**A Comprehensive Study on Learning Management System (LMS) in Education: Present and Future**

**Bijetree Basak1, Tanushree Jati2, Asoke Nath3**

Postgraduate and Research Department of Computer Science

St. Xavier’s College (Autonomous), Kolkata, India

**Abstract**

**The demand for remote and inherently flexible education systems has become a growing trend since the pandemic. The solution to this problem is the indispensable learning management system. With the ever-growing demand for robustness and security for online study platforms, the demand for newer innovations in learning management systems are anticipated. The next-generation learning management system is the new future that we are looking towards in the field of education. The authors performed a comprehensive comparison between the existing approaches pertaining to the current learning management systems, with the evolving requirements to cater to the users personally, thereby forming a new system which bridges the distance between it and the existing systems. The authors also focus on the essential practical aspect of effectively implementing a next-gen learning management system in Education in the present scenario. The authors have proposed a model system constructed utilizing Python’s Django framework for backend development, to ensure speed and high performance, while using native HTML framework for frontend development which ensures light-weightedness and cross-platform independence. Unlike traditional platforms, the proposed application aims to seamlessly transition to the ideal remote classes and dedicates itself entirely for educational purposes. Future work on this topic includes incorporating an online library resource system, automated checking of certain submitted assignments and grading system, attendance tracker based on physical presence and gamification methods amongst other immersive learning techniques.**

**Keywords: Conference call, Django, Group and Meeting management, Learning management system**.

1. **INTRODUCTION**

This research paper presents a basic model for the Learning Management System (LMS) which would help the audience to implement and be capable of enhancing the basic model with advanced features. The objective of the model is to ensure both role-based access and a secure, efficient virtual learning platform. Using a multi-tiered architecture, the LMS would allow user access via the client interface and data server, managing information regarding the user data and meeting details, whereas requests are streamed through the web server [1].

The LMS has role-based access control (RBAC) [2], where it can grant permissions based on a user's role-being admin, instructor, and student. Admins can create, update, and remove users, groups, and meetings; instructors can create and schedule meetings within their groups; and students have a very limited view to see and join meetings. This is achieved with a custom user model built upon Django's AbstractBaseUser for secure authentication and permission management.

The LMS has group and meeting management. For scheduling meetings, an approved user may be capable of creating a new meeting, as well as can update that meeting settings and also has the permission to cancel it. For any meeting, its status has to be updated automatically, that is, whether the meeting is upcoming, ongoing or cancelled. This way, each user will get real-time updates of their scheduled meetings in the calendar view.

With this, the LMS is developed using a Django MVC framework aside from Python, HTML, CSS, and JavaScript, which, of course, utilizes Django's built-in authentication framework but it was customized to comply with our requirements. Therefore, it has the capability to manage both user authentication, that is, logging in and logging out, and role-based access control to ensure only authorized users can access specific views to perform certain actions. This model, therefore, supports robust and flexible collaboration, bringing about ease of streamlined experience to everyone within the virtual learning environment.

1. **METHODOLOGY**

The authors will now outline the system design and model of the proposed Learning Management System. It provides a detailed description of the architecture, system design, role-based access structure, group management and scheduling as well as conducting meetings. In this model, both use case diagrams and class diagrams are also discussed.

1. *Web Application Architecture*

The proposed LMS model utilizes a multi-tier structure for effective communication among the client, middleware, and database [3]. It consists of three primary elements:

* Client Application, which enables the users to interact with the web application interface via a web browser.
* A protected data server that will manage the essential data and handle user information, resources, scheduled meetings and other elements.
* The web server will control the overall functionality, including user queries, and is also responsible for dynamic responses.

The architecture allows for seamless communication and transfer of information, which leads to the creation of a web platform that is expandable, reliable, and secure [4][5]. The functional requirements are illustrated using a use case diagram to represent both the static and dynamic components of the system.

1. *User authentication*

The user Authentication system is a critical component of the basic LMS model, as it is responsible for ensuring secure access to resources and infrastructure. The authentication process begins with the user attempting to log in through the platform's custom login view. This view extends Djangos built in login view and allows for customized login pages which fits the UI requirements of the platform. When the user submits login credentials, that is, email ID and password, the system will verify these details with the data stored in the backend database. If the credentials are correct and matches with the backend database, Django authentication will create a session for that user, marking them as authenticated. A session ID will be created and used for all the subsequent requests. If the login attempt is unsuccessful, then the user will be redirected to the login page and display a message that informs the user the credentials entered were incorrect. Once after successfully completing the authentication process, the user will be redirected to their personalized dashboard which will be determined by their user role. The customized login page has been shown in Fig. 1.

Similarly, for the logout page we extend Django's logout view with our customized view to meet the requirements. When the user interacts with the logout button, the session is immediately invalidated and effectively allowing the user to exit the platform. It will also remove the session cookie from the browser, preventing further access until the user logs in again. After logout, the user will be redirected to the login page allowing a smooth transition.

1. *Role-Based Access Control (RBAC)*

The learning management systems (LMS) model employs a role-based access control (RBAC) system that enforces permissions according to user roles which would be manually assigned by the administrators. Some functionalities may be performed only by users with specific roles: only an admin user is allowed to create and manage users and class groups, and both admin and instructor users are allowed to schedule meetings. Students can only view the meetings that have been scheduled in their calendar and join those meetings on the scheduled time. Hence the admin user will have the highest level of access, the instructor user has a slightly restricted access and the student user has the most limited access to the system [6].

