

Report

Anthony Winata Salim

Advanced Web Development

1. Introduction

Learn Together eLearning App

As the dynamic era of education keeps evolving, the web applications have become the tools of the era for disseminating knowledge and acquiring skills. The COVID-19 pandemic only accelerated the process, requiring steady, user-friendly eLearning systems that could host successful teacher-student interactions beyond classrooms. Here is a chronicle of designing, developing, and testing "Learn Together," an eLearning web application in Django and newer web technologies.

Project Overview

Learn Together is a comprehensive eLearning platform that seeks to bring together learners and educators in an easy-to-use virtual environment. The site enables educators to create and publish courses, upload course materials, and interact with learners in real time. Learners are able to search courses, track their learning activity, provide feedback, and interact with other learners and educators. At a bare minimum, the site seeks to mirror the essential dynamics of conventional learning while capitalizing on the intrinsic strengths of digital technology.

Scope and Objectives

The main aim of this project was to create and design an operational eLearning website that meets the requirements stated below:

Implement a dual-role user system (students and teachers) with the relevant permissions

Develop a course management system with enrollment and feedback facilities

Create user profiles with status and discoverable info

Implement real-time notifications for user activities

Facilitate real-time user-to-user interaction using web sockets

Provide a RESTful API to fetch user information

Provide a responsive, user-friendly interface

Its coverage was front-end and back-end development with a focus on Django's built-in functionality and extension through third-party libraries. Although the application covers all fundamental functionality of an eLearning app, some advanced features such as streaming video or comprehensive assessment tools were outside the scope of this version.

Technologies and Methodology

The app was built on the Django web framework (version 5.1.2) due to its solid architecture, security features out of the box, and huge ecosystem. The main technologies used are:

Django: Underlying framework that offers the Model-View-Template framework

Django REST Framework: To create the API endpoints

Django Channels: For enabling WebSocket functionality

Bootstrap 5: Utilized for responsive front-end development SQLite: As the development backend database

JavaScript: For making user interface interactive Redis: As the planned channel layer for WebSocket communication (with a demo JavaScript implementation) The iterative method was employed during the development process, beginning with basic user management functionality, then course management, communications, and lastly the REST API. This permitted ongoing testing and development of each module.

2.Application Design

2.1 Database Design

Learn Together database schema is a relational schema, which has been designed for minimum redundancy, data integrity, and well-defined relationships among entities. The most significant requirements of distinguishing between various types of users, management of courses, and two-way communication are captured in the schema.

Entity Relationship Model :

User (extends Django's User):

- Profile attributes (user_type, bio, , profile_pic)
- 1 : N -> StatusUpdate
- 1 : N -> Taught courses (as a teacher)
- M : N -> Enrolled courses (as a student) through enrolment
- 1 : N -> Notification

Courses:

- Basic attributes (title, description, created_at)
- N : 1 -> Teacher
- M : N -> students through enrolment
- 1 : N -> materials
- 1 : N -> feedback
- 1 : N -> notification

Material:

- Content Attribute (title, description, file)
- N : 1-> course

Feedback:

- Content Attributes (content, rating_
- N : 1 -> Course
- N : 1 -> Student

Notification:

- Type attributes (notification_type, is_read)
- N:1 → Recipient (User)
- N:1 → Course

Chatroom:

- M:N → Participants (Users)
- 1:N → Messages

Message:

- Content attributes (content, timestamp)
- N:1 → ChatRoom
- N:1 → Sender (User)

Normalization Technique

To prevent data redundancy or duplication and ensuring data integrity, the database is designed to be in the third normal form(3NF) :

1. First Normal Form (1NF) requires each attribute to be of atomic value and does not allow repeating groups. For instance, course content is handled separately and not as arrays in the Course model.
2. Second Normal Form (2NF) requires the non-key attributes to have total functional dependence on the primary key. For example, feedback information goes into another table with composite dependency on course and user.
3. Third Normal Form (3NF): There is no transitive dependency of non-key attributes. User profile information is dependent on the user ID directly, not indirectly because of an intervening dependency.

