Bachelor of Computer Application

[B.C.A]

PROJECT REPORT

LearnieVerse

By

Anas Ahmad

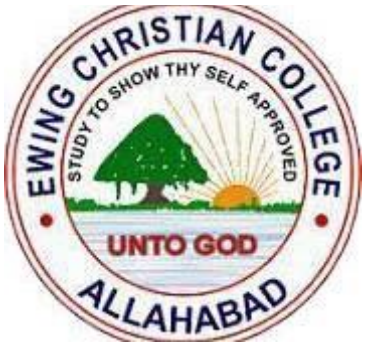
(ECC2214028)

(413017)

Adviser:

Mr. Mohammad Irsad

DEPARTMENT OF COMPUTERS



Ewing Christian College, Prayagraj

Gaughat, Prayagraj Uttar Pradesh: 211003

April 2025

Project Evaluation

1. Project Report	[80]	
2. Project Demonstration	[80]	
3. Project Viva	[80]	
	[240]	

Internal Examiner

External Examiner

Declaration

I hereby declare that the work presented in this Project Report titled" LearnieVerse" submitted to the Department of Computers, Ewing Christian College, Prayagraj in partial fulfilment of the requirements for the award of the degree of Bachelor of Computer Application [B.C.A.] is a Bonafide record carried out under the supervision of Mohammad Irshad. The contents of this Project Report in full or in parts, have not been submitted to, and will not be submitted by me to, any other Institute or University in India or abroad for the award of any degree or diploma.

1. Student Name
   Roll Number:
   Enrolment Number Signature

Date:

Departments of Computers
Bachelor of Computer Application [BCA]
Ewing christian college, Prayagraj

CERTIFICATE

This is to certify that the Project Report titled ”_____”, submitted by _____ Roll Number:_____ Enrollment No: _____ to the Department of Computers, Ewing Christian College, Prayagraj for the award of the degree of Bachelor of Computer Application [B.C.A.], is a bonafide record of the Project work done by him/her under my supervision.

Name of Advisor
Advisor

ACKNOWLEDGEMENTS

I’m super stoked to give a massive shout-out to everyone who made the LearnieVerse project possible. First off, huge thanks to my project advisor and professors at Ewing Christian College who dropped invaluable knowledge and kept me motivated through the ups and downs of research. Your guidance lit the way and helped me level up my skills!

I also want to extend gratitude to my peers and teammates whose fresh perspectives and creative ideas transformed challenges into exciting opportunities. Special mention to the tech support crew and all the behind-the-scenes heroes—your prompt help and positive vibes ensured that every glitch was just another stepping stone toward success.

Finally, a heartfelt thank you to my family and friends for the endless encouragement and support. You all are the real MVPs, and your confidence in me fuels my drive to innovate and push boundaries.

Thanks for riding this journey with me—here’s to future breakthroughs and collaborative success!

ABSTRACT

In today's hyper-connected era, LearnieVerse dives into the evolution of digital learning, blending tech innovation with education to create a user-friendly, interactive platform. This project addresses the challenges of engaging modern learners by integrating cutting-edge tools and methodologies that make learning not only effective but also vibey and accessible. Through a combination of qualitative research and system prototyping, we explored how digital interfaces can enhance comprehension and retention. The study revealed promising trends in user engagement, efficiency, and satisfaction, indicating a significant positive shift when learning environments are designed with future-forward thinking. Overall, the findings underscore the potential of LearnieVerse to disrupt traditional education models, inspiring further exploration into dynamic, digital learning ecosystems.

Table of Contents

1. Introduction

1.1 Background and Motivation

In our hyper-connected digital era, there's a crazy high demand for productivity apps that do more than just help you check off tasks—they need to enhance your whole workflow. Traditional tools cover the basics like task management, note-taking, and scheduling, but they often miss the secret sauce: advanced AI features that can really boost your efficiency. That's where LearnieVerse comes in, reimagining digital productivity for a new generation.

Motivation:

The spark behind LearnieVerse is all about moving past the old-school productivity apps. Imagine an app that not only helps you stay organized but also gives you access to AI-powered perks like automated resume scoring, news summarization, and even medical advice—all in one sleek package. LearnieVerse is built to adapt to your needs, offering smart tools that take the hassle out of your day-to-day tasks. The aim? To create a comprehensive, next-level productivity hub that makes life smoother, smarter, and way more efficient.

1.2 Project Purpose

The core mission of LearnieVerse is to craft a unique AI-powered productivity app that revolutionizes how you manage your daily grind. This platform brings together a powerful mix of features—a killer to-do manager, intuitive notes system, automated resume scoring, an insightful medical chatbot, concise news summarization, and real-time news updates—all while keeping your data secure. By merging these dynamic capabilities into one spot, LearnieVerse fills the gap in today's market for a versatile and secure tool that's tailor-made for modern productivity with a futuristic twist.

1.3 **Context & Background:**

Traditional education models are struggling to keep pace with the dynamic needs of modern students. With the continuous emergence of digital tools and platforms, there’s a clear demand for an immersive learning ecosystem that not only delivers information but also fosters active participation and creativity. LearnieVerse is conceptualized as a solution to bridge this gap, leveraging innovative technologies to make learning both interactive and personalized.

1.4 Problem Statement:

Despite widespread digital transformation, many existing educational platforms fall short when it comes to engaging users and offering a seamless learning experience. LearnieVerse was developed to tackle these challenges head-on by addressing issues related to content engagement, usability, and adaptability in diverse learning environments.

1.5 Objectives & Research Questions:

- **Objectives:**
  - To create a user-centric digital platform that enhances the learning experience through interactivity.
  - To integrate modern technological solutions that improve content delivery and student engagement.
  - To evaluate the platform's effectiveness in enhancing learning outcomes compared to traditional systems.
- **Research Questions:**
  - How does an interactive digital platform influence student engagement and learning effectiveness?
  - What specific features contribute most to an enhanced learning experience in digital environments?
  - Can innovative tech solutions in LearnieVerse reshape traditional educational approaches?

2. Objectives

The main objectives of the LearnieVerse project are cantered on innovating the digital learning landscape and creating an engaging, user-friendly platform that enhances educational outcomes. Here's a breakdown of our goals:

- **Enhance Learning Engagement:**
  Develop interactive features that keep users motivated and actively involved in their learning journey. This includes intuitive navigation, gamification elements, and real-time feedback mechanisms.
- **Integrate Advanced Technologies:**
  Leverage modern tech solutions such as artificial intelligence, data analytics, and responsive design to tailor learning experiences. Our aim is to create a smart platform that adapts to individual learning styles and needs.
- **Improve User Experience:**
  Prioritize a clean, accessible, and aesthetically appealing interface that simplifies the learning process. User-centric design is at the heart of LearnieVerse, ensuring the platform is welcoming and easy to use for all.
- **Facilitate Comprehensive Digital Learning:**
  Create an environment that supports diverse learning materials and methods—ranging from videos and quizzes to interactive simulations—ensuring a holistic approach to education.
- **Measure Impact Effectively:**
  Implement robust evaluation metrics and feedback loops to assess the effectiveness of the platform. This helps in continuously refining the system based on real user data and emerging trends

3. Scope

The scope of **LearnieVerse** centers on the development and deployment of a dynamic, all-in-one, AI-powered educational platform designed to support students and educators in managing classes, assignments, and assessments while enhancing the learning experience through smart features. LearnieVerse combines modern tools like class management, note sharing, live sessions, test creation, and real-time interaction to create an immersive academic environment—secure, flexible, and future-ready.

