UNIVERSITY
OF LONDON

CM3035 - Advanced Web Development

Final Report

Contents

1. Introduction

This report will cover the describe the eLearning platform application and the reasoning for its design and functionality. This platform, eLearn, provides various interactive and useful features aimed at improving the educational experience for both students and teachers.

This eLearning platform was developed to make online education more engaging by enhancing the interaction between students and teachers. Instead of just being another platform that only provides course materials for self-study, eLearn allows students and teachers to communicate through a real-time chat feature. This makes the learning process more interactive and helps students receive real-time feedback on their progress.

With the implementation of user roles and access controls, different users can perform different tasks based on their roles. Teachers can manage their courses, upload course materials, manage students all in one place. On the other hand, students can enroll in multiple courses, access their learning resources, and provide feedback on their course experiences.

This project was mainly built using Django to handle the backend operations. The Django REST Framework (DRF) was used to create REST API endpoints, making it easier to expand the application to other platforms in the future. WebSockets and Django Channels were implemented for the real-time chat feature, allowing instant communication between users. Additionally, Celery and Redis were used to run background tasks asynchronously, improving the overall performance and efficiency.

For the frontend, Tailwind CSS and Flowbite were used to design a nice and clean user interface. Tailwind CSS makes it easier to create a modern design without having to write too much custom CSS. Flowbite was also used to enhance the UI with some pre-built components, improving the overall user experience and making the development smoother.

2. Project Structure and System Design

2.1 Project Structure

This eLearning platform is structured using Django's app-based structure, where different features are separated into individual Django apps instead of keeping everything in one place. This makes the project more organized and easier to manage.

Instead of writing everything in a single app, this project's features were divided into smaller apps, with each apps handle their corresponding function. The authentication app manages user login, registration, and authentication logic, while the users app handles user profile, roles, and their respective permissions. The courses app contains logics for course creation, course materials, enrollment, notifications, and the chat app implements the real-time

messaging feature between students and teachers using WebSocket. Additionally, the api app contains API endpoints for users to access their data.

This structure is much better than cramming everything into a single app because it makes the code a lot more organized, making it easier to find and modify specific features without messing with other parts of the project. It also simplifies the debugging process since each app contains only the code related to its function. For instance, if there is a bug in the chat feature, it is easy to locate the error source since all chat-related code is inside the chat app.

Initially, parts of the code were kept in two Django app, courses and users, but as more features were added, it became significantly more difficult to track down specific errors. Having everything in only two apps made the codebase unnecessarily long and complicated, making it harder to locate specific functions. Dividing the project into multiple apps provided a cleaner and more structured way to handle development and maintain.

2.2 Models and Relationships

This eLearning platform is designed using a relational database model, where the entities such as users, courses, feedback, and notifications are connected through relationships. Each model defines how data is stored and retrieved from the database.

User Model and Role-Based Access Control

The platform includes a AppUser model to extend Django's built-in User model. This model allows users to store extra information, such as real name, profile photo, bio, and their user role as a student or a teacher.

To ensure that each user is unique, the OneToOne relationship links AppUser to default User model. This makes authentication easier while still allowing custom role-based permission. Additionally, profile photos filenames will be uniquely generated using UUIDs to prevent filename conflicts that overwrite profile photos with the same filename. The model also includes a get_photo_url function, which returns the link to a default profile photo if a user hasn't uploaded one.

Another one is the UserStatusUpdate model, which allows user to post their status updates. These updates are linked to the built-in User through a Foreign Key, making it easy to track user-specific posts and display them correctly.

Course Model and Enrollment

The platform also includes a Course model, designed to store courses created by teachers. Each course has a title, description, and a created_by field that link to the AppUser model, ensuring only teachers can create courses. On the other hand, students can enroll in multiple courses through a ManyToMany field relationship to the AppUser model. The system automatically generates timestamps for each course upon creation, making it easier to verify the latest courses.

The CourseMaterial model allows teachers to upload files related to their courses. Each uploaded file is saved with a unique name using a similar UUID function that generates unique profile photo filenames previously. This prevents duplicates and ensure that materials with the same filename would not overwrite each other. The model also records who uploaded each file and the timestamp, providing useful information for students

Course Feedback and Notification Model

The CourseFeedback model connects students and courses through a Foreign Key relationship, allowing students to leave feedback only for the course they are enrolled in. Each feedback is automatically timestamped upon submission, making it easier for teachers to improve from students' feedback.

The Notification model is linked to a specific user and can be marked as read or unread. This keeps users informed so that they don't miss the latest updates for their courses. The platform also includes a context processor that makes notifications accessible globally, so they can be displayed anywhere disregarding the page the users are on.

2.3 Database Design and Normalisation

The eLearning platform is designed using SQLite, a relational database. The database structure follows the Third Normal Form (3NF) to minimize duplicate data and speed up queries across the database.

To achieve First Normal Form (1NF), all the tables must have a primary key, and every field should contain atomic values. For example, in the UserStatusUpdate model, each status updates is stored as a separate row instead of cramming them in a single field. The same also applies to CourseFeedback, where each feedback is stored as an individual entry.