A custom user model was used based on Django's AbstractBaseUser and PermissionsMixin, which enables proper authentication and permission capabilities. The CustomUserManager class inherits BaseUserManager in providing specific methods of how the user can be created. The method first ensures that an email gets normalized before throwing a non-existent error in case the email does not exist; it then proceeds with saving the user's details after storing their password into the database. The superuser method is meant for admin users, to set the additional flags is\_staff and is\_superuser for permissions purposes, and then call user creation method to save the admin users permissions.

The custom user class itself further inherits the functionalities of a Django user model. One can define the user\_type field with possible choices, namely 'admin', 'instructor', and 'student'. Other relevant fields are email, first name, last name, staff, active status, and date of joining. It makes the USERNAME\_FIELD set to 'email' for login with email rather than a username, and it sets REQUIRED\_FIELDS to an empty list because no other fields are needed during user creation. By utilizing the user manager, this model allows users and admins to be easily created and supports flexible role assignments to allow for role-based access control within the LMS [7].

In order to achieve this control, the built-in authentication and permission decorators of Django are used along with some additional logic as that has been illustrated in the pseudo code provided below:

FUNCTION is\_admin(user)

IF user is authenticated AND user type is 'admin':

RETURN true

ELSE:

RETURN false

FUNCTION is\_admin\_or\_instructor\_required(function)

function wrapper(request, args, kwargs)

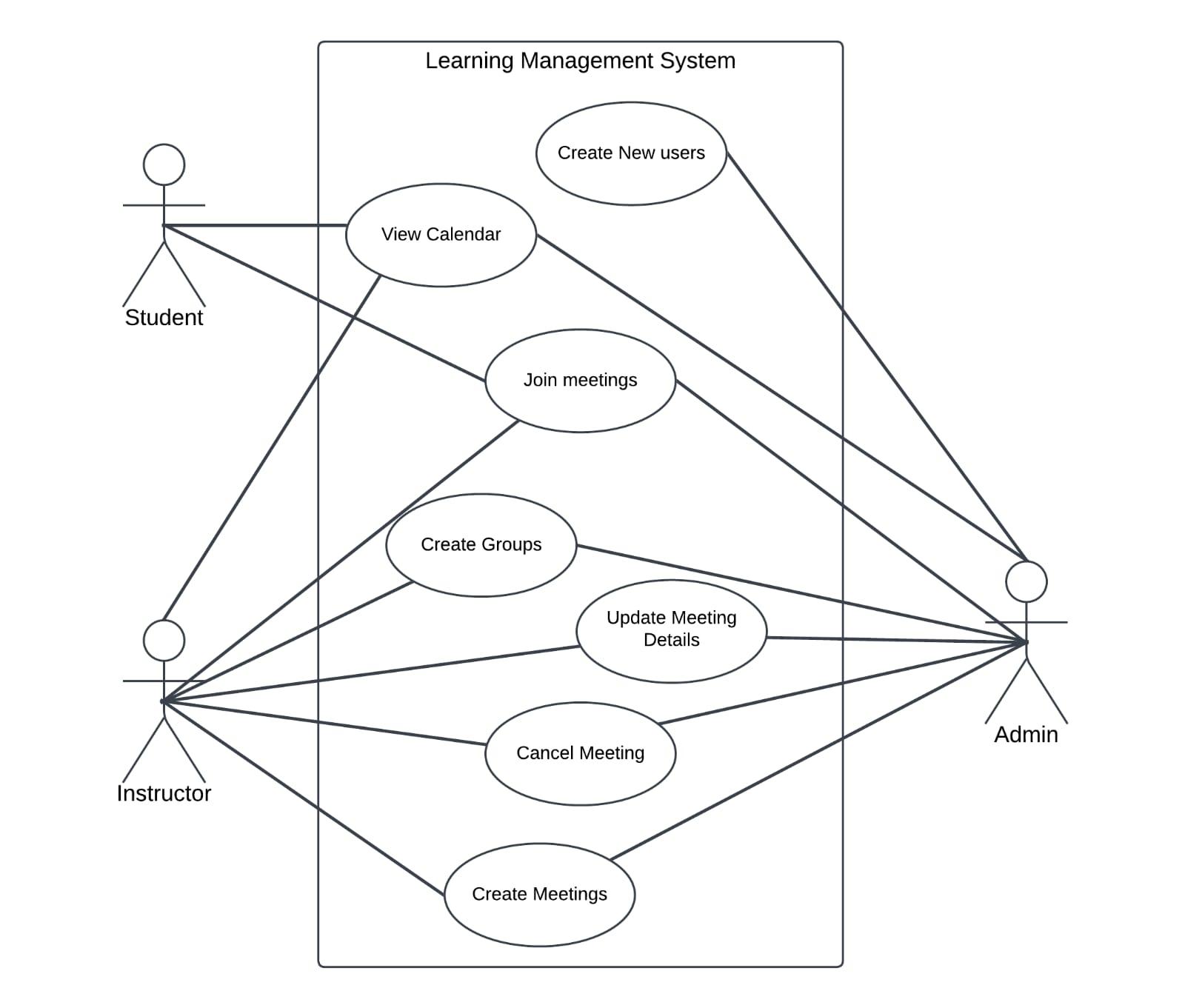
IF user is authenticated AND (user type is 'admin' OR user type is 'instructor'):

RETURN function(request, args, kwargs)

ELSE:

redirect to 'some\_error\_page'

