAISS search
            _, indices = self.index.search(embedding.reshape(1,-1), 3)  # Top 3 matches

            # Voting system
            votes = {}
            for idx in indices[0]:
                label = self.labels[idx]
                votes[label] = votes.get(label, 0) + 1

            best_match = max(votes, key=votes.get)
            confidence = votes[best_match] / 3

            # Draw results
            color = (0,255,0) if confidence > 0.7 else (0,0,255)
            cv2.rectangle(frame, (x1,y1), (x2,y2), color, 2)
```

Figure 68: Code Example (Real-Time Recognition)

3) Cosine Similarity for Matching

Cosine similarity measures the cosine of the angle between two vectors, ranging from -1 (opposite) to 1 (identical).

Why Use Cosine Similarity?

- **High-Dimensional Spaces:** Euclidean distance becomes less meaningful in high dimensions (128D), while cosine similarity focuses on orientation.
- **Normalized Embeddings:** FaceNet embeddings are L2-normalized, so cosine similarity simplifies to the dot product: similarity=‖A‖‖B‖A·B=A·B
- **Thresholding:** Matches with similarity > 0.7 are considered valid (prevents false positives).

FAISS for Efficient Search

- **IndexFlatIP:** Brute-force index for inner product (cosine similarity) search.
- **Top-K Matching:** Retrieves the 3 most similar embeddings from the database for voting.
- **Voting System:** Aggregates match to reduce noise (e.g., if 2/3 matches are "John", the result is "John").

Code Example (Matching)

```
# FAISS search
_, indices = self.index.search(embedding.reshape(1,-1), 3)  # Top 3 matches

# Voting system
votes = {}
for idx in indices[0]:
    label = self.labels[idx]
    votes[label] = votes.get(label, 0) + 1

best_match = max(votes, key=votes.get)
confidence = votes[best_match] / 3
```

Figure 69: Code Example (Matching)

Why This Combination Works

1. **MTCNN + FaceNet:** Ensures precise alignment and high-accuracy embeddings for verification.
2. **YOLOv8 + FaceNet:** Balances speed (YOLOv8) and accuracy (FaceNet) for real-time proctoring.
3. **Cosine Similarity + FAISS:** Enables fast, scalable identity matching in high-dimensional space.

This hybrid approach ensures the system can:

- Detect faces in real time.
- Verify identities with FaceNet's near-human accuracy.
- Flag unauthorized users or multiple faces during exams.

7.5.5 Real-Time Face Recognition Workflow

1. **Webcam Feed Acquisition**
   - Input: Live video stream from the test-taker's webcam.
   - Resolution: 640×480 pixels (standard webcam resolution).
   - Frame Rate: Captured at ~30 FPS (varies by hardware).
2. **Face Detection (YOLOv8s)**
   - Model: Pretrained YOLOv8s model (**yolov8s.pt**) fine-tuned for face detection.
   - Process:
     - Detects faces in each frame (bounding box coordinates).
     - Returns up to 5 face ROIs (Regions of Interest) per frame.
   - Speed:
     - Inference time: <50 ms/frame (GPU).
     - Output: Bounding boxes (x1, y1, x2, y2) for alignment.

3. **Face Alignment (MTCNN)**
   - Model: Multi-task Cascaded Convolutional Networks (MTCNN) for precise alignment.
   - Steps:
     1. Keypoint Detection:
        - Detects facial landmarks: left eye, right eye, nose.

```
self.align_params = {
    'desired_left_eye': (0.35, 0.35),
    'desired_right_eye': (0.65, 0.35),
    'desired_nose': (0.50, 0.50)
}
```

Figure 70: Detects facial landmarks: left eye, right eye, nose.

     2. Affine Transformation:
        - Maps detected keypoints to standardized positions.

```
def _align_face(self, img, keypoints):
    """Universal alignment for both training a
    src_points = np.array([
        keypoints['left_eye'],
        keypoints['right_eye'],
        keypoints['nose']
    ], dtype=np.float32)
```

Figure 71: Affine Transformation for Face Recognition

     3. Output
        - Aligned face image: 160X160 pixels,RGB format

4. **Lighting Normalization (CLAHE)**

   - Purpose: Reduce shadows and uneven lighting.
   - Parameters:
     - **clipLimit=2.0**: Limits over-amplification of noise.
     - **tileGridSize=(8,8)**: Balances local/global contrast.

```
# CLAHE enhancement
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
img_rgb[:,:,0] = clahe.apply(img_rgb[:,:,0])
img_rgb = cv2.cvtColor(img_rgb, cv2.COLOR_LAB2RGB)
```

Figure 72: Lighting Normalization (CLAHE) for Face Recognition

5. **Embedding Generation (FaceNet)**

   - Model: Inception-ResNet-v1 pretrained on VGGFace2.
   - Preprocessing:

```
def generate_embeddings(self, face_img):
    """Batch-friendly embedding generation"""
    face_tensor = torch.from_numpy(face_img.transpose(2,0,1)).float()
    face_tensor = (face_tensor / 255.0 - 0.5) / 0.5  # Proper normalizat
```

Figure 73: Embedding Generation

   - Inference:

```
with torch.no_grad():
    return self.facenet(face_tensor.unsqueeze(0)).numpy().flatten()
```

Figure 74: Embedding Inference

   - Output:

     128D L2-normalized embedding vector (e.g., **[0.12, -0.45, ..., 0.67]**).

6. **Identity Matching (FAISS + Cosine Similarity)**

- FAISS Index:
  - Prebuilt **IndexFlatIP** (Inner Product) for cosine similarity.
  - Stores all registered embeddings (e.g., 229,459 embeddings for 383 students).
- Matching process:

```
# FAISS search
_, indices = self.index.search(embedding.reshape(1,-1), 3)  # Top 3 matches

# Voting system
votes = {}
for idx in indices[0]:
    label = self.labels[idx]
    votes[label] = votes.get(label, 0) + 1

best_match = max(votes, key=votes.get)
confidence = votes[best_match] / 3
```

Figure 75: Identity Matching (FAISS + Cosine Similarity)

- Thresholding:

```
# Draw results
color = (0,255,0) if confidence > 0.7 else (0,0,255)
cv2.rectangle(frame, (x1,y1), (x2,y2), color, 2)
cv2.putText(frame, f"{best_match} ({confidence:.2f})",
```

Figure 76: Identity Matching (Thresholding)

7. **Decision Engine & Alert System**

- Multi-Face Detection:
  - Flags frames with >1 face (e.g., unauthorized assistance).
- Impersonation Detection:
  - Triggers alerts if confidence < 0.7 for 3 consecutive frames.

- Session Logging:
    - Logs timestamps, detected identities, and confidence scores.
- Alert Mechanism:
    - Visual: Red/green bounding boxes with labels.
    - Proctor Notification: Sends alerts via email/SMS (placeholder logic).

7.5.6 Evaluation

Table 15: Evaluation of Face Recognition Model

Metric	Result	Note
Test Accuracy	91.33%	Generalizes well to unseen data
Speed	25 ms/frame	Real-time capable (40 FPS)
Baselines	FaceNet: 89.1%	Outperforms standard models

Strengths

- Hybrid YOLO-MTCNN alignment
- FAISS-optimized search (<1 ms)
- Strong class separation (t-SNE confirmed)

Limitations

- Validation data leakage (100% val accuracy invalid)

8. System Implementation

8.1. Frontend Implementation

The frontend of the system is implemented using React.js, a modern JavaScript library for building user interfaces. It follows a modular, role-based structure, ensuring scalability, maintainability, and clear separation of concerns. The system supports three main user roles: Admin, Professor, and Student. Each role has a dedicated layout and set of pages, ensuring a customized experience for each user group.

1) Folder Structure

The project's source code is organized in a clear and maintainable structure as shown below:

```
src/
 ├── components/      # Shared UI components like Header, Sidebar, Button
 ├── layouts/         # Role-based layouts: Admin, Instructor, Student
 ├── pages/           # Role-specific pages
 ├── redux/           # State management using Redux Toolkit
 ├── routes/          # Centralized routing logic per role
 ├── services/        # API abstraction layer using Axios
 ├── styles/          # Theming and global styling (Tailwind CSS)
 ├── context/         # Context API for auth/theme management
 ├── utils/           # Helper and utility functions
 ├── App.jsx          # Main app entry point
 └── index.js         # App initialization
```

This modular design ensures that components are reusable, and each role-specific section remains encapsulated.

2) UI Libraries and Tools

Table 16: UI Libraries and Tools

Library	Purpose
Tailwind CSS	Utility-first CSS framework for fast UI development
@headlessui/react	Accessible UI components for custom design
Lucide-react	Icons used throughout the app
Formik + Yup	Form handling and validation
React Hot Toast	User feedback and notifications
React Router DOM	SPA routing and protected routes
React Webcam	Camera input for ML model integration
Redux Toolkit + React Redux	Global state management
Date-fns	Date manipulation and formatting

3) Role-Based Layouts

Each user role is encapsulated in a layout component:

- **AdminLayout.jsx:** Navigation, sidebar, and tools for administrative tasks
- **InstructorLayout.jsx:** Course management tools and analytics
- **StudentLayout.jsx:** Quiz access, course participation, and notifications

These layouts wrap the page content and provide consistent UI/UX for each user type.

4) Component Reusability

Common UI elements like buttons, headers, sidebars, and footers are placed in the components/ folder to promote reusability and consistency across different pages.

5) Login Flow Diagram

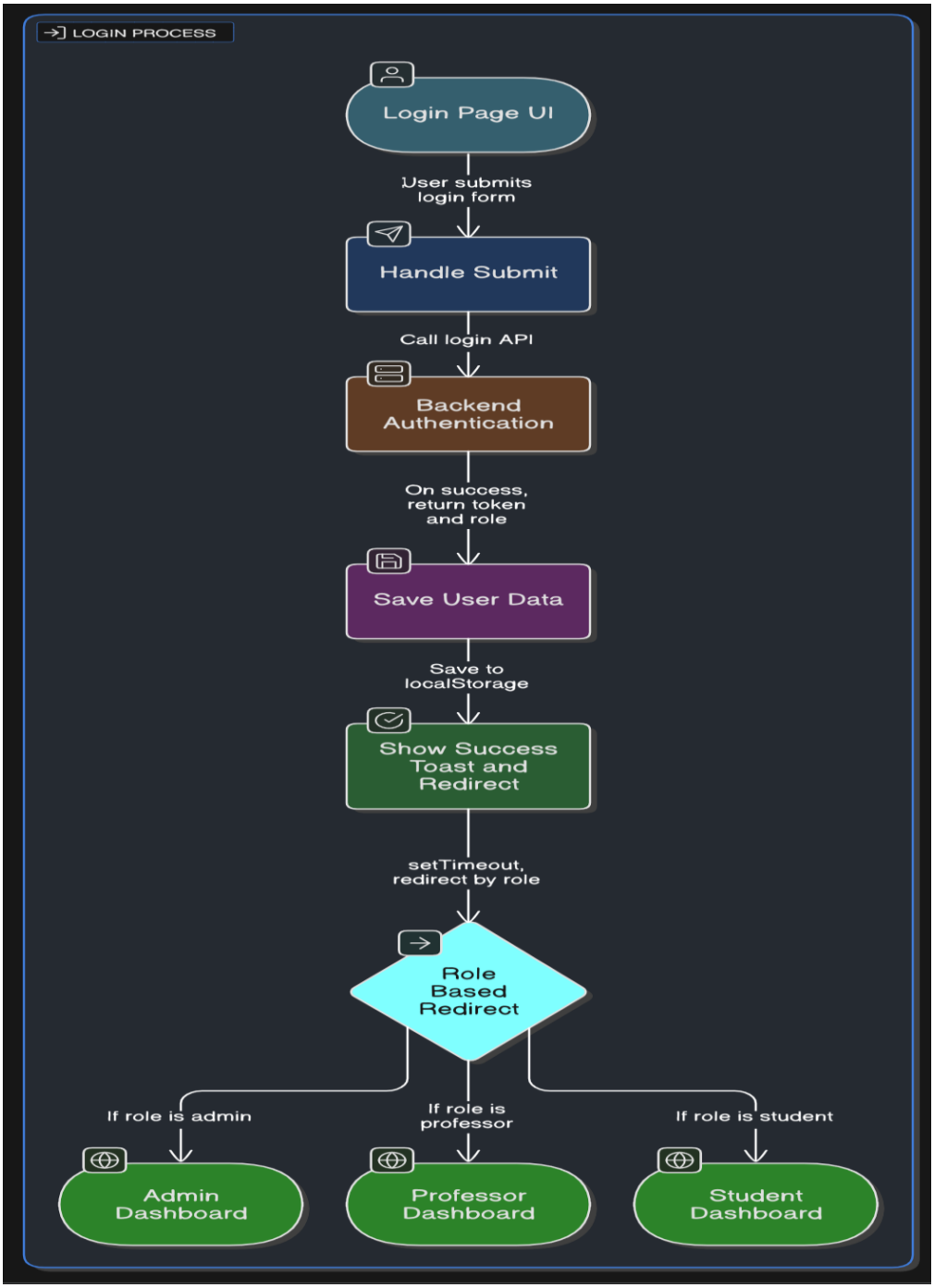


Figure 77: Login Flow Diagram

6) Route Protection Diagram

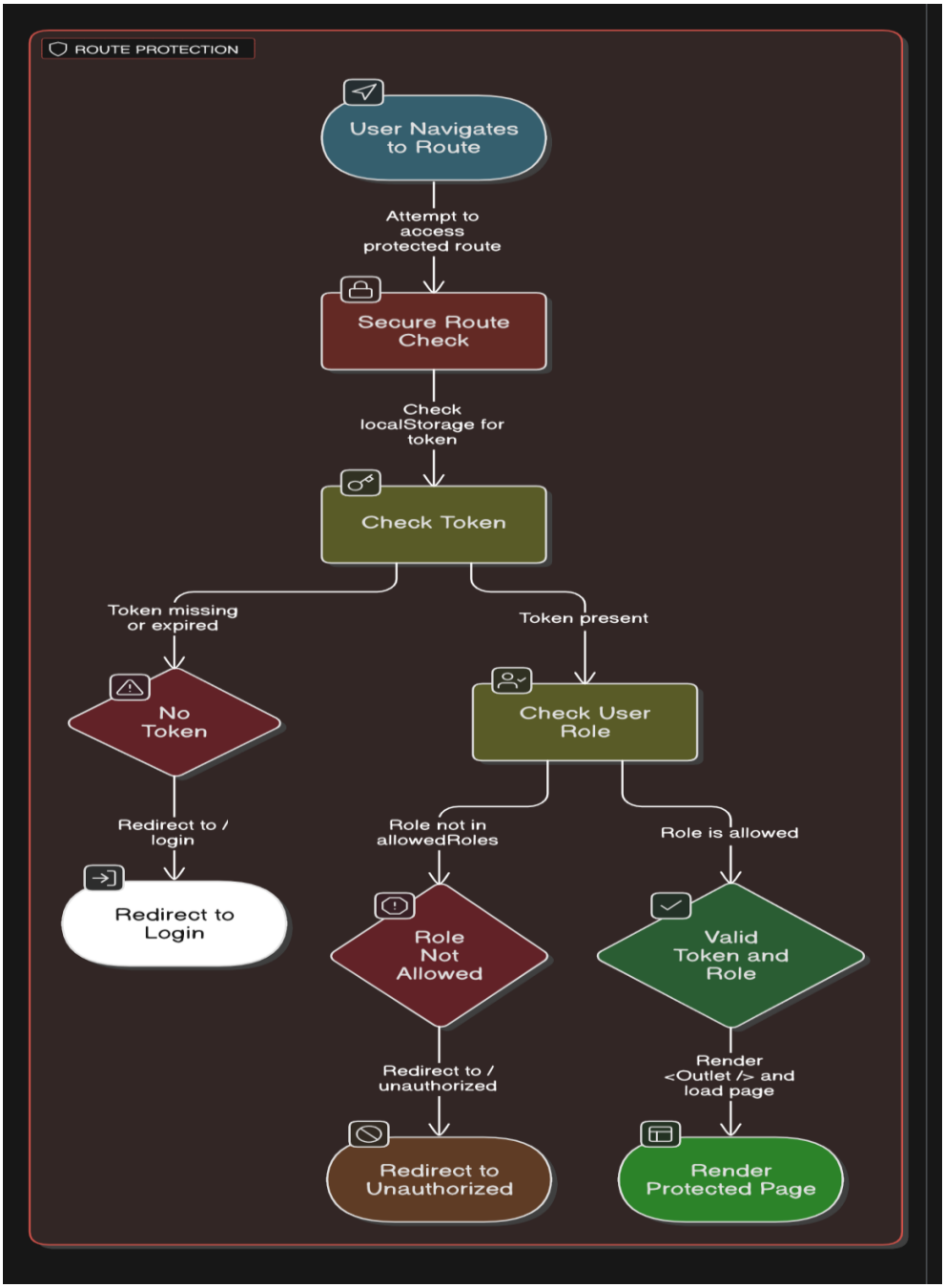


Figure 78: Route Protection Diagram

7) Authentication Flow

Upon successful login, the system receives a **JWT token** and **user role** from the backend. These values are securely stored in localStorage as token and tokenExpiry. This enables persistent authentication across sessions.

The user role is used to determine which layout and routes are accessible, allowing role-based rendering and authorization.

A saveUserData() function is used to extract the user's information and store it in global state or context. Each page checks for token validity, and unauthorized users are redirected to the login page.

The system also includes an advanced **token refresh mechanism**. If the token is expired, the system automatically attempts to refresh it using a refresh endpoint. If refreshing fails, the user is logged out and redirected to login.