Design Decisions

for Keys Certain key decisions impacted the database schema:

1. User Model Extension: Instead of a completely customized user model, I extended Django's built-in User model by introducing a user_type field with choices 'student' and 'teacher'. This introduces user role-based functionality over Django's auth system.
2. The one-to-many relationship from student to course is implemented via an intermediary model Enrollment instead of a ManyToManyField. This allows for the addition of enrollment dates and other enrollment information.
3. A different model called Notification is used to record events such as enrollments of courses and publication of new content. This is carried out in order to centralize the management of notifications so that it will be simpler to include other kinds of notifications in the future.
4. Chat System: The Message module records an individual message, and the ChatRoom module maintains conversation context between participants. The gap offered ensures follow-up and one-to-one group discussion.

2.2 Application Architecture

The app is developed on Django's Model-View-Template (MVT) architectural pattern with a clearly separated concern among multiple apps.

Application Structure

Learn Together is organized into three main Django apps with the following responsibilities:

- user_accounts: Handles user authentication, profiles, and status updates
- courses: Manages course creation, registration, materials, and feedback
- live-chat: Offers live chat functionality

This modular structure maximizes maintainability and is supportive of the Django philosophy of organization by business domain instead of technical features.

Model-View-Template Implementation

The MVT pattern is applied as follows:

- Models: Specify data structure and business logic, which is in each application's models.py file
- Views: Handle HTTP requests and responses using function-based and class-based views in views.py files
- Templates: Provide the HTML rendering using Django template language, in app-specific directories
- URLs: Link directly to views in each app's urls.py file, included centrally in the project's top-level URL configuration

In addition, Django REST Framework's serializer-viewset pattern is employed for API elements, with serializers translating model data to/from JSON and viewsets managing API requests.

Authentication and Authorization System

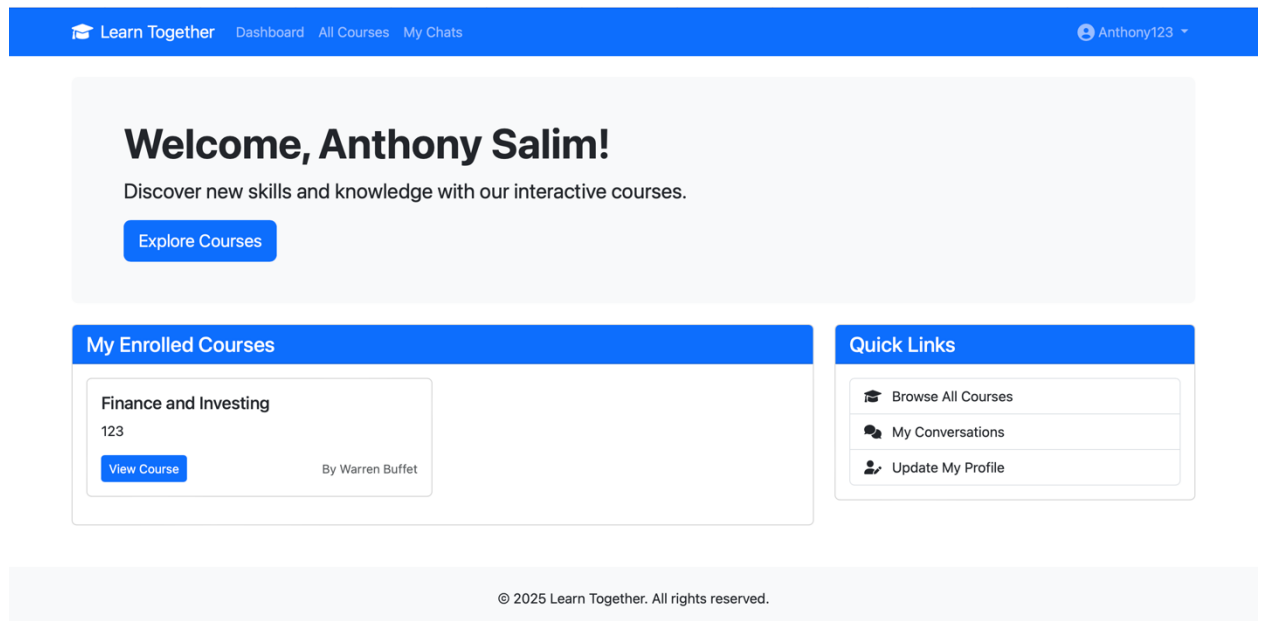
The system of authentication utilizes Django's built-in authentication with custom permissions for different types of users:

