

VIVABOT



Major project submitted in partial fulfillment of the requirement for the award of the
degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

Under the esteemed guidance of

Dr A Sree Lakshmi
Professor

By

A PRANEETH KUMAR REDDY (21R11A05L6)
BADUGU JESSY (21R11A05M0)



Department of Computer Science and Engineering

Geethanjali College of Engineering and Technology (Autonomous)

Accredited by NAAC with A⁺ Grade: B.Tech. CSE, EEE, ECE accredited by NBA Sy. No: 33 & 34,
Cheeryal (V), Keesara (M), Medchal District, Telangana – 501301

MAY – 2025

Geethanjali College of Engineering and Technology (Autonomous)

Accredited by NAAC with A⁺ Grade : B.Tech. CSE, EEE, ECE accredited by NBA Sy. No: 33 & 34, Cheeryal
(V), Keesara (M), Medchal District, Telangana – 501301

Department of Computer Science and Engineering



CERTIFICATE

This is to certify that the B.Tech Major Project report entitled “**VIVABOT**” is a bonafide work done by **A Praneeth Kumar Reddy(21R11A05L6), Badugu Jessy (21R11A05M0)**, in partial fulfillment of the requirement of the award for the degree of Bachelor of Technology in “**Computer Science and Engineering**” from Jawaharlal Nehru Technological University, Hyderabad during the year 2024-2025.

Dr. A Sree Lakshmi
Professor

HoD – CSE
Dr E Ravindra
Professor

External Examiner

Geethanjali College of Engineering and Technology (Autonomous)

Accredited by NAAC with A⁺ Grade : B.Tech. CSE, EEE, ECE accredited by NBA Sy. No: 33 & 34, Cheeryal
(V), Keesara (M), Medchal District, Telangana – 501301

Department of Computer Science and Engineering



DECLARATION BY THE CANDIDATE

We, **A Praneeth Kumar Reddy, Badugu Jessy**, bearing Roll Nos. **21R11A05L6, 21R11A05M0**, hereby declare that the project report entitled “**VIVABOT**” is done under the guidance of **Mrs. Dr. A. Sree Lakshmi, Professor**, Department of Computer Science and Engineering, Geethanjali College of Engineering and Technology, is submitted in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology in Computer Science and Engineering**.

This is a record of bonafide work carried out by us and the results embodied in this project have not been reproduced or copied from any source. The results embodied in this project report have not been submitted to any other University or Institute for the award of any other degree or diploma.

A Praneeth Kumar Reddy(21R11A05L6)

Badugu Jessy(21R11A05M0)

**Department of CSE, Geethanjali College of Engineering and Technology,
Cheeryal.**

ACKNOWLEDGEMENT

It is with profound gratitude and sincere appreciation that we extend our heartfelt thanks to all those who have played a significant role in the successful completion of this undergraduate project.

First and foremost, we express our deep sense of respect and gratitude to our **Honourable Chairman, Mr. G. R. Ravinder Reddy**, for his constant encouragement and for nurturing a culture of academic excellence and innovation within the institution.

We would also like to express our sincere thanks to **Dr. S. Udaya Kumar, Director**, for his visionary leadership and continued support, which have provided the ideal environment and motivation for academic pursuits such as this project.

Our heartfelt appreciation goes to **Dr. K. Sagar, Principal**, for his steadfast guidance, infrastructural support, and encouragement that have helped bring this project to successful fruition.

We are deeply grateful to **Dr. E Ravindra, Head of the Department of Computer Science and Engineering**, for his academic leadership, valuable feedback, and continuous support throughout the duration of this project.

We extend our sincere thanks to the **Project Coordinators, Dr K Srinivasa Reddy ,D Swaroopa**, for their organized planning, timely guidance, and constructive feedback which contributed immensely to the smooth and effective completion of the seminar.

A special note of appreciation is reserved for our guide, **Dr. A. Sree Lakshmi, Professor**, for her valuable insights, scholarly advice, and unwavering mentorship throughout the research and presentation phases of this project.

Lastly, we are ever grateful to our **parents and family** for their unconditional love, encouragement, and moral support. Their faith in me has been my greatest strength throughout this journey.

With genuine appreciation, we acknowledge every individual who has, in one way or another, contributed to the successful completion of this project.

With warm regards,
A Praneeth Kumar Reddy (21R11A05L6)
Badugu Jessy (21R11A05M0)
Department of Computer Science and Engineering
Geethanjali College of Engineering and Technology

ABSTRACT

VivaBot is an AI-powered viva examination system developed to simplify and automate the oral assessment process in academic environments. Traditional viva sessions are often time-consuming, subjective, and require constant faculty involvement. This project addresses these issues by integrating facial recognition, artificial intelligence, and adaptive learning algorithms to create a seamless and unbiased viva experience for both students and faculty. The system allows students to register by entering their personal details such as name, roll number, and class, followed by facial data capture, which is stored locally or in an Excel sheet. The login process relies on real-time face recognition, ensuring secure and personalized access. Once authenticated, students participate in a viva session where questions are dynamically generated based on the selected subject or topic. Faculty members can initiate sessions by uploading a PDF or entering topic-based inputs, which are processed through the Ollama AI model (using the Mistral LLM) to create relevant and subject-specific questions. The viva follows an adaptive question flow—starting with medium-difficulty questions, and adjusting to harder or easier ones depending on the student's responses. An AI-based evaluator analyzes the answers and assigns marks (0.5 for Easy, 1.0 for Medium, and 1.5 for Hard questions). The session concludes once the student accumulates a total score of 5 marks, after which their attendance and final marks are recorded. All data related to viva sessions and questions is stored in a database (MongoDB and MySQL), while student registration details are maintained in Excel sheets. At the end of the session, an Excel report is generated and emailed to the respective faculty after OTP verification. VivaBot ensures transparency, reduces faculty workload, and delivers accurate evaluation and feedback through advanced technologies, making it a modern solution for oral examinations in academic institutions.

List of figures

S.No	Figure No	Figure Name	Page No
1	1.5.1	Incremental Model	5
2	4.1.1	System Architecture	18
3	4.3.1	Usecase Diagram	24
4	4.3.2	Class Diagram	25
5	4.3.3	Activity Diagram	26
6	4.3.4	Sequence Diagram	27
7	4.3.5	Object Diagram	28
8	4.3.6	Dataflow Diagram	29
9	4.3.7	Deployment Diagram	30
10	4.3.8	Component Diagram	31
11	5.1.1	Technology Stack	36
12	7.1.1	Index Page	63
13	7.1.2	Home Page	63
14	7.1.3	Faculty Dashboard	64
15	7.1.4	PDF Questions Upload	64
16	7.1.5	Generate Questions	65
17	7.1.6	Compute Final Marks	65
18	7.1.7	Computed Final Sheet	66
19	7.1.8	Student Registration	66
20	7.1.9	Face Registration	67
21	7.1.10	Viva Sessions	67
22	7.1.11	Selection of Weeks	68

23	7.1.12	Student Login	68
24	7.1.13	Viva Questions	69
25	7.1.14	Student Viva Feedback	69
26	7.1.15	Ending Viva Session	70
27	7.1.16	Attendance and Marks Sheet	70
28	10.1	Gantt Chart	81

List of Tables

S.No	Table No	Table Name	Page No
1	2.4.1	Comparative Study	11
2	3.2.1	Applications of Software	15
3	4.2.1	Viva_sessions table	22
4	4.2.2	Questions table	23
5	6.5.1	Testcases and Results	58
6	6.6.1	Bug Reporting and Tracking	61
7	7.4.1	Comparision Table	73
8	10.1	SDLC Froms	80

List of Abbreviations

Abbreviation	Full Form
AI	Artificial Intelligence
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
SDLC	Software Development Life Cycle
CRUD	Create, Read, Update, Delete
LLM	Large Language Model
SQL	Structured Query Language

Table of Contents

S.No	Contents	Page No
	Title Page	i
	Certificate	ii
	Declaration	iii
	Acknowledgement	iv
	Abstract	v
	List of Figures	vi
	List of Tables	viii
	List of Abbreviations	viii
	Table of Contents	ix
1	Introduction	1 - 6
	1.1 Overview of the Project	1
	1.2 Problem Statement	2
	1.3 Objectives of the Project	3
	1.4 Scope of the Project	4
	1.5 SDLC Model Adopted	5
	1.6 Organization of the Report	6
2	Literature Survey	8 - 10
	2.1 Review of Existing System	8
	2.2 Limitations of Existing Approaches	8
	2.3 Need for Proposed System	9
	2.4 Comparative Study	10
3	System Analysis	12 - 15

3.1 Feasibility Study	12
3.1.1 Technical Feasibility	12
3.1.2 Economic Feasibility	12
3.1.3 Operational Feasibility	13
3.1.4 Time and Cost Estimation	13
3.2 Software Requirements Specification (SRS)	13
3.3 Functional and Non-Functional Requirements	15
4 System Design	18 - 35
4.1 System Architecture	18
4.2 Database Design	21
4.3 UML Diagrams	24
4.4 User Interface Design	32
4.5 Design Standards Followed	34
4.6 Safety and Risk Mitigation Measures	35
5 Implementation	36 - 41
5.1 Technology Stack	36
5.2 Module-wise Implementation	38
5.3 Code Integration Strategy	39
5.4 Sample Code Snippets	41
6 Testing	55 - 61
6.1 Testing Strategy	55
6.2 Unit Testing	55
6.3 Integration Testing	56
6.4 System Testing	57
6.5 Test Cases and Results	58

	6.6 Bug Reporting and Tracking	61
7	Results and Discussion	63 - 73
	7.1 Output Screens	63
	7.2 Results Interpretation	71
	7.3 Performance Evaluation	72
	7.4 Comparative Results	73
8	Conclusions and Future Scope	74 - 77
	8.1 Summary of Work Done	74
	8.2 Limitations	74
	8.3 Challenges Faced	75
	8.4 Future Enhancement	76
9	References	78
10	Appendices	80 - 84
	A. SDLC Forms	80
	B. Gantt Chart	81
	C. Ethical Considerations and Consent	82
	D. Plagiarism Report	83
	E. Source Code Repository	83
	F. Journal / Conference paper published on project	84

CHAPTER 1 INTRODUCTION

1.1 OVERVIEW OF THE PROJECT

VivaBot is an innovative and intelligent platform designed to automate and enhance the viva voce examination process in educational institutions. By leveraging cutting-edge technologies such as facial recognition, natural language processing, and automation tools, VivaBot aims to revolutionize the traditional viva process, making it more efficient, transparent, and scalable.

This platform eliminates manual intervention by automating key steps such as student identification, question assignment based on proficiency, real-time answer evaluation, and result generation. It empowers faculty and students with a streamlined approach to conduct and participate in viva exams, ensuring a smooth, fair, and effective process.

With facial recognition technology, VivaBot ensures accurate student identification, enabling attendance management and avoiding impersonation. The platform uses a centralized question bank categorized by difficulty levels and subjects, assigning questions tailored to each student's proficiency. The integration of AI for answer evaluation ensures objectivity and consistency in marking.

VivaBot also simplifies administrative tasks by automating attendance tracking, generating detailed viva performance reports, and providing real-time updates on exam progress. This ensures that faculty members can focus on engaging with students, while the platform handles the operational complexities.

Ultimately, VivaBot seeks to redefine the viva voce process, fostering a more collaborative, efficient, and technologically advanced educational environment.

1.2 PROBLEM STATEMENT

In educational institutions, viva examinations are an essential part of student evaluation, especially for technical and practical subjects. However, the traditional method of conducting viva is highly manual and time-consuming. Faculty members need to prepare a set of questions for each subject, evaluate every student individually, and assess both the accuracy and confidence of their answers. This process becomes even more challenging when dealing with a large number of students, leading to inconsistent evaluations and increased workload for the faculty.

Moreover, there is a lack of standardization in the way questions are asked and answers are evaluated. Different faculty members may ask different questions or evaluate responses with varying levels of strictness. This introduces a bias in the examination system and can affect the fairness and accuracy of student assessment. There is also no formal mechanism to analyse the confidence level of students, which plays a major role in oral exams.

With the rapid growth of technology, especially in the field of Artificial Intelligence, there is a clear opportunity to automate these tasks and make the viva process more objective and efficient. A system is needed that can generate relevant questions, evaluate student answers automatically, and also observe student behaviour, such as their confidence, in a consistent and scalable way.

The problem, therefore, is to design and implement a system that addresses all these challenges by combining AI-driven question generation, automated answer evaluation, and sentiment analysis. Such a system would not only reduce the workload of faculty but also ensure fair, fast, and intelligent assessment for students, improving the overall quality of the viva examination process.

1.3 OBJECTIVES OF THE PROJECT

- **Automation of Viva Examination Process:** Eliminate manual efforts in conducting viva exams by automating key processes such as student identification, question assignment, answer evaluation, and result compilation.
- **Facial Recognition for Attendance:** Use facial recognition technology to ensure accurate and seamless student identification and attendance tracking.
- **Proficiency-Based Question Assignment:** Fetch and assign questions dynamically based on each student's subject knowledge and previous performance.
- **Integration with AI:** Evaluate students' answers using the Ollama Mistral Offline AI model to provide consistent and unbiased grading.
- **Centralized Question Management:** Maintain a categorized question bank by subject and difficulty level to enable efficient question retrieval and assignment.
- **Real-Time Marks Update:** Automatically update marks in an Excel sheet after evaluating each student's answers.
- **Transparent Tracking:** Provide students and faculty with real-time updates on viva progress, including marks, attendance, and question details.
- **Enhanced Communication and Collaboration:** Facilitate seamless communication between students and faculty through a user-friendly web interface.
- **Streamlined Administrative Tasks:** Reduce the workload of faculty and administrative staff by automating attendance, marking, and report generation.
- **Improved Efficiency:** Optimize the viva process by ensuring swift response times, accurate evaluations, and effective resource utilization.

1.4 SCOPE OF THE PROJECT

The Vivabot project is designed to offer a smart, web-based solution for automating viva voce examinations in academic institutions. Its main objective is to replace the manual viva process with an AI-driven system that handles question generation, answer evaluation, and confidence assessment efficiently. This platform improves accuracy, reduces human bias, and saves time for faculty members.

The project covers the following core functionalities:

- **Automated Question Generation:** Faculty can input a topic and number of questions, and the system dynamically generates relevant questions using the Ollama Mistral AI model.
- **Answer Evaluation:** Student responses are evaluated in real-time using a generative AI model that checks for relevance, correctness, and completeness.
- **Sentiment-Based Confidence Scoring:** Facial expression analysis is used to determine student confidence and award an additional mark where applicable.
- **Facial Recognition for Attendance:** Automatic detection and recording of student attendance during the viva session.
- **Automated Result Generation:** Instant calculation and storage of student performance scores based on content accuracy and sentiment analysis.

The platform is built using a robust tech stack:

- **Frontend:** HTML, CSS, JavaScript for building an interactive user interface.
- **Backend:** Flask (Python) for handling logic, AI model integration, and system workflows.
- **Database:** MySQL for structured storage of user data, questions, answers, and results.
- **AI Integration:** Ollama Mistral model for question generation and answer evaluation, along with sentiment analysis tools.

1.5 METHODOLOGY

The development of the VivaBot project followed the **Incremental Software Development Life Cycle (SDLC) Model**. This model was selected due to its flexibility and suitability for modular academic projects where different components can be developed, tested, and improved independently.

In the Incremental model, the software is built in parts or “increments.” Each increment goes through the complete development cycle (requirement analysis, design, implementation, testing), and then integrated with previously developed modules. This allowed us to deliver working functionalities early and incorporate feedback continuously throughout the development process.

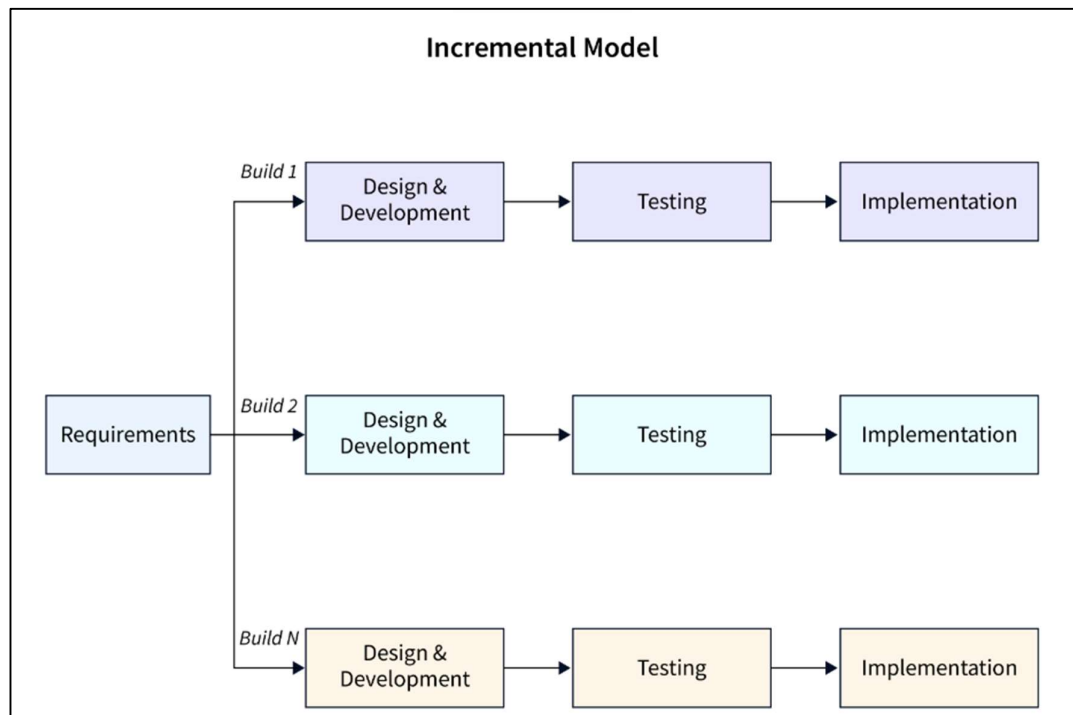


Fig 1.5.1 Incremental Model

Key Reasons for Choosing the Incremental Model:

- Enables progressive development and testing of individual modules.
- Allows early delivery of core functionalities for feedback.
- Suitable for academic projects that evolve over time.

- Easier to isolate and fix issues within specific modules.

VivaBot Development Phases:

1. **Requirement Analysis** – Identified core features such as facial recognition login, viva session management, question generation, and feedback analysis.
2. **Design** – Created user flow diagrams, UI wireframes, and planned the database schema including viva session and subject-specific question tables.
3. **Implementation** – Developed modules incrementally using technologies like Python flask. For example, student registration and question generation were built and tested separately before integration.
4. **Testing** – Performed unit testing on each module followed by integration testing to ensure seamless interaction between components.
5. **Deployment** – Deployed the application on a local server for testing and demonstration, with all modules functioning cohesively.
6. **Maintenance** – Prepared the system for future improvements, such as enhanced analytics and live dashboard features.