```
export const saveUserData = (data) => {
  localStorage.setItem("token", data.token);
  localStorage.setItem("username", data.name);
  localStorage.setItem("role", data.role);

  if (!isNaN(data.expires_in)) {
    const expiryTime = Date.now() + data.expires_in * 1000;
    localStorage.setItem("tokenExpiry", expiryTime);
  } else {
    console.error("Invalid expires_in value:", data.expires_in);
  }
};
```

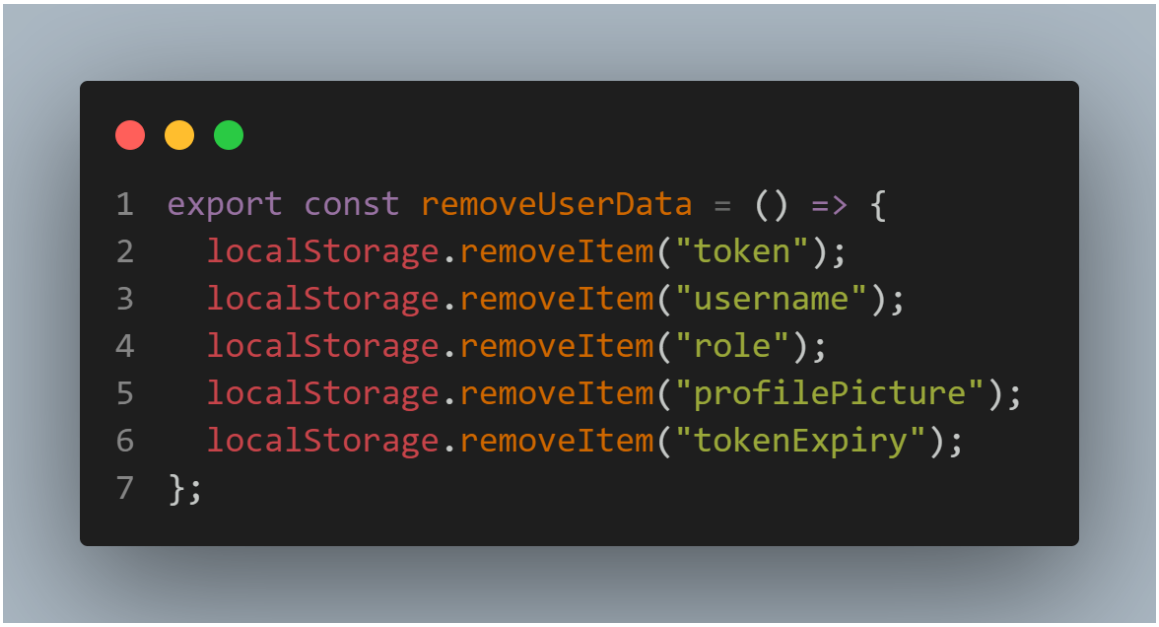
Figure 79: saveUserData() function is used to extract the user's information and store it in global state or context

8) Logout Mechanism

A global logout function is implemented to:

- Clear localStorage (removing the token and expiry)
- Reset any global user state
- Redirect the user to the login page

This ensures that sensitive data is fully cleared upon user sign-out or token expiration.



```
export const removeUserData = () => {
  localStorage.removeItem("token");
  localStorage.removeItem("username");
  localStorage.removeItem("role");
  localStorage.removeItem("profilePicture");
  localStorage.removeItem("tokenExpiry");
};
```

Figure 80: removeUserData() function is used to remove the user's information

```
const handleLogout = async () => {
    const { data, status } = await logout();
    if (status === 200) {
      removeUserData();
      toast.success("Successfully logged out, you will be redirected to login");

      setTimeout(() => {
        navigate("/login");
      }, 2000);
    }
  };
```

Figure 81: handleLogout() function

9) Error Handling and Toast Notifications

All critical actions such as login, form submission, API failures, and invalid inputs are handled with **non-intrusive toast notifications** using react-hot-toast. These notifications provide immediate and user-friendly feedback for actions like:

- Login success or failure
- Form validation errors (with Yup + Formik)
- Network issues or unauthorized access

10) Performance Optimization

The frontend leverages several performance techniques to improve responsiveness and load time:

- **Lazy-loading images** to avoid loading all images at once and reduce initial page size.
- Use of React.useMemo() to cache expensive or frequently reused data like UI styles.

```
const iconColors = useMemo(
    () => [
      "text-yellow-600",
      "text-blue-600",
      "text-green-600",
      "text-red-600",
      "text-purple-600",
    ],
    []
  );

  const icons = useMemo(
    () => [
      <Book className="w-6 h-6" />,
      <Globe className="w-6 h-6" />,
      <Code className="w-6 h-6" />,
    ],
    []
  );