To achieve the Second Normal Form (2NF), all non-key attributes should depend on the primary key. Since each model uses a single-column key 'id', so there is no composite primary key, in which, 2NF is already achieved. Each model uses a single-column primary key and other non-key attributes always rely on that single primary key. For instance, the field feedback_content in the CourseFeedback model depends only on the 'id' of the entry, and references to course and student are stored as Foreign Keys, preventing unnecessary data duplication in the feedback table.

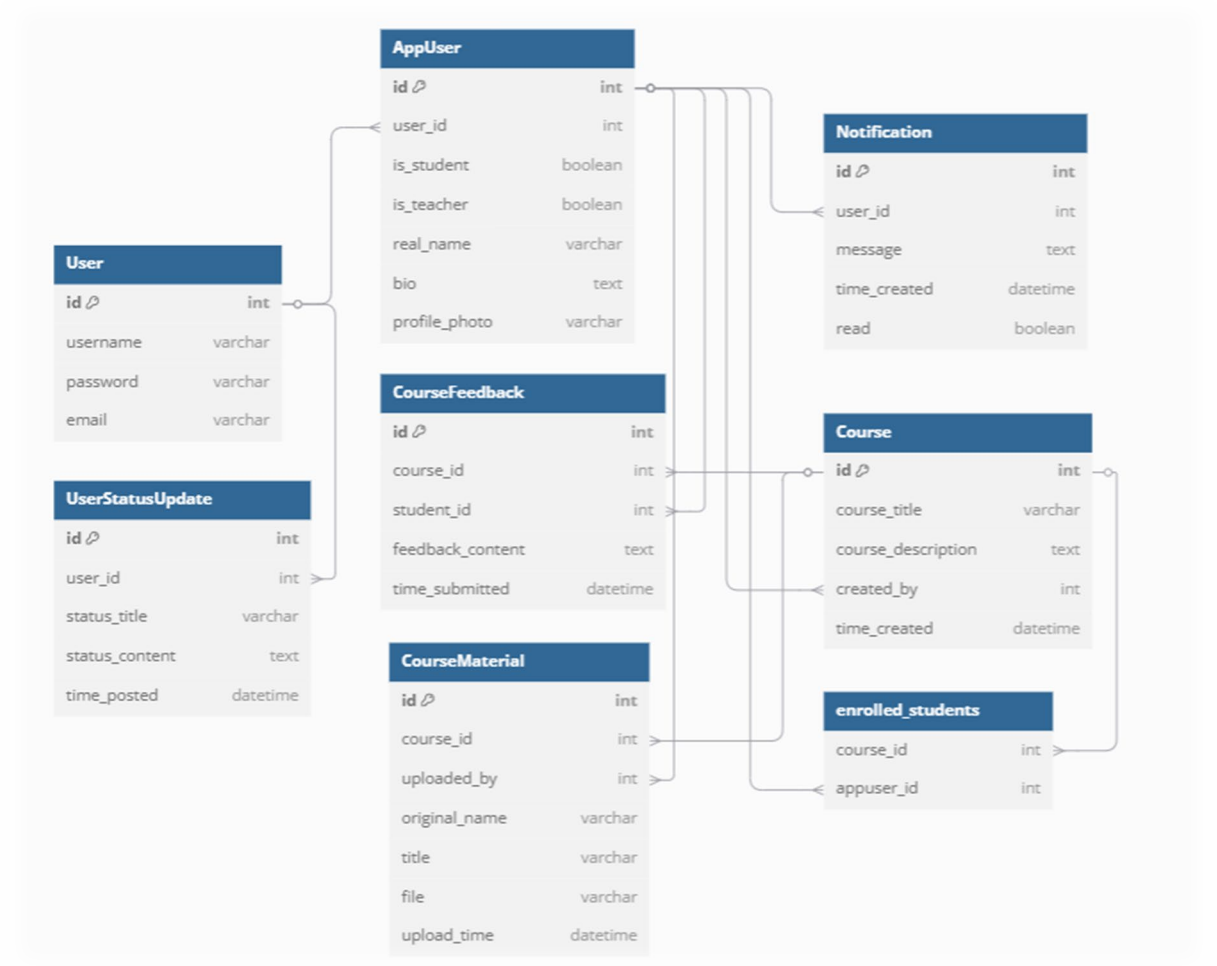
For the Third Normal Form (3NF), the database should avoid transitive dependencies, which means non-key attributes do not depend on another non-key attribute. For example, the CourseFeedback model shows that each feedback entry links only to a student and a course, instead of including extra details like the student's information. By doing so, the feedback table does not need to be updated every time a student updates their profile because it only stores the reference.

Going beyond 3NF to Fourth Normal Form (4NF) or Boyce-Codd Normal Form (BCNF) is unnecessary for this platform because the current database structure already minimizes

code redundancy and data integrity. 4NF aims to remove multi-valued dependencies, for instance, a course having multiple lists of students and course materials. In this structure, relationships like enrolled_students fields are already handled using ManyToMany field, avoiding that issue. BCNF is designed to handle cases where a table has overlapping candidate keys. Since this databases structure use single-column primary keys, normalising it into BCNF isn't necessary. Going beyond 3NF would just add extra unnecessary complexity to the database without many advantages.

The overall database structure follows 3NF, reducing data redundancy and enforcing data integrity. With these design structure, the platform can handle large data while maintaining speed and consistency.