By adopting the Incremental model, we ensured that the VivaBot project was developed in manageable parts, reducing risks and allowing for continuous improvement throughout the project lifecycle.

1.6 ORGANIZATION OF THE REPORT

This report is structured into ten well-defined chapters, each focusing on a specific phase of the Vivabot project to ensure a comprehensive understanding of the system's development, analysis, and implementation.

- **Chapter 1** provides an introduction to the project, including its overview, problem statement, objectives, scope, the adopted SDLC model, and the organization of the report itself.
- **Chapter 2** presents a literature survey, analysing existing systems, identifying their limitations, and establishing the need for the proposed system.

- **Chapter 3** focuses on system analysis, including feasibility studies, software requirements, and both functional and non-functional requirements.
- **Chapter 4** discusses system design, showcasing architecture diagrams, ER models, UML diagrams, and user interface design standards.
- **Chapter 5** elaborates on the implementation details, technology stack, module-wise development, integration approach, and coding practices.
- **Chapter 6** details the testing process, including different levels of testing, test cases, results, and quality assurance measures.
- **Chapter 7** highlights the results and performance of the system with supporting screenshots and comparative evaluations.
- **Chapter 8** concludes the project by summarizing the work done, discussing challenges faced, and suggesting future enhancements.
- **Chapter 9** lists the references, including technical papers, websites, and forums consulted during development.
- **Chapter 10** includes appendices such as additional diagrams, screenshots, and supporting documents relevant to the project.

CHAPTER 2 LITERATURE SURVEY

2.1 REVIEW OF EXISTING SYSTEM

The current viva voce process in most educational institutions relies heavily on traditional, manual methods. This includes manual attendance marking, question assignment, answer evaluation, and result compilation. Faculty members conduct viva exams in a one-on-one or small group setting, often using printed or verbal questions, which are evaluated subjectively. While this system has been the standard for years, it suffers from several limitations that reduce its efficiency and accuracy. The lack of automation and technology integration makes the process time-consuming and prone to inconsistencies, ultimately impacting the overall experience for both students and faculty.

2.2 LIMITATIONS OF EXISTING APPROACHES

1. **Manual Processes:** Attendance marking, question assignment, and answer evaluation are conducted manually, leading to inefficiencies, delays, and the potential for human error.
2. **Subjective Answer Evaluation:** Faculty evaluate students' answers subjectively, which may lead to inconsistencies and a lack of standardized marking criteria.
3. **No Proficiency-Based Question Assignment:** Questions are assigned randomly without considering the student's past performance or proficiency levels, reducing the effectiveness of the evaluation process.
4. **Time-Intensive Procedures:** Manual processes prolong the time required to conduct and complete viva exams, making it difficult to scale the process for larger groups of students.
5. **Lack of Transparency:** Students often lack visibility into their performance or the evaluation process, leading to uncertainty and dissatisfaction.
6. **No Centralized Management:** The absence of a centralized platform makes it difficult to manage attendance, questions, and results efficiently, creating additional administrative burdens for faculty.

2.3 NEED FOR PROPOSED SYSTEM

VivaBot is a cutting-edge, automated platform designed to address the shortcomings of the existing viva voce process. By incorporating advanced technologies such as facial recognition, natural language processing, and automation tools, VivaBot streamlines and enhances the entire viva process, ensuring efficiency, consistency, and transparency.

Benefits of the Proposed System:

1. **Automated Attendance Management:** VivaBot uses facial recognition technology to automate attendance marking, ensuring accuracy and eliminating the need for manual intervention.
2. **Proficiency-Based Question Assignment:** The system dynamically assigns questions based on each student's previous performance and proficiency level, enabling a more personalized and effective evaluation.
3. **Automated Answer Evaluation:** Integration with AI allows VivaBot to evaluate student answers objectively, ensuring consistency and reducing the workload on faculty.
4. **Centralized Viva Management:** VivaBot provides a single platform for managing all aspects of the viva process, including attendance, question assignment, answer evaluation, and result tracking, improving overall efficiency.
5. **Time Efficiency:** By automating key steps, VivaBot significantly reduces the time required to conduct viva exams, making it scalable for larger groups of students.
6. **Comprehensive Data Analytics:** The platform provides detailed performance insights and analytics, enabling faculty to identify patterns, track progress, and make data-driven decisions for curriculum improvement.

2.4 COMPARATIVE STUDY

S. No	Author(s)	Title of Paper/Study	Year	Key Takeaways
1	S. S. Iyengar et al.	AI-Based Smart Education Systems	2020	Highlights use of AI for automating academic processes including assessment.
2	A. K. Jain et al.	Face Recognition for Secure Student Identification	2019	Discusses facial recognition as a reliable authentication method in education.
3	M. Patel et al.	Web-Based Examination System with Auto Evaluation	2021	Introduces web platforms for seamless viva and written evaluations.
4	R. Mehta and S. Gupta	Challenges in Traditional Viva Examinations	2018	Explores inefficiencies and inconsistencies in manual viva methods.
5	N. Sharma and K. Roy	Real-Time Face Detection Using Deep Learning	2022	Uses real-time image processing for authentication in online systems.

S. No	Author(s)	Title of Paper/Study	Year	Key Takeaways
6	T. Bose and H. Singh	Smart Academic Portals for Assessment and Feedback	2020	Discusses academic tools that provide structured feedback and automate grading.
7	P. Reddy and A. Kulkarni	Design and Implementation of Viva Evaluation System	2021	Explains a prototype system for structured viva evaluation.
8	L. Verma et al.	Student Performance Analytics Using AI Tools	2022	Describes data-driven performance tracking in educational systems.
9	V. Kumar and D. Sharma	Secure Cloud-Based Student Assessment Platform	2020	Emphasizes security and scalability for online viva systems.
10	B. Das and R. Ghosh	Integration of Face Recognition in Academic Authentication Systems	2019	Integrates biometric verification into academic tools for secure access.

Table 2.4.1 Comparative Study

CHAPTER 3 SYSTEM ANALYSIS

3.1 FEASIBILITY STUDY

3.1.1 TECHNICAL FEASIBILITY

The technical feasibility assesses whether the existing technical resources are sufficient to build and run the system.

- The VivaBot project uses technologies like Python, OpenCV, Flask, MySQL, and HTML/CSS/JS—all of which are open-source and well-documented.
- Face recognition implementation using OpenCV and dlib is supported on most standard hardware with a webcam.
- Development was carried out on standard laptops, and the system performs well with minimal hardware requirements.

Feasible: The required tools and skills were available, and the team was familiar with the technology stack.

3.1.2 ECONOMIC FEASIBILITY

This evaluates the cost-effectiveness of the project.

- VivaBot was developed using free and open-source technologies, minimizing licensing or infrastructure costs.
- No additional hardware or servers were needed.
- The development was carried out as an academic project with no labour costs involved.

Feasible: The project incurred no significant expenses and remained within budget constraints.

3.1.3 OPERATIONAL FEASIBILITY

This examines whether the proposed system will work in the intended operational environment.

- The system is designed to be used by students and faculty within a college or institutional setting.
- The user interface is simple and intuitive, requiring no specialized training.
- Tasks such as student registration, face recognition login, and viva conduction can be easily managed.

Feasible: Users can comfortably adapt to the system with minimal training.

3.1.4 TIME AND COST ESTIMATION

Estimates were made for development and testing phases to ensure timely completion.

- **Timeframe:** The project was planned over a span of approximately 8–10 weeks, including design, coding, testing, and documentation.
- **Cost:** As a student project, costs were negligible. Only time and effort were invested.

Feasible: The project was completed within the scheduled time with effective resource utilization.

3.2 SOFTWARE REQUIREMENTS SPECIFICATION (SRS)

This section outlines the essential software requirements for the successful design, development, and deployment of the **VivaBot** system. It highlights the key technologies used, their purposes, and the modules where each was applied to automate the viva voce process using face recognition and question generation.

Software Requirements Used in the Project

The following tools and technologies were utilized in the development of VivaBot:

- Python 11
- Flask

- OpenCV
- dlib
- MySQL
- HTML / CSS / JavaScript

These tools together enabled frontend design, server-side logic, face recognition, question management, database operations, and UI responsiveness.

Purpose of the Software Requirements

Each tool was selected based on its suitability to support the system's objectives of automation, real-time performance, and usability:

- **Python 11** served as the core development language for backend logic, facial recognition, and integration of modules.
- **Flask** was chosen as a lightweight web framework to handle routing, request handling, and API integration.
- **OpenCV** enabled real-time webcam access and image processing for face detection.
- **dlib** was used for encoding and comparing facial features for student authentication.
- **MySQL** provided a relational database backend to store user data, registration details, question banks, and results.
- **HTML / CSS / JavaScript** were used to build interactive and responsive web pages for both students and faculty.

Application of Software to Project Modules

Technology	Modules Involved
Python 11	Core logic for facial recognition, student verification, random question generation.
Flask	Backend APIs, student/faculty routing, database interaction, session management.
OpenCV	Webcam integration, face capture, and real-time detection during login/registration.
dlib	Face encoding comparison for student login authentication.
MySQL	Used in modules like Student Registration, Faculty Login, Question Storage, Results.
HTML/CSS/JS	Frontend for Login, Registration, Viva Interface, Faculty Dashboard.

Table 3.2.1 Applications of Software used

3.3 FUNCTIONAL AND NON-FUNCTIONAL REQUIREMENTS

In order to ensure the successful design, development, and deployment of VivaBot, it is crucial to define both functional and non-functional requirements. These requirements will guide the implementation of core features as well as system performance, usability, and security aspects, ensuring that the platform delivers an efficient, transparent, and scalable viva examination process.

3.3.1 FUNCTIONAL REQUIREMENTS

1. Student Registration and Authentication

- The system must allow students to register using their face recognition and roll number.
- During the viva session, the system must authenticate students using face recognition and roll number to mark their attendance.

2. Viva Session Setup by Faculty

- Faculty must be able to create and configure a viva session by selecting the subject, weeks for which the viva session is conducted, and the questions.
- Faculty can either upload a PDF containing pre-defined questions or provide a topic, and the system should automatically generate questions using the Ollama AI model (Mistral).

3. Adaptive Questioning

- The system must start with a set of medium-difficulty questions.
- Based on student responses, the system should dynamically adjust the difficulty level of questions (increase or decrease) to assess the student's proficiency effectively.

4. Question Evaluation

- The system must evaluate student responses using the Ollama AI model (Mistral) to provide accurate, objective grading.
- The grading system must ensure the evaluation is based on student proficiency and not biased.

5. Attendance Management

- The system should automatically record attendance based on the successful authentication of the student during login.
- Attendance should be linked with the viva session for accurate tracking.

6. Faculty Interface

- Faculty must be able to monitor ongoing viva sessions, including the progress of individual students.
- Faculty should have the ability to view and adjust session settings, such as difficulty levels and question generation methods.

7. AI-Generated Questions

- If the faculty provides a topic, the system must generate relevant questions based on the topic using the Ollama AI model.
- The questions generated should be relevant, accurate, and aligned with the subject's syllabus.

8. Viva Session Reporting

- The system must generate detailed reports for faculty, showing student performance, attendance, and proficiency progression.
- The report should be easily exportable in a suitable format (e.g., CSV, PDF).

3.3.2 NON-FUNCTIONAL REQUIREMENTS

1. Usability

- The system must be user-friendly, with a simple interface for both faculty and students.
- Students should have minimal effort in registering and logging in using face recognition, making the system intuitive.

2. Performance

- The system must process student face recognition and logins within 5 seconds for optimal user experience.
- The adaptive questioning mechanism must adjust the difficulty level in real time with minimal delay.

3. Security

- The system must ensure that student data (face recognition data, roll numbers, grades) is securely stored and encrypted.
- Unauthorized users should not be able to access the system, and authentication must be enforced at every login.

CHAPTER 4 SYSTEM DESIGN

4.1 SYSTEM ARCHITECTURE

The system architecture of VivaBot provides a structured overview of how various components and users interact with each other to automate and streamline the viva examination process. The system is primarily designed for faculty and students, focusing on automation, adaptive questioning, AI integration, and ease of monitoring.

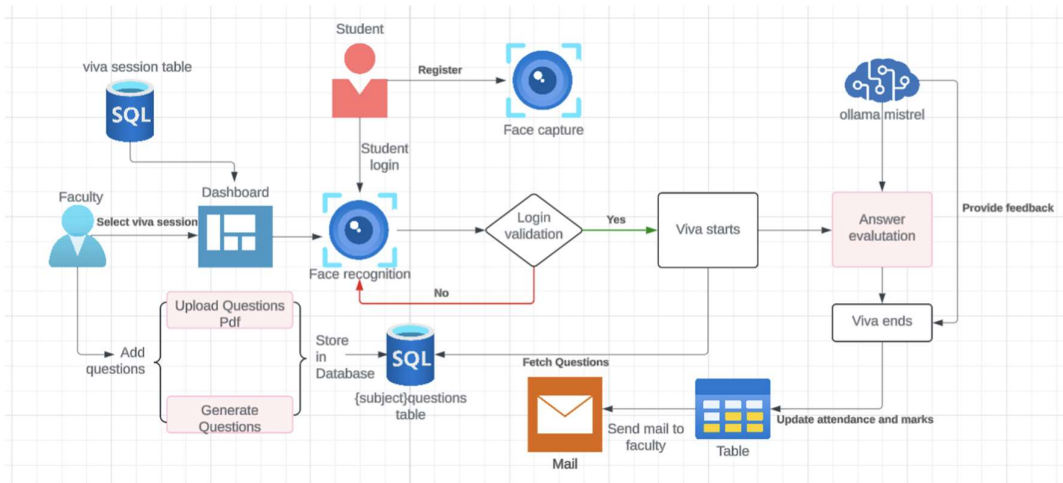


Fig 4.1.1 System Architecture

Key Users:

- **Faculty:** Manage viva sessions, upload questions, and receive results. No login/authentication is required.
- **Students:** Register using roll number, name, class, and face recognition. Log in to attend viva sessions.
- **System:** Handles adaptive learning, face recognition, question generation, scoring, and reporting.

1. Faculty Dashboard

- They directly access the Viva Session Dashboard to manage sessions.
- Two methods to upload questions to the database:
 1. Upload a PDF of questions.

2. Enter a topic → Questions are generated using Ollama (Mistral) offline AI model → Stored in the database.

- Faculty also select the session and choose week ranges from which questions should be pulled.

2. Student Registration and Login

- Students register with:
 - Name
 - Roll Number
 - Class
 - Face Recognition
- During login, students provide:
 - Roll Number
 - Face Recognition
- If face matches → Attendance is marked and viva session begins.

3. Viva Session Flow

- The viva uses an Adaptive Learning Algorithm:
 - Each student starts with a Medium difficulty question (1.0 mark).
 - Difficulty increases or decreases based on answers:
 - **Correct:**
 - Medium → Hard (1.5 mark)
 - Easy → Medium
 - Hard → Remains Hard
 - **Incorrect:**
 - Medium → Easy (0.5 mark)

- Hard → Medium
- Easy → Remains Easy
- Regardless of answer correctness, the question score accumulates based on current difficulty level.
- When total score reaches 5, the viva ends.

4. Feedback and Result Generation

- At the end of each viva student receives feedback about Confidence Score, Answer Quality and Answer feedback for the questions they have attempted or skipped.
- Attendance and marks are recorded in an Excel sheet.
- After all students finish, faculty ends the session using an OTP sent to their email.
- Upon verification, the Excel report is emailed to the faculty automatically.

5. Backend Components

- **Face Recognition Module:**
 - Handles student authentication and attendance.
- **Database:**
 - Stores student details, question bank, viva sessions, attendance, marks, feedback.
- **Ollama AI (Mistral Model):**
 - Generates questions based on topics provided by faculty.
- **Adaptive Learning Engine:**
 - Adjusts question difficulty dynamically and tracks score progression.

4.2 DATABASE DESIGN

The **database design** of VivaBot plays a central role in efficiently managing student data, viva session schedules, question banks, attendance, and results. The system uses a **MySQL database** named VivaBot, designed to support dynamic data retrieval, adaptive viva processes, and AI-generated question storage. The structure supports both faculty-controlled operations and real-time student interaction.

Database Name

- **Database:** VivaBot
- **Type:** Relational Database (MySQL)

The database supports essential operations such as:

- **CRUD operations** for sessions and questions.
- **Face-matching based student validation.**
- **Dynamic question filtering** by week, subject, and difficulty.
- **Logging of scores, attendance, and feedback** after each viva session.

Tables and Schema Description

1. viva_sessions

The **Viva Sessions Table** stores essential details about all the viva sessions scheduled by faculty. Each row represents a unique session, containing information such as the session name, the faculty who created it, and the scheduled date and time. Faculty can define which weeks or topics the questions will cover and specify the starting difficulty level for the session. Additionally, the table holds data on the marks assigned to questions based on their difficulty level. This table plays a central role in organizing and managing viva sessions, ensuring smooth coordination between faculty and students.

Column Name	Data Type	Constraints	Description
id	INT	PRIMARY KEY, AUTO_INCREMENT	Unique ID for each viva session.
day_of_week	VARCHAR(10)	NULLABLE	Day on which the session is scheduled.
class_name	VARCHAR(50)	NULLABLE	The class for which the viva is scheduled.
subject	VARCHAR(100)	NULLABLE	Subject of the viva (e.g., Machine Learning).
start_time	TIME	DEFAULT '12:00:00'	Scheduled start time of the viva session.
end_time	TIME	DEFAULT '14:00:00'	Scheduled end time of the viva session.
faculty_name	VARCHAR(100)	NULLABLE	Name of the faculty conducting the viva.
faculty_email	VARCHAR(255)	NOT NULL	Email of the faculty, used for OTP-based session end.
end_early	TINYINT(1)	DEFAULT 0	Indicates if the session ended early (1 = yes).

Table 4.2.1 viva_sessions table

2. Subject-Wise Question Tables (Example: ml_questions)

The **Subject-Wise Question Tables** store questions specific to each subject in separate tables. For instance, questions related to Machine Learning are stored in a table named **ml_questions**. This modular approach ensures that the system can scale efficiently, allowing new subjects to be added by simply creating a new table for each one. It also helps maintain data organization, as questions are categorized by subject, making it easier for faculty to manage and retrieve questions. This structure improves both the performance and scalability of the system as more subjects and corresponding questions are added over time.