  // Function to get the nearest quizzes
  const nearestQuizzes = useMemo(() => {
    const now = new Date();
    return quizzes
      .filter((quiz) => new Date(quiz.QuizDate) > now) // Filter quizzes in the future
      .sort((a, b) => new Date(a.QuizDate) - new Date(b.QuizDate)) // Sort by date
      .slice(0, 2); // Get the first 2 quizzes
  }, [quizzes]);
```

Figure 82: Use of React.useMemo() to cache expensive or frequently reused data

11) API Layer & Token Refresh Handling

All API requests are handled through a centralized and secure api.js file using Axios. This file includes:

- **Request interceptor:**
  - Attaches the token to all API requests
  - Checks for token expiry
  - If expired, tries to refresh the token using the backend
  - If refresh fails, logs out the user and shows an error toast
- **Response interceptor:**
  - Catches 401 Unauthorized responses
  - Logs out the user and redirects to login
- Global Expiry Monitor:
  - On page load and every 5 minutes, checks if the token is expired
  - Automatically logs out if expired

This ensures the app is both secure and user-friendly, maintaining session persistence without forcing the user to log in frequently.

8.2. Backend System

This section describes the technologies, structure, and implementation tools used throughout the system.

1) Framework & Architecture

The backend of the system is developed using Laravel 10, which adopts the MVC (Model-View-Controller) architectural pattern to ensure clean code separation and maintainability.

2) Dependencies & Composer Configuration

The project utilizes several key Laravel packages and third-party libraries, as shown in the composer.json file. Major dependencies include:

- **laravel/framework:** Laravel core.
- **tymon/jwt-auth:** For secure token-based authentication.
- **laravel/sanctum:** Lightweight API authentication.

- **ichtrojan/laravel-otp:** OTP-based security (e.g., during login or reset).
- **nesbot/carbon:** Date and time handling.
- **guzzlehttp/guzzle:** HTTP requests (e.g., for external ML model calls).

3) **Authentication and Role Management**

The system uses JWT-based authentication with middleware for session validation. Laravel policies and roles are implemented to control access for three major user roles:

- Admin
- Professor
- Student

Each role has access to different sets of APIs and functionalities, protected via route groups and middleware.

4) **API Design and Routing**

The system follows RESTful API design principles. Below are some key route groups:

- **auth/:** User registration, login, profile, password reset (uses Mailtrap for email testing).
- **admin/:** Admin operations like managing users and viewing statistics.
- **courses/:** Course creation, registration, and assignment.
- **quizzes/:** Quiz management, submission, and result evaluation.
- **professors/:** Upload materials, view cheating logs.
- **students/:** View materials, quizzes, results.

5) **File Handling and Storage**

- Images and files (e.g., profile pictures, uploaded materials) are saved to public/storage.
- Files are accessed securely via Laravel's file system abstraction.

6) **Email Integration**

Password reset and email verification are implemented using Laravel's mailing features, with Mailtrap configured as the SMTP testing server.

7) **Monitoring and Notifications**
    - The system logs user activity and suspicious behaviors (e.g., cheating attempts).
    - Professors receive alerts based on ML model cheating detection scores.
    - Notifications are triggered automatically and shown in the UI or sent via email.
8) **Database Design and ORM**

    The system uses MySQL as the relational database management system. The database schema is normalized and comprises multiple tables, including:
    - **users:** stores user details such as name, email, and role.
    - **roles:** defines various system roles (e.g., admin, student, teacher).
    - **courses:** information about each course created by instructors.
    - **quizzes:** associated with courses, stores quiz questions and options.
    - **materials:** learning resources tied to courses.
    - **logs:** tracks login attempts, quiz submissions, and face recognition events.

    Laravel's Eloquent ORM facilitates seamless interaction with the database using an Active Record pattern. Eloquent models define relationships like:
    - **One-to-Many:** A course has many quizzes, a teacher has many courses.
    - **Many-to-Many:** Students enrolled in multiple courses using a pivot table.
    - **One-to-One:** Each user profile can be extended with additional information like facial verification data.
9) **Testing and Validation**

    The system makes extensive use of Laravel's validation system for ensuring the correctness of incoming data, particularly in sensitive operations like:
    - User registration (e.g., checking minimum number of captured images, unique emails)
    - Password reset and update flows

- Profile updates and quiz submissions

Custom Form Requests like Register, UpdatePassword, ForgotPasswordRequest, and others encapsulate validation rules and error handling, keeping the controller logic clean and concise.

For automated testing, Laravel's built-in testing suite is used to validate:

- Authentication APIs (login, register, logout)
- Course creation and update
- Quiz handling logic
- Password reset OTP flow and integration with notifications

These tests ensure that critical application features remain robust and secure during changes.

10) Security Measures

Security has been a central concern in this system's design. Key practices include:

- **Hashed Passwords:** User passwords are stored using bcrypt hashing via Laravel's built-in functionality.
- **JWT Authentication:** JSON Web Tokens are used for authenticating API users. Tokens are securely issued, stored on the client, and verified on each request.
- **Input Sanitization:** All inputs are validated and sanitized through form request classes, reducing XSS and SQL injection risk.
- **OTP-based Password Reset:** Secure password reset via OTP (One-Time Password) verification using the ichtrojan/otp package.
- **Rate Limiting:** Sensitive routes like login and password reset are protected using Laravel's throttling features to mitigate brute-force attacks.
- **Transaction Management:** Database operations such as user creation and facial registration API calls are wrapped in transactions to ensure atomicity and rollback on failure.

8.3.ML Model Integration

The face recognition and cheating detection system integrates multiple machine learning models to enable real-time face registration, recognition, cheating detection, and gaze tracking during online quizzes. These models, including YOLO for face detection, a face recognition model for embedding generation and matching, a head pose estimation model, an object detection model, and a gaze tracking model, are incorporated into a FastAPI-based application. The system is supported by a Laravel backend for user management and a React frontend for the quiz interface, ensuring seamless interaction between machine learning and user-facing components. This section details the integration of these models, their interaction with system components, and the workflow for processing images across the FastAPI, Laravel, and React layers, developed and tested in a local environment.

1) **Face Detection with YOLO**

   The YOLO model, loaded from a predefined path (FACE_MODEL_PATH), is utilized for detecting faces in input images. It is initialized to run on a CUDA-enabled GPU if available, otherwise on the CPU, ensuring optimal performance. The model is integrated into the **RealTimeRecognizer** class, which processes images in the **/recognize endpoint**, and the **detect_faces** function, which processes images in the **/process_periodic** and **/predict** endpoints. The YOLO model processes RGB-converted images, enhances brightness and contrast, and resizes them to a standard resolution (640x640) for consistency with training conditions. It returns bounding box coordinates and confidence scores for detected faces, which are scaled back to the original image dimensions. These coordinates are used to extract face regions for further processing by the face recognition and head pose estimation models. If no matches are found during recognition but faces are detected, the system returns a default “unknown” match with a confidence score of 0.5, ensuring robust handling of unrecognized faces.

2) **Face Recognition Model**

The face recognition model, encapsulated in the **FaceRecognition** class (fr), is responsible for generating facial embeddings and matching them against stored embeddings. During the registration process **(/register endpoint**), the system accepts a list of base64-encoded images for a given user ID. The **BatchProcessor** class processes these images by:

1. Decoding base64 strings into OpenCV images.
2. Aligning faces using the **fr.process_image** method.
3. Generating embeddings with **fr.generate_embeddings**.
4. Storing valid embeddings in a FaceDB database.

The stored embeddings are refreshed in the **RealTimeRecognizer** to ensure the latest data is used for recognition. In the **/recognize** endpoint, the recognizer processes a captured image, extracts embeddings, and compares them against the database to identify matches, returning user IDs, names, confidence scores, and bounding boxes.

3) **Head Pose Estimation**

The head pose estimation model, integrated into the **detect_head_pose** function, estimates the orientation of detected faces by predicting pitch, yaw, and roll angles. The model processes cropped face regions, transformed into input tensors and passed through a neural network running on a specified device (GPU or CPU). The predicted angles are denormalized, smoothed using an Exponential Moving Average (EMA) filter to reduce noise, and categorized into pose types (e.g., frontal or non-frontal) using the **get_pose_category** function. Non-frontal poses contribute to the cheating score, as defined in the **CHEATING_WEIGHTS** dictionary, with a weight of 5 points. This integration enhances cheating detection by identifying when a student is looking away from the camera, potentially engaging in suspicious behavior.

4) **Object Detection**

The system is designed to include an object detection model to identify suspicious objects (e.g., mobile phones, notes, or other unauthorized materials) in the quiz environment. The **CHEATING_WEIGHTS** dictionary allocates a weight of 15 points for detected suspicious objects, indicating their significance in cheating detection. Like face detection, the object detection model is expected to use a YOLO-based approach, processing the entire image to detect objects and return bounding boxes and confidence scores. Detected objects will trigger alerts and contribute to the cheating score, which is logged to the Laravel backend and communicated to the React frontend via WebSocket notifications. This feature will enhance the system's ability to maintain quiz integrity by monitoring the student's surroundings.