2.4 ER Diagram



3. Authentication and Role-Based Access

3.1 Authentication System

The authentication system is handled using Django's built-in authenticate system, which includes login, logout, automatic password hashing, as well as session management. Users register through a custom registration form where they can provide a username, email, and password to create an account. The form was also extended to allow additional inputs such as real name, bio, as well as their role as either a student or teacher. Upon submission, the password is automatically hashed and stored using Django's default hashing algorithm, reducing the risk of data breaches.

When a new user register, their data is stored in two different models, User model, a Django's built-in authentication model, and AppUser model, a custom model extending the user functionality. The AppUser model assigns the selected role to the user, setting the is_student or is_teacher Boolean field accordingly. If the username already exists, an error message is displayed to prevent duplicate registrations. Upon successful registration, users are redirected to the login page, where they can enter their newly registered credentials to access the platform.

To ensure the platform remains secured, Django's built in session authentication is enabled, once a user logs in, their session remains active until they log out or when it expires. Additionally, Django's Cross Site Request Forgery (CSRF) protection is also implemented in every POST request form in the platform to prevent unauthorized form submissions from external sources. Upon logging out, the user session is cleared, and the user be redirected back to the login page. The logout function is locked behind Django's @login_required decorator, ensuring that only logged in user can perform the log out action.

3.2 Role-Based Access Controls and Permissions

The eLearning platform system implements Role-Based Access Control to manage permissions based on user roles. There are two main roles in the platform, student and teacher. Students can enroll in courses, post feedback, and access course materials, while teachers can create courses, manage students, and upload course materials.

To enforce proper access control, the platform utilises Django's @login_required decorator on almost every view function to ensure that only authenticated users can access them. For role-based permissions, Django's user_passes_test decorator is used. For example, teachers are the only ones allowed to search and view other user's profile, so the the user_passes_test decorator is called before the search_users view function to check if the logged-in user is a teacher before allowing them to perform the search. Before redirecting to the profile page, the user_passes_test decorator is called again to double check that the logged in user is a teacher before rendering the target user's profile page. If a student tries to access this URL or feature, they are redirected back to their respective homepage.

Access control is also implemented at form submissions, ensuring that students can only interact with courses they are enrolled in, and teachers can only manage courses they created. For instance, when submitting feedback, the system checks if the student is enrolled in the course before allowing the feedback submission.

By implementing authentication and role-based access control, the platform ensrues that users can only perform actions relevant to their respective role, maintaining the platform's security.

4. Course Management System

4.1 Course Creation and Deletion

Teachers can create various courses for the students, which they can do using the platform's web form. The Course model is created to store the details of the course, such as its title, description, creation timestamp. Each course entry in the database table is linked to a specific teacher using a Foreign Key relationship. This ensures that only the teacher who create the course has control over its content and actions.

The create_course view function is designed to handle course creation. It checks whether the logged-in user is a teacher using user_passes_test decorator to ensure that only authorised teacher can create courses. If a student tries to access this view via the URL, they are redirected back to their homepage. The form handling is done using the CourseForm, which validates the course title and description before saving it to the database.

The delete_course view function allows teachers to delete courses they created. When a teacher deletes a course, the respective course id is passed in the URL. The database then searches for a course with that specific id and perform the deletion. The view also ensures that the logged-in user matches the created_by field of the Course model before allowing the deletion. This prevents unauthorized users from removing courses that aren't theirs.

4.2 Course Enrollment and Unenrollment

Students can enroll in courses using a Many-To-Many relationship between the AppUser model and the Course model. This allows students to enroll in multiple courses while also allowing each course to have multiple enrolled students.

The enroll_course view function is structured to process enrolment requests, ensuring that the user is a student before adding them to the Course model's enrolled_students field. When a student enrolls, an notification is sent to the course teacher using the Notification model, and a success message toast is displayed to the student. This function also prevent overlapping enrollments by check if the student is already enrolled in the course. If the student is already enrolled, an error message is shown to inform the student.

The unenroll_course view function allows students to unenroll themselves from a course. Similar to delete_course view function, the course id of the course the student aims to unenroll from is passed to the URL. The function checks if the student is enrolled in the course before unenrolling them. This prevents students from unenrolling from a course they are not enrolled in.

4.3 Course Materials Upload

Teachers can upload various types of course materials, including PDFs, images, videos, and other file types to their courses. The CourseMaterial model is structured to store file details such as the title, original filename, upload timestamps. Initially, there was an issue where two files with the same filename would overwrite each other, causing data loss. This was fixed by storing uploaded files using UUID filenames, ensuring there is a unique name for each file.

The upload_course_materials view function ensures that only teachers who created the course can upload course materials by using user_passes_test decorator and by verifying that the logged-in user matches the created_by field in the Course model. The frontend displays two separate buttons, one for previewing the course material, another for downloading it. The preview button shows the original filename and extension and are used to open the file in a new browser tab. However, when the download button is clicked, the system downloads the file using its UUID-generated filename as stored in the database. This prevents filename conflicts while ensuring users can still recognize the file based on its original name when previewing.