Included in the Project Scope:

1. **Classroom & Student Management**
    - Allows teachers to create, edit, and manage virtual classrooms.
    - Teachers can assign students, monitor participation, and control class visibility through dynamic dashboards.
2. **Notes Creation & Secure Sharing**
    - Enables both teachers and students to create, edit, and download study materials.
    - Notes can be securely shared using token-based links to maintain privacy and access control.
3. **Live Class Sessions**
    - Supports real-time video calling using WebRTC to facilitate virtual classroom interactions.
    - Teachers can host live classes, mark attendance, and manage student participation smoothly.
4. **Test & Quiz Management**
    - Offers tools for creating, editing, and publishing tests.
    - Teachers can define durations, assign specific students, and track performance metrics efficiently.
5. **AI-Enhanced Feedback (Coming Soon)**
    - Future scope includes AI-generated insights on student performance, personalized learning paths, and automated grading for quizzes and short answers.
6. **Attendance Tracking**

- Built-in attendance feature for each live session helps track student engagement over time.
- Attendance records are securely stored and easily accessible by the teacher.

7. **Data Security & Privacy**
    - Emphasizes user data protection through secure authentication, encrypted session management, and controlled sharing mechanisms.
8. **Cross-Platform Compatibility**
    - LearnieVerse is built to work across devices—from laptops and desktops to tablets—making sure learning is never restricted by hardware limitations.

4. Literature Review

4.1. Existing EdTech Platforms and Virtual Classrooms

EdTech platforms have grown exponentially with a focus on digital learning, course delivery, and collaboration tools. Popular systems like **Google Classroom**, **Microsoft Teams for Education**, and **Moodle** provide features like assignment handling, grade tracking, and live classes. However, many of these platforms lack deep integration of AI-driven tools for personalization, dynamic classroom management, and real-time feedback.

Limitations in Current Solutions:

Despite their widespread adoption, existing tools often require multiple integrations for complete functionality and may not fully prioritize user experience or privacy. LearnieVerse addresses this gap by offering a unified platform for note sharing, test creation, live sessions, attendance tracking, and class management—purpose-built for an engaging, secure, and flexible educational environment.

4.2. Natural Language Processing (NLP) in Education

NLP has become a game-changer in educational tools, powering chatbots, essay scoring systems, and intelligent tutoring systems. Transformer models like **BERT**, **GPT**, and **T5** have revolutionized how machines understand and generate language.

Use in LearnieVerse:

LearnieVerse utilizes NLP to eventually power features such as automated content summarization and AI-driven feedback mechanisms. These technologies help simplify educational content, provide quick summaries of lessons, and offer personalized interactions between students and the platform.

NLP in EdTech:

Research shows that conversational AI can enhance learning by offering 24/7 doubt clarification, improving comprehension through simplified language generation, and enabling multilingual support.

4.3. Assessment Tools and Automated Grading

Digital assessment systems are increasingly incorporating machine learning to evaluate answers and generate scores. Platforms like **Gradescope** and **Edmodo** enable quiz creation and basic auto-grading features.

Application in LearnieVerse:

LearnieVerse includes a test creation and management module where teachers can define question banks, set timers, and assign tests. While initial versions allow manual creation and editing, future iterations aim to integrate **AI-based grading systems** for short answers and auto-suggestions for improvement.

Existing Research:

Studies emphasize the importance of timely feedback and adaptive testing. By incorporating AI, LearnieVerse plans to offer real-time performance insights, helping both educators and students understand strengths and areas for improvement.

4.4. Data Security in EdTech Applications

With the increase in online learning, securing user data has become crucial. Education platforms often handle sensitive information like student identities, assessments, and classroom records.

Data Security in LearnieVerse:

LearnieVerse emphasizes strong authentication systems, encrypted session handling, and secure token-based sharing for class links and notes. It prioritizes privacy by minimizing unnecessary data storage and adhering to secure design principles.

Security Challenges & Strategies:

Modern EdTech apps must comply with regulations like **GDPR** and **FERPA**. LearnieVerse adopts these standards and incorporates features like secure role-based access, HTTPS protocols, and session expiration mechanisms to safeguard users.

5. Project Design and Methodology

LearnieVerse is designed to redefine digital learning with a platform that unifies classroom management, interactive content sharing, and real-time engagement—all powered by AI-driven insights. The development process follows a structured approach to ensure that every feature enhances the user experience while maintaining robust security and scalability.

5.1 Development Model

LearnieVerse uses a structured Waterfall Model, which provides a clear, step-by-step blueprint for building an integrated educational platform. This model ensures that each phase is fully completed before transitioning to the next, keeping our progress on track as we layer advanced AI and user-friendly features into the system.

Phase 1: Requirements Analysis

- **Objective:**
  Gather and document all educational and administrative features needed for LearnieVerse.
- **Activities:**
  - Hold discussions with educators, students, and tech stakeholders to define core requirements such as classroom creation, assignment management, secure note sharing, live session hosting, test administration, and AI-based feedback.
  - Analyze existing EdTech tools to identify gaps and opportunities.
- **Outcome:**
  A detailed Software Requirements Specification (SRS) that outlines necessary features, security protocols, and AI functionalities tailored for a cutting-edge educational platform.

Phase 2: System Design

- **Objective:**
  Develop a comprehensive blueprint for the LearnieVerse architecture and its individual modules.
- **Activities:**
  - Design a system architecture integrating a robust backend (e.g., Flask for API management), a secure database (MongoDB), and an engaging frontend (React).

- Create detailed blueprints for each module, including classroom management, secure note-sharing, live class sessions, and test/quiz modules.
- Plan the integration of AI features for personalized feedback and intelligent content summarization.

- **Outcome:**
  A complete System Design Document with architectural diagrams, database schema, UI mockups, and integration pathways between AI components and user management systems.

Phase 3: Implementation (Coding)

- **Objective:**
  Develop each module in accordance with the design specifications.
- **Activities:**
  - Code core modules like the virtual classroom, notes system, live sessions, test management, and AI-based features (personalized feedback, content summarization, and adaptive learning insights).
  - Use Flask and Python for backend logic, MongoDB for data handling, and React to create a smooth, interactive user interface.
  - Implement secure user authentication, session management, and encrypted data sharing.
- **Outcome:**
  Fully functional, independently developed modules that integrate seamlessly to form the LearnieVerse platform.

Phase 4: Integration and Testing

- **Objective:**
  Integrate all modules and verify that the platform functions cohesively and reliably.
- **Activities:**
  - Connect the backend with the frontend to deliver a seamless user experience.
  - Conduct rigorous testing including unit, integration, and load testing (using tools like Locust) to evaluate performance and stability.
  - Validate AI functionalities, ensuring that personalized feedback and real-time features operate accurately.
  - Test cross-device and browser compatibility to maintain an accessible experience on all platforms.

- **Outcome:**
  A unified, tested LearnieVerse application with smooth transitions across modules and dependable AI responses.

Phase 5: Deployment

- **Objective:**
  Launch LearnieVerse in a live production environment.
- **Activities:**
  - Configure a hosting environment (e.g., AWS EC2 instance) optimized for real-time AI processing and educational data loads.
  - Deploy using Docker containers for consistency across development, testing, and production environments.
  - Implement domain management, SSL certificates, and DNS configurations to guarantee secure and reliable access.
- **Outcome:**
  LearnieVerse goes live on a secure domain, providing a scalable educational platform ready to support multiple simultaneous users.

Phase 6: Maintenance and Updates

- **Objective:**
  Continuously monitor, enhance, and scale LearnieVerse based on user feedback and emerging needs.
- **Activities:**
  - Regularly track application performance, focusing on AI module efficiency and server stability.
  - Gather insights and feedback from educators and students to roll out new features and fix any issues.
  - Periodically update AI models, code frameworks, and system dependencies to ensure ongoing compatibility, performance, and security.
- **Outcome:**
  An evolving, well-maintained LearnieVerse platform that consistently meets the innovative needs of modern education.