1. Base Permissions: Inherited from Django's auth system, handling views' access depending on whether they're logged in
2. Role-Based Permissions: Custom permissions filter on the user_type column to limit action (e.g., only the instructor can create a course)
3. Object-Level Permissions: In certain domains such as course administration, permissions check if the user is an instructor for the course
4. API Permissions: Custom REST API permission classes provide that users only receive access to suitable data What these various levels of permission do is allow users to be capable of performing tasks that are consistent with what they're attempting to do relative to what's required of them by the content.

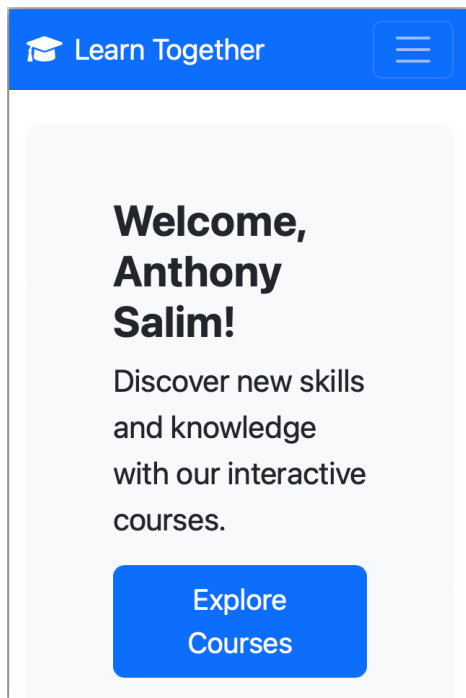
2.3 User Interface Design

The user interface is designed for simplicity, consistency, and responsiveness to produce an easy-to-use learning experience across multiple devices.

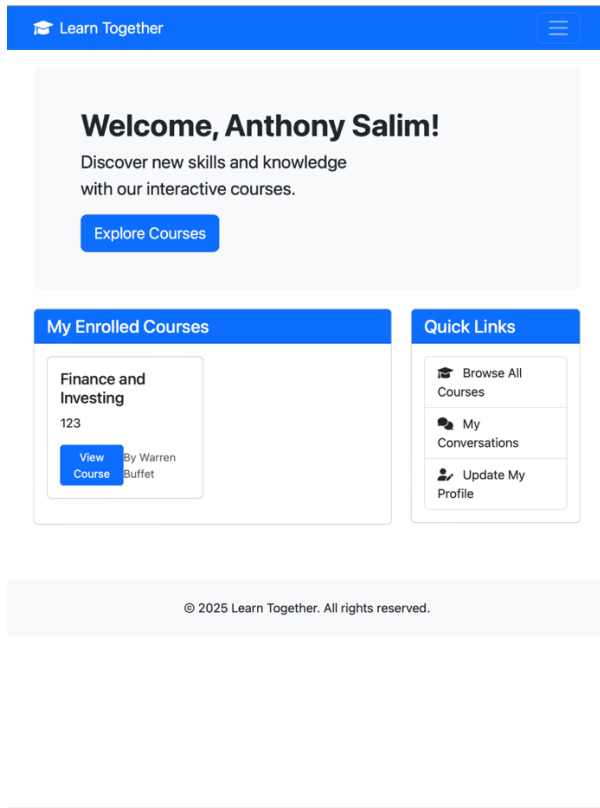
Overall Layout:



Responsive Design Mobile View:



Responsive Design Tablet View:



Design Principles

The interface design follows these key principles:

- 1.Consistency: All the common elements such as navigation, cards, and buttons are consistently styled throughout the app
- 2.Clarity: Strong visual hierarchy guides users through important actions and information
- 3.Feedback: Visual feedback reacts to user interactions with alerts, badges, and state changes
- 4.Accessibility: Color contrast, font sizes, and interactive elements adhere to accessibility best practices

Responsive Implementation

Bootstrap 5 offers the basis for a fully responsive design that can be adapted to various screen sizes:

- 1.Grid System: Fluid layouts adapt from multi-column on computers to single-column on mobiles

- 2.Component Sizing: UI components like cards, forms, and buttons resize as expected
- 3.Navigation: Primary navigation is minimized to a hamburger menu on smaller screens
- 4.Media Queries: Custom CSS extends Bootstrap for app-specific responsive behaviors

User Experience Considerations

Several UX features enhance the overall experience:

- 1.Dashboard Approach: Home pages of students and teachers are personalized dashboards with respective information and quick actions
- 2.Card-Based Layout: Course and course material listings utilize card elements to create visual separation and prominence
- 3.Status Indicators: Badges and icons represent states such as unread notifications or enrollment status
- 4.Progressive Disclosure: Sophisticated features disclose information progressively to prevent users from being overwhelmed The design of the interface melds beauty with utilitarian practicality in a way that allows both teachers and students to think about content and learning, not website use.


3.Implementation Details

3.1 Core Functionality Implementation (R1)

User Authentication System

The authentication system implements a dual-role approach (teachers and students) using Django's built-in authentication framework with customizations.

Account Creation Process

 Learn Together

Log InSign Up

Log In


Username*

Password*

Log In

New here? [Sign up for an account](#)

© 2025 Learn Together. All rights reserved.

 Learn Together

Log InSign Up

Join Our Learning Community

Username*

Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

Email*

First name*

Last name*

Create Account

Already have an account? [Log in here](#)

User type*

Student

Password

• Your password can't be too similar to your other personal information.

• Your password must contain at least 8 characters.

• Your password can't be a commonly used password.

• Your password can't be entirely numeric.

Password confirmation

Enter the same password as before, for verification.

© 2025 Learn Together. All rights reserved.

1. User Creating Form:

```

class UserRegistrationForm(UserCreationForm):

    USER_TYPE_CHOICES = [

        ('student', 'Student'),

        ('teacher', 'Teacher'),

    ]

    user_type = forms.ChoiceField(choices=USER_TYPE_CHOICES, required=True)

    email = forms.EmailField(required=True)

    class Meta:

        model = User

        fields = ['username', 'email', 'first_name', 'last_name', 'user_type', 'password1',
'password2']

```

2. Registration Process :

```

def register(request):

    if request.method == 'POST':

        form = UserRegistrationForm(request.POST)

        if form.is_valid():

            form.save()

            messages.success(request, 'Account created successfully! You can now log in.')

            return redirect('login')

    else:

        form = UserRegistrationForm()

    return render(request, 'user_accounts/register.html', {'form': form})

```

User type and Permission

1. User Model:

```
class User(AbstractUser):
```

```
    USER_TYPE_CHOICES = [  
        ('student', 'Student'),  
        ('teacher', 'Teacher'),  
    ]
```

```
    user_type = models.CharField(max_length=10, choices=USER_TYPE_CHOICES,  
                                default='student')
```

```
    profile_pic = models.ImageField(upload_to='profile_pics', blank=True, null=True)
```

```
    bio = models.TextField(blank=True, null=True)
```

```
    def is_teacher(self):
```

```
        return self.user_type == 'teacher'
```

```
    def is_student(self):
```

```
        return self.user_type == 'student'
```

2. Permission Checks:

```
@login_required
```

```
def create_course(request):
```

```
    if not request.user.is_teacher():
```

```
        messages.error(request, 'Only teachers can create courses.')
```

```
        return redirect('home')
```


User Profile and Status Update

Learn Together

DashboardAll CoursesMy Chats

Anthony123

Profile Info



Anthony Salim

Student

anthonysalim85@gmail.com

Enrolled in

[Finance and Investing](#)

Update Profile

First name

Anthony

Last name

Salim

Email address

anthonysalim85@gmail.com

Save Changes

Profile pic

Choose File | no file selected

Bio

Post a Status Update

What's on your mind?