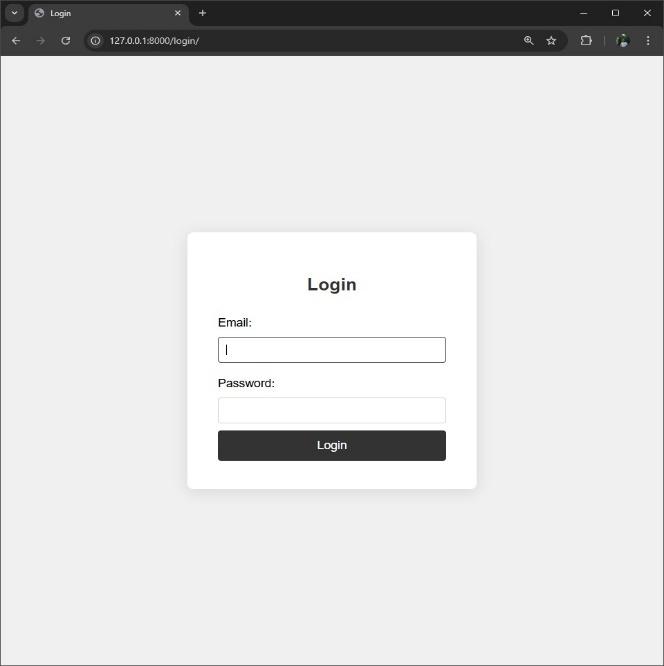
We have used a use-case diagram to show the relations between the different categories of users that we see in RBAC as in Fig. 1.



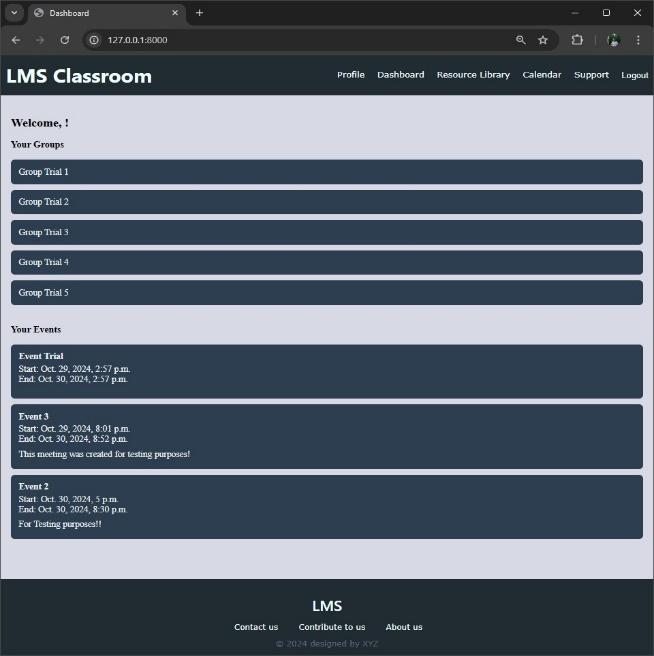
**Fig. 1.** Use-case diagram for RBAC.

1. *Application Development*

The application was developed using Python, HTML, CSS, and JavaScript,   utilizing   the   Django   framework's Model-View-Controller (MVC) architecture for the development process. The LMS is currently undergoing testing and development on a localhost server where it is being hosted. The login page where users must input their email and password to access the dashboard of the system has been shown in Fig. 2 and Fig. 3.



**Fig. 2.** :Login Page of the LMS Classroom.



**Fig. 3.** After successful login, display of the dashboard.

After the users successfully log in, they will be directed to the dashboard. The dashboard is an interactive page that allows the user to see the scheduled meetings, the list of class groups the user belongs to.

1. *Group Management*

Group management is one of the core features in the LMS. The system is designed so that users can manage and interact with different class groups according to their roles. The functionalities of Group Management ensure that every kind of role has appropriate permission such as admin and instructor can build and manage groups whereas students can only view the groups he belongs to and participate in scheduled meetings.

There are different views in Django used for group management in the system. One of the views provides instructors and admin with an interface to create new class groups. This will allow them to input necessary information regarding the class group such as the group name, description and add the relevant members for the group and the user that creates the group is added automatically to the list of users in the group. To do so, the process has been illustrated in the following algorithm provided below:

FUNCTION create\_group(form, user)

new group with information from the form

set the logged-in user as a group creator

add user currently logged in as a member

add other relevant members to the group

save the group to the database

Instructors and admins have the permission to assign students to the necessary groups manually. As for the student users, they can only see groups that they are members of, admins and instructors view all of them. To do so the following pseudocode was implemented which has been provided below:

FUNCTION get\_queryset(user)

IF the user is a 'student':