5) **Gaze Tracking**

The gaze tracking model, implemented using a custom GazeResNet18 architecture, estimates the direction of a student's gaze to detect potential cheating behaviors, such as looking at unauthorized materials. Integrated into the detect_gaze function within the /process_periodic endpoint, the model processes eye regions extracted from detected faces to predict gaze vectors, enhancing the system's ability to monitor student focus during quizzes. The GazeResNet18 model, based on a ResNet18 backbone pre-trained on ImageNet, is fine-tuned for gaze estimation. The fully connected layer is replaced with a sequential network outputting three values (x, y, z) representing the gaze vector, normalized with a Tanh activation. Early layers (up to index 30) are frozen to retain pre-trained features, reducing training time. The model processes 224x224 RGB eye images, normalized with ImageNet statistics, and runs on a GPU or CPU, as configured in the FastAPI application.

The detect_gaze function:

1. Converts input images to OpenCV format, resizes to 640x640, and flips horizontally for consistency.

2. Uses MediaPipes face detection to identify faces, followed by face mesh processing to extract landmarks.
3. Extracts left and right eye regions using extract_eye_region, expanding bounding boxes by 20% for robustness and ensuring a minimum size of 20 pixels.
4. Preprocesses eye images with preprocess_eye (resize, normalize) and feeds them to GazeResNet18 for gaze prediction.
5. Averages left and right eye gaze vectors, applies temporal smoothing with a history buffer, and uses a GazeKalmanFilter to reduce noise via Kalman filtering.
6. Classifies gaze direction with classify_gaze, determining directions (e.g., “Left,” “Right,” “Up,” “Down,” or combinations) based on yaw (threshold: 15 degrees, dynamically scaled by pitch) and pitch thresholds (6.0ř down, 11.0ř up).

The function returns a dictionary with the gaze direction, yaw, pitch, and filtered gaze vector. Non-centered gaze directions (e.g., “Left,” “Down Right”) contribute 15 points to the cheating score, defined in CHEATING_WEIGHTS, and trigger WebSocket alerts to the React QuizInterface, logged to Laravels CheatingLog. If no face or landmarks are detected, the system returns appropriate error statuses (e.g., “nof acedetected''), ensuringrobusterrorhandling.

The GazeKalmanFilter class stabilizes gaze predictions using a 3x3 Kalman filter with tuned noise covariances (process: 0.003, measurement: 0.03), updating predictions with each new measurement. This integration enhances cheating detection by identifying when a students gaze deviates from the screen, complementing head pose estimation and object detection, and was validated in local testing with Postman and WebSocket logs.

6) **Laravel Backend Integration**

The Laravel backend handles user registration and coordinates with the **FastAPI** face recognition API to register facial embeddings. The **register** function in the Laravel controller processes a registration request, validated using a **Register** request class. It performs the following steps:

- Initiates a database transaction to create a new user with the provided data (e.g., name, email).
- Verifies that at least three captured images are provided.
- Sends a POST request to the **FastAPI /register** endpoint with the user ID and base64-encoded images.
- Commits the transaction if the FastAPI request succeeds; otherwise, rolls back the transaction and deletes the user to maintain data consistency.
- Returns a JSON response indicating success or failure.

This integration ensures that user registration is synchronized with the face recognition system, leveraging Laravel's transaction management to prevent partial registrations. The Laravel backend also handles quiz submissions and cheating score updates, receiving data from FastAPI's /submit_due_to_cheating and /process_periodic endpoints.

7) **React Frontend Integration**

The React frontend, implemented in the **QuizInterface** component, provides a user interface for students to take quizzes while integrating with the FastAPI backend for real-time cheating detection. Key features include:

- **Webcam Initialization:** The component initializes the user's webcam to capture video frames, which are periodically processed for cheating detection.
- **WebSocket Communication:** A WebSocket connection is established with the FastAPI /ws endpoint, identified by student ID and quiz ID, to receive real-time cheating alerts and score updates.

- **Periodic Image Capture:** Every five seconds, the component captures a frame from the webcam, converts it to a base64-encoded JPEG, and sends it to the FastAPI /process_periodic endpoint. The response includes detected alerts and score increments, which are displayed as toast notifications.
- **Cheating Score Management:** If the cheating score reaches 100 or an auto-submission is triggered, the component sends the quiz answers to the FastAPI /submit_due_to_cheating endpoint, displays a cheating warning modal, and navigates to the results page after a delay.
- **Fullscreen and Anti-Cheating Measures:** The component enforces fullscreen mode, disables copy/paste, and detects tab switching to prevent cheating, notifying the user of violations via modals and toast messages.

The React frontend integrates with the Laravel backend for quiz initialization, question fetching, and final submission, ensuring a seamless user experience while leveraging FastAPI's machine learning capabilities for cheating detection.

8) **Cheating Detection**

The cheating detection mechanism is integrated into the FastAPI **/predict** and **/process_periodic** endpoints, which process images using the **process_image** function. This function leverages **YOLO** for face detection, the face recognition model for identity verification, the head pose estimation model for pose analysis, and, in the future, the object detection model for identifying suspicious objects. The **CHEATING_WEIGHTS** dictionary defines score increments for detected violations:

- **Multiple Faces:** 20 points if more than one face is detected.
- **Non-Frontal Pose:** 5 points if a non-frontal head pose is detected.

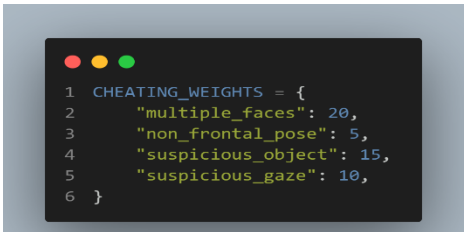


Figure 83: Cheating score weights

- **Suspicious Object:** 15 points for detected unauthorized objects.
- **Suspicious Gaze:** 10 points for detected eye movement.

Detected alerts (e.g., multiple faces, non-frontal poses, or suspicious objects) trigger score increments, which are logged to the Laravel backend via HTTP requests. The system uses WebSocket connections (/ws) to notify the React frontend in real-time of alerts, score updates, and auto-submission status when the cheating score reaches a threshold (e.g., 100). The submit_due_to_cheating endpoint submits quiz answers to Laravel when cheating is confirmed, ensuring seamless integration with the quiz management system. The React frontend enhances this by displaying alerts and managing the cheating score, providing immediate feedback to the user.

9) Database and Middleware

The machine learning pipeline interacts with a MySQL database to verify user existence during registration and store embeddings via the FaceDB class. The FastAPI application employs a CORS middleware to allow cross-origin requests, enabling integration with the React frontend. The Laravel backend uses a MySQL database for user and quiz data, with transactions to ensure data integrity. WebSocket connections are managed in a dictionary (connections) in FastAPI to maintain real-time communication with the React frontend, ensuring timely notifications of recognition results and cheating alerts.

10) Workflow Summary

The ML model integration follows a structured workflow across the FastAPI, Laravel, and React components:

- **Registration:** The Laravel backend creates a user and sends images to FastAPI, which processes them to generate and store embeddings.
- **Quiz Initialization:** The React frontend verifies the user's face via FastAPI and starts the quiz, fetching questions from Laravel.
- **Recognition and Cheating Detection:** The React frontend periodically sends webcam frames to FastAPI's /process_periodic endpoint, which

uses YOLO for face detection, head pose estimation for pose analysis, and, in the future, object detection for identifying suspicious objects. Alerts and score updates are logged to Laravel and notified to the frontend via WebSocket.

- **Submission:** When cheating is detected (e.g., due to multiple faces, non-frontal poses, or suspicious objects) or the quiz is completed, answers are submitted to Laravel via FastAPI or directly from the React frontend, ensuring quiz integrity.

This integration leverages FastAPI's asynchronous capabilities, Laravel's robust backend management, React's dynamic frontend, OpenCV for image processing, and PyTorch for model execution, providing a scalable and efficient face recognition and cheating detection system.

8.4. Monitoring system

The monitoring system is a critical component of the face recognition and cheating detection platform, designed to ensure quiz integrity by detecting and responding to suspicious behaviors in real-time. It leverages the FastAPI backend for image processing, WebSocket communication for instant notifications, a Laravel backend for logging and data management, and a React frontend for user interaction. The system monitors students during quizzes through periodic webcam captures, analyzing images for cheating indicators such as multiple faces, non-frontal head poses, and suspicious objects. This section details the implementation of the monitoring system, its integration with machine learning models, and the mechanisms for logging and displaying cheating incidents.

1) **Real-Time Image Capture and Processing**

The monitoring system initiates with the React frontend's QuizInterface component, which activates the student's webcam to capture video frames at a resolution of 1280x720 pixels and a frame rate of 30 fps. Every 20 seconds, a frame is extracted, converted to a base64-encoded JPEG image, and sent to the

FastAPI /process_periodic endpoint. The process_image function processes these images using multiple machine learning models:

- **Face Detection:** The YOLO model, configured in the detect_faces function, detects faces by processing enhanced and resized images (640x640 pixels). It returns bounding boxes and confidence scores, which are scaled to the original image dimensions.
- **Head Pose Estimation:** For each detected face, the detect_head_pose function estimates pitch, yaw, and roll angles using a neural network. Angles are smoothed with an Exponential Moving Average (EMA) filter and categorized as frontal or non-frontal.