Post Update

Recent Updates

No status updates yet.

[Dashboard](#)
[All Courses](#)
[My Chats](#)
[Create Course](#)
[Find Users](#)

teacher123

Profile Info

Warren Buffet
Teacher
teacher@gmail.com

Teaches

[Finance and Investing](#)

Update Profile

First name

Last name

Email address

Profile pic
 no file selected

Bio

Post a Status Update

What's on your mind?

Recent Updates

No status updates yet.

1. User profile:

```
class StatusUpdate(models.Model):
```

```
    user = models.ForeignKey(User, on_delete=models.CASCADE,
related_name='status_updates')
```

```
    content = models.TextField()
```

```
    created_at = models.DateTimeField(auto_now_add=True)
```

```
class Meta:
```

```
    ordering = ['-created_at']
```

2. Profile View:

```
@login_required
```

```
def profile(request, username=None):
```

```
    # Determine which user's profile to show
```

```
if username:
    profile_user = get_object_or_404(User, username=username)
    editable = request.user == profile_user
else:
    profile_user = request.user
    editable = True

# Handle form submissions
if request.method == 'POST' and editable:
    if 'update_profile' in request.POST:
        # Profile update logic...
    elif 'post_status' in request.POST:
        # Status update logic...

# Render profile with statuses and courses
statuses = profile_user.status_updates.all()
```

Course Management

Create Course

Title*

Description*

Create Course

Cancel

Finance and Investing



Taught by: [Warren Buffet](#)

Created: March 4, 2025

123

Add Learning Material


Course Materials


Security Analysis


Added: Mar 05, 2025

 Download

Enrolled Students (1)

 [Anthony Salim](#)





Learn Together
Dashboard
All Courses
My Chats
Anthony123

Finance and Investing

Taught by: [Warren Buffet](#)
Created: March 4, 2025

123

Course Materials

Security Analysis
Added: Mar 05, 2025

Download

Leave Feedback

Content*

Rating*

☒ 1
☐ 2
☐ 3
☐ 4
☐ 5

Submit Feedback

1. Course Form:

```
class CourseForm(forms.ModelForm):

    class Meta:

        model = Course

        fields = ['title', 'description']

        widgets = {

            'description': forms.Textarea(attrs={'rows': 5}),

        }
```

2. Course Model:

```
class Course(models.Model):

    title = models.CharField(max_length=200)

    description = models.TextField()
```

```
teacher = models.ForeignKey(User, on_delete=models.CASCADE,  
related_name='taught_courses')
```

```
created_at = models.DateTimeField(auto_now_add=True)
```

```
students = models.ManyToManyField(User, through='Enrollment',  
related_name='enrolled_courses')
```

Material Upload System

1. Materials:

```
class Material(models.Model):
```

```
    course = models.ForeignKey(Course, on_delete=models.CASCADE,  
related_name='materials')
```

```
    title = models.CharField(max_length=200)
```

```
    description = models.TextField(blank=True, null=True)
```

```
    file = models.FileField(upload_to='course_materials/')
```

```
    upload_date = models.DateTimeField(auto_now_add=True)
```

2. Upload View:

```
@login_required
```

```
def upload_material(request, course_id):
```

```
    course = get_object_or_404(Course, id=course_id)
```

```
    # Check if user is the teacher of this course
```

```
    if not request.user.is_teacher() or course.teacher != request.user:
```

```
        return HttpResponseForbidden("You don't have permission to upload materials.")
```

Enrollment System

1. Enrollment:

```
class Enrollment(models.Model):  
    student = models.ForeignKey(User, on_delete=models.CASCADE)  
    course = models.ForeignKey(Course, on_delete=models.CASCADE)  
    enrollment_date = models.DateTimeField(auto_now_add=True)  
  
    class Meta:  
        unique_together = ('student', 'course')
```