RETURN groups where user is a member

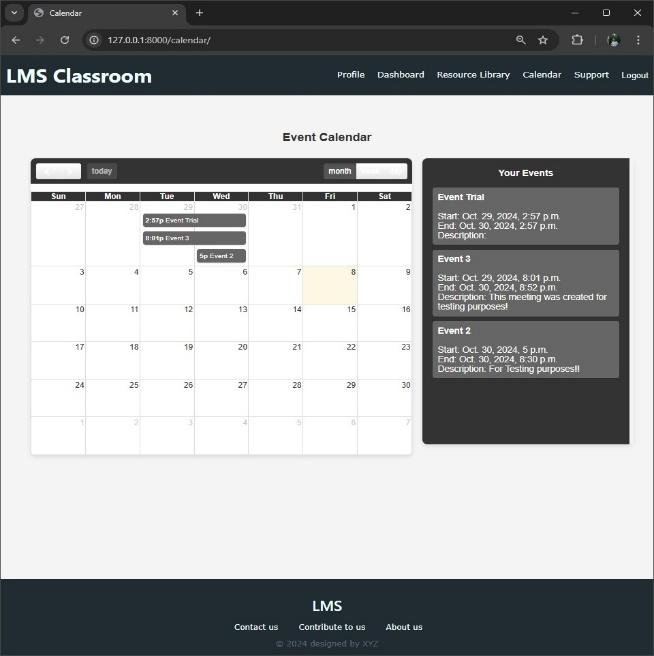
ELSE

RETURN all the groups in the model

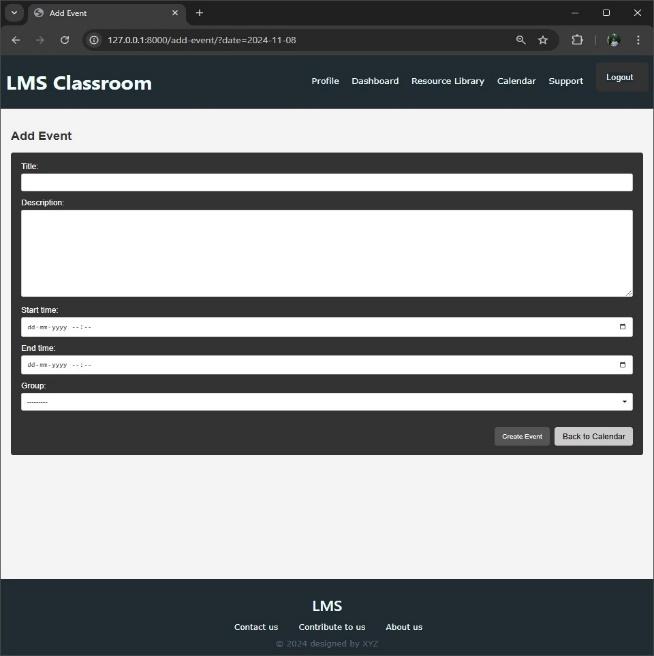
Updating the group details will allow the users to change the group name, description or the list of members and deletion of the group removes the group and its associated meetings and data from the platform but these actions are available to admin and instructors only so that student users do not accidentally delete important resources from the platform's database.

1. *Scheduling Meetings*

A model is designed to schedule and manage meetings within a group-based environment. The fields such as title and description for basic meeting details, along with start\_time and end\_time for specifying the timespan of the meeting. The created\_by field will link each event to the user who created it. This model will also be linked with the group model so that when the user creates a meeting for a group all the members of that group will also be notified as well. The status field helps track the meeting state by labeling it as “upcoming”, “ongoing”, or “canceled”. It will allow the system to provide real time updates for users on upcoming or current events.



**Fig. 4.** Calendar view.



**Fig. 5.** To schedule a meeting through a calendar view.

The process to schedule meetings, retrieval, updating, and deletion is permitted to only authorized users like instructors and admins. Now the meeting creation is accessible to only admins and instructors if it is any other type of user then it will render a Forbidden page. To schedule a meeting a calendar was implemented in the html page, shown in Fig. 4, such that whenever one of the dates of the calendar has been selected the site will redirect it to the Add event page, shown in Fig. 5, where the user can schedule the meeting according to their preference. Otherwise, a form is populated with the submitted data, validated, and saved, linking the event to the creator. In case of GET requests, an empty form is displayed with a class group field limited to groups the user belongs to. Then renders a template with form context, this way it can ensure a secure and structured way to schedule meetings. This process can be implemented using the following pseudo code.

FUNCTION add\_meeting(request):

IF request method is POST:

IF request user type is NOT ('admin' OR 'teacher'):

RETURN HTTP response forbidden

create a new instance of meeting form using request POST data

IF form is valid:

set meeting's created\_by attribute to the request user

save the meeting to the database

redirect to the calendar page

ELSE:

initialize a blank meeting form instance

retrieve groups where the user is a member

set the form’s 'group' field queryset to the user’s groups

render meeting creation template with form context

The next method retrieves a list of events related to the groups that the user is a member of, then formats those events with details such as meeting ID, title, start time, and end time in Indian standard formats and thus ready to be spliced together with frontend tools such as FullCalendar. It does not permit editing and meeting details to users who do not have administration rights or are not the creators of the event. The system redirects the user to the calendar page at the end of successful edit. Analogous restrictions put a ban on deleting events: this type of operation is restricted to users with the right of deleting events and once the meeting is deleted it needs to be ensured that the meeting details are also removed from the database and simultaneously in the frontend the status of the meeting should be change to ‘cancelled’. In case of successful deletion, the user is also redirected to the calendar page. This ensures secure and controlled access to data in preventing unauthorized users from making changes while allowing users to manage events based on their roles and permissions. This design maintains the integrity and security of the scheduling management system.

1. *Room Segregation*

The LMS model has an essential feature that allows users to create rooms through which they will initiate a call or they can join an already existing call. When creating a room, the user is required to provide a valid username (words should not be separated by a blank), which is then registered in the backend in the “Room” database. It automatically generates an ID and a password for the room. The ID is generated using a UUID (Universally unique identifier) string for each instance of a room that cannot be edited even in the database itself. This maintains security, uniqueness, and generalization. The password, on the other hand, is a 10-character generated using the “random” function used in Python. To ensure more safety, we should use a bigger password size and use all Unicode characters to maximize randomness. This ID and password are then given to the respective users to allow them to join the call. While a user is trying to join a call, they will be asked to enter the ID and password for the room they want to join. On successful authentication, they will be taken to the room which they can join. The frontend for this is constructed using HTML, CSS, and JavaScript. The algorithm for this section is given below.