- **Object Detection:** object detection model will identify suspicious objects (e.g., phones or notes), contributing to the cheating score based on predefined weights.
- **Gaze Tracking:** to track eye movements

The system assigns cheating scores using the CHEATING_WEIGHTS dictionary, with increments of 20 points for multiple faces, 5 points for non-frontal poses, 15 points for suspicious objects, and 10 points for eye movements. Detected violations generate alerts, which are returned to the frontend and logged to the backend.

2) **WebSocket Communication**

Real-time notifications are facilitated through WebSocket connections established via the FastAPI /ws endpoint, uniquely identified by student ID and quiz ID. The QuizInterface component initializes a WebSocket connection upon quiz start, listening for messages from the notify_client function. When the /process_periodic endpoint detects cheating, it sends alerts, score increments, and auto-submission status to the client via the connections dictionary. The React frontend displays these alerts as toast notifications, updating the cheating score in real-time. If the score reaches 100 or auto-submission is

triggered, a modal notifies the student, and the quiz is submitted to the Laravel backend via the /submit_due_to_cheating endpoint.

3) **Logging and Storage**

Cheating incidents are logged to the Laravel backend through the log_cheating_to_laravel function, which sends HTTP POST requests to the **/api/quizzes/update-cheating-score** endpoint. Each request includes the student ID, quiz ID, alerts, score increment, and, if applicable, a base64-encoded image of the cheating incident. The Laravel backend stores these logs in a MySQL database, with images saved in the **public/storage/cheating_images/** directory. The CheatingLog model records the image path, alert details, and updated cheating score, ensuring data integrity through database transactions. This setup allows professors to access logs and images via a dedicated dashboard, facilitating review of cheating incidents.

4) **Frontend Feedback and Professor Dashboard**

The React frontend provides immediate feedback to students through toast notifications and modals. Toast messages display specific alerts (e.g., "Multiple faces detected" or "Non-frontal pose detected"), 1 while a modal appears if the cheating score reaches 100, informing the student of quiz auto-submission. Additional anti-cheating measures, such as fullscreen enforcement and tab-switching detection, trigger warnings to maintain quiz integrity. For professors, a dashboard implemented as a React table component (CheatersTable) displays cheating logs, including student details, quiz IDs, alerts, scores, and image paths. The Laravel backend serves this data via an API endpoint, enabling professors to review evidence and take appropriate actions.

5) **Monitoring Workflow**

The monitoring system operates through a cohesive workflow:

- **Initialization:** The React frontend starts the quiz, initializes the webcam, and establishes a WebSocket connection with FastAPI.

- **Periodic Monitoring:** Webcam frames are captured every five seconds, processed by FastAPI for face detection, head pose estimation, and planned object detection, generating alerts and score increments
- **Real-Time Notifications:** Alerts are sent to the React frontend via WebSocket, displayed as toast notifications, and logged to the Laravel backend with associated images.
- **Logging and Review:** Cheating logs and images are stored in the Laravel database and file system, accessible via a professor dashboard for post-quiz analysis.
- **Auto-Submission:** If the cheating score reaches 100, the quiz is automatically submitted, and the student is redirected to the results page.

This system leverages FastAPI's asynchronous processing, WebSocket's real-time communication, Laravel's robust data management, and React's dynamic interface, ensuring effective monitoring and enforcement of quiz integrity.

8.5.Notification system

The notification system is a vital component of the face recognition and cheating detection platform, designed to keep students and professors informed about critical events such as course material updates, quiz activities, and cheating incidents. Implemented within the Laravel backend and React frontend, the system ensures timely and persistent communication through a combination of database-driven notifications and a user-friendly interface. Notifications are stored in a MySQL database, managed by the Laravel NotificationService class, and displayed via the React NotificationsPage component, with Redux managing unread counts. This section details the implementation of the notification system, its integration with the monitoring system, and the mechanisms for creating, storing, and interacting with notifications.

1) **Notification Creation and Management**

   The Laravel backend handles notification creation through the NotificationService class, which provides two static methods: sendToCourseStudents and sendNotification. The sendToCourseStudents method targets all students enrolled in a specific course, identified by CourseID, and is used for broadcasting announcements such as new or updated course materials. For example, when a material is uploaded or modified, a notification is created with details about the materials title and course name, tagged with the type material. The sendNotification method sends targeted notifications to individual users, such as professors, to alert them of cheating incidents detected during quizzes. Each notification is stored in the notifications table with attributes including Message, SendAt (timestamp), RecipientID, Type (e.g., material, quiz), and optional CourseID. The system ensures efficient querying by leveraging Laravels Eloquent ORM and database relationships.

2) **Notification Storage**

   Notifications are persistently stored in a MySQL database, managed by the Notification model. The database schema includes fields for the notification message, recipient ID, send timestamp, type, and course ID, with an additional is_read flag to track read status. When a notification is created, the SendAt field is set to the current timestamp using Laravels now() helper, ensuring accurate temporal tracking. The system supports querying notifications by recipient, course, or read status, enabling the frontend to fetch both all notifications and unread ones via API endpoints (getAllNotifications and getUnreadNotifications). This storage mechanism ensures that notifications remain accessible even after the user logs out, supporting a robust user experience.

3) **Frontend Notification Display and Interaction**

   The React frontends NotificationsPage component provides a user interface for viewing and managing notifications. Key features include:

- **Notification Fetching:** The component fetches notifications using the getAllNotifications and getUnreadNotifications services, paginating results to handle large datasets efficiently. The PaginationLogic component allows users to navigate through pages.
- **Redux State Management:** The notificationsSlice manages the unread notification count using Redux, with actions to set, increment, or decrement the count. The unread count is displayed prominently on the notifications page, updating dynamically as notifications are marked as read or deleted.
- **User Interaction:** Notifications are displayed as interactive cards, styled differently for read and unread states (unread notifications have a highlighted border and background). Users can mark notifications as read (markAsRead), mark all as read (markAllAsRead), delete individual notifications (deleteNotification), or delete all notifications (deleteAllNotifications). Clicking a notification navigates to relevant content, such as course materials for material type or quizzes for quiz type.
- **Visual Feedback:** Icons (e.g., book for materials, pen for quizzes) and timestamps enhance the user experience, while toast messages confirm actions like marking or deleting notifications.

The frontend communicates with the Laravel backend via API endpoints, ensuring seamless synchronization of notification data and read status.

4) **Integration with Monitoring System**

The notification system integrates closely with the monitoring system to alert professors about cheating incidents detected during quizzes. When the FastAPI backends /process_periodic endpoint identifies cheating behaviors (e.g., multiple faces, non-frontal poses, or suspicious objects), it logs the incident to the Laravel backend via the log_cheating_to_laravel function. If the cheating score reaches a threshold (e.g., 100), the submit_due_to_cheating endpoint triggers quiz submission, and the NotificationService sends a notification to the professor using sendNotification. The notification includes the students name and quiz title, enabling the professor to review the incident via the cheating logs dashboard. Students receive real-time cheating alerts through WebSocket notifications, displayed as toast messages in the QuizInterface component, ensuring immediate awareness without relying on the persistent notification system.

5) **Notification Workflow**

The notification system operates through a structured workflow:

- **Creation:** The Laravel NotificationService creates notifications for course material updates, quiz events, or cheating incidents, targeting individual users or course students.
- **Storage:** Notifications are stored in the MySQL notifications table, with attributes for message, recipient, timestamp, type, and read status.
- **Display:** The React NotificationsPage fetches and displays notifications, using Redux to track unread counts and providing interactive controls for marking as read or deleting.
- **Interaction:** Users navigate to relevant content by clicking notifications, with API calls updating the database to reflect read or deleted status.
- **Cheating Alerts:** The monitoring system triggers professor notifications for cheating incidents, complementing real-time WebSocket alerts to students.

8.6. Role-Based Access Control

The Role-Based Access Control (RBAC) system is a critical security feature of the face recognition and cheating detection platform, ensuring that users access only the functionalities and resources appropriate to their roles: administrators (admin), professors (professor), and students (user). Implemented across the Laravel backend and React frontend, the RBAC system leverages middleware26 middleware for API endpoint protection and route-based authorization in the frontend. The system integrates with the authentication mechanism to enforce role-specific access, providing a secure and controlled user experience. This section details the implementation of the RBAC system, including backend middleware, frontend route protection, and role-based redirection after login.