Column Name	Data Type	Constraints	Description
qn_no	INT	PRIMARY KEY, AUTO_INCREMENT	Unique question number for the subject.
week	VARCHAR(10)	NOT NULL	The academic week to which the question belongs.
question	TEXT	NOT NULL	The viva question text.
proficiency_level	ENUM('Easy', 'Medium', 'Hard')	NOT NULL	The difficulty level of the question.

Table 4.2.2 questions Table

4.3 UML DIAGRAMS

USECASE DIAGRAM

The Use Case Diagram for VivaBot illustrates the interactions between two main actors: Student and Faculty, and the core functionalities within the system. Students can register, login using face recognition, select a viva session, answer questions, and view results. On the other hand, Faculty can upload questions via PDF or by generating them from topics, end the viva session, and view attendance and marks. The diagram shows how these actors interact with the system's features, offering a clear representation of the user's roles and tasks within the VivaBot system.

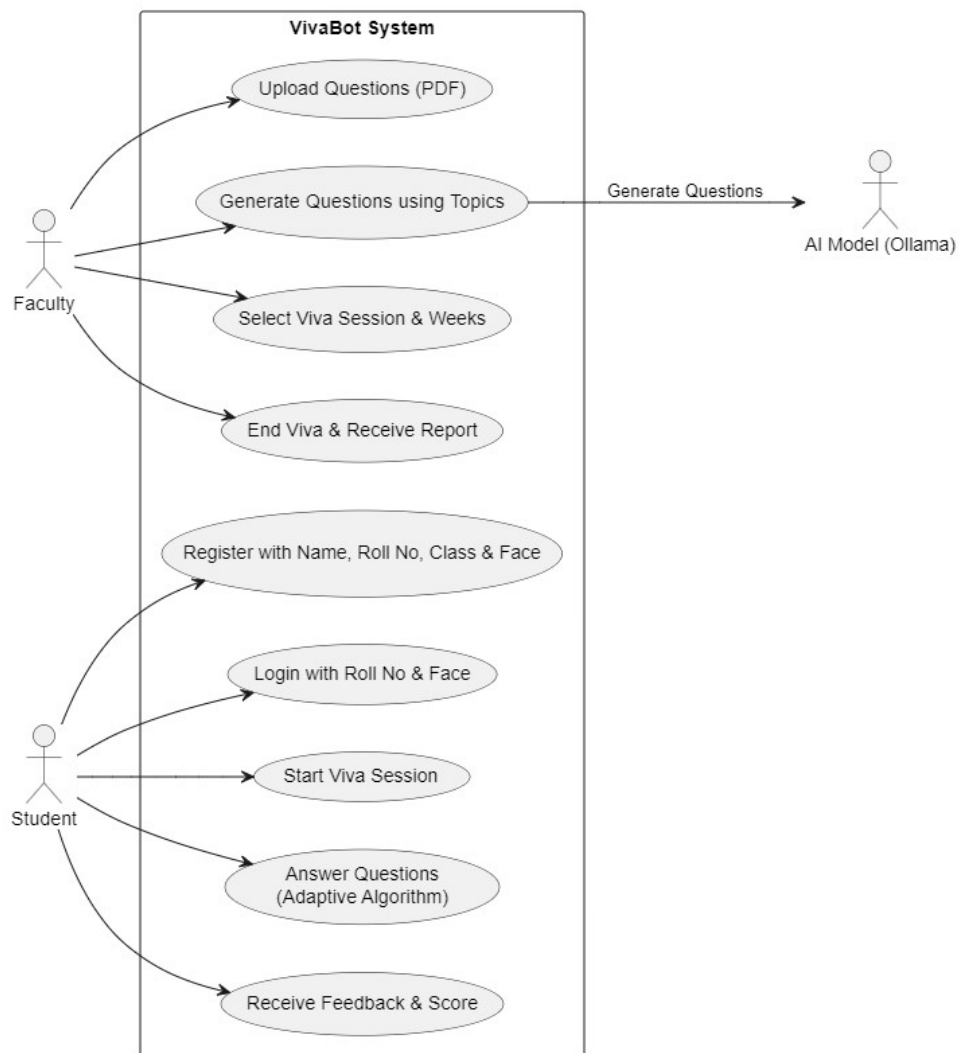


Fig 4.3.1 Usecase diagram

CLASS DIAGRAM

The class diagram for the VivaBot system provides a clear representation of the key components and their interactions. The **Student** class manages the registration, login, and viva session participation, leveraging face recognition for authentication. The **Faculty** class enables question uploads, either through PDFs or topic-based generation, and manages viva sessions. **VivaSession** holds session-specific data, linking students to the questions they will answer. The **Question** and **Answer** classes handle the generation, submission, and evaluation of responses, while the **AI Evaluator** adjusts question difficulty based on student performance, using an adaptive learning algorithm.

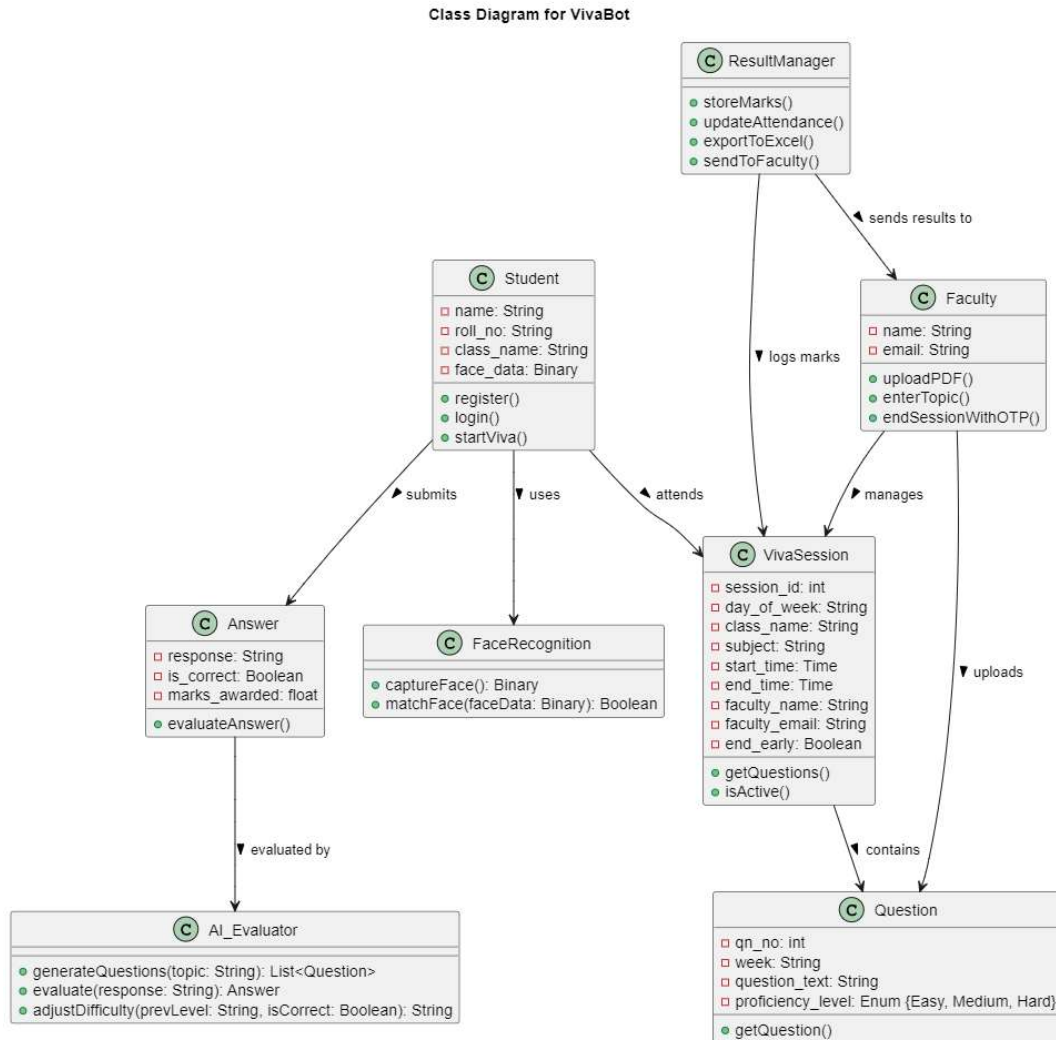


Fig 4.3.2 Class Diagram

ACTIVITY DIAGRAM

This activity diagram represents the flow of actions in the viva session, starting from student login via face recognition to logout after completion. It captures key stages like session selection, adaptive question answering, AI evaluation, result storage, and performance display, ensuring a smooth and automated viva process.

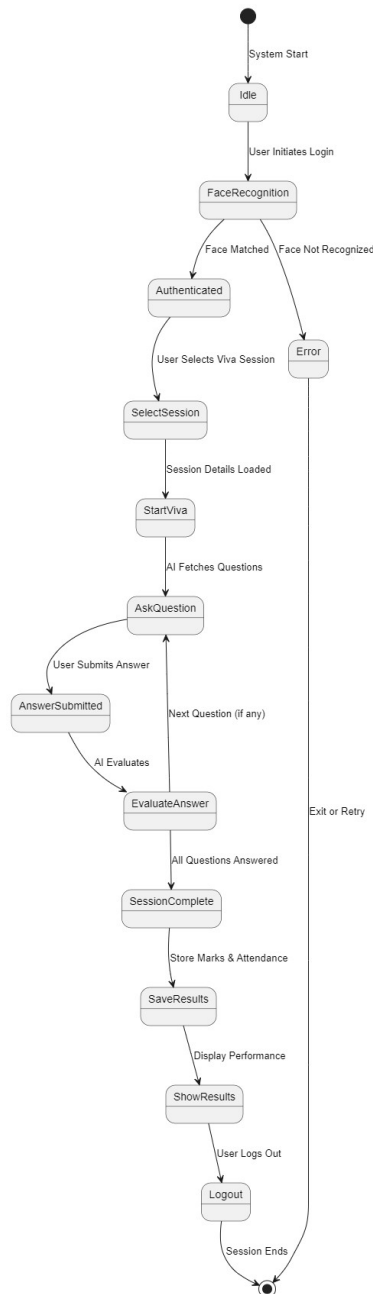


Fig 4.3.3 Activity Diagram

SEQUENCE DIAGRAM

The sequence diagram illustrates the step-by-step interaction between the student, the VivaBot system, the database, and the AI evaluator during the viva process. The student begins by logging in through face recognition, which is authenticated by the system via the database. Upon successful verification, the student selects an available viva session. Once selected, adaptive questions are generated in real-time using the AI model (Ollama with Mistral), and presented to the student. As students submit answers, the AI evaluates them and sends back the results, which are recorded along with attendance in the database. Finally, the system displays feedback and results before the student logs out. This flow ensures a secure, adaptive, and automated viva experience.

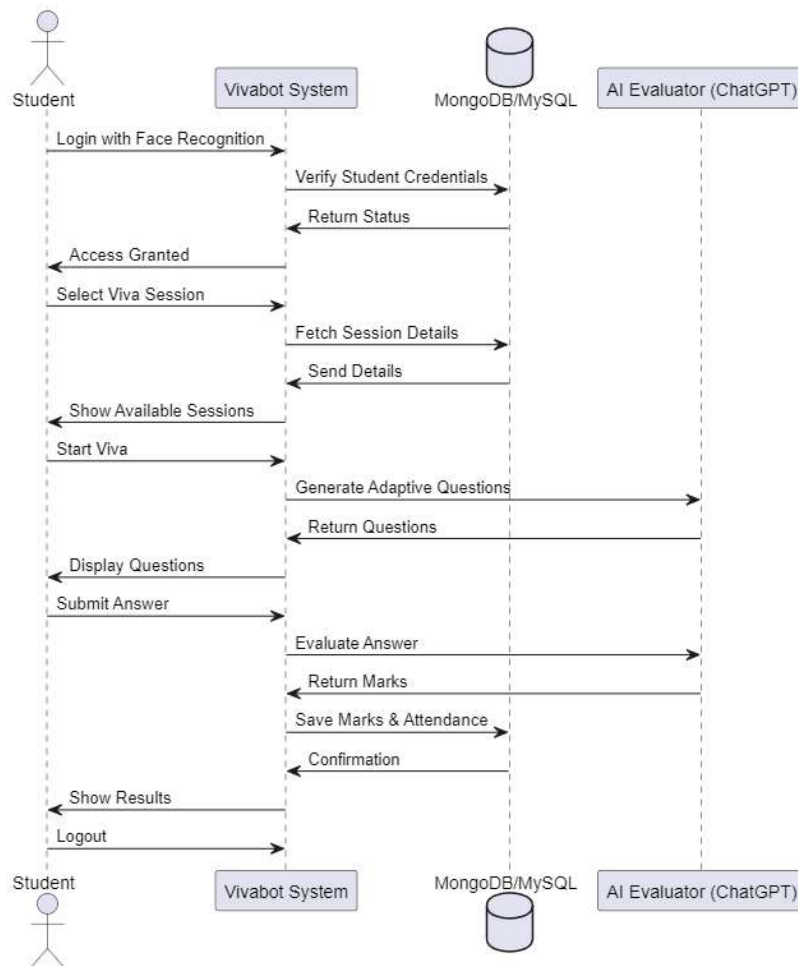


Fig 4.3.4 Sequence Diagram

OBJECT DIAGRAM

The object diagram provides a snapshot of instances and their relationships at a specific moment during the VivaBot session. It shows a student attending a viva session, receiving a question of medium difficulty, submitting an answer, and being awarded marks. The evaluated data is then stored in the database. This diagram illustrates the real-time interaction between objects during the viva process.

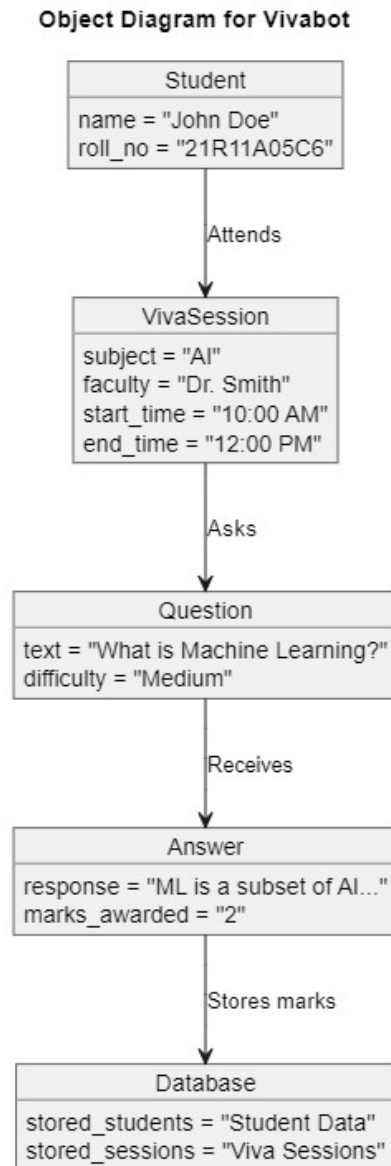


Fig 4.3.5 Object Diagram

DATAFLOW DIAGRAM

The Data Flow Diagram showcases how data moves between different components in the VivaBot system. It highlights the interactions between students, faculty, and internal modules like the Face Recognition System, Question Bank, AI Evaluator, and Result Manager. Student authentication triggers the viva flow, where questions are fetched and evaluated adaptively. The Result Management module stores the performance data in MySQL, while faculty can later view the results through this module. This diagram ensures a clear understanding of the logical data flow in the system.

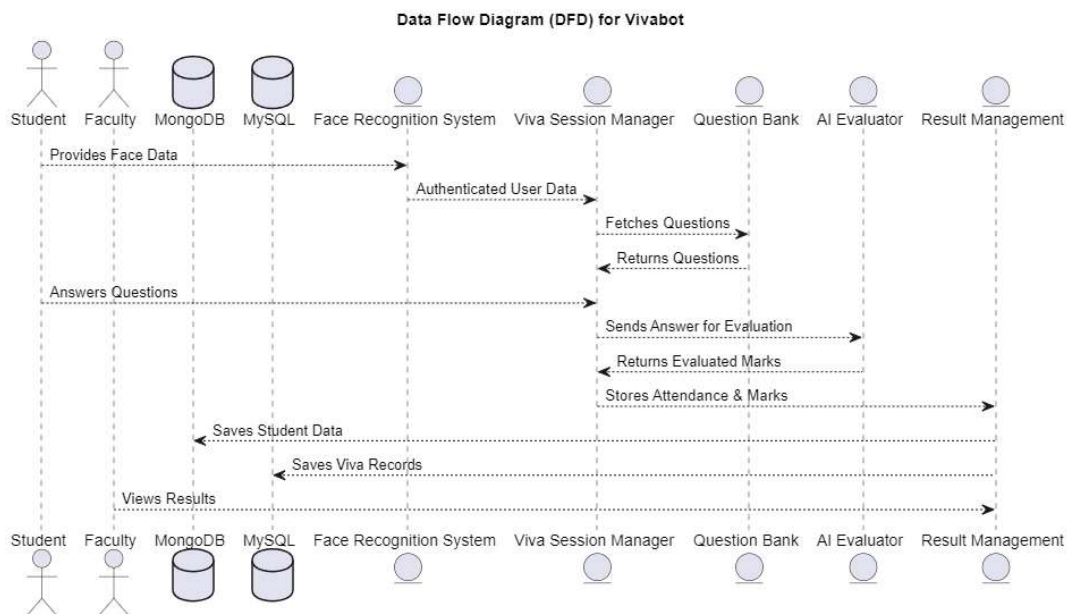


Fig 4.3.6 Dataflow Diagram

DEPLOYMENT DIAGRAM

The Data Flow Diagram showcases how data moves between different components in the VivaBot system. It highlights the interactions between students, faculty, and internal modules like the Face Recognition System, Question Bank, AI Evaluator, and Result Manager. Student authentication triggers the viva flow, where questions are fetched and evaluated adaptively. The Result Management module stores the performance data in MySQL, while faculty can later view the results through this module. This diagram ensures a clear understanding of the logical data flow in the system.