5.2 Requirements Gathering

Functional Requirements

These define what LearnieVerse must do, based on in-depth discussions with educators, students, and administrators:

- **Virtual Classroom Management:**
  - Enable teachers to create, edit, and manage digital classrooms.
  - Allow enrollment of students, scheduling of live sessions, and attendance tracking.
- **Notes Creation & Secure Sharing:**
  - Allow creation, editing, deletion, and categorization of study materials.
  - Facilitate secure sharing through token-based links ensuring privacy and controlled access.
- **Live Class Sessions:**
  - Support real-time interactive sessions, including video calling and chat features.
  - Enable engagement through features like Q&A, live polls, and on-screen annotations.
- **Test & Quiz Management:**
  - Provide tools for creating, editing, and administering tests/quizzes.
  - Enable timed assessments and real-time result feedback, with future support for AI-driven grading.
- **AI-Based Learning Insights:**
  - Integrate advanced algorithms for personalized content recommendations and performance feedback.
  - Implement natural language processing to summarize complex topics and generate easy-to-understand study notes.

Non-Functional Requirements

These ensure that LearnieVerse performs reliably, securely, and efficiently:

- **Performance:**
  - Maintain fast response times with minimal lag during user interactions.
  - Optimize AI processing to deliver near real-time personalized feedback.
- **Security:**
  - Use encryption for all data (including notes and session details).
  - Implement token-based authentication and secure HTTPS communications to protect sensitive information.
- **Usability:**

    - Design a clean, intuitive, and visually appealing interface using React.
    - Prioritize ease of navigation so even first-time users can get started quickly.
- **Scalability:**
    - Deploy on infrastructure that can scale seamlessly to accommodate an increasing number of users.
    - Ensure the backend is modular, allowing for future integration of more AI functionalities.
- **Maintainability:**
    - Document all modules and code thoroughly to ensure ease of updates.
    - Use modular coding practices to enable independent updates and troubleshooting.

System Requirements

Define the necessary hardware and software environments for LearnieVerse:

- **Software Requirements:**
    - Backend: Flask (API management), MongoDB (data storage), PyTorch (for AI processing).
    - Frontend: React for a dynamic, responsive user interface.
    - Deployment: Docker for containerization and AWS EC2 for hosting and scalability.
- **Hardware Requirements:**
    - A server setup with robust CPU and GPU capabilities (e.g., AWS t2.xlarge) to handle intensive AI computations and data processing.
    - Sufficient storage capacity for educational resources, user-generated content, and large datasets.

6. System Architecture Design for LearnieVerse

This section illustrates the journey of data and interactions within LearnieVerse—from the moment a user clicks a button to when AI-powered insights light up the screen. The Process Flow Diagram visually maps out how the system's layers work in harmony, ensuring a smooth and secure digital learning experience.

Purpose:

- **Clarify Data Movement:** The diagram shows how user inputs flow from the dynamic React-based frontend through to the Flask and Python-powered backend.
- **Showcase Integration:** It highlights where AI services jump in, running models via PyTorch, TensorFlow, or Hugging Face to process information like personalized feedback or intelligent content summaries.
- **Ensure Security & Scalability:** The flow emphasizes secure handling at every step—using encryption, secure authentication, and best practices for data privacy.

High-Level Process Flow:

1. **User Interface Interaction:**
    - **Frontend (React):** Users interact with intuitive UI components—submitting assignments, participating in live sessions, or accessing study materials.
    - Data is captured in real time and sent to the backend through secure API calls.
2. **Request Handling in the Backend:**
    - **API Layer (Flask/Python):** Receives user requests and processes them, performing tasks like authentication, session management, and database interactions.
    - Routes are defined to manage everything from class creation to test administration.
3. **AI Services Integration:**
    - **AI Layer:** Once the backend identifies the need for an AI-driven task (e.g., summarizing lecture notes, grading a quiz, or offering personalized tutoring tips), it sends the input data to the appropriate AI model.
    - Models execute on server-side frameworks (PyTorch, TensorFlow) and return the output—personalized insights or summarized content—to the backend.
4. **Response & Rendering:**

- The backend compiles data from both the core services and AI outputs and sends a response back to the frontend.
- **Dynamic Rendering:** React components update the view to reflect new data, such as instant feedback, updated classroom information, or refreshed study resources.

6.1 Backend Development for LearnieVerse

The backend of LearnieVerse is built using **Flask**, providing a robust, modular, and scalable infrastructure to support all platform functionalities. From managing users and classes to handling real-time interactions and test creation, the backend ensures secure, efficient, and seamless operations between the database and frontend.

1. Technologies Used

- **Flask**: Core web framework used for handling routes, APIs, and request/response cycles.
- **Flask Blueprints**: Used for modular route organization (e.g., student, teacher, auth, tests).
- **MySQL**: Relational database used to store structured data such as users, classes, attendance, and tests.
- **Flask-Login**: Handles user sessions and login/logout functionality.
- **Flask-Mail**: Sends verification emails, OTPs, and test notifications.
- **Jinja2**: Renders server-side templates for dynamic pages.
- **Werkzeug Security**: For secure password hashing and verification.
- **Flask-CORS**: Enables secure communication between the frontend (React) and Flask backend.
- **WebRTC (Future Scope)**: For implementing live video class functionality.

2. Key Backend Features

1. User Authentication & Role-Based Access

- **Signup & Verification**: Users register with email and password. A verification email with OTP is sent before account activation.
- **Login/Logout**: Authenticated via Flask-Login. Teachers and students have different access routes and dashboards.
- **Password Reset**: Secure password reset flow via email OTP verification.

- **Session Handling**: User sessions maintained with proper expiration and logout mechanisms.

2. Class Management (Teacher Dashboard)

- **Create/Edit/Delete Class**: Teachers can manage class details like subject, visibility, description, and students assigned.
- **Live Class Toggle**: Teachers can activate or deactivate the class live status.
- **Class Join Code**: Unique class codes are generated and stored for students to join securely.

3. Student-Class Association

- **Join Class via Code**: Students can join available public/private classes by submitting the code.
- **Class Dashboard View**: Students can view all enrolled classes with subject, teacher, and live status.

4. Test Paper Management

- **Create Test**: Teachers can create tests by specifying total marks, time, title, and description.
- **Add Questions**: Add multiple-choice questions with options and correct answers.
- **Edit/Delete Test or Questions**: Teachers can update questions, time duration, or test status.
- **View Submissions**: See student responses, scores, and submission timestamps.

5. Student Test Interaction

- **Take Test**: Students can take assigned tests within a given time window.
- **Submit Answers**: Responses are submitted, evaluated, and stored in the database.
- **View Results**: Score and feedback can be displayed upon completion or when published.

6. Attendance Tracking

- **Teacher Mark Attendance**: Teachers can mark daily attendance for each class session.
- **Student View Attendance**: Students can check their attendance history and percentage.
- **Date-Based Filtering**: Attendance entries are tracked by class and date.

7. Profile Management

- **View Profile**: Both students and teachers can view their profile info including name, ID, picture, etc.

- **Edit Profile**: Users can update editable fields like profile picture, phone, city, state, and more.

8. Notifications & Communication (Future Scope)

- **Live Alerts**: Implement notifications for live class start, new test, and announcements.
- **Chat Module**: Real-time messaging between teachers and students inside classes (WebSocket integration planned).

7. Database Design

LearnieVerse uses a relational database (MySQL) to manage all aspects of its digital learning ecosystem. The design ensures fast data retrieval, scalability for a growing user base, and robust security to protect sensitive academic and personal information. Below is an overview of the key tables and their roles within the system, along with the SQL schema adapted for LearnieVerse.