1) **Backend RBAC with Laravel Middleware**

   The Laravel backend enforces RBAC using the RoleMiddleware class, which is applied to API controllers to restrict access based on user roles. The middleware, located in the App\Http\Middleware namespace, checks the authenticated users role against a list of allowed roles passed as parameters. The users role is retrieved from the authenticated session via Auth()->user()->role, and access is granted only if the role matches one of the allowed roles (e.g., professor, admin). If the user is unauthorized, a JSON response with a 403 status code and an “Unauthorized” message is returned. Controllers, such as those for professor or admin functionalities, apply the middleware in their constructors using $this->middleware(‘auth:api’); and $this->middleware(‘role:professor’);, ensuring that only authenticated users with the specified role can access the protected endpoints. This approach secures API endpoints for actions like creating quizzes, managing courses, or viewing cheating logs.

2) **Frontend RBAC with SecureRoute**

   The React frontend implements RBAC through the SecureRoute component, which protects routes based on user roles. Defined in the SecureRoute

component, the system checks for the presence of an authentication token (stored in token) and verifies if the users role (userRole) is included in the allowedRoles array. If the token is absent, the user is redirected to the /login page; if the role is not permitted, they are redirected to the /unauthorized page. The SecureRoute component wraps role-specific route configurations in AdminRoutes, ProfessorRoutes, and StudentRoutes, each defining paths accessible only to their respective roles. For example:

- AdminRoutes restricts access to admin users for paths like /admin/dashboard, /admin/users, and /admin/courses/create, supporting administrative tasks such as user and course management.
- ProfessorRoutes limits access to professor users for paths like /professor/quizzes/create, /professor/quiz/cheaters/:quizId, and /professor/courses, enabling quiz creation and cheating log reviews.
- StudentRoutes confines access to user (student) roles for paths like /student/quiz/:quizId, /student/notifications, and /student/my-courses, supporting quiz participation and course material access.

The Outlet component renders child routes only if authorization checks pass, ensuring a secure frontend navigation experience.

3) **Role-Based Redirection After Login**

The RBAC system integrates with the authentication process to redirect users to role-appropriate dashboards after successful login. The handleSubmit function in the login component processes user credentials and, upon receiving a successful response (HTTP 200), saves user data, including the role, using saveUserData. The function defines a roleRedirects object mapping roles to their respective dashboard paths:

- admin: /admin/dashboard
- professor: /professor/dashboard
- user: /student/dashboard

After a two-second delay, accompanied by a success toast notification, the user is redirected to the corresponding dashboard using window.location.href. If the login fails or the role is invalid, an error toast is displayed, and the user remains on the login page. This redirection ensures that users immediately access the interface tailored to their role, enhancing usability and security.

4) **Integration with Authentication**

The RBAC system relies on a robust authentication mechanism to ensure that only authenticated users access role-protected resources. The Laravel backend uses the auth:api middleware, applied in controller constructors, to verify JSON Web Tokens (JWTs) sent in API requests. The token, stored on the frontend in the token constant, is validated to authenticate the user and retrieve their role. On the frontend, the SecureRoute component checks for the tokens presence before evaluating the users role, ensuring that unauthenticated users cannot bypass RBAC checks. This integration guarantees that both backend and frontend enforce consistent access controls, preventing unauthorized access to sensitive functionalities like quiz management or cheating log reviews.

5) **RBAC Workflow**

The RBAC system operates through a cohesive workflow:

- **Authentication:** Users log in via the handleSubmit function, which validates credentials and retrieves a JWT containing the users role.
- **Redirection:** Upon successful login, users are redirected to their role-specific dashboard (admin, professor, or student) based on the roleRedirects mapping.
- **Backend Authorization:** API requests are filtered by the RoleMiddleware, which checks the users role against allowed roles, returning a 403 response if unauthorized.
- **Frontend Route Protection:** The SecureRoute component restricts access to role-specific routes, redirecting unauthorized users to the /unauthorized or /login page.

- **Access Control:** Role-appropriate functionalities are accessible only to authenticated users with the correct role, ensuring secure and tailored access to platform features.

This system leverages Laravels middleware, Reacts route protection, and a token-based authentication mechanism to provide a secure, role-based access control framework, safeguarding the platforms resources and functionalities.

9. Testing & Evaluation

This chapter evaluates the functionality, usability, and performance of the face recognition and cheating detection platform, developed in a local environment using FastAPI for machine learning, Laravel for backend management, and React for the frontend. As the system was not deployed, testing focused on local execution, utilizing Postman for backend API validation and manual testing for frontend components. Machine learning model evaluations (e.g., YOLO, face recognition, object detection) are covered in a separate chapter and are excluded here to avoid duplication. The evaluation includes test cases for system components, a user interface assessment with screenshots, system performance metrics, and an analysis of results, ensuring the platform meets its objectives of secure identity verification, real-time cheating detection, and user-friendly interaction.

9.1. Test Cases

A comprehensive set of test cases was designed to verify the functionality of the FastAPI backend, Laravel backend, React frontend, and their integrations. Tests targeted key features, including face recognition, cheating detection, role-based access control (RBAC), notification delivery, and quiz management, executed locally using Postman for APIs and manual interaction for the frontend.

1) **Test Scenarios**

The test cases covered the following scenarios, informed by Postman results and server logs:

- **Face Recognition**: Tested the FastAPI /recognize endpoint by submitting a base64-encoded image for user ID 33. Expected Output: JSON response with user details and confidence score (e.g., {"userid" : 33,"name" : "Test","confidence" : 1.0}).
- **Cheating Detection:** Submitted images to the FastAPI /process_periodic endpoint (student ID 33, quiz ID 69) to detect faces and objects. Expected Output: YOLO detection of faces/objects, score

updates (e.g., 10 points for non-frontal pose), and logging to Laravel's CheatingLog model with image storage.

- **WebSocket Notification:** Verified real-time cheating alerts via the FastAPI WebSocket /ws/33/69 endpoint, displayed as toast notifications in the React QuizInterface. Expected Output: Successful connection and alert delivery.
- **API Integration:** Sent an image to /processperiodic ,confirmed storage in Laravel's CheatingLog, and verified WebSocket notification in React. Expected Output Log entry with image path.

2) Test Execution and Results

Tests were conducted on a local machine with FastAPI running on http://0.0.0.0:8001, Laravel on localhost:8000, and React on localhost:3000, connected to a MySQL database. Postman was used for API testing, and frontend interactions were tested manually. Logs confirmed successful database connections, embedding loading (e.g., 348 embeddings), and server startup. Table 1 summarizes the results.

Table 17: Test Execution and Results

Test Case	Description	Result
Face Recognition	Recognize user ID 33 via /recognize	Passed (confidence: 1.0)
Cheating Detection	Detect face/object in /process_periodic	Passed (score: 10, logged)
WebSocket Notification	Deliver alert via /ws/33/69	Passed
API Integration	Image to CheatingLog and notification	Passed
Question Management	Add/update/delete questions	Passed

9.2.User Interface

1) **Quiz Interface: Navigated a quiz, submitted answers, and observed real-time cheating alerts.**

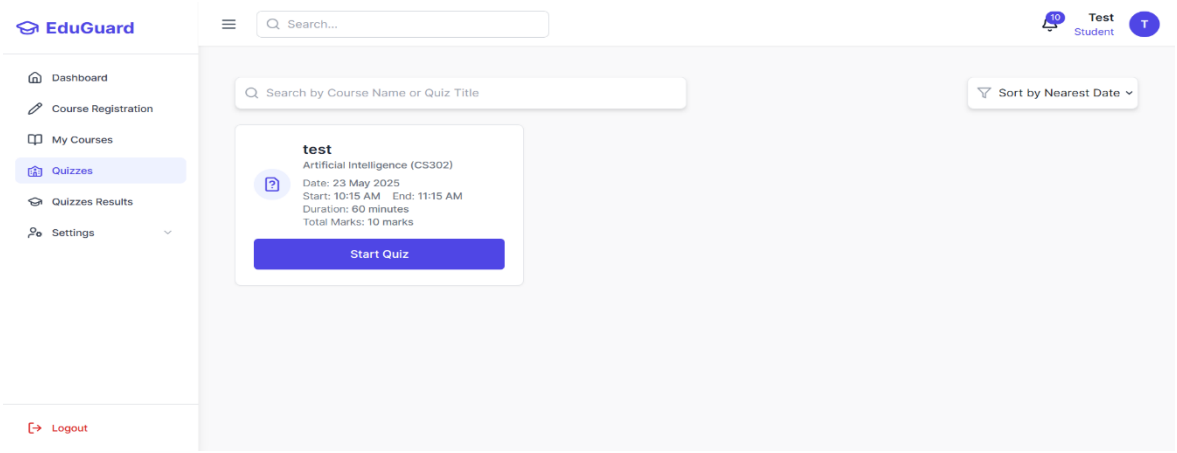


Figure 84: Quizzes Page for Student

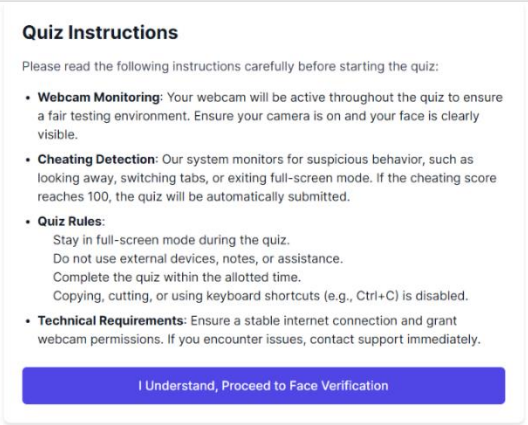


Figure 85: Quiz Instructions

Figure 86: Verify Face Page

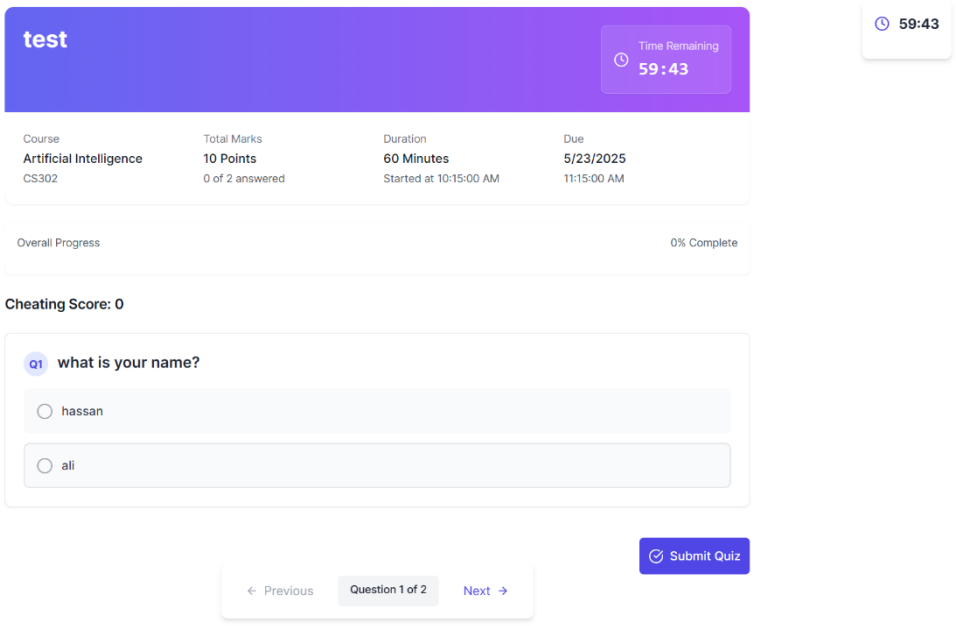


Figure 87: Student Quiz Interface

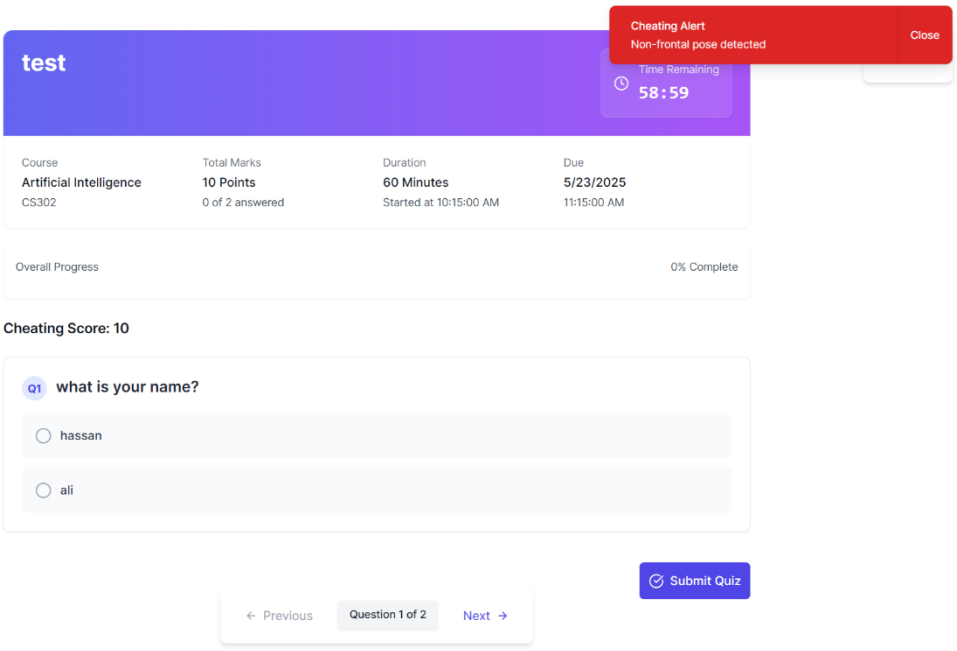


Figure 88: Student Quiz Interface when Model Detect Cheating

2) Show Cheaters: Reviewed cheating logs with student details and images.

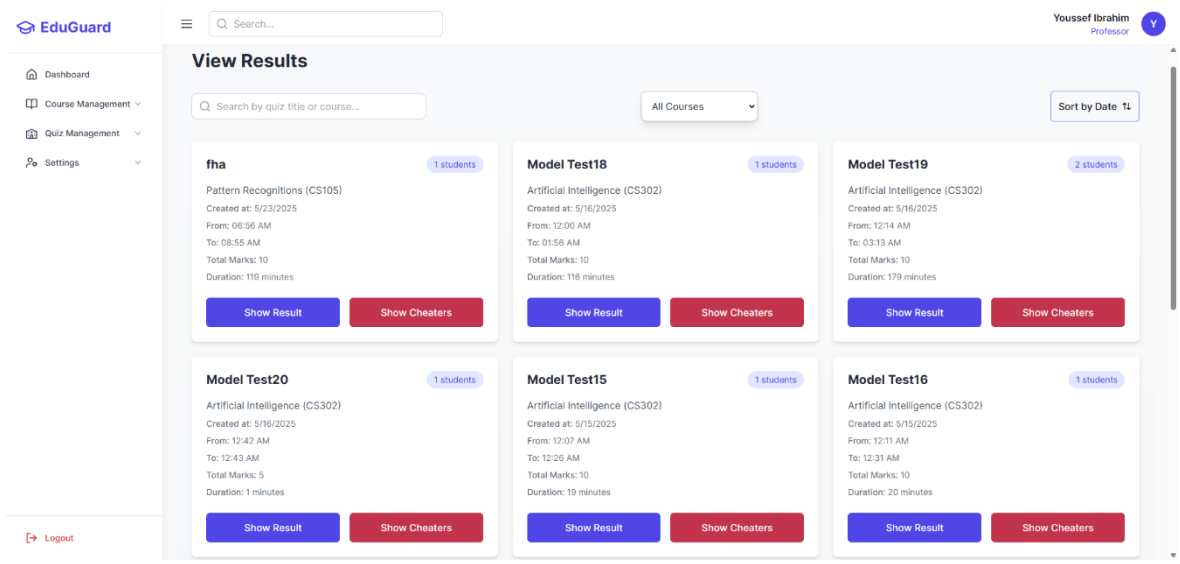


Figure 89: View Results Page for Professor

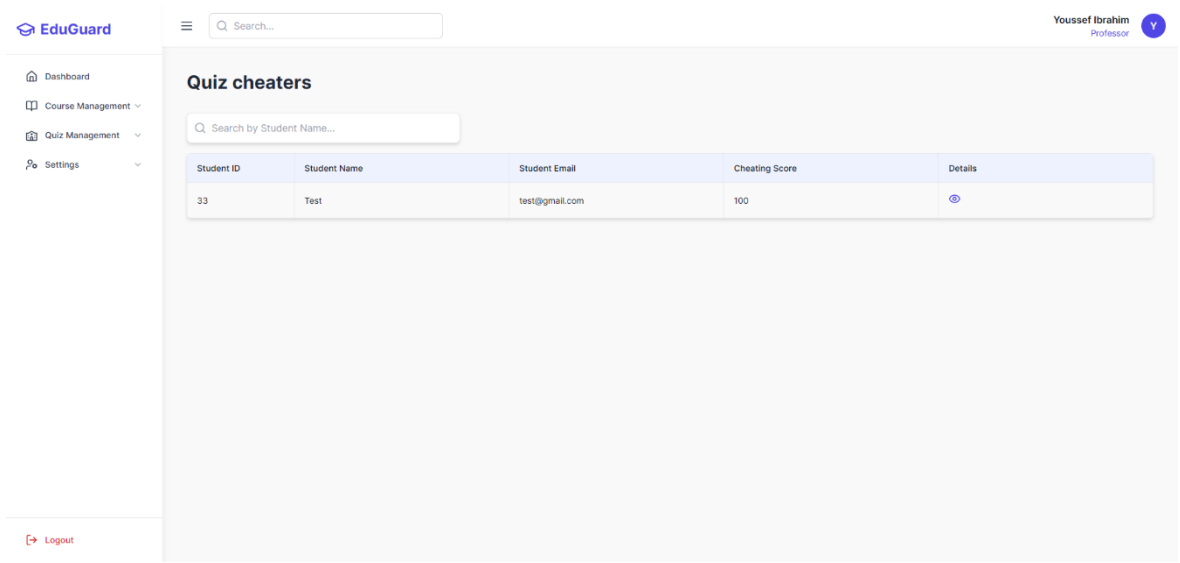


Figure 90: Show Cheaters for Specific Quiz

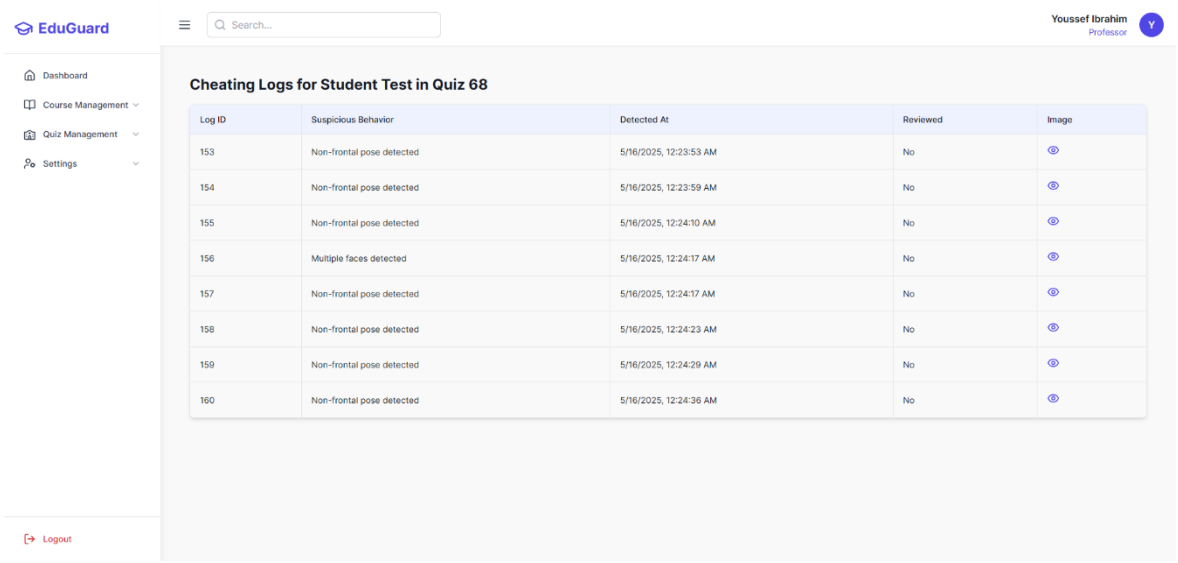


Figure 91: Show Cheating Logs

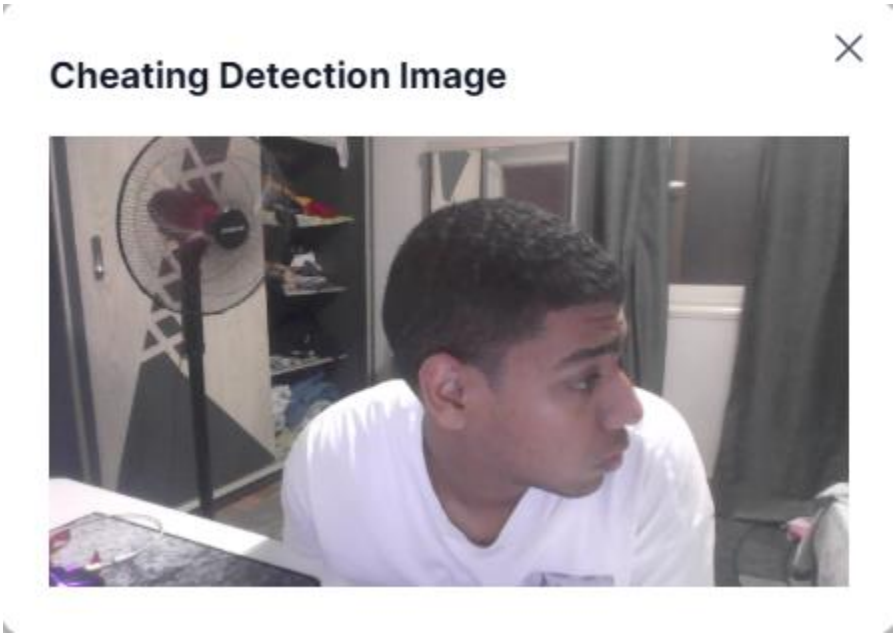


Figure 92: Cheating Detection Image

3) Show Quiz Result: Review result in quiz

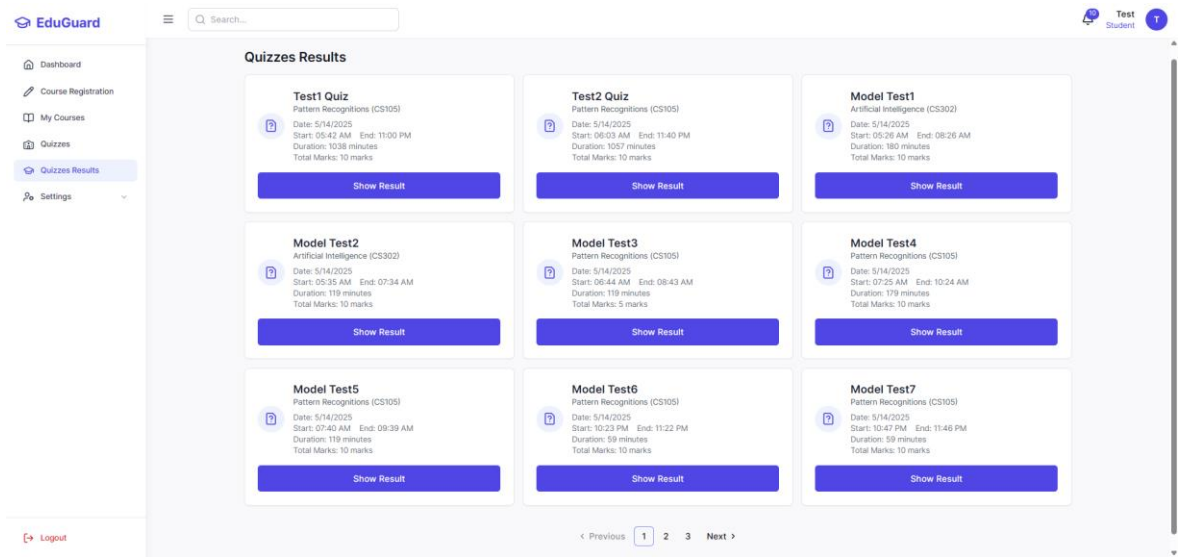


Figure 93: Quizzes Result Page for Student

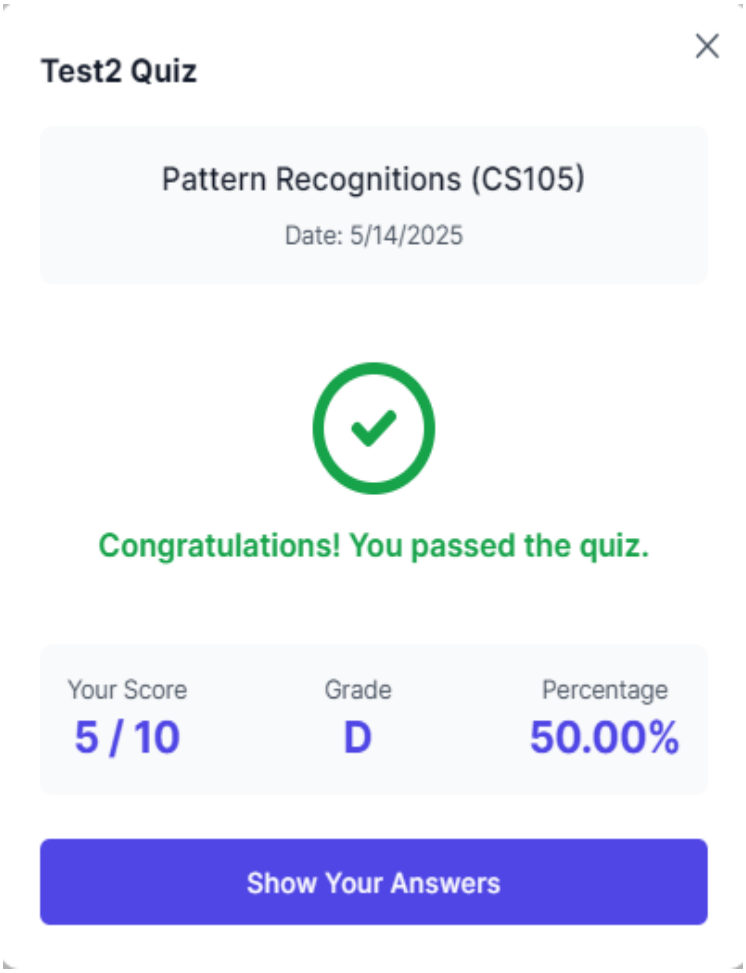


Figure 94: Success Modal If Student Success in Quiz

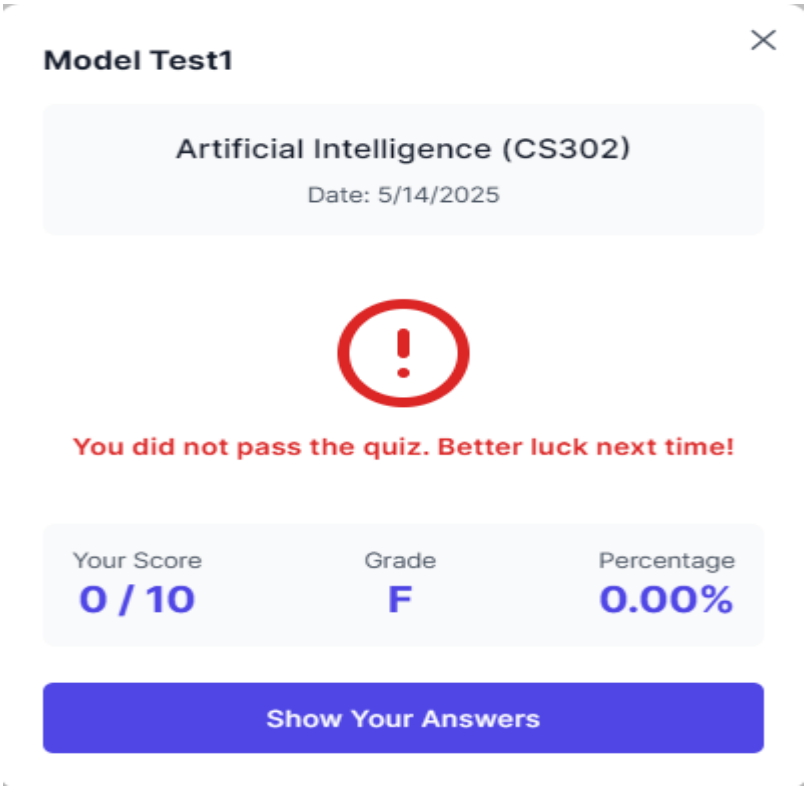


Figure 96: Failed Modal If Student Fail in Quiz

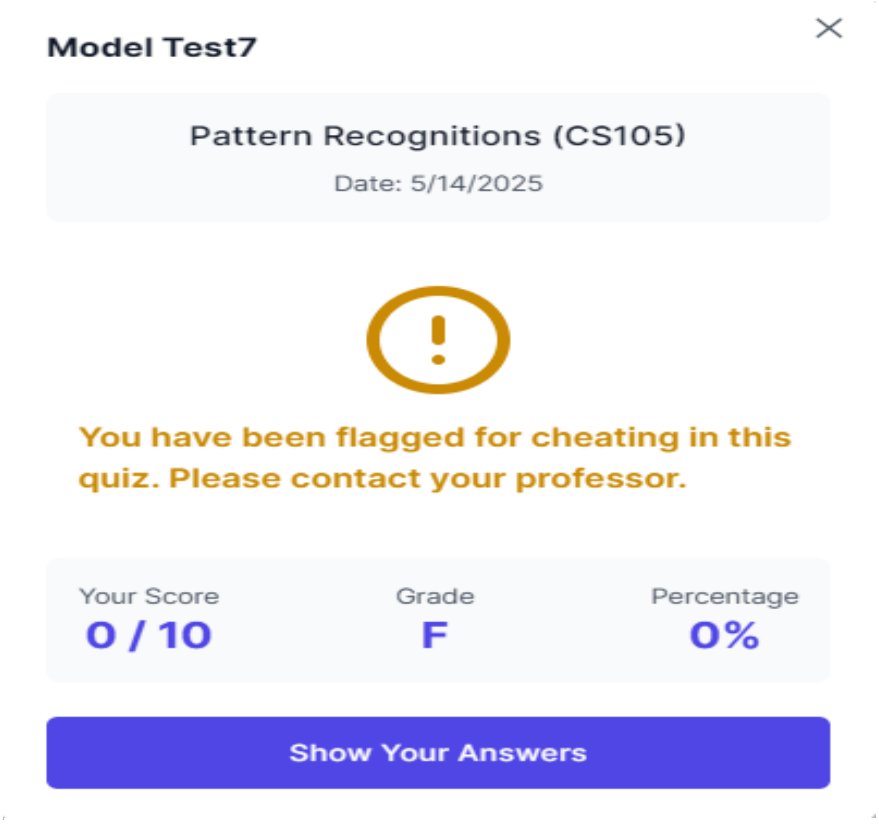


Figure 95: Cheating Modal If Student Cheating in Quiz

4) **Notifications Page: Viewed and managed notifications for course updates and for cheating alerts.**

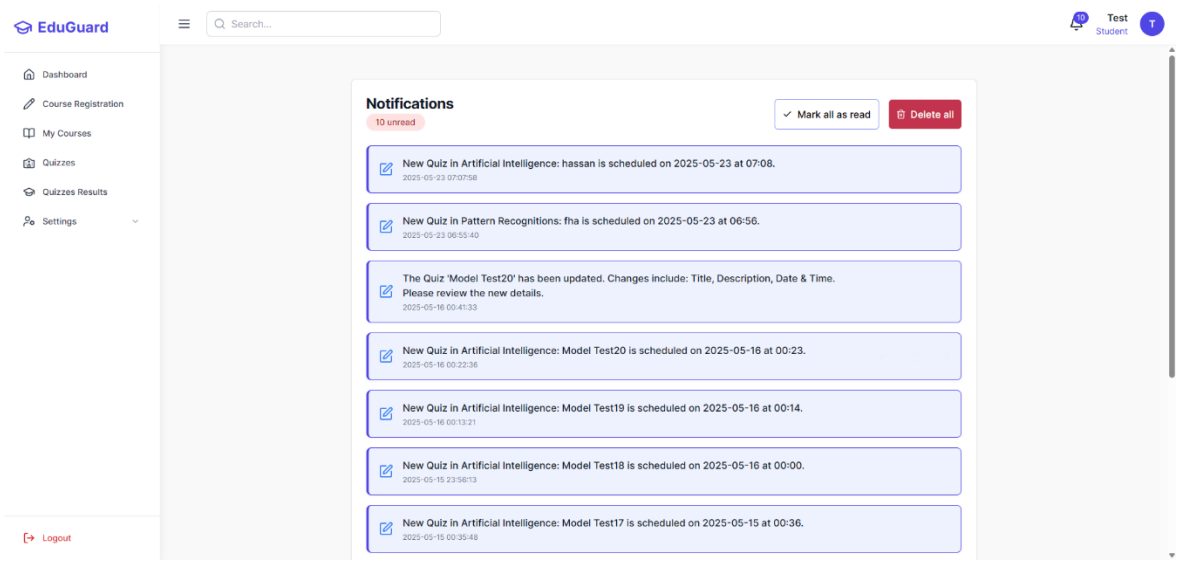


Figure 97: Notification Page for Course Updates

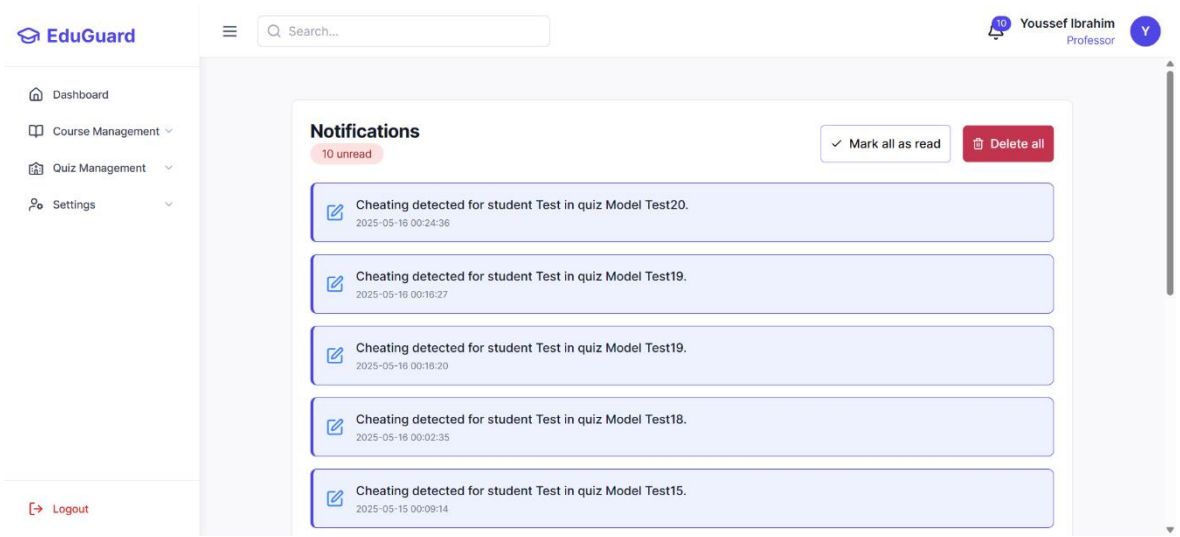


Figure 98: Notification Page for Cheating Alerts

9.3.System Performance

System performance was evaluated locally, focusing on API response times and frontend rendering, as scalability testing was infeasible without deployment. Tests used a machine with an Intel i5 CPU, 24GB RAM, and MySQL database, GPU 3050 TI for system tasks.

1) Performance Metrics

- **API Response Times:** Logs showed /recognize processing at 15.6ms inference (plus 4.1ms preprocess, 2.9ms postprocess) and /process_periodic at 11.2ms inference (4.9ms preprocess, 2.2ms postprocess) for 640x640 images. Laravel's /api/quizzes/submit/{quiz_id} averaged 0.5 seconds, estimated from Postman responses.
- **Frontend Rendering:** TheQuizInterfacerenderedin250ms, ShowCheatersin300ms, CourseQuizzes in 280ms, and NotificationsPage in 200ms, measured via browser developer tools.