FUNCTION createRoom()

get user input for room name

IF room name is empty:

show error message

send POST response to server with room name

obtain roomID and password from server

save data of room name, roomID and password to database

display roomID and password to user

display button to enter the newly created room

IF error present:

show message

FUNCTION joinRoom()

Get user input of roomID and password from user

IF roomID OR password is empty:

show error message

send POST request to server with roomID and password

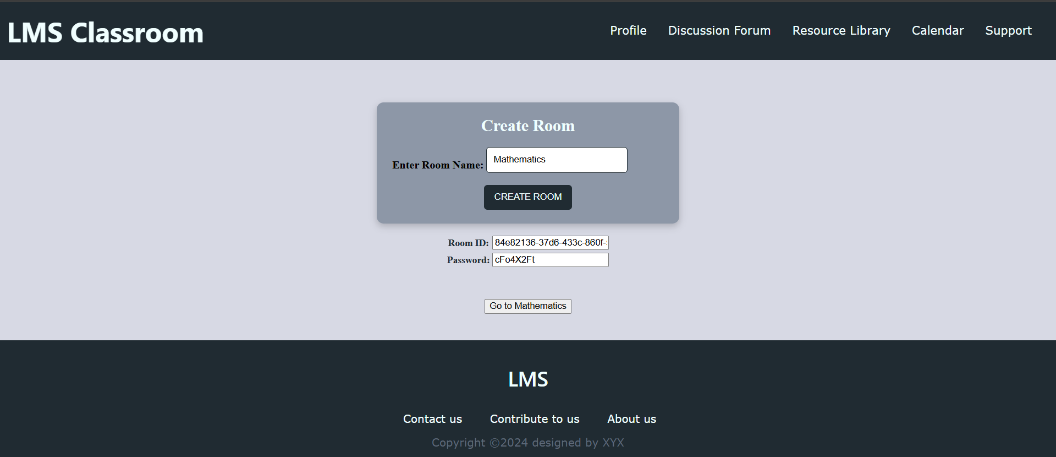
match roomID and password with database entries

IF authenticated:

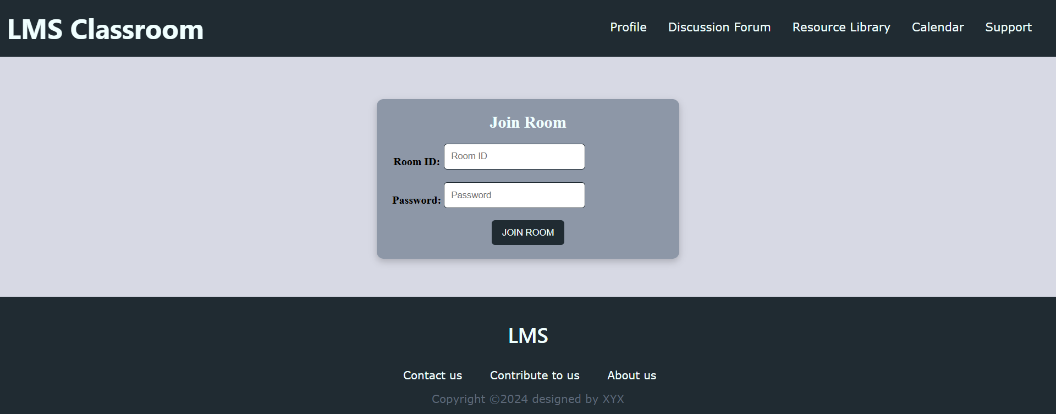
redirect user to room page

ELSE:

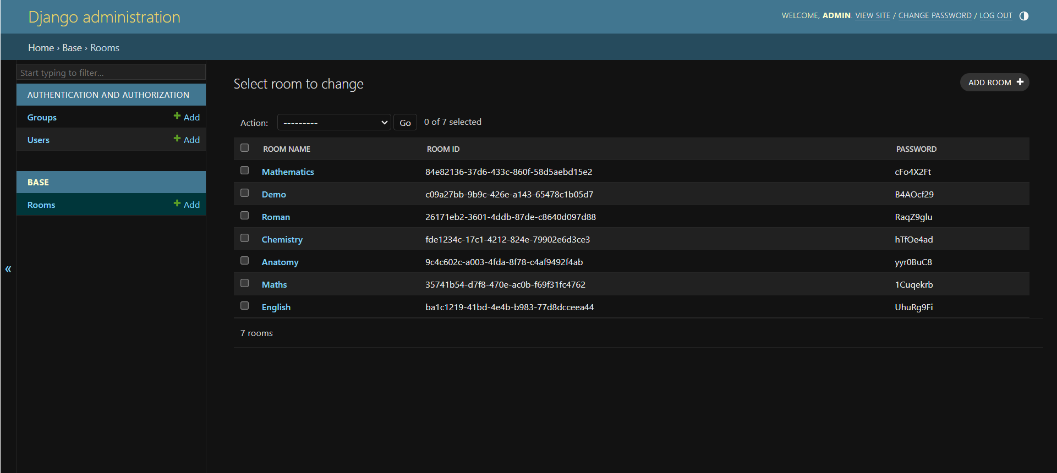
show error message



**Fig. 6.** Creation of a room.



**Fig. 7.** Joining a pre-existing room.



**Fig. 8.** The backend database stores the room name, room ID and password.

1. *Video Conference Model*

The video conference model proposed under the learning management system model is designed as a user-friendly, feature rich, web-based interactive communication tool that supports high-quality video and audio input from users. It is built using the Django framework, where the model primarily used WebRTC (Web Real-Time Communication) as the key protocol to establish peer-to-peer media interaction. This protocol is aimed to foster secure and reliable real-time connectivity between users, without the need to incorporate additional plugins to support media content. WebRTC has built-in APIs that have been used to access media devices from the user, that can be enabled or disabled as per the user’s requirement. This model features the essential functionalities such as pausing video output, muting the microphone, accessing video feeds [8].