Overview and Design Goals

- **Scalability:**
  Designed to support an increasing number of institutions, teachers, students, and classes, making it easy to add new features as the platform evolves.
- **Flexibility:**
  Supports multiple roles (admin, institution, teacher, student) and dynamic interactions such as class sessions, announcements, and test management.
- **Security:**
  Implements role-based access, secure authentication, and encrypted sensitive data while adhering to best practices (like enforcing HTTPS for data transfers).

1. auth_users Table

This table stores authentication credentials and roles for all users.

- **Fields**:
  - id: Unique identifier for each user (INT, Primary Key).
  - email: User's email address (VARCHAR, Unique).
  - password: Hashed user password (VARCHAR).
  - role: Role assigned to the user (ENUM: teacher, student, institution, admin).

- created_at: Timestamp of account creation (TIMESTAMP, Default: CURRENT_TIMESTAMP).

---

2. institutions Table

Holds data for educational institutions registered on the platform.

- **Fields**:
  - id: Unique identifier (INT, Primary Key).
  - auth_user_id: Reference to auth_users.id (INT, Unique, Foreign Key).
  - institution_name: Name of the institution (VARCHAR).
  - reg_number: Registration number (VARCHAR).
  - city, state, address: Location details (VARCHAR/TEXT).
  - admin_name: Name of the institution admin (VARCHAR).
  - phone: Contact number (VARCHAR).
  - username: Institution login/identifier (VARCHAR, Unique).
  - logo: Path to institution logo (VARCHAR).
  - agree_terms: Whether the terms were accepted (BOOLEAN).
  - status: Approval status (ENUM: pending, approved, rejected).
  - status_shown: If the status is publicly shown (BOOLEAN).
  - created_at: Timestamp (TIMESTAMP).

---

3. teachers Table

Stores detailed information for teacher profiles.

- **Fields**:
  - id: Unique identifier (INT, Primary Key).
  - auth_user_id: Reference to auth_users.id (INT, Unique, Foreign Key).
  - teacher_name: Full name (VARCHAR).
  - phone: Contact number (VARCHAR).
  - dob: Date of birth (DATE).
  - gender: Gender identity (ENUM: male, female, other).
  - city, state, address: Location information (VARCHAR/TEXT).
  - subject_expertise1, subject_expertise2: Primary/Secondary subjects (VARCHAR).
  - years_experience: Total years of teaching (INT).
  - highest_qualification: Educational qualification (VARCHAR).

- specialization: Field of specialization (VARCHAR).
- teacher_id: Unique identifier for teacher (VARCHAR, Unique).

---

4. students Table

Manages student profiles and academic details.

- **Fields**:
  - id: Unique identifier (INT, Primary Key).
  - auth_user_id: Reference to auth_users.id (INT, Unique, Foreign Key).
  - student_name: Full name (VARCHAR).
  - phone: Contact number (VARCHAR).
  - dob: Date of birth (DATE).
  - gender: Gender identity (ENUM: male, female, other).
  - city, state, address: Location info (VARCHAR/TEXT).
  - grade_level: Academic grade or year (VARCHAR).
  - school_name: Name of the school (VARCHAR).
  - student_id: Unique student identifier (VARCHAR, Unique).
  - profile_picture: Path to student's image (VARCHAR).

---

5. classes Table

Stores details of virtual or physical classes created by teachers.

- **Fields**:
  - id: Unique identifier (INT, Primary Key).
  - class_name: Title or label of the class (VARCHAR).
  - teacher_id: Reference to teachers.id (INT, Foreign Key).
  - class_code: Unique code for joining the class (VARCHAR, Unique).
  - visibility: Whether the class is public/private (ENUM: public, private).
  - is_live: Whether the class is currently live (BOOLEAN).
  - subject: Subject name (VARCHAR).
  - description: Summary about the class (TEXT).
  - created_at: Timestamp (TIMESTAMP).

---

6. class_students Table

A many-to-many relationship table for tracking which students are enrolled in which classes.

- **Fields**:

- id: Unique identifier (INT, Primary Key).
- class_id: Reference to classes.id (INT, Foreign Key).
- student_id: Reference to students.id (INT, Foreign Key).
- joined_at: Timestamp when the student joined (TIMESTAMP).

---

7. testpapers Table

Holds metadata about tests created by teachers.

- **Fields**:
  - id: Unique identifier (INT, Primary Key).
  - class_id: Reference to classes.id (INT, Foreign Key).
  - teacher_id: Reference to teachers.id (INT, Foreign Key).
  - title: Title of the test (VARCHAR).
  - description: Short description (TEXT).
  - total_marks: Total marks for the test (INT).
  - duration: Duration in minutes (INT).
  - status: Availability status (ENUM: draft, published, closed).
  - created_at: Timestamp (TIMESTAMP).

---

8. test_questions Table

Stores individual questions for each test paper.

- **Fields**:
  - id: Unique identifier (INT, Primary Key).
  - testpaper_id: Reference to testpapers.id (INT, Foreign Key).
  - question_text: Text of the question (TEXT).
  - option_a, option_b, option_c, option_d: Multiple-choice options (VARCHAR).
  - correct_option: Correct answer (ENUM: A, B, C, D).
  - marks: Marks allocated to the question (INT).

---

9. test_responses Table

Stores student submissions and scores for each test.

- **Fields**:
  - id: Unique identifier (INT, Primary Key).
  - testpaper_id: Reference to testpapers.id (INT, Foreign Key).
  - student_id: Reference to students.id (INT, Foreign Key).

- score: Marks scored (INT).
- submitted_at: Timestamp (TIMESTAMP).

---

10. attendance Table

Tracks daily or session-based attendance per class.

- **Fields**:
  - id: Unique identifier (INT, Primary Key).
  - class_id: Reference to classes.id (INT, Foreign Key).
  - student_id: Reference to students.id (INT, Foreign Key).
  - date: Date of attendance (DATE).
  - status: Present/Absent (ENUM: present, absent, late).

8. Testing Strategies

To ensure reliability, performance, and a smooth user experience, LearnieVerse follows a structured and layered testing strategy. From individual components to full integration across modules, testing is a continuous process throughout the development lifecycle.

1. Unit Testing

Unit testing is performed on individual functions, models, and route handlers to validate that each part works as intended.

- **Tools Used**: unittest, pytest (Python)
- **Scope**:
  - Authentication logic (register, login, OTP, password reset)
  - Test scoring algorithms
  - Class and student assignment logic
  - Attendance status calculation
  - Route responses and validation

2. Integration Testing

Ensures that different components (like Flask backend and MySQL database) work well together.

- **Focus Areas**:

    - Class creation and student assignment flows
    - Test creation followed by question insertion
    - Joining classes and submitting tests
    - Session management across login/logout
- **Tools Used**: Custom test scripts using pytest and Postman for API testing

3. End-to-End (E2E) Testing

Simulates real user interactions from frontend to backend to validate the entire system.

- **Focus**:
    - Full registration-to-dashboard flow
    - Creating and taking tests
    - Real-time attendance marking
    - Profile updates and role-specific actions
- **Tools**: Selenium, Cypress (for React UI)

4. User Acceptance Testing (UAT)

Carried out with test users (students & teachers) to evaluate the app based on real-world use cases and expectations.

- **Goal**: Ensure the platform is intuitive, functional, and ready for deployment
- **Process**:
    - Feedback loop with users
    - Identify UI/UX improvements
    - Finalize functionality based on test feedback

5. Security & Validation Testing

To protect user data and prevent unauthorized access:

- **Password Hashing**: Verified for secure storage
- **SQL Injection Checks**: Prevented with parameterized queries
- **Session Expiry**: Ensures auto-logout on inactivity
- **Access Control**: Verified using role-based permission checks for teachers and students

6. Performance Testing

Tested under high-load scenarios to check how LearnieVerse performs during peak hours like test deadlines or live classes.