4.4 Student Removal System

The question paper specifies that teachers should be able to remove or block students from courses. In this platform, instead of completely blocking students by blocking preventing them from enrolling in courses taught by that teacher again, teachers are given the ability to remove students from courses. The decision is due to blocking a student from all courses under a specific teacher could be too serious. On the other hand, removing a student from a specific course allows teachers to manage their class more effectively without completely removing a student's opportunity to learn. This ensures fairness gives teachers control over their current class while keeping future learning opportunities for the students.

At the course page, other than viewing the course materials, it also includes a button that displays all enrolled students in a pop-up modal. If the logged-in user is the teacher that created the course, they can see an additional Remove button next to each enrolled student's name. This provides a interactive way for teachers to manage their students.

The remove_student view function allows teachers to remove students from their course. The same decorator and checks are performed to verify that the logged-in user is the teacher that created the course. When a student is removed from a course, their student id and the course

id were passed to the function to process the removal. Once the checks passed, they are removed from the enrolled_students Many-To-Many relationship.

4.5 Course Feedback System

The course feedback system allows students to provide feedback on courses they are enrolled in. This feature helps teachers improve their course content based on the student experiences feedback, ensuring that their course remain high standard. The CourseFeedback model links each feedback to a specific course and student using Foreign Key relationships.

The submit_feedback view function is responsible for handling feedback submissions. It ensures that students can only submit feedback for courses they are enrolled in. It also ensures that teachers cannot submit feedback but can read feedback for the courses they created. Any unauthorised feedback are blocked and gets the user redirected back to their homepage with an error message.

5. Status Updates, Profile Management and Notifications

5.1 Status Updates System

Students and teachers both can post status updates to homepage, sharing their learning and teaching progress, discuss topics related to their enrolled courses. These updates are stored in the UserStatusUpdate model, which links each post to a specific AppUser entry and timestamps the submissions. The form includes a status_title and status_content field, allowing users to add a heading along with their main content.

The homepage dynamically loads status updates so that students can recent posts without needing to manually refresh the page. The homepage view function fetches the status updates by querying through UserStatusUpdate objects that were linked to the logged-in user, ensuring that only status updates from the correct user are displayed. Initially, it was sorted by default, causing the latest updates to be displayed at the bottom. To counter this, an additional order_by was utilised to sort them from the latest to the oldest, allowing the latest status updates to appear at the top of the section, improving visibility and relevancy.

5.2 Profile Management

Users can modify their profile details, including their real name, bio, and profile photo. The AppUser model stores these data, and users can update them through the edit_profile view function which renders the UserProfileUpdate form.

The profile photos are uploaded and processed asynchronously using Celery and Redis, which ensures that uploading large image files do not slow down the system. By utilising

them, multiple users can also update their profile photo at the same time. The process_profile_photo is a shared task that resizes and optimizes images before saving them. Initially, profile photos were stored without any size restrictions, leading to high storage consumption. This was fixed by image size and dimension compression, ensuring that all uploaded photos are optimized before being stored. The task uses Pillow (PIL) python library to open, resize, and crop images, maintain a maximum dimension 300px, then crops them into square images. Cropping them into square images prevents the images from getting stretched in display due to the modifications to its aspect ratio.

Users can view their own profiles through the user_profile view function, which fetches the latest logged-in user's profile details and renders them in the template. The profile page dynamically updates whenever changes are made, providing a better user experience.

5.3 User Search System

Teachers have access to a search feature, allowing them to render students and other teachers profile page by their username. The search_user view function retrieve user profiles based on the query input, and if a match is found, the teacher is redirected to the user's homepage.

Initially, when a teacher searched for a student, the system rendered a student homepage template used for a student's view. This resulted in an issue where teachers had access to the buttons that allowed them to enroll students in courses, submit feedback, and perform other actions meant for students, even when viewing someone else's profile. To address this issue, a check was added to both teacher's and student's homepage template to check if the logged-in user is the profile owner. If the teacher is viewing another user's profile, all buttons are disabled, ensuring that teachers can only view the user's homepage without any interactions to the account.

Additionally, when a teacher searches for a student or another teacher, a floating navigation bar appears at the top of the page. This floating bar shows information on which user the teacher is viewing on, as well as another two option buttons, returning to their own homepage or access the searched user's profile information page. This profile page is similar to a user's normal profile page view, displaying details such as real name, bio, profile photo, but instead of showing the logged-in teacher's information, it displays the searched user's profile instead. This allows teacher to navigate between their own profile and searched users more clearly.

5.4 Notification System

The notification system ensures that users receive updates about events such as course enrollment, and new course materials. Whenever a user enrolled in a course, or when a new course material has been uploaded, the respective notifications are then created and saved.

Notifications are stored in the Notification model, linking each notification to a specific user and tracking whether it has been read.

Initially, the notifications were only able to be displayed on the homepage, requiring users to visit the homepage every time they want to check for notification updates. To improve the usability, a context processor was used to make the notification data available globally. The function retrieves all unread notifications from the Notification model, then displays then in a dropdown when the notification button is clicked. This change allows notification to be accessible from any page, enhancing the user experience.

At the bottom of the notification dropdown, there is a view all notification link the redirects users to a page displayed all read and unread notifications. The all_notifications view function fetches all notifications for a user and marks them as read when they are viewed.