The core feature of our model is the establishment of the platform to host a conference call, along with the capability to initiate and terminate a call. We have used WebRTC’s RTCPeerConnection API to construct the architecture of the real-time call component, enabling safe and secure connections. These connections are set up by exchanging user session descriptions and ICE (Interactive Connectivity Establishment) candidates through the signaling server. A point to note here is that we will be required to set up Redis (Remote Directory Server) beforehand which will be acting as a fast-signaling server and storage for session status. Along with Redis, we apply the concepts of WebSockets via Python, to handle the connection setup, which in turn allow the clients to locate each other and interact smoothly. Once initiation of a call has been successfully established, an option for termination of that same call appears, which allows the user to leave the call. The end call button triggers the connection to terminate and end the session. The corresponding pseudocode is given below.

Set up configuration with ICE server

FUNCTION connect()

initialize WebSocket connection to server with room name

IF WebSocket is open:

send reconnect message

IF WebSocket message is received:

IF message type is “offer”:

call handleOffer()

ELSE-IF message type is “answer”:

call handleAnswer()

ELSE-IF message type is “ice-candidate” AND candidate is existing:

add that candidate to peer connection

IF WebSocket is closed:

reconnect after 1 second

IF there exists WebSocket error:

show error message

call connect() to start the connection

FUNCTION handleOffer(offer, sender)

create new peer connection

store the new connection along with sender ID to locate the sender

create video element for that user along with ID

append that video element to video container

add local media track to peer connection

IF peer connection is ice candidate event:

If candidate exists AND WebSocket is open:

send message to ice-candidate

IF peer connection is track event:

assign media to video element

set up remote description with offer

create and set up local answer

send the answer to server

FUNCTION handleAnswer(answer, sender)

IF peer connection is existing for sender:

set up remote description with answer

FUNCTION startCall()

IF hang-up state is active:

call initiation not available

TRY

acquire user media stream

assign media stream to local video player

send message to server about new user

show required additional features

Catch error:

show error

FUNCTION hangUp()

IF hang up state is active:

perform nothing

set up hang up in active state

IF WebSocket is open:

send hang-up message to server

close connection

hide additional features

show start call button and hide hang up button

stop and clear local media tracks

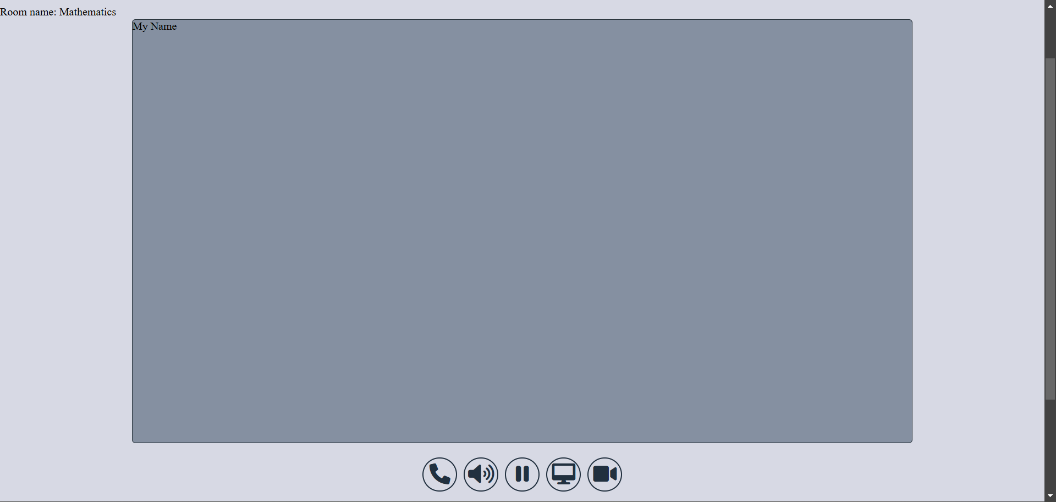
clear local video player media

FOR EACH peer connection existing:

close connection

remove connection from list of all connections

reset the hang up state



**Fig. 9.** Conference model setup before the call has started.



**Fig. 10.** Conference model setup after call has started from admin-end.

We utilize WebRTC’s MediaStream API to control the user's audio and media stream on the user-end. The functionalities included under these are muting the audio, pausing the video output and sharing the screen as and when deemed by the user. When the user clicks on the mute button, it immediately disables the user's audio track, whose status can again be changed by toggling the button. In a similar way, video stream is also disabled on clicking the appropriate button, which can also be enabled again by toggling the same button. Both these functions can be successfully performed without hindering the action of the other or dropping the call, thereby ensuring that flexibility along with privacy. Screen sharing is applied using WebRTC’s getDisplayMedia API, which allows the user to share a certain part of their screen on their entire window. It toggles in a similar way to the mute and pause video feed button, allowing the user to stop the screen sharing function at any time, without hindering the audio and video stream. All these features are seamlessly integrated into the video conference model. Needless to say, these features are utmost necessary to be incorporated in a system that would be used in an educational environment that prioritizes privacy, flexibility and seamless real-time interaction. These features are implemented according to the provided algorithm.