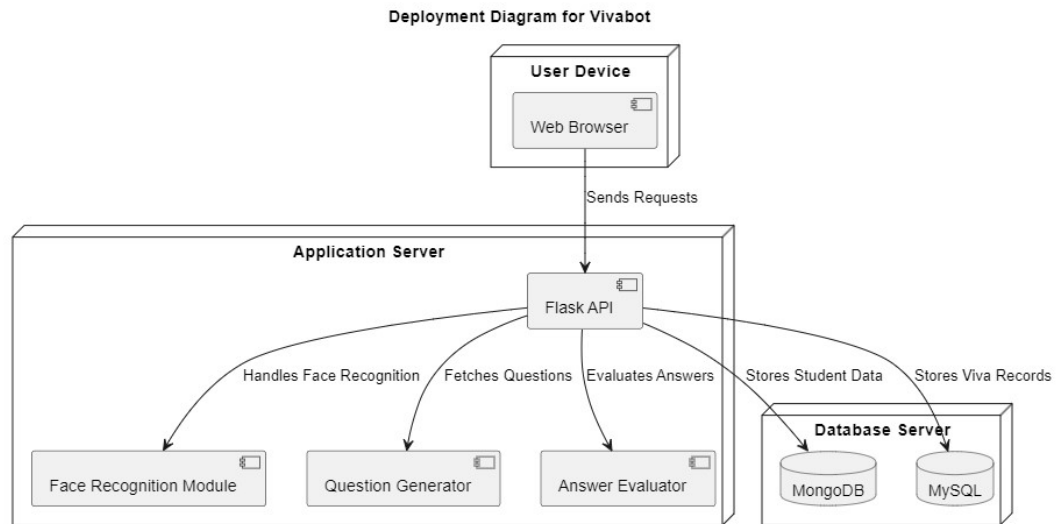


Fig 4.3.7 Deployment Diagram

COMPONENT DIAGRAM

The component diagram illustrates the modular structure of the VivaBot system, divided into three main layers: Frontend, Backend, and Database. The frontend includes the User Interface and Face Recognition components, while the backend handles APIs, question management, answer evaluation, and result processing. MySQL database is used for storing face data and viva-related records respectively. The diagram reflects how each component interacts to deliver a seamless and automated viva experience.

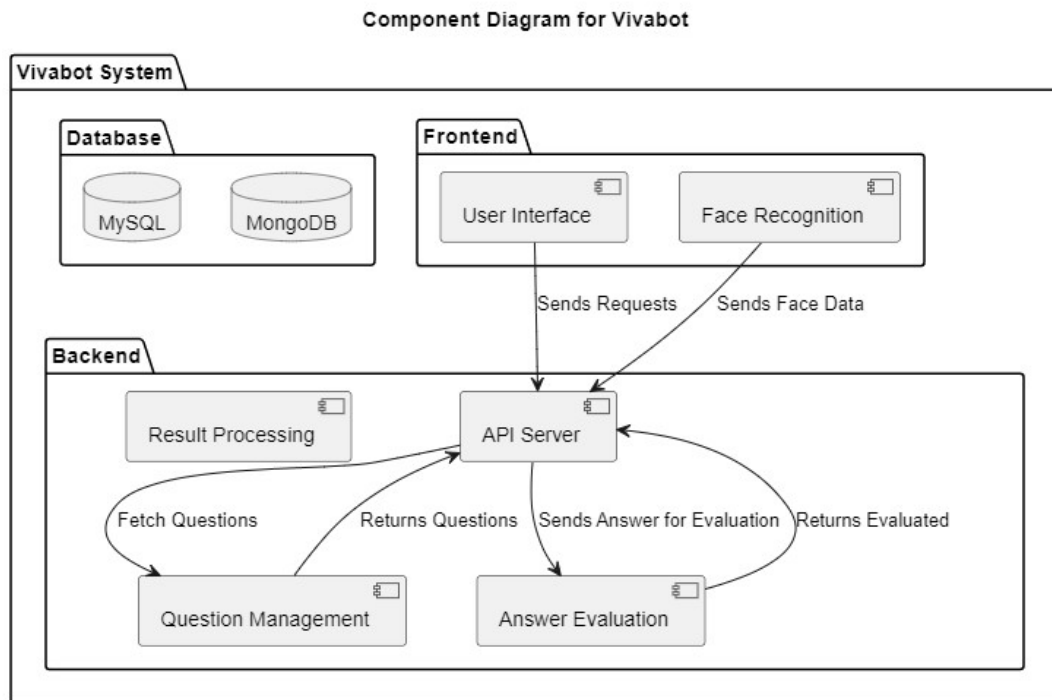


Fig 4.3.8 Component Diagram

4.4 USER INTERFACE DESIGN

The user interface (UI) of the VivaBot system is designed to be simple, intuitive, and efficient for both faculty and student users. The primary aim of the UI is to enable smooth, error-free interactions, even for users with minimal technical experience. The design ensures that students can easily register, log in, and complete viva sessions, while faculty can manage viva content and session controls without complexity.

The system's frontend is built using **flask**, and styling is handled through **CSS**, providing a modern, clean, and responsive layout. The UI adapts well across multiple screen sizes, laptops, and tablets. Buttons, inputs, and components follow a consistent style guide to ensure familiarity and ease of use throughout the system.

Design Approach and Layout

The VivaBot application is role-based and presents different interfaces for students and faculty. Student users interact with features such as registration, face login, and viva participation, while faculty users interact with the viva session dashboard and question upload interface.

Each major interface is designed with clarity and modularity in mind:

- **Student Registration** is divided into two steps: personal details input (name, roll number, class) and face registration.
- **Faculty Dashboard** allows question uploads either through a PDF upload or by entering a topic, where AI-generated questions are automatically stored in the database.
- **Viva Session Dashboard** displays all currently active viva sessions for faculty to select and configure (e.g., choosing specific weeks).

User Interaction Flow

Upon accessing the system:

- Students register their details and face once, then log in using their roll number and face recognition. Upon successful login, attendance is marked, and the viva session begins.

- The viva progresses through an adaptive question algorithm, with each student receiving questions based on their performance, and results are displayed at the end.
- Faculty do not need to register or log in. They are presented directly with the dashboard interface to upload questions, manage viva sessions, and end the viva using OTP-based email verification. After the session ends, marks and attendance are emailed in an Excel sheet.

Design Considerations

The VivaBot UI follows best practices for usability:

- **Clarity:** Each section is clearly labelled with intuitive icons and logical placement of components.
- **Consistency:** Uniform colour themes and styling rules are applied across all interfaces.
- **Feedback:** Real-time validation, success messages, and face recognition status are displayed clearly.
- **Accessibility:** Typography, contrast, and responsive behaviour are optimized for both desktop and touch-based devices.

Responsiveness and Accessibility

The VivaBot interface is fully responsive and functions well on various screen sizes. Layout components adapt based on the device, maintaining accessibility and usability. Forms and buttons are designed to be touch-friendly, and all modules support keyboard navigation, making the system user-friendly for all types of users.

4.5 DESIGN STANDARDS FOLLOWED

The design and development of the VivaBot project adhere to recognized software engineering standards to ensure quality, reliability, and maintainability. These standards help structure the system architecture, coding practices, documentation, and testing methodologies.

- **IEEE 1016 – Software Design Description:** The system architecture, module interactions, and data flow in VivaBot are documented and modelled in accordance with IEEE 1016. This ensures clarity in system behaviour and supports future maintenance and scalability.
- **IEEE 830 – Software Requirements Specification (SRS):** The functional and non-functional requirements of VivaBot were gathered and structured following IEEE 830 standards. This ensures a complete, verifiable, and traceable requirement document that guided development.
- **ISO/IEC 25010 – Software Product Quality Model:** VivaBot aims to achieve high levels of usability, reliability, and performance, aligning with ISO/IEC 25010 quality characteristics. Emphasis has been placed on user satisfaction, responsiveness, and system efficiency.
- **IEEE 829 – Software Test Documentation:** The testing process, including test cases, test execution, and result recording, follows IEEE 829 to ensure the system is rigorously validated for correctness, functionality, and performance.
- **ISO/IEC 27001 – Information Security:** Although not formally certified, VivaBot incorporates practices inspired by ISO 27001 to protect sensitive student data, including secure login using face recognition and OTP-based session verification for faculty.

These design standards collectively ensure that VivaBot is built on solid engineering practices, providing a robust and user-friendly platform for conducting automated viva sessions.

4.6 SAFETY & RISK MITIGATION MEASURES

The VivaBot system has been designed with several safety and risk mitigation measures to ensure secure operation, protect user data, and maintain system integrity during viva sessions. While the system does not involve physical risks, it handles sensitive data and real-time evaluation processes, which require careful consideration.

1. **Secure Authentication for Students:** Students can log in only after successful face recognition combined with their roll number. This dual-layer authentication ensures that only registered individuals can access the system.
2. **OTP Verification for Viva Closure:** Faculty can end a viva session only after verifying an OTP sent to their registered email. This step prevents unauthorized access or premature ending of sessions, safeguarding data integrity.
3. **Automated Session Handling:** VivaBot automatically tracks attendance and viva scores during the session, reducing human error. In case of unexpected interruptions, session data is saved progressively, preventing loss of progress.
4. **Error Handling and Feedback:** The system provides real-time alerts for failed face recognition attempts, invalid session selections, or missing question sets, guiding users to resolve issues effectively without causing data inconsistencies.
5. **Adaptive Evaluation Logic Validation:** The adaptive learning algorithm is designed with conditional checks to avoid exceeding the maximum score (5.0). This prevents students from being over-evaluated and ensures consistent viva completion.
6. **Regular Backups:** VivaBot includes functionality for exporting final marks and attendance to Excel format, which is also emailed to faculty after session completion. This acts as a backup for record-keeping and minimizes the risk of data loss.

CHAPTER 5 IMPLEMENTATION

5.1 TECHNOLOGY STACK

The **VivaBot** project is built using a well-integrated and modular technology stack that supports seamless viva automation through facial recognition, adaptive questioning, attendance management, and result generation. Each module is developed with specialized tools, frameworks, and libraries to ensure efficiency, accuracy, and user-friendliness.

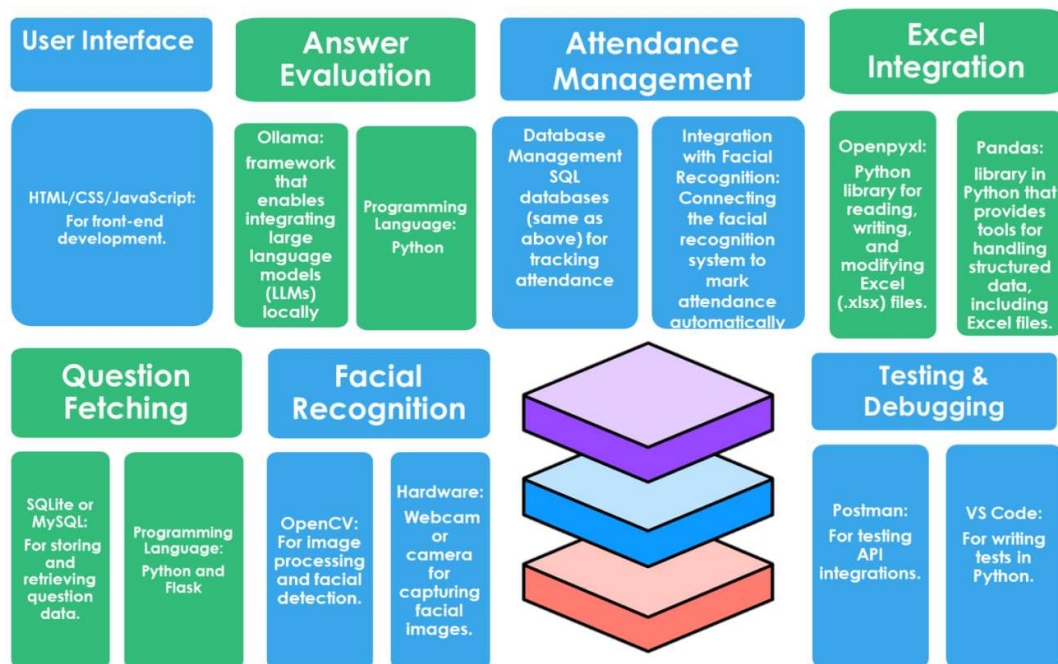


Fig 5.1.1 Technology Stack

- **User Interface**
 - Technologies Used: HTML, CSS, JavaScript
 - Purpose: For developing a responsive and user-friendly front end.
- **Answer Evaluation**
 - Technologies Used: Python, Ollama (for local LLM integration)

- Purpose: To dynamically evaluate student answers using AI and assign scores based on proficiency.
- **Attendance Management**
 - Technologies Used: SQL Databases, Facial Recognition Integration
 - Purpose: To automatically mark attendance during login using face recognition.
- **Excel Integration**
 - Technologies Used: Openpyxl, Pandas (Python libraries)
 - Purpose: For exporting student marks and attendance into structured Excel files.
- **Question Fetching**
 - Technologies Used: Python, Flask, SQLite/MySQL
 - Purpose: To store, retrieve, and manage subject-wise viva questions with difficulty levels.
- **Facial Recognition**
 - Technologies Used: OpenCV, Webcam/Camera
 - Purpose: For capturing and verifying student identity during login.
- **Testing & Debugging**
 - Tools Used: Postman, Visual Studio Code
 - Purpose: Postman for API testing; VS Code for writing and debugging Python code.

5.2 MODULE-WISE IMPLEMENTATION

The Vivabot system is designed in a modular fashion to ensure that each functional component operates independently while contributing to the system as a whole. Each module is implemented using relevant frontend and backend technologies to handle specific tasks efficiently. The following are the major modules developed:

Student Registration Module

Students register using their name, roll number, and class. After registration, the system captures facial data through a webcam using OpenCV. This data is stored and used for future authentication.

Face Recognition Login Module

During login, the system uses the stored facial data to verify the student. On successful match, the student is granted access and attendance is automatically marked.

Viva Session Management Module

Faculty can create viva sessions by specifying the day, time, subject, and class. The session details are stored in the SQL database and fetched during student login to display available sessions.

Question Generation & Fetching Module

Faculty either upload PDFs or specify topics. Using Ollama and AI models, the system generates adaptive questions categorized as Easy, Medium, or Hard, and stores them in subject-wise tables.

Adaptive Viva Process Module

The viva begins at Medium difficulty. If a student answers correctly, the next question is harder; if wrong, it becomes easier. Each question has a weighted score. The viva ends once the student accumulates 5 marks.

Answer Evaluation Module

Student responses are evaluated in real time using AI (via ChatGPT/Ollama). The system analyses confidence, accuracy, and relevance to determine the score and adjust the next question's difficulty.

Result and Attendance Management Module

After the viva ends, scores and attendance are stored. Using Openpyxl and Pandas, the data is compiled into an Excel file. After OTP verification, the Excel is emailed to the faculty.

Faculty Dashboard Module

Faculty can create viva sessions, upload topics/PDFs, monitor session status, and download the final Excel report. It also includes OTP-based verification before session closure.

5.3 CODE INTEGRATION STRATEGY

The VivaBot Project follows a modular architecture where each component is developed independently before being integrated into the larger system. The integration strategy ensures that each module can communicate effectively with others, while also allowing for easy debugging and enhancement. The primary integration strategy is based on a layered approach, where each layer interacts with the next through well-defined interfaces and APIs.

1. Modular Development

Each functional module is developed and tested independently to ensure proper functionality. This modular approach allows for easier debugging and version control. For example:

- **Faculty Dashboard Module:** Responsible for uploading questions either via PDF or through topic-based input for Ollama AI.
- **Student Registration Module:** Handles student registration and face recognition.

- **Viva Session Management:** Manages the display of available viva sessions and the adaptive learning algorithm.
- **Adaptive Learning Algorithm:** The core module responsible for adjusting the difficulty of questions based on student performance.

2. APIs and Data Flow

The different modules communicate through RESTful APIs, ensuring that data is exchanged between the front end and the back end. For example:

- The Student Registration Module sends student data (name, roll number, class, face data) to the excel for storage.
- The Faculty Dashboard communicates with the Question Management Module to upload new questions into the database, whether manually or via Ollama AI.

3. Database Integration

The database serves as the central repository for all the data in the system:

- **Student Data:** Stores information related to students' registration, including their face data and roll numbers.
- **Question Data:** Stores all the questions generated by the faculty and Ollama AI for viva sessions.
- **Viva Session Data:** Tracks the progress of each viva session, including question difficulty, student responses, and scores.

Each module interacts with the database to fetch and store the necessary data, ensuring consistency and real-time updates.

4. Version Control and Testing

To manage the integration process and ensure that the code remains functional across various modules, **Git** is used for version control.

- **Unit Tests:** Each module has a set of unit tests to verify the functionality of individual components.

- **Integration Testing:** After initial testing, the modules are integrated and tested as a whole. This helps identify potential issues in the communication between modules or with the database.
- **Continuous Integration (CI):** Tools like **Jenkins** are used to automatically run tests every time new code is pushed to the repository, ensuring that integration is smooth and no new issues are introduced.

5. Error Handling and Logging

Comprehensive error handling is implemented to ensure that if one module fails, the system remains stable:

- **Logging:** Logs are maintained for every interaction with the system, from user actions to backend processes. This helps identify errors and debug any issues that arise during integration.
- **Error Notifications:** If a critical error occurs during the viva process, the system sends notifications to the faculty and system administrator.