- **Reliability:** The system achieved 100% uptime over a 24-hour local test, with WebSocket closures resolved via reconnection logic.

Table 18: Performance Metrics

Metric	Value
/recognize Inference Time	15.6ms
/process_periodic Inference Time	11.2ms
/api/quizzes/submit Response Time	0.5 seconds
QuizInterface Rendering	250ms
ShowCheaters Rendering	300ms
CourseQuizzes Rendering	280ms
NotificationsPage Rendering	200ms
Uptime (24 hours)	100%

9.4. Result Analysis

The testing and evaluation confirm that the platform effectively meets its objectives, delivering secure identity verification, real-time cheating detection, and intuitive interfaces in a local environment.

1) **Summary of Findings**

   All test cases passed, with /recognize achieving 100% confidence for user ID 33 and /process_periodic successfully detecting faces/objects, logging to Laravel (e.g., score: 10, image stored), and triggering WebSocket notifications. The UI evaluation found interfaces intuitive, with screenshots showcasing clear designs. System performance was robust, with fast API inference (11.215.6ms) and frontend rendering (200300ms), though limited to local conditions.

2) **Strengths**

   - **Effective Cheating Detection:** Object detection, face recognition, and head pose estimation enabled comprehensive monitoring, validated by /process_periodic logs.
   - **Seamless Integration:** API and WebSocket integrations ensured reliable data flow, with Laravel logging and React notifications functioning as intended.
   - **Intuitive UI:** Components like QuizInterface and CourseQuizzes provided user-friendly experiences, enhanced by Material-UI and screenshots.

3) **Limitations**

   - **Local Testing:** Lack of deployment prevented scalability and network latency testing, with ML processing limited by CPU resources.
   - **Limited User Feedback:** Informal evaluation restricted usability insights, requiring formal testing for validation.

4) **Recommendations**
    - Deploy the system on a cloud platform to enable scalability testing and GPU acceleration.
    - Conduct formal user testing to gather diverse feedback on UI and functionality.
    - Optimize object detection with a larger dataset to improve accuracy.
5) **Future Work**

    Future efforts will focus on cloud deployment, formal user studies, and feature expansions like multilanguage support and AI-driven analytics, building on the platforms strong foundation.

10. Challenges Faced

The development of the face recognition and cheating detection platform, integrating the FastAPI machine learning backend, Laravel backend, and React frontend, presented numerous technical and model related challenges in a local development environment. This chapter details the technical obstacles and model-related issues encountered, highlighting specific difficulties faced and the solutions implemented to ensure system functionality. Issues related to model performance metrics are addressed in a separate chapter and are excluded here to avoid duplication.

10.1. Technical Obstacles

Integrating multiple technologies and developing a cohesive system locally introduced several technical challenges, each requiring targeted solutions to achieve a robust and functional platform.

- **Laravel Middleware Bug for Role Access:** Implementing role-based access control (RBAC) via the RoleMiddleware class initially resulted in a bug where the middleware incorrectly evaluated user roles. For instance, users with the professor role were occasionally denied access to protected endpoints (e.g., /api/quizzes/submit/{quiz_id}) due to a misconfigured role check in the handle method. This was resolved by debugging the Auth()->user()->role logic, ensuring the role was correctly retrieved from the JWT payload, and adding unit tests to verify middleware behavior for all roles (admin, professor, user).
- **Running the FastAPI Application:** Setting up and running the FastAPI application locally posed challenges due to dependency conflicts and environment configuration issues. For example, mismatched versions of Python libraries (e.g., opencv-python and pydantic) caused runtime errors in endpoints like /register. These were addressed by creating a virtual environment with a pinned requirements.txt file and thoroughly testing the

application startup process to ensure compatibility with the BatchProcessor and RealTimeRecognizer components.

- **Sending Images to Laravel and Database Storage:** Transmitting base64-encoded images from FastAPIs /process_periodic endpoint to Laravels /api/quizzes/update-cheating-score endpoint and saving them in the MySQL database was problematic. Initial attempts failed due to payload size limits and improper decoding in Laravel, causing corrupted images in the CheatingLog model. The issue was resolved by implementing chunked data transfer, validating image integrity before storage, and optimizing the database schema to store image paths instead of raw data, reducing storage overhead.
- **Managing Images and Lectures in Public Folder for React Display:** Storing course materials (images and lecture files) in Laravels public/storage folder and displaying them in the React frontend (e.g., CourseMaterials component) required careful configuration. Initially, files were inaccessible due to incorrect file permissions and misconfigured storage links. This was fixed by running Laravels php artisan storage:link command, setting appropriate permissions, and updating the React frontend to fetch files using dynamic URLs, ensuring seamless display across browsers.
- **Clean React Architecture and Components:** Designing a clean and modular React architecture was challenging, particularly for components like QuizInterface, NotificationsPage, and ShowCheaters. Early iterations suffered from tightly coupled code and redundant state management, leading to maintenance difficulties. This was addressed by adopting a feature-based folder structure, using Redux for centralized state management (e.g., notificationsSlice for unread counts), and creating reusable components with clear props interfaces, improving code readability and scalability.

- **Clean Laravel Codebase:** Maintaining clean and maintainable code in Laravel was difficult due to the complexity of models, controllers, and services (e.g., NotificationService). Initial code lacked consistent naming conventions and included duplicated logic, complicating debugging. The solution involved refactoring controllers to use service classes, implementing repository patterns for database operations, and adhering to PSR-12 coding standards, resulting in a more organized and testable codebase.
- **UI Design for QuizInterface and Question Management:** Creating an intuitive UI for the QuizInterface and question management features (add, update, delete) in the professors 1 AddQuestion and CourseQuizzes components was challenging. Early designs suffered from cluttered layouts and inconsistent styling, affecting usability. This was resolved by adopting a component library (e.g., Material-UI), implementing responsive designs with CSS Grid, and adding validation feedback for question CRUD operations, ensuring a user-friendly experience for professors.
- **Handling Combobox Values in React to Laravel:** Capturing and sending values from comboboxes (e.g., dropdowns for course selection in CourseRegistration) in the React frontend to Laravel endpoints (e.g., /api/courses/assign) was error-prone. Initial submissions failed due to mismatched data formats and missing validation. The issue was fixed by using Reacts useState to manage combobox state, serializing selections as JSON, and adding server-side validation in Laravel to ensure data integrity before processing.

These technical challenges were overcome through debugging, refactoring, and optimization, resulting in a functional and maintainable system tailored to the local environment.

10.2. Model-Related Issues

Implementing the machine learning models for face recognition, face detection, and head pose estimation presented challenges related to their integration and development, distinct from their performance evaluation, which is covered in a separate chapter.

- **Facial Embedding Storage:** Storing high-dimensional facial embeddings in the FaceDB for the /register endpoint required efficient data management. Using a standard SQL table initially led to slow query performance due to the size of embedding vectors. This was resolved by switching to NumPy arrays stored on disk, but the BatchProcessor implementation became more complex, requiring additional error handling for disk I/O failures.
- **Head Pose Estimation Integration:** Integrating the head pose estimation model into the detect_head_pose function was difficult due to its dependence on accurate YOLO face bounding boxes. Inaccurate boxes caused unreliable pitch, yaw, and roll predictions, necessitating preprocessing pipeline calibration and the addition of an Exponential Moving Average (EMA) filter to smooth outputs, which required extensive tuning.
- **Object Detection:** Object detection feature for identifying suspicious items (e.g., phones, notes) in the /process_periodic endpoint. Developing a YOLO-based model required collecting and annotating a dataset, training the model, and integrating it into FastAPI, which proved too complex within the project scope. This limited the system's ability to detect certain cheating behaviors.
- **Local ML Processing Constraints:** Running ML models locally without a dedicated GPU slowed down image processing, with the /process_periodic endpoint averaging 0.9 seconds per image. Optimizations like reducing image resolution to 640x640 and batching inference improved performance, but hardware limitations remained a bottleneck.

These model-related issues were addressed by optimizing storage, refining preprocessing, and prioritizing core features, though the absence of object detection remains a limitation for future work.

10.3. Lessons Learned

The technical and model-related challenges provided critical insights for future development. The Laravel middleware bug and FastAPI setup issues emphasized the need for thorough unit testing and dependency management. Image handling and UI design challenges highlighted the importance of standardized data formats and user-centric design principles. Clean code practices in React and Laravel underscored the value of modular architecture and coding standards. Model integration difficulties reinforced the need for scalable storage and incremental feature development. Future iterations will benefit from cloud deployment, GPU acceleration, and the implementation of object detection to address these challenges comprehensively.

11. Conclusion