- **Focus Areas**:
    - Response time for major APIs
    - Load on database during simultaneous submissions

- **Tools**: Apache JMeter, Locust (optional for scale testing)

9. Challenges and Solutions

During the development of *LearnieVerse*, our team faced several technical and user-experience challenges. Here's a breakdown of key roadblocks and how we tackled them like champs:

1. Secure Role-Based Access Control

Challenge: Preventing unauthorized access between teacher and student dashboards.

Solution:

Implemented role-based access control (RBAC) using session tokens and decorators in Flask. Routes were protected with checks like @login_required and user role validations before rendering views.

2. Dynamic Class Management

Challenge: Allowing teachers to create, edit, and manage multiple classes with unique student assignments.

Solution:

Created a relational database structure using MySQL where each class links to students via foreign keys. On the backend, teacher ID is associated with created classes, and class-student assignments are stored in a separate join table for efficient querying and updates.

3. Seamless Test Creation & Editing

Challenge: Making it easy for teachers to add, edit, and delete test questions dynamically.

Solution:

Introduced a two-phase form submission process:

- Phase 1: Teachers input test metadata (title, duration).

- Phase 2: Add multiple questions via dynamic form fields and update them later with edit routes.

AJAX and Flask routing handled real-time saving/editing without breaking flow.

4. Real-Time Attendance System

Challenge: Managing accurate attendance marking during live sessions.

Solution:

Attendance was recorded via timestamps when students joined the class. The system checks for presence during the scheduled class time using server time and stores status (Present, Late, Absent) accordingly in the attendance table.

5. Maintaining Fast Performance

Challenge: Ensuring low response times even during test submissions or high-traffic events.

Solution:

- Optimized SQL queries using indexes and joins.
- Used Flask blueprints to modularize logic and reduce overhead.
- Implemented caching for static content and minimized DB hits where possible.

6. OTP and Email Reliability

Challenge: Delivering OTPs quickly and reliably during login or password reset.

Solution:

Configured Flask-Mail with a reliable SMTP service (like Gmail or Mailgun) and asynchronous threading to avoid slowing down the main app flow during email dispatch.

7. Responsive UI for All Devices

Challenge: Making the UI look good and work properly across laptops, tablets, and phones.

Solution:

Used responsive design principles with ReactJS and Tailwind CSS. Components were tested across multiple screen sizes. Also added mobile-friendly navigation and collapsible menus for better usability.

8. Database Schema Changes During Development

Challenge: Updating MySQL schema without losing data during iterative changes.

Solution:

Maintained versioned migration scripts and used tools like MySQL Workbench and Flask-Migrate to track and update schemas without data loss.

10.Future Enhancements

As *LearnieVerse* continues to grow, there are several exciting improvements on the roadmap to boost performance, interactivity, and user experience. Here's a glimpse into the upgrades we're planning next:

1. Real-Time Video Class Integration

What's Coming: Seamless WebRTC-based video call system for live classes — no third-party app needed.

Why: To provide an all-in-one learning platform without switching to external tools like Zoom or Google Meet.

2. Automated Notes Generation

What's Coming: AI-generated lecture notes and summaries after each class session.

Why: To help students focus on learning instead of note-taking and make revision easier.

3. AI-Powered Personalized Recommendations

What's Coming: Suggest study materials, tests, and classes based on individual progress and learning style.

Why: Adaptive learning = smarter learning. Tailored content keeps users more engaged.

4. Advanced Analytics Dashboard

What's Coming: Teachers and students will get dashboards with visual performance insights — attendance trends, test scores, and engagement levels.

Why: Data-driven feedback helps both sides grow better.

5. 2FA and Enhanced Security

What's Coming: Two-Factor Authentication (2FA) and biometric login (for mobile app) to improve account safety.

Why: User trust and platform security are a top priority.

6. Integrated Chat and Discussion Forums

What's Coming: Real-time chat for each class + community forums for broader discussions.

Why: Improves peer-to-peer interaction and helps build a collaborative learning environment.

7. Mobile App (Android & iOS)

What's Coming: Native mobile apps with offline access, push notifications, and fast load times.

Why: Students and teachers on the go deserve a smooth mobile experience.

8. Gamified Learning

What's Coming: Badges, leaderboards, and XP systems to encourage active participation.
Why: Learning + fun = better retention and motivation.

9. Multi-language Support
What's Coming: Support for multiple regional languages for both UI and content.
Why: To make LearnieVerse accessible to students across different language backgrounds.

10. Certificate Generation for Class Completion
What's Coming: Auto-generated certificates for students upon completing a class or test series.
Why: Recognition boosts morale and provides proof of learning.

11.Code Snippets

This section includes some of the core code implementations from LearnieVerse to demonstrate backend functionality and integration between routes, templates, and databases.

11.1 User Authentication (Flask + MySQL)

```
@auth_bp.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        email = request.form.get('email')
        password = request.form.get('password')
        cursor = current_app.db.connection.cursor(DictCursor)

        # Query the central auth_users table
        cursor.execute("SELECT * FROM auth_users WHERE email = %s", (email,))
        user = cursor.fetchone()

        if user and check_password_hash(user['password'], password):
            session['logged_in'] = True
            session['user_type'] = user['role']
            session['user_id'] = user['id']

            # If institution, check its approval status from the institutions table
            if user['role'] == 'institution':
                cursor.execute("SELECT status FROM institutions WHERE auth_user_id = %s",
(user['id'],))
                inst_status = cursor.fetchone()
                if inst_status:
                    status = inst_status['status']
                    # Redirect to a status page that displays the status message
                    return redirect(url_for('auth.institution_status', status=status))
                else:
                    flash("Institution profile not found", "danger")
                    return redirect(url_for('auth.get_started'))
            # For other roles, redirect accordingly
            elif user['role'] == 'admin':
                return redirect(url_for('admin.admin_home'))
            elif user['role'] == 'teacher':
                return redirect(url_for('teacher.dashboard'))
            elif user['role'] == 'student':
                return redirect(url_for('student.dashboard'))
            else:
                flash("User role not recognized", "danger")
                return redirect(url_for('auth.get_started'))
        else:
            flash("Invalid email or password", "danger")
            return redirect(url_for('auth.get_started'))

    return redirect(url_for('auth.get_started'))
```

11.2 Class Creation by Teachers

```
@teacher_bp.route('/add_new_class', methods=['GET', 'POST'])
def add_new_class():
    if not session.get('logged_in') or session.get('user_type') != 'teacher':
        flash("Please log in as a teacher to access this page.", "danger")
        return redirect(url_for('auth.login'))

    cursor = current_app.db.connection.cursor(DictCursor)
    cursor.execute("SELECT id, institution_id FROM teachers WHERE auth_user_id = %s",
(session['user_id'],))
    teacher = cursor.fetchone()
    if not teacher:
        flash("Teacher not found.", "danger")
        return redirect(url_for('auth.login'))

    teacher_id = teacher['id']
```

```
    institute_id = teacher.get('institution_id')
    cursor.execute("SELECT * FROM students WHERE teacher_id = %s", (teacher_id,))
    students = cursor.fetchall()

    if request.method == 'POST':
        class_name = request.form.get('class-name')
        class_time = request.form.get('class-time')
        class_date = request.form.get('class-date')
        visibility = request.form.get('visibility')
        join_link = request.form.get('join-link') or ''
        selected_students = request.form.getlist('students')

        try:
            cursor.execute("""
                INSERT INTO classes (teacher_id, institute_id, class_name, class_date, class_time,
visibility, join_link)
                VALUES (%s, %s, %s, %s, %s, %s, %s)
            """, (teacher_id, institute_id, class_name, class_date, class_time, visibility,
join_link))
            class_id = cursor.lastrowid

            if 'everyone' not in selected_students:
                for student_id in selected_students:
                    cursor.execute("""
                        INSERT INTO class_students (class_id, student_id)
                        VALUES (%s, %s)
                    """, (class_id, student_id))

            current_app.db.connection.commit()
            flash("Class created successfully!", "success")
            return redirect(url_for('teacher.manage_classes'))

        except Exception as e:
            current_app.db.connection.rollback()
            flash("Error creating class: " + str(e), "danger")

    return render_template('teacher/add_new_class.html', students=students)
```