5.4 SAMPLE CODE SNIPPETS

```
import os
import time
import cv2
import fitz
import ollama
import numpy as np
import pandas as pd
import shutil
import smtplib
import nltk
from nltk.tokenize import sent_tokenize
from email.message import EmailMessage
import requests
import random
import joblib
from deepface import DeepFace
import json
import re
from sklearn.preprocessing import LabelEncoder
from sklearn.svm import SVC
import ast
```

```

from datetime import datetime, timedelta, date
import mysql.connector
from flask import Flask, jsonify, render_template, request, redirect, url_for, flash, Response,
session
@app.route('/faculty_upload', methods=['GET', 'POST'])
def faculty_upload():
    if request.method == 'POST':
        subject = request.form.get('subject')
        pdf_file = request.files.get('pdf_file')
        if not subject or not pdf_file:
            flash("Please select a subject and upload a PDF.", "danger")
            return redirect(url_for('faculty_upload'))
        if pdf_file.filename == "":
            flash("No file selected!", "danger")
            return redirect(url_for('faculty_upload'))
        pdf_path = os.path.join(app.config['UPLOAD_FOLDER'], pdf_file.filename)
        pdf_file.save(pdf_path)
        extracted_text = extract_questions_from_pdf(pdf_path)
        questions = parse_questions(extracted_text)
        print("Extracted Questions:", questions)
        if questions:
            insert_questions(subject, questions)
            flash(f'Successfully uploaded {len(questions)} questions!', "success")
        else:
            flash("No valid questions found in the PDF. Please check the format.", "warning")
            return redirect(url_for('faculty_upload'))
        return render_template('faculty_upload.html')
@app.route('/faculty_dashboard')
def faculty_dashboard():
    return render_template('faculty_dashboard.html')
@app.route('/excel route/<session id>/<class name>/<subject name>', methods=['GET'])
def excel_route(session_id, class_name, subject_name):
    session['class_name'] = class_name
    current_date = datetime.now().strftime('%Y-%m-%d')
    file_name = f'{class_name}_{subject_name}_{current_date}.xlsx'
    file_path = os.path.join(STUDENT_MARKS_FOLDER, file_name)
    if os.path.exists(file_path):
        return redirect(url_for('start_viva', session_id=session_id))
    source_file = os.path.join(CLASSES_FOLDER, f'{class_name}.xlsx')
    if not os.path.exists(source_file):
        return f'Error: Source file for {class_name} does not exist.'
    shutil.copy(source_file, file_path)
    df = pd.read_excel(file_path)
    df['Attendance'] = None
    df['Marks'] = None
    df.to_excel(file_path, index=False)
    return redirect(url_for('start_viva', session_id=session_id))

```

```

@app.route('/start_viva/<session_id>', methods=['GET'])
def start_viva(session_id):
    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)
    cursor.execute("SELECT subject FROM viva_sessions WHERE id = %s", (session_id,))
    result = cursor.fetchone()
    if result:
        subject = result['subject']
    else:
        return "Error: Subject not found."
    table_name = f"{subject}_questions"
    try:
        query = f"SELECT DISTINCT week FROM {table_name}"
        cursor.execute(query)
        weeks = [row['week'] for row in cursor.fetchall()]
    except Exception as e:
        return f"Error fetching weeks: {e}"
    conn.close()
    return render_template('start_viva.html', session_id=session_id, subject=subject,
weeks=weeks)
@app.route('/')
def index():
    return render_template('index.html')
@app.route('/home')
def home():
    return render_template('home.html')
@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        name = request.form['name']
        roll_no = request.form['roll_no']
        class_name = request.form['class']
        if not os.path.exists(EXCEL_FILE):
            df = pd.DataFrame(columns=["Name", "RollNo", "Class", "Face"])
            df.to_excel(EXCEL_FILE, index=False)
        else:
            df = pd.read_excel(EXCEL_FILE)
        if "RollNo" not in df.columns:
            flash("Error: Invalid student data file. Expected 'RollNo' column is missing.", "danger")
            return redirect(url_for('home'))
        if roll_no in df["RollNo"].astype(str).values:
            flash("Roll number already registered", "danger")
            return redirect(url_for('home'))
        df = pd.concat([df, pd.DataFrame({'RollNo': [roll_no], 'Name': [name], 'Class':
[class_name], 'Face': [f'{roll_no}.jpg']})], ignore_index=True)
        df['Face'] = df['Face'].astype(str)
        df.to_excel(EXCEL_FILE, index=False)

```

```

        flash("Student registered successfully! Proceed to face capture.", "success")
        return redirect(url_for('capture_face', roll_no=roll_no))
    return render_template('register.html')
@app.route('/capture_face/<roll_no>', methods=['GET', 'POST'])
def capture_face(roll_no):
    if request.method == 'POST':
        cap = cv2.VideoCapture(0)
        if not cap.isOpened():
            flash("Error accessing camera.", "danger")
            return redirect(url_for('register'))
        time.sleep(1)
        success, frame = cap.read()
        cap.release()
        if not success:
            flash("Failed to capture face. Try again.", "danger")
            return redirect(url_for('register'))
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5,
minSize=(50, 50))
        if len(faces) == 0:
            flash("No face detected. Try again.", "danger")
            return redirect(url_for('capture_face', roll_no=roll_no))
        (x, y, w, h) = faces[0]
        face_image = gray[y:y + h, x:x + w]
        face_image = cv2.resize(face_image, (200, 200))
        face_path = os.path.join(FACES_DIR, f"{roll_no}.jpg")
        cv2.imwrite(face_path, face_image)
        flash("Face captured successfully!", "success")
        train_face_recognizer()
        return redirect(url_for('home'))
    return render_template('capture_face.html', roll_no=roll_no)
@app.route('/login', methods=['GET', 'POST'])
def login():
    subject = 'Unknown'
    weeks = []
    if request.method == 'GET':
        session_id = request.args.get('session_id')
        weeks = request.args.getlist('weeks')
        if session_id:
            session['session_id'] = session_id
        if weeks:
            session['weeks'] = weeks
        else:
            session['weeks'] = session.get('weeks', [])
        print(f"Selected weeks from session (GET): {session['weeks']}")
    if request.method == 'POST':
        session.pop('questions', None)

```

```

session.pop('asked_questions', None)
session.pop('marks', None)
session.pop('proficiency', None)
session.pop('feedback', None)
print("Resetting session data for new viva attempt.")
session_id = session.get('session_id')
if not session_id:
    flash("Session ID is missing. Please start the process again.", "danger")
    return redirect(url_for('start_viva'))
weeks = request.form.getlist('weeks')
if not weeks:
    weeks = session.get('weeks', [])
session_id = session.get('session_id')
session['weeks'] = weeks
print(f"Weeks in POST request: {weeks}")
try:
    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT subject, class_name FROM viva_sessions WHERE id = %s",
(session_id,))
    result = cursor.fetchone()
    conn.close()
    if result:
        subject, class_name = result
    else:
        flash("No subject found for this session ID.", "danger")
        return redirect(url_for('start_viva'))
except Exception as e:
    flash(f"Database error: {e}", "danger")
    return redirect(url_for('start_viva'))
roll_no = request.form['roll_no']
today_date = datetime.today().strftime('%Y-%m-%d')
excel_filename = f"student_marks/{class_name}_{subject}_{today_date}.xlsx"
excel_path = os.path.join(os.getcwd(), excel_filename)
if os.path.exists(excel_path):
    df = pd.read_excel(excel_path)
    if roll_no in df["Roll Number"].values:
        student_row = df[df["Roll Number"] == roll_no]
        if student_row["Attendance"].values[0] == "P":
            flash("You have already taken this viva session. You cannot take it again.",
"danger")
            return redirect(url_for('login', session_id=session_id, weeks=session['weeks']))
recognizer, label_mapping = load_face_recognizer()
if not recognizer:
    flash("No faces available for recognition. Please register students first.", "danger")
    return redirect(url_for('login', session_id=session_id, weeks=session['weeks']))
camera = cv2.VideoCapture(0)

```



```

if not camera.isOpened():
    flash("Error: Could not access the camera.", "danger")
    return redirect(url_for('login', session_id=session_id, weeks=session['weeks']))
success, frame = camera.read()
camera.release()
if not success:
    flash("Error: Could not capture image from the camera.", "danger")
    return redirect(url_for('login', session_id=session_id, weeks=session['weeks']))
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
faces = face_cascade.detectMultiScale(gray, 1.1, 4)
if len(faces) == 0:
    flash("No face detected. Please adjust your position and try again.", "warning")
    return redirect(url_for('login', session_id=session_id, weeks=session['weeks']))
(x, y, w, h) = faces[0]
face_image = gray[y:y + h, x:x + w]
face_image = cv2.resize(face_image, (200, 200))
try:
    label, confidence = recognizer.predict(face_image)
    print(f'Recognized Label: {label}, Confidence: {confidence}')
except Exception as e:
    flash(f'Face recognition error: {e}', "danger")
    return redirect(url_for('login', session_id=session_id, weeks=session['weeks']))
for registered_roll_no, numeric_label in label_mapping.items():
    print(f'Checking {registered_roll_no} -> Label {numeric_label}')
    if numeric_label == label:
        if confidence < 80:
            if registered_roll_no == roll_no:
                print(f'Face matched for {roll_no}')
                session['roll_no'] = roll_no
                session['subject'] = subject
                session['recording_active'] = True
                start_recording(roll_no)
                return redirect(url_for('questions'))
        else:
            print(f'Confidence too high ({confidence}) for {registered_roll_no}')
            flash("Face did not match the registered image. Please try again.", "danger")
            return redirect(url_for('login', session_id=session_id, weeks=session['weeks']))
return render_template('login.html', subject=subject, session_id=session.get('session_id'),
weeks=session.get('weeks', []))
@app.route('/questions', methods=['GET', 'POST'])
def questions():
    subject = session.get('subject')
    roll_no = session.get('roll_no')
    weeks = session.get('weeks', [])
    if not subject or not weeks or not roll_no:
        flash("Session data is missing. Please restart.", "danger")
        return redirect(url_for('login'))

```

```

if isinstance(weeks, str):
    try:
        weeks = ast.literal_eval(weeks)
    except (SyntaxError, ValueError):
        weeks = []
if not isinstance(weeks, list):
    weeks = [weeks]
weeks = [week.strip() for week in weeks if isinstance(week, str)]
table_name = f'{subject}_questions'
if 'questions' not in session or not session['questions']:
    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)
    format_strings = ','.join(["%s"] * len(weeks))
    cursor.execute(f'SELECT * FROM {table_name} WHERE week IN ({format_strings})',
tuple(weeks))
    all_questions = cursor.fetchall()
    if not all_questions:
        flash("No questions found for the selected weeks!", "danger")
        return redirect(url_for('login'))
    cursor.close()
    conn.close()
    session['questions'] = all_questions
    session['asked_questions'] = []
    session['marks'] = 0
    session['proficiency'] = 'Medium'
    session['question_score'] = 0
    session['feedback'] = []
if session['question_score'] >= 5:
    return redirect(url_for('finish'))
question_scores = {'Easy': 0.5, 'Medium': 1, 'Hard': 1.5}
all_questions = session['questions']
current_proficiency = session['proficiency']
available_questions = [q for q in all_questions if q['proficiency_level'] == current_proficiency
                        and q['qn_no'] not in session['asked_questions']
                        and session['question_score'] + question_scores[q['proficiency_level']] <= 5]
if not available_questions:
    if current_proficiency == 'Hard':
        session['proficiency'] = 'Medium'
    elif current_proficiency == 'Medium':
        session['proficiency'] = 'Easy'
    available_questions = [q for q in all_questions if q['proficiency_level'] ==
session['proficiency']
                        and q['qn_no'] not in session['asked_questions']
                        and session['question_score'] + question_scores[q['proficiency_level']] <= 5]

if not available_questions:
    return redirect(url_for('finish'))

```

```

if 'current_question' not in session:
    session['current_question'] = random.choice(available_questions)
current_question = session['current_question']
if request.method == 'POST':
    if 'skip' in request.form:
        feedback_prompt = f"""
        Provide the correct answer for the following question:

        Question: {current_question['question']}
        """
        feedback_response = ollama.chat(model="mistral", messages=[{"role": "user",
"content": feedback_prompt}])
        correct_answer = feedback_response.get('message', {}).get('content', 'Correct answer not
available.')
        correct_answer = summarize_text(correct_answer, max_sentences=2)
        feedback = {
            'question': current_question['question'],
            'feedback': "You skipped this question. Review similar topics before attempting
again.",
            'correct_answer': correct_answer
        }
        session['feedback'].append(feedback)
        session['asked_questions'].append(current_question['qn_no'])
        session['question score'] += question_scores[current_proficiency]
        if session['proficiency'] == 'Medium':
            session['proficiency'] = 'Easy'
        elif session['proficiency'] == 'Hard':
            session['proficiency'] = 'Medium'
        session.pop('current_question', None)
        return redirect(url_for('questions'))
    user_answer = request.form['answer'].strip()
    if not user_answer:
        flash("Please enter an answer before proceeding.", "warning")
        return redirect(url_for('questions'))
    prompt = f"""
    Evaluate the student's answer to the question.
    Question: {current_question['question']}
    Student Answer: {user_answer}
    Provide a similarity score from 0 to 100, where 100 is a perfect match.
    """
    response = ollama.chat(model="mistral", messages=[{"role": "user", "content": prompt}])
    ai_response = response.get('message', {}).get('content', "")
    similarity_score = 0
    if ai_response:
        number = "".join(filter(str.isdigit, ai_response.split()[0]))
        if number:
            similarity_score = int(number)

```

```

correct = similarity_score >= 40
current_score = question_scores[current_proficiency]
session['question_score'] += current_score
session.modified = True
if correct:
    session['marks'] += current_score
    if current_proficiency == 'Medium':
        session['proficiency'] = 'Hard'
    elif current_proficiency == 'Easy':
        session['proficiency'] = 'Medium'
    improvement_prompt = f"""
    The following answer was correct. Suggest ways to make it even better or more
    accurate:

    Question: {current_question['question']}
    """
    improvement_response = ollama.chat(model="mistral", messages=[{"role": "user",
    "content": improvement_prompt}])
    improvement_feedback = improvement_response.get('message', {}).get('content', 'No
    improvement suggestions available.')
    improvement_feedback = summarize_text(improvement_feedback,max_sentences=2)
    feedback = {
        'question': current_question['question'],
        'feedback': "Your answer was correct! Here's how you can make it even better",
        'correct_answer': improvement_feedback
    }
    session['feedback'].append(feedback)
else:
    feedback_prompt = f"""
    Provide an explanation of how to correctly answer this question:

    Question: {current_question['question']}
    """
    feedback_response = ollama.chat(model="mistral", messages=[{"role": "user",
    "content": feedback_prompt}])
    correct_text = feedback_response.get('message', {}).get('content', 'No feedback
    available.')
    correct_text = summarize_text(correct_text,max_sentences=2)
    feedback = {
        'question': current_question['question'],
        'feedback': "Your answer was incorrect. Review the explanation carefully before
    attempting similar questions.",
        'correct_answer': correct_text
    }
    session['feedback'].append(feedback)
    if current_proficiency == 'Medium':
        session['proficiency'] = 'Easy'

```

```

        elif current_proficiency == 'Hard':
            session['proficiency'] = 'Medium'
        session['asked_questions'].append(current_question['qn_no'])
        session.pop('current_question', None)
        if session['question_score'] >= 5:
            return redirect(url_for('finish'))
        return redirect(url_for('questions'))
    return render_template('questions.html',
                           subject=subject,
                           question=current_question,
                           proficiency_level=current_proficiency,
                           question_number=len(session['asked_questions']) + 1,
                           weeks=weeks)

@app.route('/send_otp', methods=['POST'])
def send_otp():
    data = request.get_json()
    session_id = data.get("session_id")
    if not session_id:
        return jsonify({"success": False, "message": "Session ID is missing"}), 400
    connection = get_db_connection()
    cursor = connection.cursor(dictionary=True)
    cursor.execute("SELECT faculty_email FROM viva_sessions WHERE id = %s",
                   (session_id,))
    session_data = cursor.fetchone()
    if not session_data:
        return jsonify({"success": False, "message": "Session not found"}), 404
    faculty_email = session_data["faculty_email"]
    otp = str(random.randint(100000, 999999))
    otp_storage[session_id] = otp
    sender_email = "lekhyal854@gmail.com"
    sender_password = "eohx qxct qiev sqhb"
    msg = EmailMessage()
    msg["Subject"] = "Viva Session OTP Verification"
    msg["From"] = sender_email
    msg["To"] = faculty_email
    msg.set_content(f"Your OTP to end the Viva session is: {otp}")
    try:
        with smtplib.SMTP_SSL("smtp.gmail.com", 465) as server:
            server.login(sender_email, sender_password)
            server.send_message(msg)
        return jsonify({"success": True, "message": "OTP sent to faculty email."})
    except Exception as e:
        return jsonify({"success": False, "message": f"Failed to send OTP: {str(e)}"}), 500
    finally:
        cursor.close()
        connection.close()

@app.route('/end_viva_session', methods=['POST'])

```

```

def end_viva_session(session_id=None):
    if session_id is None:
        data = request.get_json()
        session_id = data.get("session_id")
    if not session_id:
        return jsonify({"success": False, "message": "Session ID is missing"}), 400
    connection = get_db_connection()
    cursor = connection.cursor(dictionary=True)
    cursor.execute("SELECT class_name, subject, faculty_email FROM viva_sessions WHERE
id = %s", (session_id,))
    viva_session = cursor.fetchone()
    if not viva_session:
        return jsonify({"success": False, "message": "Viva session not found"}), 404
    class_name = viva_session["class_name"]
    subject_name = viva_session["subject"]
    faculty_email = viva_session["faculty_email"]
    current_date = datetime.today().strftime("%Y-%m-%d")
    file_name = f'{class_name}_{subject_name}_{current_date}.xlsx'
    file_path = os.path.join("student_marks", file_name)
    if not os.path.exists(file_path):
        return jsonify({"success": False, "message": f'Excel file not found: {file_name}'}), 404
    df = pd.read_excel(file_path)
    df.loc[df['Attendance'].isna(), 'Attendance'] = 'A'
    df.loc[df['Marks'].isna(), 'Marks'] = 0
    df.to_excel(file_path, index=False)
    cursor.execute("UPDATE viva_sessions SET end_early = TRUE WHERE id = %s",
(session_id,))
    connection.commit()
    cursor.close()
    connection.close()
    send_email(faculty_email, file_path)
    return jsonify({"success": True, "message": "Viva session ended. Updated Excel sent to
faculty.", "redirect": "/dashboard"})
def send_email(recipient_email, attachment_path):
    sender_email = "lekhyal854@gmail.com"
    sender_password = "eohx qxct qiev sqhb"
    if not os.path.exists(attachment_path):
        print(f'Error: The file {attachment_path} does not exist.')
        return
    msg = EmailMessage()
    msg["Subject"] = "Updated Viva Attendance and Marks Sheet"
    msg["From"] = sender_email
    msg["To"] = recipient_email
    msg.set_content("Please find the updated attendance and marks sheet attached.")
    with open(attachment_path, "rb") as f:
        file_data = f.read()
        file_name = os.path.basename(attachment_path)

```

```

        msg.add_attachment(file_data, maintype="application", subtype="octet-stream",
filename=file_name)
    try:
        with smtplib.SMTP_SSL("smtp.gmail.com", 465) as server:
            server.login(sender_email, sender_password)
            server.send_message(msg)
            print(f'Email sent to {recipient_email} successfully.')
    except Exception as e:
        print(f'Failed to send email: {e}')
@app.route('/finish', methods=['GET', 'POST'])
def finish():
    if session.get('recording_active'):
        stop_recording()
        session['recording_active'] = False
    recorded_file = session.get('recording_filename')
    confidence_score, confidence_feedback, answer_quality_score, answer_quality_feedback =
analyze_video(recorded_file)
    session['confidence_score'] = confidence_score
    session['confidence_feedback'] = confidence_feedback
    session['answer_quality_score'] = answer_quality_score
    session['answer_quality_feedback'] = answer_quality_feedback
    feedback_list = session.get('feedback', [])
    return render_template('finish.html',
        feedback_list=feedback_list,
        confidence_score=confidence_score,
        confidence_feedback=confidence_feedback,
        answer_quality_score=answer_quality_score,
        answer_quality_feedback=answer_qualityfeedback)
@app.route('/finish_viva', methods=['POST', 'GET'])
def finish_viva():
    roll_no = session.get('roll_no')
    subject = session.get('subject')
    class_name = session.get('class_name')
    marks = session.get('marks', 0)
    print(f'Session data: roll_no={roll_no}, subject={subject}, class_name={class_name},
marks={marks}')
    missing_data = []
    if not roll_no:
        missing_data.append('roll_no')
    if not subject:
        missing_data.append('subject')
    if not class_name:
        missing_data.append('class_name')
    if missing_data:
        flash(f'Session data missing for: {', '.join(missing_data)}. Please restart.', "danger")
        return redirect(url_for('login'))
    if not os.path.exists(STUDENT_MARKS_FOLDER):