2. Enroll Course:

```
@login_required  
def enroll_course(request, course_id):  
    if not request.user.is_student():  
        messages.error(request, 'Only students can enroll in courses.')  
        return redirect('home')  
  
    course = get_object_or_404(Course, id=course_id)  
  
    # Check if already enrolled  
    if Enrollment.objects.filter(student=request.user, course=course).exists():  
        messages.info(request, f'You are already enrolled in "{course.title}")')  
        return redirect('course_detail', course_id=course.id)  
  
    # Create enrollment and notification  
    Enrollment.objects.create(student=request.user, course=course)
```

```

Notification.objects.create(

    recipient=course.teacher,

    course=course,

    notification_type='enrollment'


)


messages.success(request, f'You have successfully enrolled in "{course.title}"!')

return redirect('course_detail', course_id=course.id)


```

Feedback Mechanism


[Learn Together](#)
[Dashboard](#)
[All Courses](#)
[My Chats](#)


Anthony123

Finance and Investing



Taught by: [Warren Buffet](#)
Created: March 4, 2025

123

Course Materials

Security Analysis

Added: Mar 05, 2025


Download

Leave Feedback

Content*

Rating*

☒ -----

☐ 1
☐ 2
☐ 3
☐ 4
☐ 5

Submit Feedback

1. Feedback:

```

class Feedback(models.Model):

    course = models.ForeignKey(Course, on_delete=models.CASCADE,
related_name='feedbacks')

```

```

student = models.ForeignKey(User, on_delete=models.CASCADE)
content = models.TextField()
rating = models.IntegerField(choices=[(i, i) for i in range(1, 6)])
created_at = models.DateTimeField(auto_now_add=True)

class Meta:
    unique_together = ('student', 'course')

```

2. Handling feedback submission:

Inside course_detail view

if request.method == 'POST' and 'submit_feedback' in request.POST and is_enrolled:

```

    feedback_form = FeedbackForm(request.POST)

```

```

    if feedback_form.is_valid():

```

```

        # Check if feedback already exists

```

```

        existing_feedback = Feedback.objects.filter(course=course,
student=request.user).first()

```

```

        if existing_feedback:

```

```

            # Update existing feedback

```

```

            existing_feedback.content = feedback_form.cleaned_data['content']

```

```

            existing_feedback.rating = feedback_form.cleaned_data['rating']

```

```

            existing_feedback.save()

```

```

            messages.success(request, 'Your feedback has been updated!')

```

```

        else:

```

```

            # Create new feedback

```

```

            feedback = feedback_form.save(commit=False)