6. Real-Time Chat with WebSockets

6.1 Chat System Overview

The chat system is implemented using Redis, Django Channels, Daphne, and WebSockets. Redis is a server that acts as the channel layer, allowing messages to be sent between WebSocket consumers. Django Channels allows the project to support WebSocket communication, handling real-time connections asynchronously. The WebSocket is used for continuous two-way communication between the client and the server.

Daphne is a component that serves as the ASGI server, allowing Django to handle WebSocket connections. Initially, when the chat system was first implemented, the WebSocket connections would fail immediately and could not form the handshake, which made it impossible for users to send or receive messages. After scrolling through Django forums, it was discovered that I did not have Daphne installed, which is required for Django Channels to function properly, as it serves as the ASGI server. Once Daphne was installed and set up, WebSocket connections were established successfully, allowing real-time communication to work.

The routing file ensures that each chat room has a separated individual WebSocket connection, uniquely identified by the chat room name. When a user enters a chat room, their browser establishes a WebSocket connection to the route, allowing them to communicate in real-time.

W

6.2 Entering a Chat Room

Users can access the chat feature through the chat lobby, where they enter a room name to join a chat. The search_room.html template provides a search input where users enter a room name, and clicking the search button redirects them to the chat room. The chat_room view

function renders the chat room template, passing the room_name as a variable to the frontend to be displayed. Upon entering a chat room, the WebSocket connection is established, and users can start exchanging messages.

6.3 Real-Time Message Handling

The chat system is handled by the ChatConsumer class, which manages the WebSocket connections and chat messages asynchronously. When a user joins a chat room, they are automatically connected to a group channel that is uniquely connected to that specific chat room. This ensures that all users in the same chat room can receive and send messages instantly.

The connect method in the ChatConsumer class is designed to add user to a chat group. Each chat room is identified by its room name, and users who enter the same chat room are added to the same WebSocket group. When a user connects, the WebSocket retrieve the room_name to form room_group_name, ensuring that all users in the same chat room are part of the same group. The WebSocket connection, channel_name, is then registered to the Redis channel layer, linking the user to the chat group. Once the connection is registered, it is accepted, allowing the user to start exchanging messages. On the other hand, the disconnect method removes the user from the WebSocket group when they leave the chat room to reduce unnecessary connections.

When a user sends a message, the receive method in the class extracts the message data in JSON. The sender's username and profile photo URL are also retrieved asynchronously to enhance the message display. Additionally, a timestamp is also generated and attached to the message, ensuring that each message is stamped with the exact time it was sent. The group_send is then utilised to send the message to all users connected to the same chat group.

The chat_message method handles the messages received by the group and forwards them to the appropriate WebSocket connections. Initially, the messages were being sent and received without usernames, timestamps, and profile photos, making the chat confusing and cause impersonation issue. To address this issue, sender's username, profile photo URL and a timestamp are also sent alongside the message. On the receiving end, the data are then retrieved and formatted, ensuring that users can see who sent each message alongside with their respective profile picture and the exact time of the message.

6.4 Chat Interface

The chat_room.html template dynamically displays the messages using Javascript. When a message is received, a new HTML container is created, and the message is inserted into it. The container is then appended to the message log section, ensuring that new messages appear in the section without needing a manual page refresh. Initially, the message log did not

automatically scroll to the new messages, requiring users to manually scroll down to see the latest messages. This was fixed by adding a simple Javascript code that adjusts the scroll position of the message log to match its scroll height, ensuring that the message log automatically scrolls to display the new messages.

7. REST API and Data Access

7.1 REST API Overview

The REST API in this eLearning platform is built using Django REST Framework (DRF) to allow users to retrieve their data. According to the requirements in the question paper, the application should implement appropriate code for a REST interface that allows users to access their data only, so the implementation focuses on GET requests. All other functionalities, such as course enrollment, course feedback submissions, and a lot of other features are handled through web-request based views.

7.2 API Permissions and Security

The APIs ensures that users can access their data securely while maintaining working role-based access control. The DRF's built-in authentication and permission classes are used to restrict access, ensuring that users can only access the data they are allowed to access.

Two custom permission classes such as IsTeacher and IsStudent has been added to permission.py, which are used to restrict access to specific API endpoints. The IsTeacher permission class checks whether the authenticated user has an associated AppUser model, and whether the is_teacher attribute is set to true. If both conditionas are met, the request is approved and is granted access to the API endpoint. Similarly, the IsStudent permission class checks whether the authenticated user has an AppUser model with the is_student attribute set to True. This ensures that students and teachers can only access the API endpoints relevant to their roles.

7.3 API Endpoints

The following API endpoints are implemented to allow users to retrieve their data:

User Profile API (/api/user/profile/)

User Profile API allows authenticated users to retrieve their profile information, including their username, real name, bio and role. This API call is open to authenticated user, meaning that both teachers and students can access this endpoint to get their own profile data.

User Status Updates API (/api/user/status-updates/)

User Status Updates API allows authenticated users to retrieve all status updates they have posted. This endpoint ensures that users can only access their own status updates, maintaining privacy.