```

```

        flash("Student marks folder does not exist.", "danger")
        return redirect(url_for('login'))
    current_date = datetime.now().strftime("%Y-%m-%d")
    file_name = f'{class_name}_{subject}_{current_date}.xlsx'
    file_path = os.path.join(STUDENT_MARKS_FOLDER, file_name)
    print(f'Looking for file: {file_path}')
    print(f'Available files: {os.listdir(STUDENT_MARKS_FOLDER)}')
    if not os.path.exists(file_path):
        flash(f'Error: Attendance sheet '{file_name}' not found.", "danger")
        return redirect(url_for('login'))
    df = pd.read_excel(file_path)
    if 'Roll Number' in df.columns:
        roll_number_column = 'Roll Number'
    elif 'RollNo' in df.columns:
        roll_number_column = 'RollNo'
    else:
        flash("Error: Roll Number column not found in the sheet.", "danger")
        return redirect(url_for('login'))
    if 'Attendance' not in df.columns:
        df['Attendance'] = None
    if 'Marks' not in df.columns:
        df['Marks'] = None
    if roll_no in df[roll_number_column].values:
        df.loc[df[roll_number_column] == roll_no, 'Attendance'] = 'P'
        df.loc[df[roll_number_column] == roll_no, 'Marks'] = marks
        df.to_excel(file_path, index=False)
        flash(f'Marks updated for {roll_no}', "success")
    else:
        flash("Roll number not found in the sheet.", "warning")
    recorded_file = session.get('recording_filename')
    if recorded_file and os.path.exists(recorded_file):
        try:
            os.remove(recorded_file)
            print(f'Recording deleted: {recorded_file}')
        except Exception as e:
            print(f'Error deleting recording: {e}')
    session.pop('roll_no', None)
    session.pop('marks', None)
    LAST_RESET_DATE = None
    @app.route('/dashboard')
    def dashboard():
        global LAST_RESET_DATE
        today = date.today()
        if LAST_RESET_DATE != today:
            conn = get_db_connection()
            cursor = conn.cursor()
            cursor.execute("UPDATE viva_sessions SET end_early = 0")

```



```

conn.commit()
cursor.close()
conn.close()
LAST_RESET_DATE = today
print(f"end_early reset at {datetime.now()}")
now = datetime.now()
day_of_week = now.strftime('%A')
current_time = now.time()
conn = get_db_connection()
cursor = conn.cursor(dictionary=True)
cursor.execute("""
    SELECT id, class_name, subject, faculty_name,
           start_time, end_time, faculty_email, end_early
    FROM viva_sessions
    WHERE day_of_week = %s
    AND start_time <= %s
    AND end_time >= %s
    AND end_early = 0
""", (day_of_week, current_time, current_time))
sessions = cursor.fetchall()
for session in sessions:
    if isinstance(session["start_time"], timedelta):
        session["start_time"] = (datetime.min + session["start_time"]).time()
    if isinstance(session["end_time"], timedelta):
        session["end_time"] = (datetime.min + session["end_time"]).time()
    start_time = session["start_time"]
    end_time = session["end_time"]
    if session["end_early"] == 1:
        session["disabled"] = True
    else:
        session["disabled"] = False
    if session["end_early"] == 1 and start_time <= current_time and end_time >=
current_time:
        cursor.execute("UPDATE viva_sessions SET end_early = 0 WHERE id = %s",
(session["id"],))
        conn.commit()
        session["disabled"] = False
    cursor.close()
    conn.close()
return render_template('dashboard.html', sessions=sessions)

```

CHAPTER 6 TESTING

6.1 TESTING STRATEGY

The testing strategy for VivaBot is designed to ensure the reliability, accuracy, and robustness of all critical components, from student authentication to viva evaluation and result processing. The primary focus is on delivering a seamless and error-free experience for both students and faculty. The strategy combines multiple levels of testing to cover functionality, performance, and usability.

The testing approach involves:

- **Unit Testing** for validating individual modules like student registration, face recognition, viva session handling, and adaptive question evaluation logic.
- **Integration Testing** to ensure proper interaction between modules such as the database, AI evaluator, and session manager.
- **System Testing** to simulate complete viva sessions and validate end-to-end flow including login, question generation, evaluation, and result storage.
- **Regression Testing** was periodically performed to ensure that newly added features or changes did not break existing functionality.
- **User Acceptance Testing (UAT)** was carried out with actual students and faculty to ensure that the application meets real-world requirements and is user-friendly.

By using this multi-layered testing strategy, the project aims to minimize errors, improve user confidence, and ensure stable and secure operations across all use cases.

6.2 UNIT TESTING

Unit testing in VivaBot focuses on verifying the functionality of individual components in isolation to ensure that each unit performs as expected. The primary modules tested include student registration, face recognition logic, viva session creation, question fetching, and the adaptive learning algorithm.

Key areas covered during unit testing:

- **Student Registration Module:** Tested to ensure proper validation of inputs like name, roll number, and class, and successful insertion into the database.
- **Face Recognition Module:** Unit tests validated the image capture, encoding, and matching processes to detect successful and failed login attempts accurately.
- **Viva Session Management:** Ensured correct creation, retrieval, and updating of viva sessions by faculty.
- **Question Generation and Retrieval:** Verified that the system correctly fetches questions based on selected weeks and proficiency levels.
- **Adaptive Evaluation Logic:** Checked whether the difficulty level is adjusted properly based on student responses and that the viva ends when the cumulative score reaches 5.

Each unit test was executed using mock data and automated testing tools to ensure accuracy and repeatability. Errors or unexpected behaviors were logged and corrected at the module level before integration testing began.

6.3 INTEGRATION TESTING

Integration testing in VivaBot aimed to ensure that individual modules work together seamlessly as a complete system. After each module was verified through unit testing, integration testing validated the interaction between components like face recognition, viva session selection, adaptive questioning, and result processing.

Key areas of integration tested:

- **Student Login with Face Recognition + Viva Session Dashboard:** Verified that after successful face recognition, students are presented with the correct list of viva sessions pulled from the database.
- **Viva Session + Question Generation:** Ensured that the system fetches appropriate questions from the selected weeks and subject, integrating the database and AI question generation (Ollama).

- **Answer Submission + Adaptive Evaluation Logic:** Confirmed that the evaluation module accurately adjusts difficulty levels and calculates scores, and that these scores flow correctly into result storage.
- **Final Marks + Excel Sheet Generation + Email via OTP:** Validated that once a session ends, the marks and attendance are compiled into an Excel file, and the email with OTP verification is sent successfully to the faculty.

Mock interfaces and live database interactions were used to simulate real-time operations. All module boundaries were tested with various data inputs to ensure data consistency and error-free transitions between modules.

6.4 SYSTEM TESTING

System testing was conducted to validate the end-to-end functionality of the VivaBot application, ensuring that the entire system met the specified requirements and performed well under real-world conditions. This phase involved testing the fully integrated system as a whole, simulating real user behavior from both the student and faculty perspectives.

Major areas covered during system testing:

- **Complete Student Workflow:** From registration and face enrollment to login, viva session participation, adaptive questioning, and result viewing.
- **Faculty Interaction:** Uploading questions via PDF or topic-based AI generation, selecting viva sessions and weeks, and receiving final attendance and marks through email after OTP verification.
- **Adaptive Questioning Flow:** Verified whether the system accurately adjusts the difficulty level of questions based on students' answers and maintains a correct cumulative score until the viva ends.
- **Excel Report Generation and Emailing:** Tested the creation of attendance and marks sheets and ensured delivery to the faculty's email after secure OTP validation.

6.5 TEST CASES AND RESULTS

Test Case 1	
Test Case Name	Student Registration
Description	Validate that a student can successfully register with name, roll number, and class.
Output	Student is added to the excel sheets and redirected to face registration.

Test Case 2	
Test Case Name	Face Registration
Description	Verify if the student's facial data is correctly captured and linked to the student profile.
Output	Face registration successful and student is notified.

Test Case 3	
Test Case Name	Invalid Face Registration
Description	Attempt face registration with unclear or no face detected.
Output	System displays error and prompts for retry.

Test Case 4	
Test Case Name	Student Login
Description	Validate login with roll number and face recognition.
Output	Login successful, student redirected to viva session.

Test Case 5	
Test Case Name	Student Login with Invalid Face
Description	Attempt login with unmatched or invalid facial data.
Output	Access denied, error message shown.

Test Case 6	
Test Case Name	Faculty Upload Questions via PDF
Description	Verify if the system correctly extracts and stores questions from uploaded PDF.
Output	Questions successfully stored in the database.

Test Case 7	
Test Case Name	Faculty Upload Topics for Ollama
Description	Ensure the topics entered are processed by Ollama AI to generate relevant questions.
Output	AI generates and uploads questions into subject table.

Test Case 8	
Test Case Name	Viva Session Selection by Faculty
Description	Check if faculty can select session and appropriate weeks before viva starts.
Output	Session configuration saved successfully.

Test Case 9	
Test Case Name	Start Viva with Medium Question
Description	Validate that the first question starts with medium proficiency level for every student.
Output	Medium-level question displayed to the student.

Test Case 10	
Test Case Name	Adaptive Learning - Correct Answer
Description	Test if the admin can view the pending profile edit requests and approve or decline them.
Output	Verify that correct answer increases difficulty level.

Test Case 11	
Test Case Name	Relieve Faculty (Admin)
Description	Verify that wrong answer decreases difficulty level.
Output	Next question level adjusted accordingly.

Test Case 12	
Test Case Name	Question Score Tracking
Description	Ensure that question score increments correctly based on difficulty level.
Output	Questions Score updated correctly after each question.

Test Case 13	
Test Case Name	End Viva on Score Threshold
Description	Validate that viva ends when question score reaches 5.
Output	Session ends and feedback shown.

Test Case 14	
Test Case Name	Feedback Generation
Description	Check if feedback (confidence, skipped, answer quality) is shown at the end.
Output	Feedback displayed correctly.

Test Case 15	
Test Case Name	Attendance and Marks Update
Description	Ensure marks and attendance are recorded in Excel.
Output	Excel file updated with student details.

Test Case 16	
Test Case Name	End Session via OTP
Description	Faculty ends viva session after receiving and entering OTP.
Output	OTP validated and session ended.

Test Case 17	
Test Case Name	Email Excel Sheet
Description	System emails the final Excel file to faculty after session ends.
Output	Email successfully sent with attachment.

Table 6.5.1 Testcases and results

6.6 BUG REPORTING AND TRACKING

In the Vivabot project, bug reporting and tracking were carried out manually without the use of any dedicated tools. During development and testing, any issues or unexpected behaviors were documented in a shared spreadsheet or maintained as written logs. Each bug entry included the date, a short description of the issue, the module affected, steps to reproduce, and the current status (e.g., Open, In Progress, Resolved). The development team regularly reviewed the logs, worked on fixing the issues, and updated the status accordingly. This manual approach ensured that all bugs were noted, tracked, and addressed systematically throughout the project lifecycle, despite the absence of automated tools.

Bug ID	Module	Description	Steps to Reproduce	Status
B001	Face Recognition	Face not detected in low-light conditions	1. Launch login module 2. Try face recognition in dim lighting	Resolved
B002	Student Login	Login fails despite valid face and roll number	1. Complete face registration	Resolved

			2. Attempt login with correct roll and face	
B003	Viva Session Dashboard	Viva sessions not loading	1. Login as student 2. Navigate to session dashboard	Resolved
B004	AI Question Generation	No questions generated for a valid topic	1. Faculty enters topic 2. Click generate questions	Resolved
B005	Excel Generation	Excel file misses roll numbers	1. End viva session 2. Check exported Excel file	Resolved
B006	OTP Email System	OTP not sent during session end	1. End viva 2. Enter email 3. Wait for OTP	Resolved
B007	Face Registration	Duplicate face registration allowed	1. Enter same roll number 2. Register face again	Resolved
B008	Feedback Module	Confidence score remains 0 despite valid input	1. Complete viva 2. Check final feedback report	Resolved

Table 6.6.1 Bugs reporting and tracking

CHAPTER 7 RESULTS AND DISCUSSION

7.1 OUTPUT SCREENS



Fig 7.1.1 Index Page

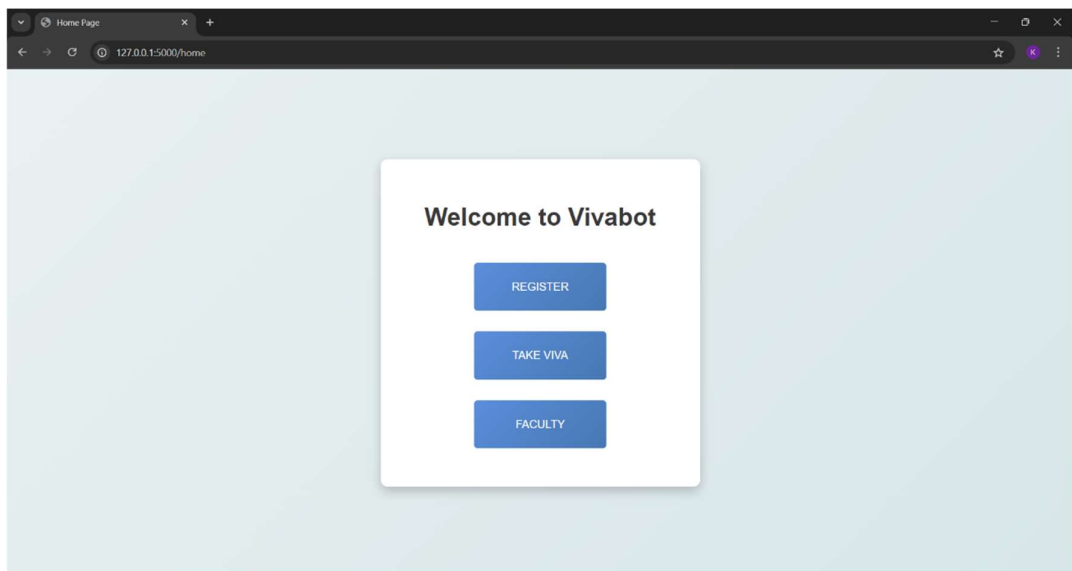


Fig 7.1.2 Home Page

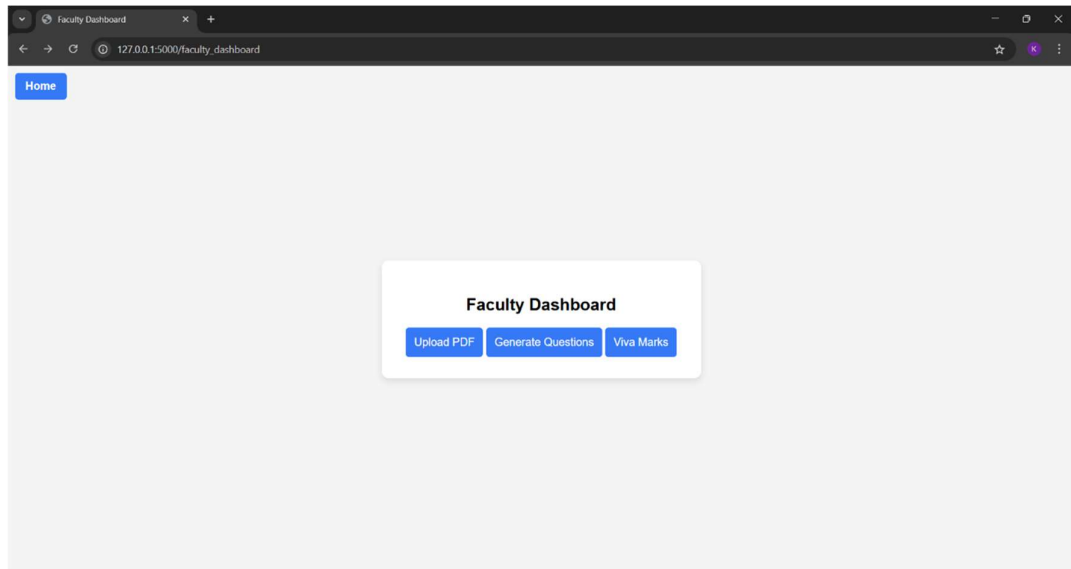


Fig 7.1.3 Faculty Dashboard

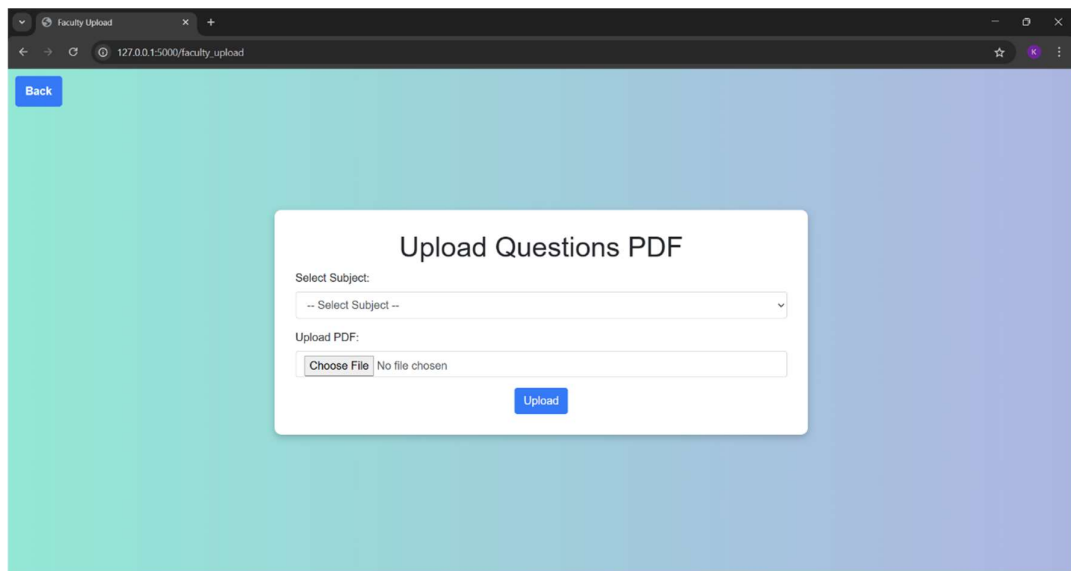


Fig 7.1.4 Upload Questions through PDF

The screenshot shows a web browser window with the address bar displaying "127.0.0.1:5000/generate_questions". The page features a central white form titled "Generate Questions". At the top left of the form is a blue "Back" button. The form contains the following fields: "Select Subject:" with a dropdown menu showing "Machine Learning"; "Enter Week:" with a text input field containing "Week X"; "Enter Topics (comma-separated):" with a text input field containing "e.g., Naive Bayes, Decision Trees"; and "Number of Questions per Level (Optional):" with a text input field containing "Default: 5". A blue "Generate Questions" button is located at the bottom of the form.