```

```

            feedback.course = course

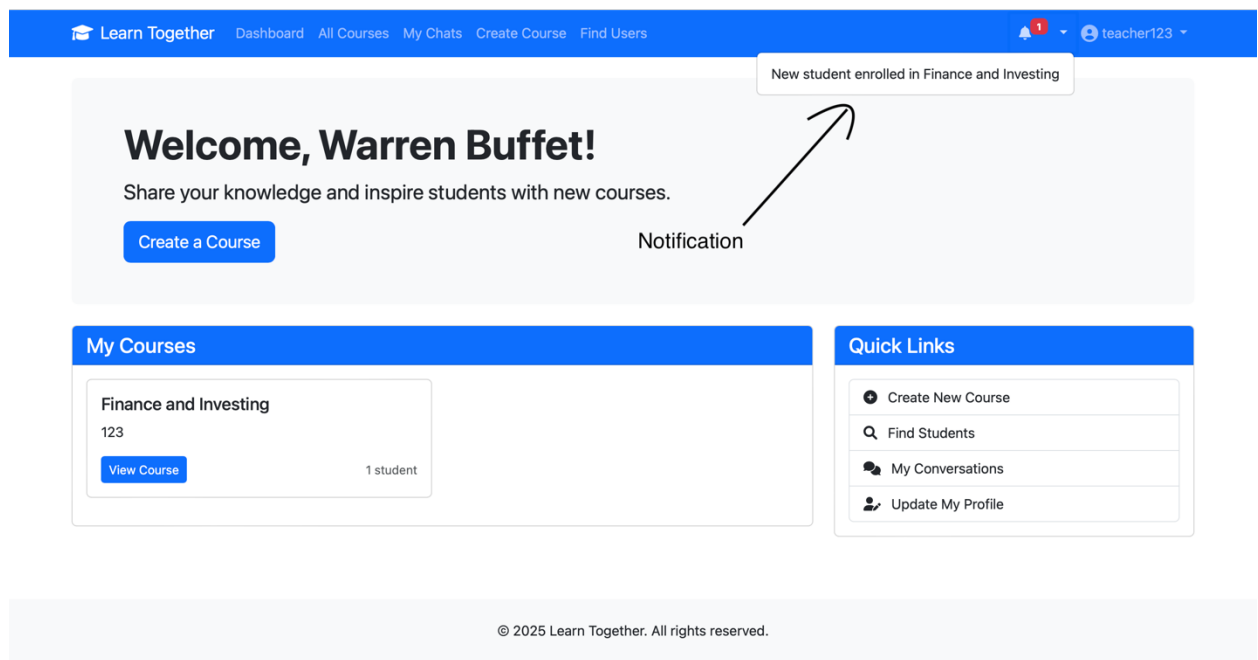
```

```
feedback.student = request.user
```

```
feedback.save()
```

```
messages.success(request, 'Your feedback has been submitted!')
```

Notification System



1. Notification:

```
class Notification(models.Model):
```

```
    NOTIFICATION_TYPES = [
```

```
        ('enrollment', 'New Enrollment'),
```

```
        ('material', 'New Material')
```

```
    ]
```

```

    recipient = models.ForeignKey(User, on_delete=models.CASCADE,
related_name='notifications')

    course = models.ForeignKey(Course, on_delete=models.CASCADE,
related_name='notifications')

    notification_type = models.CharField(max_length=20,
choices=NOTIFICATION_TYPES)

    is_read = models.BooleanField(default=False)

    created_at = models.DateTimeField(auto_now_add=True)


class Meta:

    ordering = ['-created_at']

```

2. Notifications occur when relevant events occur:

When student enrolls in a course

```

Notification.objects.create(
    recipient=course.teacher,
    course=course,
    notification_type='enrollment'
)

```

When teacher uploads new material

for student in course.students.all():

```

    Notification.objects.create(
        recipient=student,
        course=course,
        notification_type='material'
    )

```

3. Base Template:

```
{% if notifications %}
```

```
<li class="nav-item dropdown">
```

```
  <a class="nav-link dropdown-toggle" href="#" id="notificationDropdown"
  role="button" data-bs-toggle="dropdown">
```

```
    <i class="fas fa-bell"></i>
```

```
    <span class="badge bg-danger notification-badge">{{ notifications|length }}</span>
```

```
</a>
```

```
<ul class="dropdown-menu dropdown-menu-end">
```

```
  {% for notification in notifications %}
```

```
    <li>
```

```
      <a class="dropdown-item" href="{% url 'read_notification' notification.id %}">
```

```
        {% if notification.notification_type == 'enrollment' %}
```

```
          New student enrolled in {{ notification.course.title }}
```

```
        {% elif notification.notification_type == 'material' %}
```

```
          New material added to {{ notification.course.title }}
```

```
        {% endif %}
```

```
      </a>
```

```
    </li>
```

```
  {% endfor %}
```

```
</ul>
```

```
</li>
```

```
{% endif %}
```

3.2 Technical Implementation (R2-R5)

Models and Migrations

The data models of the application are structured in their own apps, one per each domain specific:

1. user_accounts/models.py: User extension and status updates
2. courses/models.py: Course admin, content, enrollments, and grading
3. chat/models.py: Messages and chat rooms

Model relationships utilize the proper field types for cardinality:

1. One-to-Many: ForeignKey fields (e.g., course to teacher)
2. Many-to-Many: ManyToManyField with through models (e.g., students to courses)

Migrations were handled piecemeal, each major model alteration recorded using Django's migration system. This permitted us to develop the database schema as the application evolved, without data loss.