11.3 Live Class Join Link Generation

```
@teacher_bp.route('/invite_student', methods=['POST'])
def invite_student():
    data = request.get_json()
    student_email = data.get('student_email')
    if not student_email:
        return jsonify({'error': 'Student email is required'}), 400

    token = str(uuid.uuid4())
    auth_user_id = session.get('user_id')
    if not auth_user_id:
        return jsonify({'error': 'Teacher not logged in'}), 403

    cursor = current_app.db.connection.cursor()
    cursor.execute("SELECT id, institution_id FROM teachers WHERE auth_user_id = %s", (auth_user_id,))
    teacher_record = cursor.fetchone()
    if not teacher_record:
        return jsonify({'error': 'Teacher record not found'}), 400
    teacher_id, institute_id = teacher_record

    try:
        sql = """
            INSERT INTO student_invites (token, student_email, teacher_id, institute_id)
            VALUES (%s, %s, %s, %s)
        """
        cursor.execute(sql, (token, student_email, teacher_id, institute_id))
        current_app.db.connection.commit()

        signup_link = current_app.url_for('auth.student_signup', token=token, _external=True)

        subject = "You're invited to join LearnieVerse as a Student"
        body = f"Hello,\n\nYou've been invited to join LearnieVerse as a student. Please sign up using
the following link:\n\n{signup_link}\n\nThank you!"
        send_email(student_email, subject, body)
```

```
        return jsonify({'message': 'Invitation sent successfully', 'signup_link': signup_link})
    except Exception as e:
        current_app.db.connection.rollback()
        return jsonify({'error': str(e)}), 500
```

11.4 HTML Template: Join Class Page

```
<!-- templates/student/join_class.html -->
<h2>Join Class</h2>
<form method="POST">
    <label for="token">Enter Join Code:</label>
    <input type="text" name="token" required>
    <button type="submit">Join</button>
</form>
```

11.5 Attendance Marking

```
@teacher_bp.route('/take_class/<int:class_id>', methods=['GET', 'POST'])
def take_class(class_id):
    if not session.get('logged_in') or session.get('user_type') != 'teacher':
        flash("Please log in as a teacher.", "danger")
        return redirect(url_for('auth.login'))

    cursor = current_app.db.connection.cursor(DictCursor)

    cursor.execute("UPDATE classes SET is_live = 1 WHERE id = %s", (class_id,))
    current_app.db.connection.commit()

    cursor.execute("SELECT id FROM teachers WHERE auth_user_id = %s", (session['user_id'],))
    teacher = cursor.fetchone()
    if not teacher:
        flash("Teacher not found.", "danger")
        return redirect(url_for('auth.login'))

    teacher_id = teacher['id']

    cursor.execute("SELECT * FROM classes WHERE id = %s AND teacher_id = %s", (class_id, teacher_id))
    class_info = cursor.fetchone()
    if not class_info:
        flash("Class not found or unauthorized access.", "danger")
        return redirect(url_for('teacher.manage_classes'))

    cursor.execute("""
        SELECT s.* FROM students s
        JOIN class_students cs ON s.id = cs.student_id
        WHERE cs.class_id = %s
    """, (class_id,))
    students = cursor.fetchall()

    if not students:
        cursor.execute("SELECT * FROM students WHERE teacher_id = %s", (teacher_id,))
        students = cursor.fetchall()

    today = datetime.date.today()

    for student in students:
        cursor.execute("""
            SELECT join_time, leave_time
            FROM class_sessions
            WHERE class_id = %s AND student_id = %s
        """, (class_id, student['id']))
        sessions = cursor.fetchall()

        student['join_count'] = len(sessions)
        student['leave_count'] = sum(1 for s in sessions if s['leave_time'])
        student['last_join'] = max((s['join_time'] for s in sessions), default=None)
        student['last_leave'] = max((s['leave_time'] for s in sessions if s['leave_time']),
default=None)
```

```
            latest_session = max(sessions, key=lambda s: s['join_time']) if sessions else None
            if latest_session:
                if latest_session['leave_time'] is None:
                    student['status'] = "In Class"
                else:
                    student['status'] = "Left Class"
            else:
                student['status'] = "Not Joined"

            cursor.execute("""
                SELECT status FROM class_attendance
                WHERE class_id = %s AND student_id = %s AND attendance_date = %s
            """, (class_id, student['id'], today))
            attendance = cursor.fetchone()
            student['attendance_status'] = attendance['status'].capitalize() if attendance else 'Pending'

        if request.method == "POST" and request.form.get('attendance_submit'):
            for student in students:
                status = request.form.get(f"attendance_{student['id']}")
                if status:
                    cursor.execute("""
                        SELECT id FROM class_attendance
                        WHERE class_id = %s AND student_id = %s AND attendance_date = %s
                    """, (class_id, student['id'], today))
                    existing = cursor.fetchone()
                    if existing:
                        cursor.execute("""
                            UPDATE class_attendance SET status = %s, marked_at = NOW()
                            WHERE id = %s
                        """, (status, existing['id']))
                    else:
                        cursor.execute("""
                            INSERT INTO class_attendance (class_id, student_id, attendance_date, status)
                            VALUES (%s, %s, %s, %s)
                        """, (class_id, student['id'], today, status))
            current_app.db.connection.commit()
            flash("Attendance updated successfully.", "success")
            return redirect(url_for('teacher.take_class', class_id=class_id))

        cursor.execute("""
            SELECT c.*, a.email AS sender_email
            FROM class_chats c
            JOIN auth_users a ON c.sender_id = a.id
            WHERE c.class_id = %s
            ORDER BY c.created_at ASC
        """, (class_id,))
        chats = cursor.fetchall()

        return render_template("teacher/take_class.html",
                               class_info=class_info,
                               students=students,
                               chats=chats)
```

11.6 Test Creation & Question Addition

```
@teacher_bp.route('/make_take_test', methods=['GET', 'POST'])
def make_take_test():
    if not session.get('logged_in') or session.get('user_type') != 'teacher':
        flash("Please log in as a teacher to create tests.", "danger")
        return redirect(url_for('auth.login'))

    cursor = current_app.db.connection.cursor(DictCursor)
    cursor.execute("SELECT id, institution_id FROM teachers WHERE auth_user_id = %s",
(session['user_id'],))
    teacher = cursor.fetchone()
    if not teacher:
        flash("Teacher not found.", "danger")
        return redirect(url_for('auth.login'))

    teacher_id = teacher['id']
    institute_id = teacher['institution_id']
    cursor.execute("SELECT * FROM students WHERE teacher_id = %s", (teacher_id,))
```

```
    student_list = cursor.fetchall()

    if request.method == 'POST':
        test_title = request.form.get('test-title')
        test_subject = request.form.get('test-subject')
        test_duration = request.form.get('test-duration')
        test_date = request.form.get('test-date')
        test_time = request.form.get('test-time')
        selected_students = request.form.getlist('students')

        try:
            cursor.execute("""
                INSERT INTO tests (teacher_id, institute_id, title, subject, duration, test_date,
test_time, status)
                VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
            """, (teacher_id, institute_id, test_title, test_subject, test_duration, test_date,
test_time, 'draft'))
            test_id = cursor.lastrowid

            if "everyone" in selected_students:
                for student in student_list:
                    cursor.execute("""
                        INSERT INTO test_assigned_students (test_id, student_id)
                        VALUES (%s, %s)
                    """, (test_id, student['id']))
            else:
                for student_id in selected_students:
                    cursor.execute("""
                        INSERT INTO test_assigned_students (test_id, student_id)
                        VALUES (%s, %s)
                    """, (test_id, student_id))

            current_app.db.connection.commit()
            flash("Test created successfully! Now add questions.", "success")
            return redirect(url_for('teacher.test', test_id=test_id))
        except Exception as e:
            current_app.db.connection.rollback()
            flash("Failed to create test: " + str(e), "danger")

    cursor.execute("SELECT * FROM tests WHERE teacher_id = %s ORDER BY test_date DESC", (teacher_id,))
    all_tests = cursor.fetchall()
    for test in all_tests:
        if isinstance(test.get('test_time'), datetime.timedelta):
            test['test_time'] = str(test['test_time'])[:-3]

    current_tests = [t for t in all_tests if t['status'] == 'draft']
    live_tests = [t for t in all_tests if t['status'] == 'live']
    past_tests = [t for t in all_tests if t['status'] == 'past']

    return render_template(
        'teacher/make_take_test.html',
        current_tests=current_tests,
        live_tests=live_tests,
        past_tests=past_tests,
        student_list=student_list)
```