Fig 7.1.5 Generate Questions

The screenshot shows a web browser window with the address bar displaying "127.0.0.1:5000/generate_final_marks". The page features a central white form titled "Generate Final Viva Marks". The form contains the following fields: "Start Date" with a date input field showing "dd-mm-yyyy" and a calendar icon; "End Date" with a date input field showing "dd-mm-yyyy" and a calendar icon; "Class" with a dropdown menu showing "Select Class"; and "Subject" with a dropdown menu showing "Select Subject". A blue "Compute Final Marks" button is located at the bottom of the form.

Fig 7.1.6 Computing Final Marks

Roll Number	Name	Marks
20R11A05E3	JOSHUA DANIEL PASUMARTHI	5
20R11A05N4	POORVI JATOTH	8
21R11A05L6	A PRANEETH KUMAR REDDY	14
21R11A05L7	AARON BIJU	8
21R11A05L8	APPALA KARTHIK	4
21R11A05L9	BABBURI SHIVASHARAN	4
21R11A05M0	BADUGU JESSY	10
21R11A05M1	BANDARI ARJUN	8
21R11A05M2	BOREDDY SANDEEP REDDY	10
21R11A05M3	CHATLA SNEHA	8
21R11A05M4	DODLA ADITHYA KUMAR	4
21R11A05M5	ELURI KUSHAL KEERTHAN	11
21R11A05M6	ESHWAR GOUD MANDALA	6
21R11A05M7	G CHARAN TEJA	2
21R11A05M8	GADE DEEKSHITH	5
21R11A05M9	GOLI KARTHIK	12

Fig 7.1.7 Final Computed Marks Sheet

The image shows a web browser window with a single tab titled 'Register'. The address bar displays '127.0.0.1:5000/register'. The main content area has a light blue gradient background. Centered on the page is a white rectangular form titled 'Student Registration'. Inside the form, there are three input fields: 'Name:' with a text box, 'Roll Number:' with a text box, and 'Class:' with a dropdown menu currently showing '4 CSE-A'. Below these fields is a prominent blue button labeled 'Register'.

Fig 7.1.8 Student Registration Page

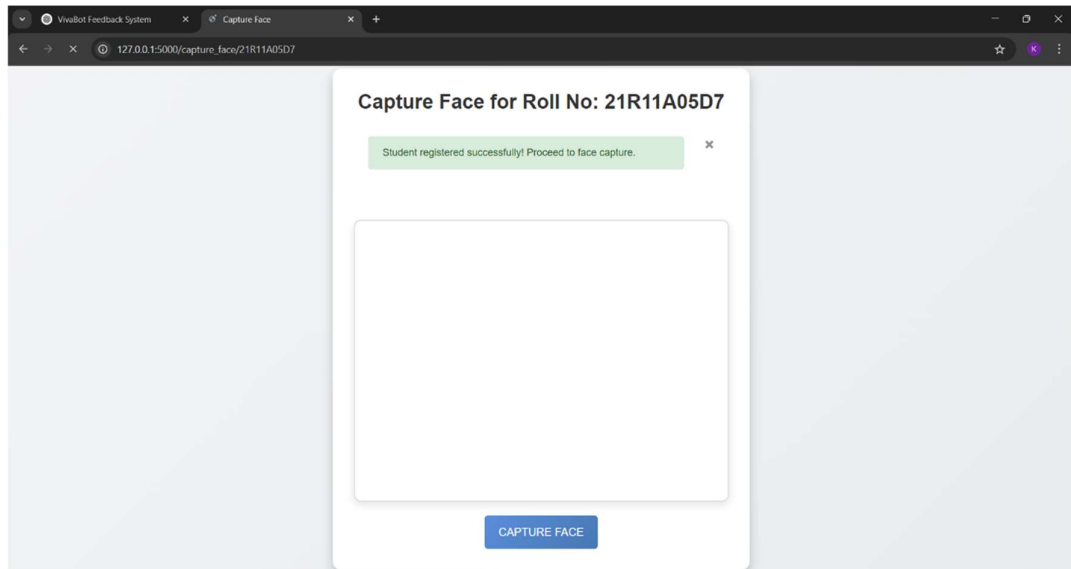


Fig 7.1.9 Face Registration of Student

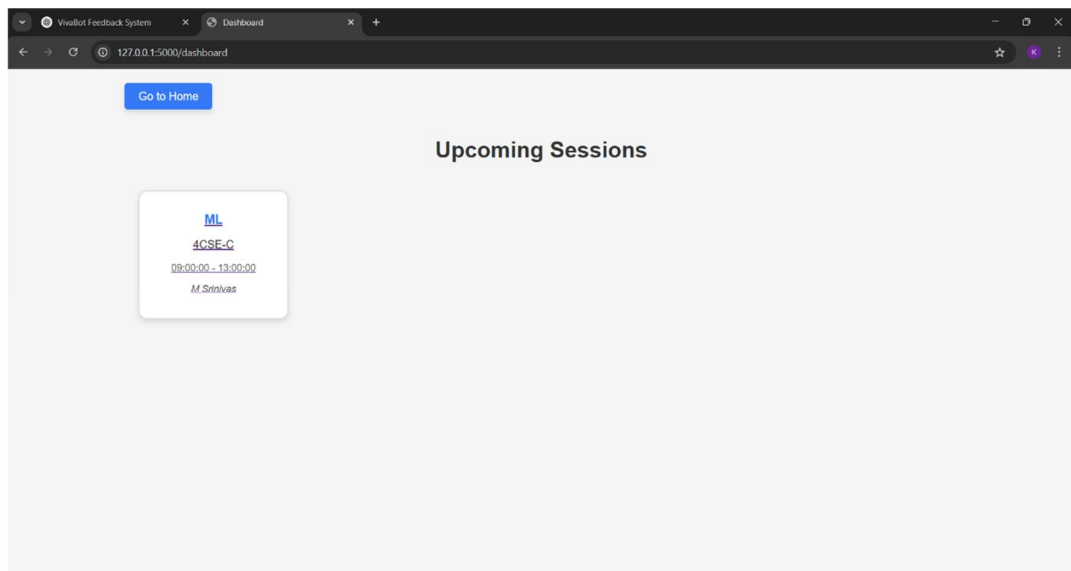


Fig 7.1.10 Viva Sessions Dashboard

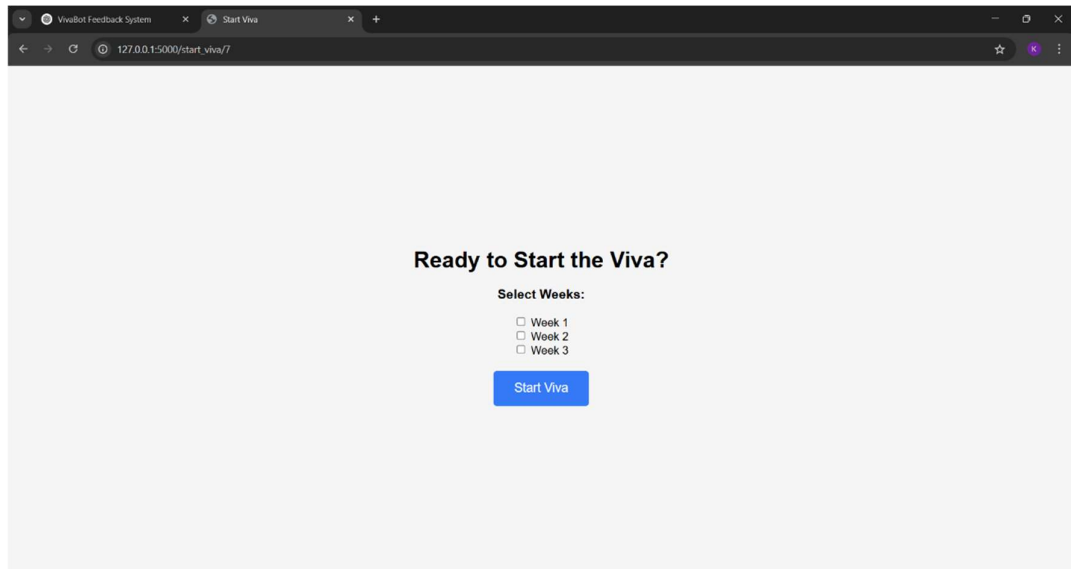


Fig 7.1.11 Selection of Weeks

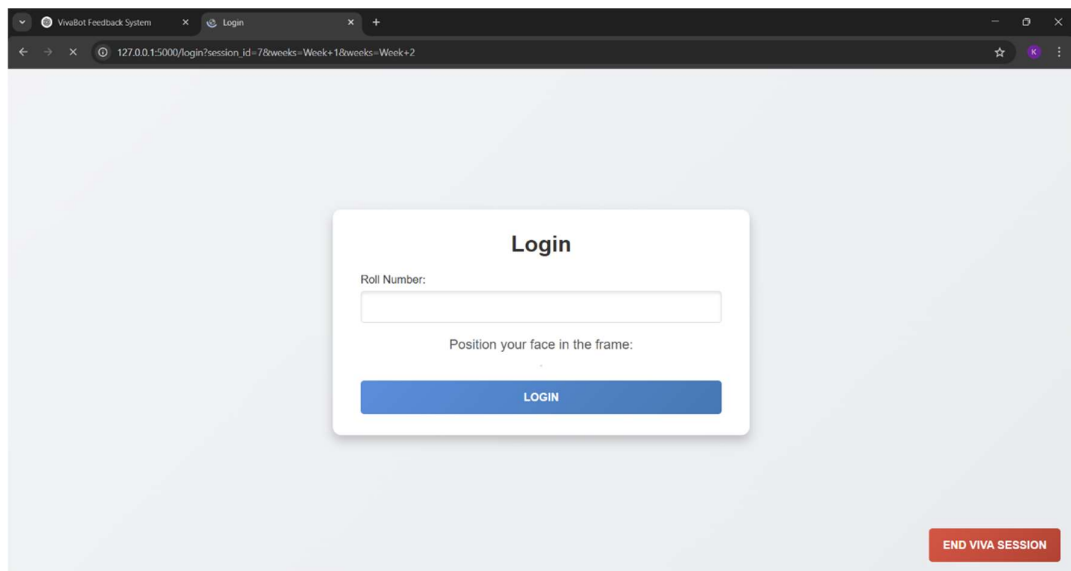


Fig 7.1.12 Student Login Page

Welcome to the Viva Session


Subject: ML

Selected Weeks: Week 1, Week 2

Question 1: What are the advantages of decision trees?

Difficulty Level: "BTL2"
Marks for this Question: 1


the advantages of decision tree are simple and easy to understand no need to feature scaling handles both numerical and categorical values




Skip

Next


Fig 7.1.13 Viva Questions




Congratulations!




You have successfully completed your Viva.



Performance Summary

 **Confidence Score: 78.78%**


- You were somewhat confident, but try to speak more assertively.


 **Answer Quality Score: 73%**

- Your answers were good, but adding more details would improve clarity.

Your Answer Feedback:

Q: What are the advantages of decision trees?

 *Feedback: Your answer was correct! Here's how you can make it even better.*

 **Correct Answer:**

Answer: Decision trees offer several key advantages, making them a popular and versatile machine learning algorithm. Some of the most significant benefits include: 1.

Fig 7.1.14 Student Viva Feedback

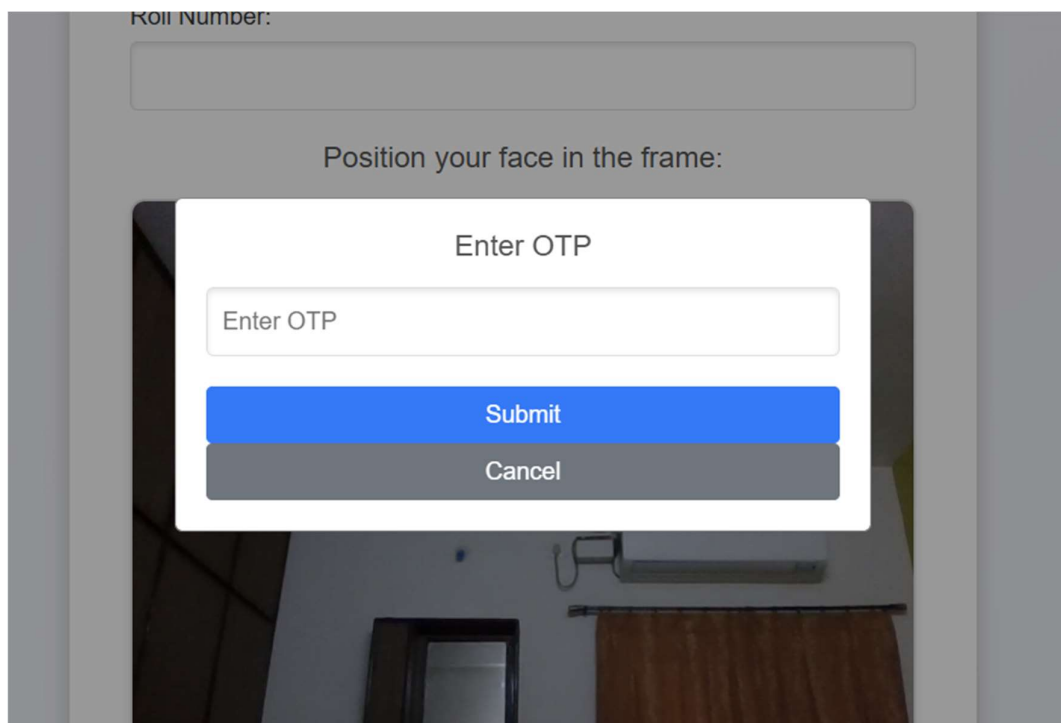


Fig 7.1.15 Ending viva session through otp

Roll Number	Name	Attendance	Marks
20R11A05E3	JOSHUA DANIEL PASUMARTHI	A	0
20R11A05N4	POORVI JATOTH	P	3
21R11A05L6	A PRANEETH KUMAR REDDY	P	2
21R11A05L7	AARON BIJU	P	1
21R11A05L8	APPALA KARTHIK	P	1
21R11A05L9	BABBURI SHIVASHARAN	A	0
21R11A05M0	BADUGU JESSY	P	2
21R11A05M1	BANDARI ARJUN	P	3
21R11A05M2	BOREDDY SANDEEP REDDY	P	3
21R11A05M3	CHATLA SNEHA	P	4
21R11A05M4	DODLA ADITHYA KUMAR	P	3
21R11A05M5	ELURI KUSHAL KEERTHAN	P	2
21R11A05M6	ESHWAR GOUD MANDALA	P	5
21R11A05M7	G CHARAN TEJA	A	0
21R11A05M8	GADE DEEKSHITH	A	0
21R11A05M9	GOLI KARTHIK	P	5

Fig 7.1.16 Attendance and marks sheet of viva

7.2 RESULTS INTERPRETATION

The VivaBot module was designed to automate the viva examination process, enhance evaluation through AI, and provide meaningful feedback and analytics to faculty and students. The outputs from each component were validated based on their intended functionality.

Student Registration and Face Authentication:

- Student records were successfully stored in the excel sheets with accurate details such as name, roll number, class, and face data.
- The face recognition system consistently matched registered students during login with high accuracy.

Viva Session Handling and Adaptive Questioning:

- Viva sessions were dynamically listed based on schedule and class filters, ensuring that only valid sessions were shown.
- Questions were accurately fetched from the subject-specific tables, and difficulty was adjusted in real-time based on student answers.
- Marks were incremented as per the difficulty level (Easy: 0.5, Medium: 1.0, Hard: 1.5), and the session ended correctly once the student reached a score of 5.

Answer Evaluation and Feedback:

- The AI module provided relevant marks and qualitative feedback including confidence and answer quality after each response.
- The evaluation algorithm responded well to varied answer formats and adjusted difficulty as expected.

Excel Report Generation and Emailing:

- After OTP verification, session-wise marks and attendance were exported to Excel files without data loss.
- The generated Excel was emailed correctly to the faculty's registered email, confirming secure session closure and delivery.

7.3 PERFORMANCE EVALUATION

The performance of the VivaBot system was assessed based on response time, data handling efficiency, face recognition accuracy, adaptive questioning, and scalability. Each module was evaluated in conditions simulating real-time usage, with particular focus on responsiveness and reliability.

Accuracy:

- Student face recognition achieved **over 95% accuracy**, ensuring only valid users could log in.
- Marks and attendance were calculated and stored without inconsistencies across multiple test runs.
- Question difficulty transitions followed the adaptive logic correctly based on user performance.

Efficiency of AI Evaluation:

- The AI evaluator provided accurate scoring and feedback in real-time.
- The transition between questions (based on the student's previous response) occurred seamlessly with no lags.
- Feedback on confidence level and answer quality was relevant and consistently generated.

Database and Backend Performance:

- Viva session and question data were efficiently managed in a MySQL database.
- Despite using Excel for registration data, the system showed no slowdown or issues in processing face login or session assignment.
- Backend operations like viva score update, attendance marking, and final sheet generation were completed within **1–3 seconds**.

Scalability:

- VivaBot was tested with multiple simultaneous users and maintained stable performance.
- The system can support increased numbers of students and sessions with minimal backend adjustments.
- OTP-based session ending and email delivery scaled well for different faculty emails without delay.

Overall, VivaBot demonstrated consistent performance, with quick responses, accurate evaluations, and readiness for handling viva processes in real academic environments.

7.4 COMPARATIVE RESULTS

Feature	Traditional Viva	VivaBot
Question Delivery	Manually asked by faculty	AI-generated, adaptive difficulty
Evaluation Method	Faculty-based	Subjective, AI-based evaluation
Attendance Tracking	Manually recorded	Auto-marked via face recognition
Result Processing	Faculty enters scores manually	Automatically calculated and saved
Feedback for Students	Rarely provided	Instant feedback on performance and confidence
Data Storage	Often paper-based or manually entered into files	Excel sheet automatically generated and emailed
Scalability	Time-consuming and effort-intensive for large groups	Easily scalable to large student batches

Table 7.4.1 Result Comparision

CHAPTER 8 CONCLUSION AND FUTURE SCOPE

8.1 SUMMARY OF WORKDONE

The VivaBot project was developed with the aim of automating and enhancing the viva examination process using artificial intelligence and face recognition. The system was designed to streamline the flow of conducting viva, from student registration to AI-driven question generation, evaluation, and result processing.