FUNCTION mute()

IF mute button is clicked:

toggle state of mute button

FOR EACH audio track in localStream:

enable or disable track based on mute state

update button title

IF mute icon exists:

remove unmute icon

add mute icon

ELSE:

remove mute icon

add unmute icon

FUNCTION videoToggle()

IF pause video button is clicked:

toggle video status

FOR EACH video track in localStream:

enable or disable track based on video status

update button title

FUNCTION startScreenSharing()

TRY:

get screenStream from display media

IF localStream exists:

add screen share track to localStream

ELSE:

assign screenStream to localStream

set video player source to localStream

set sharing status to true

send message to peers

CATCH error:

show error

FUNCTION stopScreenShare()

IF sharing status is true:

set sharing status to false

restart local video stream

update video player source to localStream

send message to peers

FUNCTION screenShare()

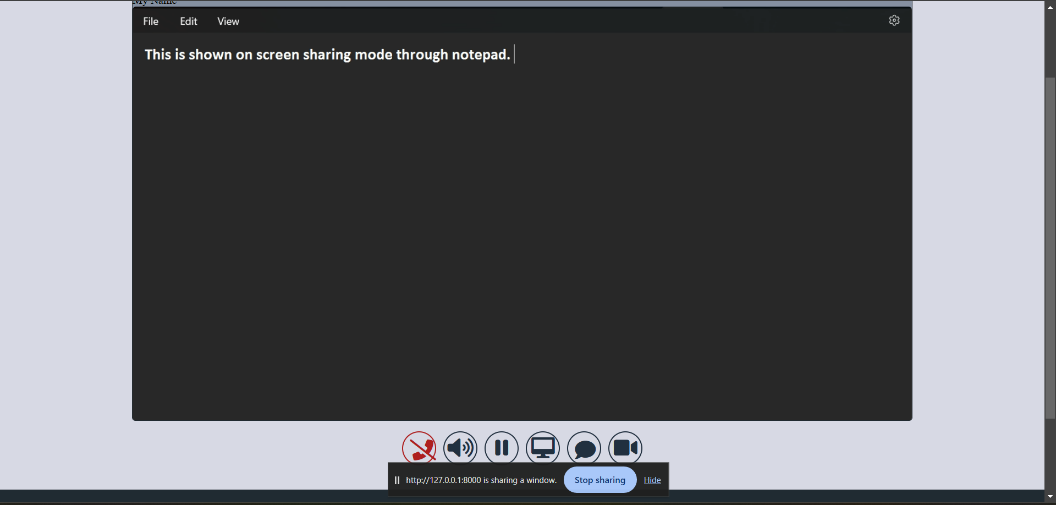
IF screen share button is clicked:

IF share status is true:

call stopScreenShare()

ELSE:

call startScreenShare()



**Fig. 11.** Screen sharing feature.

Two other additional features present in the proposed model are screen recording functionality and a live in-class chatroom. The screen recording functionality utilizes MediaRecorderAPI of WebRTC to capture both audio and video stream during the live session. Once the recording is stopped, in a similar toggle button, the recording is downloaded by the user, which can be changed so that it is stored over a cloud server. The quality is not compromised and is not expected to have issues with concurrency of audio and media streams. It is supposed to serve as a resource material to students for revisiting certain topics as they wish. The chatroom enables real-time in-call text communication, which is imperative to the fact that some users might observe poor network strength which disallows them to stream their audio or video, but the presence of a chatroom will allow them to communicate without taking a lot of network bandwidth. It is built using WebSocket technology, which allows bi-directional text communication between users, with each text accompanied by their registered names. The conversations remain private to the call and will be removed for a user after the call ends or the user disconnects themselves from the call. These additional features are implemented according to the following algorithm.

FUNCTION startScreenRecording()

get screenStream from display media

IF localStream exists:

FOR EACH audio track in localStream:

add audio track to screenStream

create mediaRecorder for screenStream

IF mediaRecorder data is available:

IF data size is NOT null:

add data to recorded array

IF mediaRecorder is stopped:

create Blob from recorded array as ‘video/webm’

create downloadable link from Blob

clear recorded

start mediaRecorder

hide start recording button

show stop recording button

FUNCTION stopScreenRecording()

stop mediaRecorder

show start recording button

hide stop recording button

FUNCTION recordingEvent()

IF start recording button is clicked:

call startScreenRecording()

ELSE-IF stop recording button is clicked:

call stopScreenRecording()

FUNCTION displayChatMessage(sender, text)

create message element

set innerHTML of message element to text from sender

add message element to chat container

FUNCTION sendChatButton()