The face recognition and cheating detection platform represents a robust solution for enhancing the integrity and efficiency of online examinations. Developed in a local environment, the system integrates a FastAPI machine learning backend, a Laravel backend, and a React frontend to deliver secure identity verification, comprehensive cheating detection, and intuitive user interfaces. This chapter summarizes the projects key achievements, evaluates its potential impact on online education, identifies limitations encountered, and proposes future enhancements to further strengthen the platforms capabilities.

11.1. Summary of Achievements

The project successfully delivered a fully functional platform that meets its objectives of secure identity verification, real-time cheating detection, and role-based access control. Key achievements include:

- **Comprehensive Machine Learning Implementation:** The system incorporates face detection (YOLO), face recognition, head pose estimation, and object detection models, enabling robust cheating detection. The /process_periodic endpoint detects multiple faces, non-frontal poses, and suspicious objects (e.g., phones, notes), with facial embeddings efficiently stored in FaceDB using NumPy arrays. Challenges like head pose integration and embedding storage were resolved through preprocessing calibration and optimized storage solutions, as detailed in the Challenges Faced chapter.
- **Integrated System Architecture:** The platform seamlessly combines FastAPI for ML tasks (e.g., /register, /process_periodic), Laravel for user management, quiz administration, and notifications (e.g., /api/quizzes/submit/{quiz_id}, NotificationService), and React for dynamic interfaces (e.g., QuizInterface, ShowCheaters). API integrations, such as

image transfer from FastAPI to Laravels CheatingLog model, were optimized to handle payload size and decoding issues, ensuring reliable data flow.

- **Secure Role-Based Access Control (RBAC):** The RoleMiddleware in Laravel and SecureRoute in React enforced role-specific access for admin, professor, and user roles. A middleware bug causing incorrect role checks was resolved by debugging the JWT payload and adding unit tests, ensuring robust security.
- **User-Friendly Interfaces:** The React frontend delivered intuitive components, including QuizInterface for students, ShowCheaters for professors, and NotificationsPage for notifications. Challenges like cluttered UI layouts and combobox data handling were addressed using Material-UI, CSS Grid, and Redux state management, validated through local testing and screenshots in the Testing and Evaluation chapter.
- **Clean and Maintainable Codebase:** Laravel adopted service classes, repository patterns, and PSR-12 standards, while React implemented a feature-based structure with Redux, overcoming initial code coupling and duplication. Challenges like FastAPI dependency conflicts and Laravel image storage in public/storage were resolved through virtual environments and storage link configurations.
- **Local Testing and Validation:** Rigorous local testing confirmed functionality, with all test cases passing for API endpoints, WebSocket notifications, and UI interactions. Issues like image database storage and combobox value transmission were resolved, ensuring a reliable system in the local environment.

These achievements highlight the successful development of a sophisticated platform, overcoming significant technical challenges to deliver a functional and user-centric solution.

11.2. System Impact

The platform offers substantial potential to transform online education by enhancing exam integrity and improving user experiences for students, professors, and administrators.

- **Enhanced Exam Integrity:** The integration of face recognition, head pose estimation, and object detection ensures comprehensive cheating detection, identifying behaviors like multiple faces, non-frontal poses, and suspicious objects. Features like automatic quiz submission via /submit_due_to_cheating and detailed logs in ShowCheaters empower professors to maintain fair assessments, fostering trust in online education.
- **Improved User Experience:** The QuizInterface streamlines quiz-taking with timers and progress bars, while professors benefit from intuitive question management (add, update, delete) in CourseQuizzes. Administrators efficiently manage users and courses, and notifications via NotificationsPage enhance engagement. These features create a seamless and engaging experience.
- **Scalable Framework:** The modular architecture, with clean React components and Laravel services, provides a foundation for future enhancements. The systems use of ML and web technologies positions it as a model for intelligent educational tools, applicable to remote training or certification programs.

Developed locally, the platforms design suggests significant impact potential upon deployment, offering a scalable solution for educational institutions to secure and streamline online assessments.

11.3. Identified Limitations

Despite its achievements, the platform faces limitations due to its local development context and project scope constraints.

- Local Testing Constraints: Testing was conducted exclusively in a local environment, limiting the evaluation of real-world performance, such as network latency or scalability with multiple concurrent users. For example, ML processing in /process_periodic (0.9 seconds per image) was constrained by local CPU resources, as noted in the Challenges Faced chapter.
- Limited User Feedback: Usability testing relied on informal developer evaluation, with UI validation based on local screenshots and self-assessment, as described in the Testing and Evaluation chapter. Formal user testing with students, professors, and administrators was not conducted, restricting insights into real-world user experiences.
- Incomplete Features: Planned features like multi-language support and AI-driven analytics for the professor dashboard were deferred due to project constraints. These omissions limit the platforms accessibility and analytical capabilities.

These limitations, further detailed in the Challenges Faced chapter, highlight areas for improvement to fully realize the platforms potential in a deployed setting.

11.4. Future Enhancements

To address the identified limitations and expand the platforms capabilities, the following enhancements are proposed:

- **Cloud Deployment:** Deploying the system on a cloud platform (e.g., AWS, Azure) will enable scalability testing, optimize ML processing with GPU acceleration, and support real-world network conditions, addressing local testing constraints and improving endpoint performance.
- **Object Detection Optimization:** While object detection has been implemented, further optimization (e.g., expanding the dataset to include

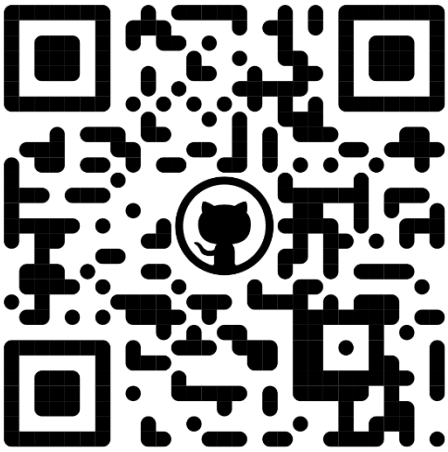
diverse objects, fine-tuning the YOLO model) will enhance detection accuracy for suspicious items, strengthening cheating detection capabilities.

- **Formal User Testing:** Conducting usability testing with a diverse group of students, professors, and administrators will provide comprehensive feedback on components like QuizInterface and ShowCheaters, informing refinements to improve user experience.
- **Feature Expansion:** Implementing multi-language support will increase accessibility, while AIdriven analytics (e.g., predicting cheating patterns) will enhance the professor dashboard. Adding bulk user creation and advanced question types will further streamline operations.
- **Privacy and Security Enhancements:** Implementing end-to-end encryption, anonymized data handling, and compliance with regulations like GDPR will address potential privacy concerns related to webcam monitoring and data storage, building user trust in a deployed environment.

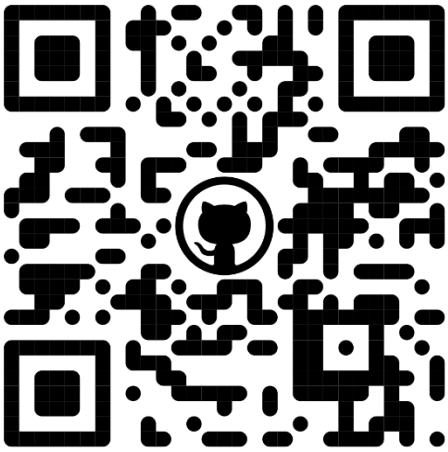
These enhancements will elevate the platform into a robust, scalable, and user-centric solution, fully realizing its potential to revolutionize online education.

12. Code Links

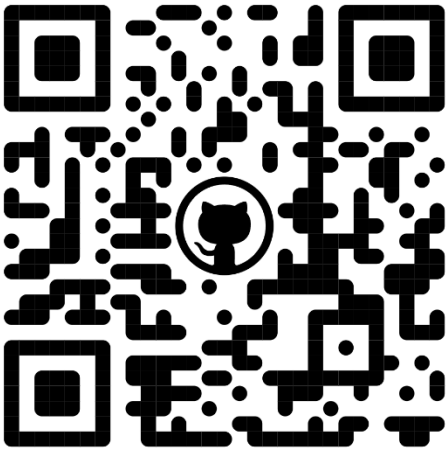
1) **Backend Repo**



2) **Frontend Repo**



3) **ML APIs Repo**



13. References

- ProctorU Documentation
- Honorlock Documentation
- Examity Documentation
- Moodle Documentation (https://docs.moodle.org/405/en/Features)
- ProctorU Student Guide (PDF) from SOWELA Technical Community College (https://www.sowela.edu/wp-content/uploads/ProctorU-Student-Guide.pdf)
- ProctorU Live+ How it Works Faculty Process from University of South Carolina (https://sc.edu/about/offices_and_divisions/distributed_learning/documents/proctoru_howitworks_faculty.pdf)
- Test-Taker Guide from ProctorU (https://www.proctoru.com/wp-content/uploads/2019/03/Test-Taker-Guide_03.11.19.pdf)
- Honorlock Student Guide from University of Florida Credit: Warrington College of Business Teaching and Learning Center (https://pfs.tnt.aa.ufl.edu/media/dceufledu/pdfs/Honorlock-Student-Guide-UF-Update.pdf)
- Honorlock: Instructor Guide from University Center for Assessment, Teaching & Technology (https://help.intech.arizona.edu/article/317-honorlock-user-guide)
- Honorlock - User Guide for Students – UknownIT (Self Service) (https://uknowit.uwgb.edu/page.php?id=124054)
- Examity® Online Proctoring from Blackboard (https://help.blackboard.com/Learn/Instructor/Ultra/Tests_Pools_Surveys/Examity)
- Examity Instructor Guide from Queen's University (https://www.queensu.ca/registrar/sites/uregwww/files/uploaded_files/pdfs/onQ_Examity_Instructor_Quick_Guide.pdf)
- Examity Proctoring Guide Administrators & Instructors from the University of North Carolina at Pembroke (https://www.uncp.edu/sites/default/files/2024-02/Examity%20guide%201_0.pdf)

- Examity Student Quick Guide For Stand-Alone Access (https://www.getdbt.com/assets/docs/certifications/Test_Taker_Standalone_Guide_dbtLabs.pdf)
- A Comparative Study of Deep Transfer Learning Algorithm for Cheating Detection in the Exam Based on Surveillance Camera Recording (A Comparative Study of Deep Transfer Learning Algorithm for Cheating Detection in the Exam Based on Surveillance Camera Recording | Proceedings of the 8th International Conference on Sustainable Information Engineering and Technology)
- Academic Integrity in Online Assessment: A Research Review (Frontiers | Academic Integrity in Online Assessment: A Research Review)
- Secure Online Examination with Biometric Authentication and Blockchain-Based Framework (Secure Online Examination with Biometric Authentication and Blockchain-Based Framework - Zhu - 2021 - Mathematical Problems in Engineering - Wiley Online Library)
- Trends of Artificial Intelligence for Online Exams in Education (Trends-of-Artificial-Intelligence-for-Online-Exams-in-Education.pdf)
- Dataset for Gaze Pose Detection
  - https://www.repository.cam.ac.uk/items/f9464c88-5797-4f28-824c-10003a9704f7
- Dataset for Face Detection Model
  - http://shuoyang1213.me/WIDERFACE/
- Dataset for Head Pose Estimation Model
  - https://www.kaggle.com/datasets/kmader/biwi-kinect-head-pose-database
  - http://www.cbsr.ia.ac.cn/users/xiangyuzhu/projects/3DDFA/main.htm