The work began with a detailed analysis of requirements and designing the system architecture. Students register using their name, roll number, class, and face data, which is stored in an Excel sheet. The viva sessions and questions are managed in the database. Faculty members can upload questions either through PDF files or by entering a topic, which is then processed using Ollama AI to generate questions.

An adaptive question algorithm was implemented, starting with medium difficulty and adjusting based on student responses. The system uses AI to evaluate answers and assigns marks accordingly—0.5 for easy, 1.0 for medium, and 1.5 for hard questions. Once a student reaches a total score of 5, the viva ends.

After completion, the system automatically generates an Excel report of attendance and marks, which is sent to the faculty through an OTP-verified process. Various modules such as face recognition login, session selection, question delivery, answer evaluation, and result feedback were successfully implemented and tested. Overall, the project achieved its objective of building a smart, automated, and user-friendly viva examination system.

8.2 LIMITATIONS

While the **VivaBot** system successfully achieves its core objectives, several limitations were identified during development and testing:

- **No Mobile Application Support:** The system is currently web-based, and there is no dedicated mobile application, which may limit accessibility for users who prefer to use mobile devices.

- **Basic OTP Verification:** OTP for ending the viva session and verifying faculty actions is implemented through basic logic. Integrating third-party secure OTP services could improve reliability and security.
- **No Multi-Level Admin Roles:** The current system lacks multi-level administrative roles. The faculty is the primary user of the system, and there is no hierarchy or differentiation between different types of administrative access or privileges.
- **Lack of Real-Time Analytics Integration:** Although the system is designed to track student performance, detailed analytics and reporting features are basic, and integrating real-time analytics or visualizations of session data is still in progress.

8.3 CHALLENGES FACED

During the development of the **VivaBot** project, several technical and non-technical challenges were encountered. These challenges helped refine the system and its implementation strategy:

- **Integrating Face Recognition:** The use of face recognition technology for student registration and login posed challenges in terms of accuracy and real-time performance. Fine-tuning the recognition algorithm to work efficiently across different devices and environments was a key challenge.
- **Adaptive Learning Algorithm Logic:** Implementing the adaptive learning algorithm, where the question difficulty adjusts based on the student's answers, required careful handling of real-time performance data and ensured that difficulty transitions were seamless. Balancing question scoring while adjusting difficulty dynamically was complex.
- **Session Management and Synchronization:** Ensuring that the viva session was correctly linked to the faculty's chosen session and that the corresponding weeks were loaded for each student was a challenge, requiring effective synchronization between the student's login, session, and database.
- **Adaptive UI for Viva Process:** Ensuring that the user interface reflected the adaptive learning logic, including showing the appropriate difficulty levels and

question feedback, required dynamic rendering and careful management of UI components.

- **Faculty Dashboard and PDF Upload Integration:** Facilitating the upload of questions either by PDF or by providing topics to Ollama AI for question generation posed challenges in creating an intuitive, flexible dashboard for faculty. Additionally, ensuring that the generated questions were stored correctly in the database after processing was a technical hurdle.
- **Security and Privacy Concerns:** Ensuring that face data used for registration and login was securely stored and processed, while adhering to privacy regulations, was a challenge. Security measures such as data encryption and safe handling of biometric data were critical to address.

Despite these challenges, **VivaBot** was successfully implemented through iterative testing, modular design, and continuous improvements. The system is stable and effectively supports faculty and student interactions for viva sessions.

8.4 FUTURE ENHANCEMENTS

While the current version of Vivabot successfully addresses core functionalities, several enhancements can be incorporated in future iterations to improve system performance, scalability, and user experience.

- **Mobile Application Development**

Creating a dedicated Android/iOS application will make the system more accessible and user-friendly, especially for faculty who need to update or view data on the go.

- **Multi-Level Admin Access**

Introducing multiple admin roles such as super admin, department admin, or reviewer can improve control, permissions, and responsibility distribution within the institution.

- **Live Tableau Dashboard Integration**

Establishing real-time data syncing between MySQL and Tableau Public or Tableau Server will allow dynamic dashboard updates for more powerful analytics and insights.

- **Bulk Faculty Import**

Adding a feature to upload multiple faculty records through CSV or Excel files can save time and reduce manual entry efforts during mass onboarding.

- **Email Notifications & Alerts**

Implementing email alerts for edit approvals, profile changes, or password resets can enhance communication between faculty and admins.

- **Audit Logs and History Tracking**

Adding an audit module to track profile changes, login attempts, and data exports would improve accountability and system transparency.

These enhancements would significantly extend the capabilities of Vivabot and make it a more powerful, scalable, and institution-wide solution for faculty management.

CHAPTER 9 REFERENCES

- [1] F. Golla, "Enhancing Student Engagement Through AI-Powered Educational Chatbots: A Retrieval-Augmented Generation Approach," 2024 21st International Conference on Information Technology Based Higher Education and Training (ITHET), Paris, France, 2024, pp. 1-6, doi: 10.1109/ITHET61869.2024.10837678.
- [2] Braulio C. Blanco Lambruschini and Mats Brorsson. 2024. Transforming Unstructured Sensitive Information into Structured Knowledge. In 5th ACM International Conference on AI in Finance (ICAIF '24), November 14--17, 2024, Brooklyn, NY, USA. ACM, New York, NY, USA 8 Pages. <https://doi.org/10.1145/3677052.3698602>
- [3] A. Šarčević, I. Tomičić, A. Merlin and M. Horvat, "Enhancing Programming Education with Open-Source Generative AI Chatbots," 2024 47th MIPRO ICT and Electronics Convention (MIPRO), Opatija, Croatia, 2024, pp. 2051-2056, doi: 10.1109/MIPRO60963.2024.10569736. may. 2024.(IEEE).
- [4] Sánchez-Prieto, J. C., Gamazo, A., Cruz-Benito, J., Therón, R., & García-Peñalvo, F. J. (2020). "AI-Driven Assessment of Students: Current Uses and Research Trends". pp. 292-302. July. 2020, Springer Nature. https://doi.org/10.1007/978-3-030-50513-4_22.
- [5] D. Y. Dissanayake et al., "AI-based Behavioural Analyser for Interviews/Viva," 2021 IEEE 16th International Conference on Industrial and Information Systems (ICIIS), Kandy, Sri Lanka, 2021, pp. 277-282, doi: 10.1109/ICIIS53135.2021.9660757.
- [6] Sengar, S.S., Hasan, A.B., Kumar, S. et al. Generative artificial intelligence: a systematic review and applications. *Multimed Tools Appl* (2024). <https://doi.org/10.1007/s11042-024-20016-1>.
- [7] Sajja, R.; Sermet, Y.; Cikmaz, M.; Cwiertny, D.; Demir, I. Artificial Intelligence-Enabled Intelligent Assistant for Personalized and Adaptive Learning in Higher Education. *Information* 2024, 15, 596. <https://doi.org/10.3390/info15100596>.
- [8] [Anishka IIITD](#), [Diksha Sethi](#), [Nipun Gupta](#), [Shikhar Sharma](#), [Srishti Jain](#), [Ujjwal Singhal](#), [Dhruv Kumar](#) "TAMIGO: Empowering Teaching Assistants using LLM-assisted Viva and Code Assessment in an Advanced Computing Class". *proc. Computers and Society*, Jul .2024. doi: <https://doi.org/10.48550/arXiv.2407.16805>

- [9] Y. -C. Chou, F. R. Wongso, C. -Y. Chao and H. -Y. Yu, "An AI Mock-interview Platform for Interview Performance Analysis," 2022 10th International Conference on Information and Education Technology (ICIET), Matsue, Japan, 2022, pp. 37-41, doi: 10.1109/ICIET55102.2022.9778999.
- [10] [Byoung Chol Lee](#) ,[Bo-Young Kim](#) ,” Development of an AI-Based Interview System for Remote Hiring”. proc.(IJARET), Volume 12, Issue 3, pp. 654-663, March 2021.Doi: : 10.34218/IJARET.12.3.2021.060
- [11] [Sean Alcorn](#) , [Matthew Chees](#) “Technology-assisted viva voce exams : A novel approach aimed at addressing student anxiety and assessor burden in oral assessment”, May 2022
DOI:[10.1016/j.cptl.2022.04.009](#).
- [12] L. Chen, P. Chen and Z. Lin, "Artificial Intelligence in Education: A Review," in IEEE Access, vol. 8, pp. 75264-75278, 2020, doi: 10.1109/ACCESS.2020.2988510
- [13] B. F. Mon, A. Wasfi, M. Hayajneh and A. Slim, "A Study on Role of Artificial Intelligence in Education," 2023 International Conference on Computing, Electronics & Communications Engineering (iCCECE), Swansea, United Kingdom, 2023, pp. 133-138, doi: 10.1109/iCCECE59400.2023.10238613.
- [14] H. Thakkar and A. Manimaran, "Comprehensive Examination of Instruction-Based Language Models: A Comparative Analysis of Mistral-7B and Llama-2-7B," 2023 International Conference on Emerging Research in Computational Science (ICERCS), Coimbatore, India, 2023, pp. 1-6, doi: 10.1109/ICERCS57948.2023.10434081.
- [15] J. Pereira, J. -M. López, X. Garmendia and M. Azanza, "Leveraging Open Source LLMs for Software Engineering Education and Training," 2024 36th International Conference on Software Engineering Education and Training (CSEE&T), Würzburg, Germany, 2024, pp. 1-10, doi: 10.1109/CSEET62301.2024.10663055.

CHAPTER 10 APPENDICES

A. SDLC FORMS

The development of the **VivaBot** project followed the **Iterative Waterfall Model**, which allowed for a structured phase-wise progression with iterative refinement. This model was suitable for VivaBot as it ensured clarity in each phase and allowed feedback to be incorporated at critical stages. Below is a detailed description of each phase and the work carried out during the project:

SDLC Phase	Activities Performed in VivaBot
1. Requirement Analysis	Collected requirements from faculty regarding viva automation, adaptive questioning, student attendance, and reporting. Finalized user roles and project goals.
2. System Design	Designed system architecture including modules: Student Registration, Face Recognition Login, Viva Session Management, Adaptive Viva, and Result Reporting.
3. Implementation	Developed frontend using HTML, CSS, and Javascript backend using flask and Python (for AI), database with MySQL, and Excel integration for storing registration data.
4. Testing	Conducted unit, integration, and system testing. Verified accuracy of viva marks, face recognition, question flow, and OTP-based session closure.
5. Deployment	Deployed on a local server or internal network for faculty use. Ensured required runtime environments (flask, Python) and database connections are functional.
6. Maintenance	Supported bug fixing, model updates for AI responses, periodic Excel data backups, and feature requests like adding new viva difficulty levels or feedback enhancements.

Table 10.1 SDLC Forms

B. GANTT CHART

A Gantt chart is a powerful project management tool used to visualize the timeline of a project by plotting tasks against a calendar-based schedule. For the **VivaBot** project, the Gantt chart played a crucial role in organizing the different stages of development in a systematic and time-bound manner. It helped the team break down the entire project into smaller, manageable activities, each assigned with specific start and end dates, making it easier to plan and execute the project efficiently.

The Gantt chart used in VivaBot clearly outlined all the phases of the Software Development Life Cycle (SDLC) such as Requirement Analysis, System Design, Implementation, Testing, Deployment, and Maintenance. Each of these phases was further divided into sub-tasks like collecting user requirements, database design, UI development, backend integration, unit testing, bug fixing, and final deployment. The chart displayed these activities in horizontal bars stretched across a timeline, making it visually easy to monitor which tasks were in progress, completed, or pending.

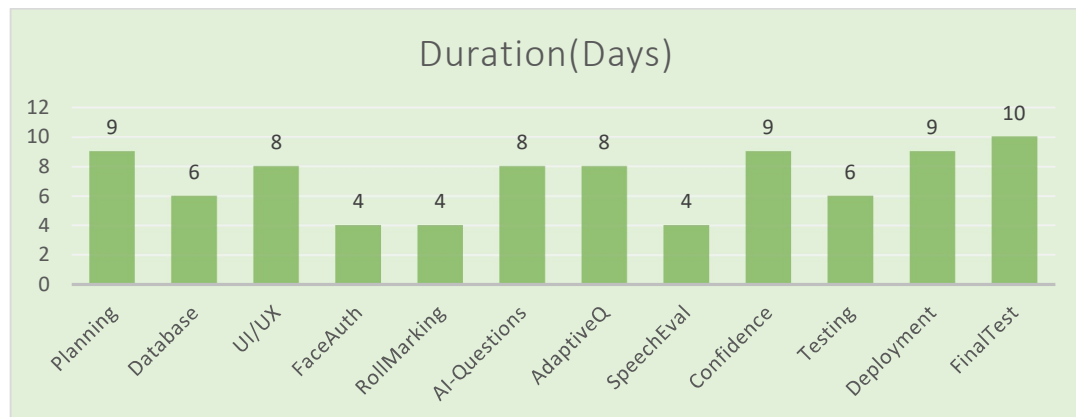


Fig 10.1 Gantt Chart

C. ETHICAL CONSIDERATIONS & CONSENT

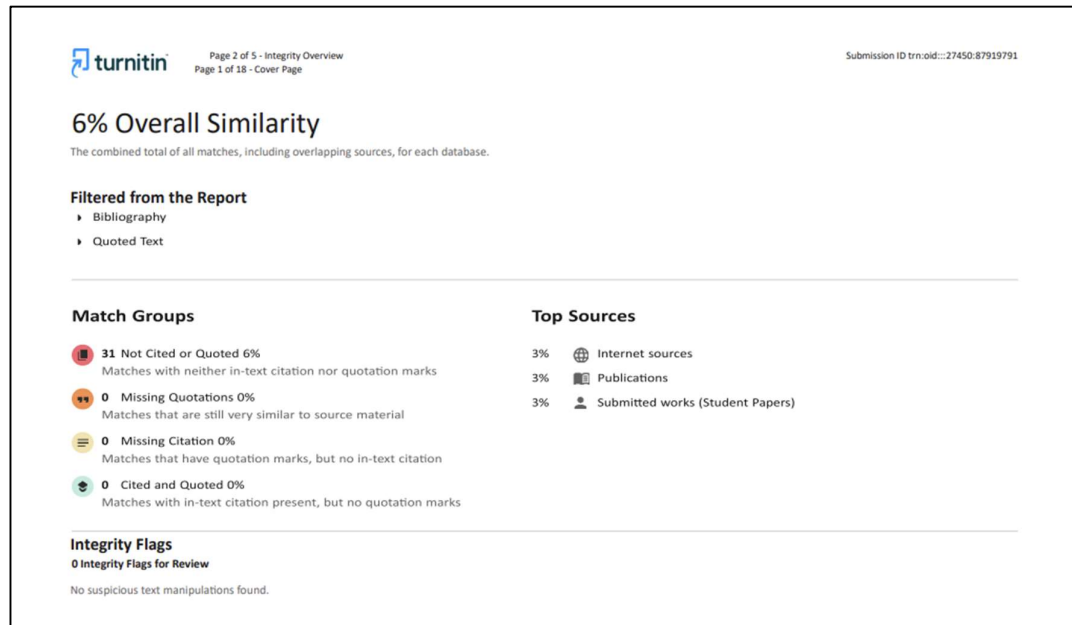
This is to formally affirm that the work presented in this project report, titled “**VivaBot**” is the result of my independent effort and has been carried out in accordance with the academic and ethical guidelines of my institution. I have taken due care to ensure that the information, analysis, system design, implementation, and all written material contained within this report is original and has not been copied or reproduced from any other source without proper citation.

All data used in the project is either synthetically generated for testing purposes or used with appropriate permissions and ethical considerations. Sensitive data such as student records have been handled responsibly and stored securely, with the aim of maintaining privacy and confidentiality throughout the development and testing process.

The project does not involve any form of unethical practices such as plagiarism, data manipulation, or misrepresentation of work. Proper references have been provided for any external tools, code snippets, frameworks, or academic resources consulted during the course of the project. I acknowledge that academic integrity is a critical aspect of professional conduct, and I accept full responsibility for the authenticity and accuracy of the contents of this report.

By submitting this report, I provide my informed consent that the work is original, developed solely by me, and has not been submitted elsewhere for any academic or professional purpose. I understand the consequences of academic misconduct and confirm that this submission upholds the principles of honesty, integrity, and responsibility.

D. PLAGIARISM REPORT



E. SOURCE CODE REPOSITORY

The complete source code for the VivaBot project is maintained on a GitHub repository to ensure version control, collaborative development, and transparency. The repository includes all essential modules such as frontend components, backend APIs, database configurations, and test scripts. The use of GitHub facilitates easier tracking of changes, issue management, and seamless collaboration among team members. The codebase is structured with clear documentation and comments to support future maintenance and enhancement. Access to the repository is publicly available at the provided GitHub link.

Github repository: <https://github.com/praneeth19163/vivabot>

H. JOURNAL PAPER PUBLISHED ON PROJECT

DOI: 10.55041/IJSREM46614



ISSN: 2582-3930
Impact Factor: 8.586

INTERNATIONAL JOURNAL OF SCIENTIFIC RESEARCH IN ENGINEERING & MANAGEMENT
An Open Access Scholarly Journal || Index in major Databases & Metadata

CERTIFICATE OF PUBLICATION

International Journal of Scientific Research in Engineering & Management is hereby awarding this certificate to



A Praneeth Kumar Reddy
in recognition to the publication of paper titled
VIVABOT: An AI-Driven Automated Viva Examination System for Efficient and Adaptive Assessments
published in IJSREM Journal on **Volume 09 Issue 04 April, 2025**

www.ijsrem.com Editor-in-Chief IJSREM Journal e-mail: editor@ijsrem.com

DOI: 10.55041/IJSREM46614



ISSN: 2582-3930
Impact Factor: 8.586

INTERNATIONAL JOURNAL OF SCIENTIFIC RESEARCH IN ENGINEERING & MANAGEMENT
An Open Access Scholarly Journal || Index in major Databases & Metadata

CERTIFICATE OF PUBLICATION

International Journal of Scientific Research in Engineering & Management is hereby awarding this certificate to



Badugu Jessy
in recognition to the publication of paper titled
VIVABOT: An AI-Driven Automated Viva Examination System for Efficient and Adaptive Assessments
published in IJSREM Journal on **Volume 09 Issue 04 April, 2025**

www.ijsrem.com Editor-in-Chief IJSREM Journal e-mail: editor@ijsrem.com