User Enrolled Courses API (/api/user/enrolled-courses/)

User Enrolled Courses API allows authenticated users to retrieve all status updates they have posted. This API endpoint can only be accessed by authenticated users that are students and ensures that they can only view courses where they are listed as an enrolled student.

User Created Courses API (/api/user/created-courses/)

User Created Courses API allows authenticated teachers to retrieve a list of courses they have created. Only users with teacher permissions can access this endpoint, ensuring that teachers can manage their own courses.

User Notifications API (/api/user/notifications/)

User Notifications API retrieves all notifications for the logged-in user. This allows users to get notified about events such as course updates, and new enrollments. Each users can only access their own notifications.

Course Feedback API (/api/course/<id>/feedback/)

Course Feedback API allows both students and teachers to retrieve feedback on a specific course. While students can access feedback for all courses, teachers can only access feedback for the course they have created.

Course Materials API (/api/course/<id>/materials/)

Course Materials API allows students and teachers to retrieve course materials uploaded for a specific course. This endpoint ensures that students can only access materials for courses they are enrolled in, while teachers can only retrieve materials from the courses they have created.

7.4 API Documentation

To provide documentation for the implemented API endpoints, OpenAPI auto-schema generation is used to automatically create the API structure, and Swagger is integrated to create an interface to test API calls. OpenAPI generates a schema that shows each endpoint's required parameters and expected responses, while Swagger renders it in a clean user interface. It allows authenticated users to test all available API endpoints directly from the browser.

8. Unit Testing

8.1 Unit Testing Overview

Unit testing is important to ensure the security, and functionality of the eLearning platform. A total of 30 unit tests were implemented across different apps, including authentication, course management, user notifications, profile interactions, real-time chat, and API endpoints. By utilising Django's TestCase framework and Django Rest Framework's APIClient, the tests ensures that all features work as expected.

8.2 Authentication Testing

Authentication is one of the most important systems of the platform, it ensures secure access for users. Multiple tests were implemented to verify the functionalities of the main functions.

Test	Purpose
test_registration_loads	Ensures that the registration page loads successfully.
test_login_loads	Checks that the login page loads successfully.
test_registration_function	Verifies that the new users can register and redirected to the login page.
test_login_function	Ensures that users with valid credentials can log in and are redirected to their homepage.
test_logout_function	Checks that authentication users can log out and are redirected to the login page.

8.3 Course Management Testing

Test	Purpose
test_create_course_function	Ensures that teachers can create courses.
test_enroll_course_function	Checks that students can enroll in a course that they have not enrolled in.
test_unenroll_course_function	Verifies that students can unenroll themselves from a course they were enrolled in.

test_delete_course_function	Ensures that teachers can delete courses they have created.
test_load_feedbacks_function	Checks that both students and teachers can retrieve feedback for courses.
test_submit_feedback_function	Verifies that students can submit feedback for enrolled courses.
test_view_course_function	Ensures that students and teachers can view course details.
test_remove_student_function	Checks that teachers can remove students from their courses.
test_upload_course_materials_function	Verifies that teachers can upload course materials to their courses.
test_enrollment_notification_function	Ensures that teachers can receive notifications when a student enrolls in their course.
test_upload_notification_function	Checks that students can receive notifications when new course materials are uploaded.

8.4 User Interaction

Test	Purpose
test_homepage_function	Ensures that students and teachers can see their respective homepage with relevant data.
test_search_users_function	Checks that teachers can search for other users.
test_show_profile_function	Verifies that teachers can view other user's profile.
test_all_notifications_function	Ensures that users can access all their notifications.
test_user_profile_function	Checks that users can view their own user profile.
test_show_information_function	Verifies that teachers can view other user's details.
test_edit_profile_function	Ensures that users can update their profile details.

8.5 API Endpoints

Test	Purpose
test_user_profile_api	Ensures that authenticated users can retrieve their profile details.
test_user_status_updates_api	Checks that users can access a list of their own status updates.
test_user_enrolled_courses_api	Verifies that students can retrieve a list of their enrolled couses.
test_user_notifications_api	Ensures that can retrieve a list of their enrolled courses.
test_user_created_courses_api	Checks that teachers can retrieve a list of courses they have created.
test_course_feedback_api	Verifies that students and teachers can access feedback of related courses.
test_course_materials_api	Ensures that users can retrieve course materials for courses they have enrolled in.