IF send chat button is clicked:

get text message from chat input

IF text message is not empty:

send text message through socket

call displayChatMessage(sender ID, text message)

clear chat input

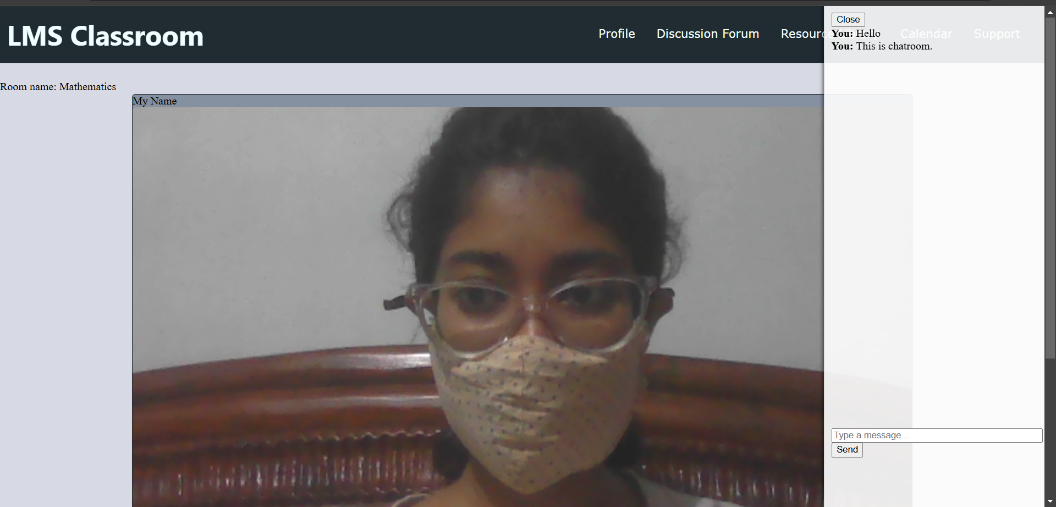
FUNCTION chatButtonClick()

IF chat button is clicked:

toggle chat sidebar container

FUNCTION chatClose()

set chat sidebar container display to ‘none’



**Fig. 12.** Chat Room in live session.

All these features are made interactive using user-friendly interfaces that have been constructed using HTML, CSS and JavaScript. They are meticulously styled to be visually intuitive, accompanied with titles to allow users to read the actual functions associated with each element. Event listeners are attached to each element that sends control signals to the session and the different WebRTC APIs to construct a smooth and secure experience.

1. **RESULTS AND DISCUSSION**

The research results highlight the successful design and implementation of the basic learning management system using Django’s framework using effective and latest web technologies and techniques to meet the requirements of the educational needs. Role-Based Access Control (RBAC) system, group management capabilities, and meeting scheduling functionalities comply with strong emphasis on secure, authenticated access to data integrity. The RBAC mechanism ensures strict access management, safeguarding sensitive data and maintaining security and privacy. Additionally, group management and meeting scheduling facilitate collaborative efforts, while real-time notifications foster active participation. Built on Django's Model-View-Controller architecture, the LMS scales efficiently, adapts to changes, and remains maintainable. The video conference architecture of the proposed LMS model prioritizes ease of use along with privacy, flexibility and seamless integration of real-time call features. All of these foster participation and intuitive approach to an education system in a virtual environment.

Such findings raise massive implications to the departments of education in improving the teaching, learning, and administrative processes toward eventual realization of impacts on more improved student engagement, better learning outcomes, and protection of data. This research is still under development and our current findings indicate that future development might be AI, machine learning, and mobile applications for further personalization and improvement of learning experiences.

1. **CONCLUSION AND FUTURE SCOPE**

The present paper describes the design, development, and functional aspects of the core section of the LMS system based on role-based access control for effective management of education by various user roles. In this model of LMS, the system promises to be secure, efficient, and provides role-specific features for admins, instructors, and students, using the multi-tiered web application architecture and MVC framework of Django. It encompasses group and meeting management, real-time updating on meetings, and controls of access to functions based on user roles. The LMS model proposed here addresses the limitations found in the existing system and suggests a model for improving the user experience for each role in a secure, scalable, and friendly environment.

This model can be used as a base for further LMS development, potentially incorporating more features and analytics as are in demand. This opens the future scope of the paper to be expanded to accommodate additional features on top of the core model to handle our concerns towards the present educational systems. The successful deployment of this system in real-world scenarios can significantly influence online learning by structuring a virtual learning platform.

**REFERENCES**

1. Graf, S., & List, B. (2005). "An evaluation of open-source e-learning platforms stressing adaptation issues." Presented in 2005 Fifth IEEE International Conference on Advanced Learning Technologies (ICALT'05) (pp. 163-165).
2. Wu, B., & Chen, X. (2008). "A security model for learning management system." Presented in Proceedings of the 2008 International Conference on Computer Science and Software Engineering (Vol. 4, pp. 832-835).
3. Sani, A. R., Adham, A., & Wicaksono, A. (2018). "Architecture Design of Web-based Learning Management System Using MVC Pattern." International Journal of Computer Applications, 180(45), 20-25.
4. Zhang, W., & Yang, Y. (2015). A cloud-based learning management system with the multi-tier architecture. International Journal of Computer Science and Network Security, 15(5), 68-73.
5. Nunes, M. R., & Rodrigues, P. (2014). A flexible, multi-tier architecture for scalable web applications. Journal of Computer and System Sciences, 80(5),1043-1055.
6. StudyGyaan: How to create RBAC in Django with multiple user types. Retrieved from [https://studygyaan.com/django/how-to-create-rbac-in-django-with-multiple-users-types](https://studygyaan.com/django/how-to-create-rbac-in-django-with-multiple-users-types) (2023).
7. MDN Web Docs: Django Tutorial Part 8: User authentication and permissions. Retrieved from [https://developer.mozilla.org](https://developer.mozilla.org) (2022).
8. Deshmukh, R., Nand, N., Pawar, A., Wagh, D., Kudale, A.: Video conferencing using WebRTC. In: 2023 International Conference on Sustainable Computing and Data Communication Systems (ICSCDS), pp. 857–864 (2023).