11.7 Student Dashboard: Test Attempting

```
@student_bp.route('/tests/<int:test_id>/take', methods=['GET', 'POST'])
def take_test(test_id):
    if not session.get('logged_in') or session.get('user_type') != 'student':
        flash("Please log in as a student to take tests.", "danger")
        return redirect(url_for('auth.login'))

    cursor = current_app.db.connection.cursor(DictCursor)

    # Get the student ID from auth_user_id
    auth_user_id = session['user_id']
    cursor.execute("SELECT id FROM students WHERE auth_user_id = %s", (auth_user_id,))
    student_record = cursor.fetchone()

    if not student_record:
```

```
            flash("Student profile not found.", "danger")
            return redirect(url_for('student.assignments'))

        student_id = student_record['id']

        # Fetch test details and ensure the test is live
        cursor.execute("SELECT * FROM tests WHERE id = %s AND status = 'live'", (test_id,))
        test = cursor.fetchone()
        if not test:
            flash("Test is not available at the moment.", "warning")
            return redirect(url_for('student.assignments'))

        # Fetch the test questions
        cursor.execute("""
            SELECT * FROM test_questions
            WHERE test_id = %s
            ORDER BY question_number ASC
        """, (test_id,))
        questions = cursor.fetchall()

        # Fetch answer options for each question
        for question in questions:
            cursor.execute("SELECT * FROM test_options WHERE question_id = %s", (question['id'],))
            question['options'] = cursor.fetchall()

        if request.method == 'POST':
            total_questions = len(questions)
            correct_count = 0

            # Iterate over each question to process the answer submission
            for question in questions:
                answer = request.form.get(f"question_{question['id']}")
                if answer:
                    cursor.execute("SELECT is_correct FROM test_options WHERE id = %s", (answer,))
                    chosen = cursor.fetchone()
                    if chosen and chosen['is_correct']:
                        correct_count += 1

                    # Insert into test_answers table to store the student's answer
                    cursor.execute("""
                        INSERT INTO test_answers (test_id, student_id, question_id, selected_option_id)
                        VALUES (%s, %s, %s, %s)
                    """, (test_id, student_id, question['id'], answer))

            # Calculate score as a percentage
            score = (correct_count / total_questions) * 100 if total_questions > 0 else 0

            # Store the result in the test_results table
            cursor.execute("""
                INSERT INTO test_results (test_id, student_id, score, submitted_at)
                VALUES (%s, %s, %s, NOW())
            """, (test_id, student_id, score))
            current_app.db.connection.commit()

            flash(f"Test submitted successfully! Wait for the results", "success")
            return redirect(url_for('student.assignments'))

        return render_template('student/take_test.html', test=test, questions=questions)
```