8.6 Unit Test Result

```
System check identified no issues (0 silenced).
test_course_feedback_api (api.tests.TestAPI.test_course_feedback_api) ... ok
test_course_materials_api (api.tests.TestAPI.test_course_materials_api) ... ok
test_user_created_courses_api (api.tests.TestAPI.test_user_created_courses_api) ... ok
test_user_enrolled_courses_api (api.tests.TestAPI.test_user_enrolled_courses_api) ... ok
test_user_notifications_api (api.tests.TestAPI.test_user_notifications_api) ... ok
test_user_profile_api (api.tests.TestAPI.test_user_profile_api) ... ok
test_user_status_updates_api (api.tests.TestAPI.test_user_status_updates_api) ... ok
test_login_function (authentication.tests.TestAuthentication.test_login_function) ... ok
test_login_loads (authentication.tests.TestAuthentication.test_login_loads) ... ok
test_logout_function (authentication.tests.TestAuthentication.test_logout_function) ... ok
test_registration_function (authentication.tests.TestAuthentication.test_registration_function) ... ok
test_registration_loads (authentication.tests.TestAuthentication.test_registration_loads) ... ok
test_create_course_function (courses.tests.TestCourses.test_create_course_function) ... ok
test_delete_course_function (courses.tests.TestCourses.test_delete_course_function) ... ok
test_enroll_course_function (courses.tests.TestCourses.test_enroll_course_function) ... ok
test_enrollment_notification_function (courses.tests.TestCourses.test_enrollment_notification_function) ... ok
test_load_feedbacks_function (courses.tests.TestCourses.test_load_feedbacks_function) ... ok
test_remove_student_function (courses.tests.TestCourses.test_remove_student_function) ... ok
test_submit_feedback_function (courses.tests.TestCourses.test_submit_feedback_function) ... ok
test_unenroll_course_function (courses.tests.TestCourses.test_unenroll_course_function) ... ok
test_upload_course_materials_function (courses.tests.TestCourses.test_upload_course_materials_function) ... ok
test_upload_notification_function (courses.tests.TestCourses.test_upload_notification_function) ... ok
test_view_course_function (courses.tests.TestCourses.test_view_course_function) ... ok
test_all_notifications_function (users.tests.TestUser.test_all_notifications_function) ... ok
test_edit_profile_function (users.tests.TestUser.test_edit_profile_function) ... ok
test_homepage_function (users.tests.TestUser.test_homepage_function) ... ok
test_search_users_function (users.tests.TestUser.test_search_users_function) ... ok
test_show_information_function (users.tests.TestUser.test_show_information_function) ... ok
test_show_profile_function (users.tests.TestUser.test_show_profile_function) ... ok
test_user_profile_function (users.tests.TestUser.test_user_profile_function) ... ok

----------------------------------------------------------------------
Ran 30 tests in 33.464s

OK
Destroying test database for alias 'default' ('file:memorydb_default?mode=memory&cache=shared')...
```

```
$  py manage.py test
Found 30 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
..............................
----------------------------------------------------------------------
Ran 30 tests in 33.633s

OK
Destroying test database for alias 'default'...
```

9. Frontend

9.1 Tailwind CSS

Tailwind CSS is s utility-first CSS framework that allows for fast UI development without the need for writing custom CSS styles from scratch. Unlike Bootstrap, Tailwind provided low-level utility classes that enable us to create fully customized components. This offers greater flexibility in UI design, as margins, paddings, stylings all can be applied directly within class attributes rather than modifying them in an external stylesheet.

9.2 Flowbite

Flowbite is an open-source collection of UI components built with the TailwindCSS utility classes. It provides a library of pre-designed UI components, such as modals, buttons, alerts, and more, accelerating the frontend development process. Since Flowbite's components are already responsive and built with Tailwind, they are fully customisable, giving complete control over their styling.

9.3 Widget Tweaks

Django's default form design is simple and does not provide styling that matches the overall design. To address this, django-widget-tweaks plugin was used to modify the form fields design dynamically in templates. This allows control over each form field's design without modifying Django's rendering logic.

10. Evaluation and Future Improvements

10.1 Authentication and Role-Based Access Control

The authentication system uses Django's built-in authentication model however, multi-factor authentication (MFA) should be added in the future to enhance security. Additionally, the platform should also provide Google or GitHub login support, simplifying the login process and provide an alternative way for users to access their account.

RBAC uses the user's role to determine their action rights and permissions, however, access logs for authentication and RBAC actions are not tracked. Future improvements should include action logging, allowing administrators to review potential unauthorized access attempts.

10.2 Course Management System

Students can freely enroll in any available courses currently. While this make the enrollment process simpler, it does not allow teachers to fully control who joins their classes. Implementing a teacher approval system for course enrollment would allow teachers to manually approve or reject enrollment requests.

The student removal system currently allows teacher to remove students from their courses, but the students can still re-enrollment. A future improvement such as temporary ban system would prevent students from re-enrolling right after removal.

The course management system allows teachers to delete a course they have created. However, the course deletion is permanent, once a teacher deletes a course, all course materials and relevant data is lost. A future improvement could introduce a disabled feature, allowing teachers to temporarily disable courses instead of deleting them.

Course feedback allows students to provide insights on their learning experience. A future improvement such as allowing anonymous feedback option could be selected to encourage unbiased feedback. Additionally, the feedback system could include a rating system, allowing students to rate courses.

10.3 Status Updates and Notifications

The status update feature allows users to share their status progress. To make this feature more interactive and engaging, future improvements such as adding likes and comments to status updates would allow users to interact with each other, creating a more interactive community.

The notification system provides user with notification updates in the web. Future improvements such as sending email notifications would ensure users stay informed even when they are not on the eLearning Platform.

10.4 Real-Time Chat System

The chat system provides real-time communication between teachers. However, the chat messages are currently not stored in a database, which means that the conversation are lost once user disconnect. Future improvements should include chat history storage to allow users to read past conversations. Additionally, a profanity filter and spam detection could also be added to the chat system to maintain a positive learning environment.