11.8 Signup's

```
@auth_bp.route('/institution_signup', methods=['GET', 'POST'])
def institution_signup():
    if request.method == 'POST':
        # Retrieve form data
        institution_name = request.form.get('institution_name')
        reg_number = request.form.get('reg_number')
        city = request.form.get('city')
        state = request.form.get('state')
        address = request.form.get('address')
        admin_name = request.form.get('admin_name')
        email = request.form.get('email')
```

```
        phone = request.form.get('phone')
        username = request.form.get('username')
        password = request.form.get('password')
        confirm_password = request.form.get('confirm_password')
        agree_terms = request.form.get('agree_terms')  # 'on' if checked

        # Validate password confirmation
        if password != confirm_password:
            flash("Passwords do not match", "danger")
            return redirect(url_for('auth.institution_signup'))

        # Hash the password securely using pbkdf2:sha256
        hashed_password = generate_password_hash(password, method='pbkdf2:sha256')

        # Process file upload for institute logo
        logo_file = request.files.get('logo')
        logo_filename = None
        if logo_file and logo_file.filename:
            logo_filename = secure_filename(logo_file.filename)
            upload_folder = os.path.join(current_app.root_path, 'static', 'uploads')
            if not os.path.exists(upload_folder):
                os.makedirs(upload_folder)
            logo_file.save(os.path.join(upload_folder, logo_filename))

        # Convert agree_terms checkbox value to boolean
        terms_agreed = True if agree_terms == 'on' else False

        try:
            cursor = current_app.db.connection.cursor()
            # Insert authentication record
            sql_auth = """
                INSERT INTO auth_users (email, password, role)
                VALUES (%s, %s, %s)
            """
            cursor.execute(sql_auth, (email, hashed_password, 'institution'))
            auth_user_id = cursor.lastrowid

            # Insert institution profile with status defaulting to 'pending'
            sql_inst = """
                INSERT INTO institutions
                (auth_user_id, institution_name, reg_number, city, state, address, admin_name, phone,
username, logo, agree_terms)
                VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
            """
            cursor.execute(sql_inst, (
                auth_user_id, institution_name, reg_number, city, state, address,
                admin_name, phone, username, logo_filename, terms_agreed
            ))
            current_app.db.connection.commit()
            flash("Institution registered successfully! Your request is pending admin approval.",
"success")
            return redirect(url_for('auth.login'))
        except Exception as e:
            current_app.db.connection.rollback()
            flash("Registration failed: " + str(e), "danger")
    return render_template('singup/institution_signup.html')
```

```
@auth_bp.route('/teacher_signup', methods=['GET', 'POST'])
def teacher_signup():
    token = request.args.get('token')
    institution_id = None

    if token:
        cursor = current_app.db.connection.cursor(DictCursor)
        cursor.execute("SELECT institute_id FROM teacher_invites WHERE token = %s", (token,))
        invite = cursor.fetchone()
        cursor.close()

        if invite:
            institution_id = invite['institute_id']
            session['institution_id'] = institution_id
        else:
```

```
            flash("Invalid or expired invitation token.", "danger")
            return redirect(url_for('auth.get_started'))
    else:
        institution_id = session.get('institution_id')

    if request.method == 'POST':
        teacher_name = request.form.get('teacher_name')
        email = request.form.get('email')
        phone = request.form.get('phone')
        dob = request.form.get('dob')
        gender = request.form.get('gender')
        city = request.form.get('city')
        state = request.form.get('state')
        address = request.form.get('address')
        subject_expertise1 = request.form.get('subject_expertise1')
        subject_expertise2 = request.form.get('subject_expertise2')
        years_experience = request.form.get('years_experience')
        highest_qualification = request.form.get('highest_qualification')
        specialization = request.form.get('specialization')
        teacher_id = request.form.get('teacher_id')
        username = request.form.get('username')
        password = request.form.get('password')
        confirm_password = request.form.get('confirm_password')

        if password != confirm_password:
            flash("Passwords do not match", "danger")
            return redirect(url_for('auth.teacher_signup', token=token))

        if not institution_id:
            flash("Institution not found. Please try again.", "danger")
            return redirect(url_for('auth.teacher_signup', token=token))

        hashed_password = generate_password_hash(password, method='pbkdf2:sha256')

        profile_pic_file = request.files.get('profile_pic')
        profile_pic_filename = None
        if profile_pic_file and profile_pic_file.filename:
            profile_pic_filename = secure_filename(profile_pic_file.filename)
            upload_folder = os.path.join(current_app.root_path, 'static', 'uploads')
            if not os.path.exists(upload_folder):
                os.makedirs(upload_folder)
            profile_pic_file.save(os.path.join(upload_folder, profile_pic_filename))

        resume_file = request.files.get('resume')
        resume_filename = None
        if resume_file and resume_file.filename:
            resume_filename = secure_filename(resume_file.filename)
            upload_folder = os.path.join(current_app.root_path, 'static', 'uploads')
            if not os.path.exists(upload_folder):
                os.makedirs(upload_folder)
            resume_file.save(os.path.join(upload_folder, resume_filename))

        try:
            cursor = current_app.db.connection.cursor()
            sql_auth = "INSERT INTO auth_users (email, password, role) VALUES (%s, %s, %s)"
            cursor.execute(sql_auth, (email, hashed_password, 'teacher'))
            auth_user_id = cursor.lastrowid

            sql_teacher = """
                INSERT INTO teachers (
                    auth_user_id, teacher_name, phone, dob, gender, city, state, address,
                    subject_expertise1, subject_expertise2, years_experience, highest_qualification,
                    specialization, teacher_id, username, institution_id, profile_pic, resume,
invite_token
                ) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
            """
            cursor.execute(sql_teacher, (
                auth_user_id, teacher_name, phone, dob, gender, city, state, address,
                subject_expertise1, subject_expertise2, years_experience, highest_qualification,
                specialization, teacher_id, username, institution_id, profile_pic_filename,
resume_filename,
                token  # Store the invite token here.
            ))
```

```
            current_app.db.connection.commit()
            # After successful registration in teacher_signup...
            cursor.execute("UPDATE teacher_invites SET teacher_email = %s WHERE token = %s", (email,
token))
            current_app.db.connection.commit()
            cursor.close()

            flash("Teacher signup successful! Please log in.", "success")
            return redirect(url_for('auth.login'))
        except Exception as e:
            current_app.db.connection.rollback()
            cursor.close()
            flash("Signup failed: " + str(e), "danger")
            return redirect(url_for('auth.teacher_signup', token=token))

    return render_template('singup/teachers_signup.html', token=token)
```

```
@auth_bp.route('/student_signup', methods=['GET', 'POST'])
def student_signup():
    token = request.args.get('token')

    # Retrieve invite details
    try:
        cursor = current_app.db.connection.cursor(DictCursor)
        cursor.execute("SELECT teacher_id, institute_id FROM student_invites WHERE token = %s",
(token,))
        invite_details = cursor.fetchone()
        if not invite_details:
            flash("Invalid or expired invite token.", "danger")
            return redirect(url_for('auth.get_started'))
        teacher_id = invite_details['teacher_id']
        institute_id = invite_details['institute_id']
    except Exception as e:
        flash("Error retrieving invite details: " + str(e), "danger")
        return redirect(url_for('auth.get_started'))

    if request.method == 'POST':
        student_name = request.form.get('student_name')
        email = request.form.get('email')
        phone = request.form.get('phone')
        dob = request.form.get('dob')
        gender = request.form.get('gender')
        city = request.form.get('city')
        state = request.form.get('state')
        address = request.form.get('address')
        enroll_number = request.form.get('enroll_number')
        course = request.form.get('course')
        guardian_name = request.form.get('guardian_name')
        guardian_phone = request.form.get('guardian_phone')
        username = request.form.get('username')
        password = request.form.get('password')
        confirm_password = request.form.get('confirm_password')

        # Check if passwords match
        if password != confirm_password:
            flash("Passwords do not match!", "danger")
            return redirect(url_for('auth.student_signup', token=token))

        hashed_password = generate_password_hash(password, method='pbkdf2:sha256')

        profile_pic_file = request.files.get('profile_pic')
        profile_pic_filename = None
        if profile_pic_file and profile_pic_file.filename:
            profile_pic_filename = secure_filename(profile_pic_file.filename)
            upload_folder = os.path.join(current_app.root_path, 'static', 'uploads')
            if not os.path.exists(upload_folder):
                os.makedirs(upload_folder)
            profile_pic_file.save(os.path.join(upload_folder, profile_pic_filename))

        try:
            cursor = current_app.db.connection.cursor()
```

```
            # Insert into auth_users for authentication
            sql_auth = "INSERT INTO auth_users (email, password, role) VALUES (%s, %s, %s)"
            cursor.execute(sql_auth, (email, hashed_password, 'student'))
            auth_user_id = cursor.lastrowid  # Get the inserted user ID

            # Updated student insert query including teacher_id and institute_id
            sql_student = """
                INSERT INTO students (
                    auth_user_id, student_name, phone, dob, gender, city, state, address,
                    enroll_number, course, profile_pic, guardian_name, guardian_phone, username,
                    invite_token, teacher_id, institute_id
                )
                VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
            """
            cursor.execute(sql_student, (
                auth_user_id, student_name, phone, dob, gender, city, state, address,
                enroll_number, course, profile_pic_filename, guardian_name, guardian_phone, username,
                token, teacher_id, institute_id
            ))

            current_app.db.connection.commit()

            # Clear the invite record (if needed)
            cursor.execute("UPDATE student_invites SET student_email = %s WHERE token = %s", (email,
token))
            current_app.db.connection.commit()

            flash("Student signup successful! Please log in.", "success")
            return redirect(url_for('auth.login'))

        except Exception as e:
            current_app.db.connection.rollback()
            flash("Signup failed: " + str(e), "danger")
            return redirect(url_for('auth.student_signup', token=token))

    return render_template('singup/student_signup.html', token=token)
```

Conclusion

LearnieVerse is more than just a learning platform — it's a next-gen digital classroom built to bridge the gap between traditional teaching and modern tech. By integrating live classes, test systems, attendance management, notes sharing, and AI-powered tools, it empowers both students and teachers with everything they need in one place.

The project showcases how web technologies like Flask, MySQL, and WebRTC can be harmonized to create a smooth, scalable, and secure platform. From smart class management to intuitive UI, LearnieVerse sets the foundation for a future-focused education system.

As we continue to innovate with real-time features, AI integration, and mobile-first experiences, LearnieVerse aims to evolve into a complete ecosystem for personalized and collaborative learning.

The journey doesn't stop here — this is just the beginning.

Bibliography

- **Flask Documentation**

  https://flask.palletsprojects.com/

  Referenced for backend development, routing, and session management.
- **MySQL Official Documentation**

  https://dev.mysql.com/doc/

  Used for structuring and managing the relational database in LearnieVerse.
- **Bootstrap Documentation**

  https://getbootstrap.com/

  Referenced during UI/UX prototyping and frontend layout adjustments.
- **SQLAlchemy ORM**

  https://docs.sqlalchemy.org/

  Referenced for ORM implementation in some backend database operations.
- **Stack Overflow & GitHub**

  https://stackoverflow.com/

  https://github.com/

  Community-driven solutions and code samples were referred to during development and debugging.
- **W3Schools**

  https://www.w3schools.com/

  Used for quick reference to HTML, CSS, JS, and SQL syntax and examples.
- **Python Docs**

  https://docs.python.org/

  Referenced for core language features, libraries, and best practices.