11. Conclusion

The eLearning platform successfully meets all the functional requirements. Through features like user authentication and role-based access control, status updates, profile, management, notifications, real-time chat feature, and REST API data access, the system creates an efficient and secured online education platform. However, while the platforms fit all the requirements, there are areas where it could be improved to enhance its functionality and user experience. For instance, the authentication system can be expanded by incorporating multi-factor authentication (MFA) and more.

In conclusion, this platform can act as a base for an eLearning system, integrating all the main eLearning tools while maintaining security. Future improvements would refine user experience and security, ensuring the platform continues to grow into a huge eLearning platform.

12. Appendix

12.1 Installation Guide

To set up and run the application, follow these steps:

1. **Unzip the Project Folder**
   - Extract the eLearning project from the ZIP file to any location.
2. **Navigate to the Project Directory**
   - Open a terminal and change directory into the project folder. Ensure that you are in the same path as manage.py:
     `cd elearning`
3. **Install Required Dependencies**
   - Install required Python packages using requirement.txt:
     `pip install -r requirements.txt`
4. **Run the Development Server**
   - Run the eLearning project:
     `python manage.py runserver`

12.2 Celery and Redis Setup

Celery is used to handle background tasks, while Redis serves as the message broker for Celery. To set up, please follow these steps:

1. **Start Redis Server via Ubuntu**
   - To start Redis:
     `sudo service redis-server start`

- To check if Redis is running:
  `sudo service redis-server status`
- To stop Redis:
  `sudo service redis-server stop`

2. **Start Celery Worker**
   - Starts Celery (Windows):
     `celery -A elearning worker --loglevel=info -P threads`
   - Starts Celery for other OS:
     `celery -A elearning worker --loglevel=info`

12.3 Installation and Setup Troubleshooting

If the application does not run correctly, please try the following solutions:

- **CSS not rendering properly**
  - Try rebuilding Tailwind CSS
    `npm run css`
- **Styling still not working**
  - Try installing the node modules
    `npm install`
- **WebSocket connection failed immediately**
  - Ensure Daphne is installed
    `pip install daphne`
  - Rerun the server
    `python manage.py runserver`

12.4 Running Unit Tests

To run the unit tests, run the following command:

```
python manage.py test
```

For more testing information, run the following:

```
python manage.py test -v 2
```

12.5 Packages and Dependencies

Python Packages

Package	Version
asgiref	3.0.1
channels	4.2.0
Django	5.1.2
djangorestframework	3.15.2
Pillow	11.1.0
daphne	4.1.2
django-compressor	4.5.1
django-tailwind	3.8.0
django-widget-tweaks	1.5.0
redis	5.2.1
celery	5.4.0
channels_redis	4.2.1

Node.JS Packages

Package	Version
@tailwindcss/cli	4.0.7
flowbite	3.1.2
tailwindcss	4.0.7

12.6 Development Environment

Operating System: Windows 11 Home 24H2
Python Version: 3.13.0

12.7 Django Admin Panel

URL: http://127.0.0.1:8000/admin/
Username: admin
Password: password

12.8 Login Credentials

The following are the login credentials for the pre-seeded user accounts:

Username	Role	Password
student_1	Student	studentone
student_2	Student	studenttwo
student_3	Student	studentthree
teacher_1	Teacher	teacherone
teacher_2	Teacher	teachertwo
teacher_3	Teacher	teacherthree

12.9 References

1. How to make global context variable in django. (n.d.). Stack Overflow. https://stackoverflow.com/questions/56181675/how-to-make-global-context-variable-in-django
2. what is the correct way to override the save method in django? (n.d.). Stack Overflow. https://stackoverflow.com/questions/61623638/what-is-the-correct-way-to-override-the-save-method-in-django
3. django - How to edit/manipulate uploaded images on the fly before saving - Bharat Chauhan. (n.d.). https://bhch.github.io/posts/2018/12/django-how-to-editmanipulate-uploaded-images-on-the-fly-before-saving/

4. ANTIALIAS vs BICUBIC in PIL(Python Image Library)? (n.d.). Stack Overflow. https://stackoverflow.com/questions/23113163/antialias-vs-bicubic-in-pilpython-image-library
5. Tailwind CSS - Rapidly build modern websites without ever leaving your HTML. (n.d.). Tailwind CSS. https://tailwindcss.com/
6. Themesberg. (n.d.). Flowbite - Tailwind CSS component library. https://flowbite.com/docs/getting-started/introduction/
7. Automatic Schema Generation. (n.d.) Coursera https://www.coursera.org/learn/uol-cm3035-advanced-web-development/lecture/frr1w/7-308-automatic-schema-generation
8. Automatic Documentation Generation. (n.d.) Coursera https://www.coursera.org/learn/uol-cm3035-advanced-web-development/lecture/BiFBn/7-310-automatic-documentation-generation
9. django channels vs daphene. (n.d.). Stack Overflow. https://stackoverflow.com/questions/59202142/django-channels-vs-daphene
10. Chris, K. (2022, December 21). Database Normalization – Normal Forms 1NF 2NF 3NF Table examples. freeCodeCamp.org. https://www.freecodecamp.org/news/database-normalization-1nf-2nf-3nf-table-examples/
11. Using the Django authentication system | Django documentation. (n.d.). Django Project. https://docs.djangoproject.com/en/5.1/topics/auth/default/