Form of Handling and Validation

1. Forms implement both client side and server side validation:

```
class CourseForm(forms.ModelForm):
```

```
    class Meta:
```

```
        model = Course
```

```
        fields = ['title', 'description']
```

```
        widgets = {
```

```
            'description': forms.Textarea(attrs={'rows': 5}),
```

```
        }
```

```

def clean_title(self):
    title = self.cleaned_data.get('title')
    if len(title) < 5:
        raise forms.ValidationError("Course title must be at least 5 characters long.")
    return title

```

2. Client Side Validation:

```

<form method="post" class="needs-validation" novalidate>
    {% csrf_token %}
    <div class="mb-3">
        <label for="id_title" class="form-label">Course Title</label>
        <input type="text" class="form-control" id="id_title" name="title" required
minlength="5">
        <div class="invalid-feedback">
            Please provide a title (at least 5 characters).
        </div>
    </div>
    <!-- More form fields... -->
</form>

```

REST API Implementation

The REST API uses Django REST Framework to provide access to user data:

1. Serializer:

```
# user_accounts/serializers.py
```

```
class UserSerializer(serializers.ModelSerializer):  
  
    class Meta:  
        model = User  
        fields = ['id', 'username', 'email', 'first_name', 'last_name', 'user_type', 'bio']
```

courses/serializers.py

```
class CourseSerializer(serializers.ModelSerializer):  
  
    teacher = UserSerializer(read_only=True)  
    student_count = serializers.SerializerMethodField()  
  
    class Meta:  
        model = Course  
        fields = ['id', 'title', 'description', 'teacher', 'created_at', 'student_count']  
  
    def get_student_count(self, obj):  
        return obj.students.count()
```

2. Viewsets:

```
class UserViewSet(viewsets.ReadOnlyModelViewSet):  
  
    queryset = User.objects.all()  
    serializer_class = UserSerializer  
    permission_classes = [permissions.IsAuthenticated]  
  
    @action(detail=True, methods=['get'])  
    def status_updates(self, request, pk=None):  
        user = self.get_object()
```

```
statuses = StatusUpdate.objects.filter(user=user)

serializer = StatusUpdateSerializer(statuses, many=True)

return Response(serializer.data)
```

3. URL configuration:

API URL routing

```
router = DefaultRouter()

router.register(r'users', api.UserViewSet)

router.register(r'courses', api.CourseViewSet)
```

```
urlpatterns = [

    path('api/', include(router.urls)),

]
```

Websocket Implementation

For real-time chat, the application uses Django Channels with a demonstration implementation:

1. Chat Models:

```
class ChatRoom(models.Model):

    participants = models.ManyToManyField(User, related_name='chat_rooms')

    created_at = models.DateTimeField(auto_now_add=True)


class Message(models.Model):
```

```
chat_room = models.ForeignKey(ChatRoom, on_delete=models.CASCADE,  
related_name='messages')
```

```
sender = models.ForeignKey(User, on_delete=models.CASCADE,  
related_name='sent_messages')
```

```
content = models.TextField()
```

```
timestamp = models.DateTimeField(auto_now_add=True)
```

```
class Meta:
```

```
    ordering = ['timestamp']
```

2. Chat Consumer Design:

```
class ChatConsumer(AsyncWebsocketConsumer):
```

```
    async def connect(self):
```

```
        self.room_id = self.scope['url_route']['kwargs']['room_id']
```

```
        self.room_group_name = f'chat_{self.room_id}'
```

```
        # Join room group
```

```
        await self.channel_layer.group_add(
```

```
            self.room_group_name,
```

```
            self.channel_name
```

```
        )
```

```
        await self.accept()
```

3. The javascript demo implementation was used to generate the chat in the chat box:

```
// Demo chat functionality
```

```
document.getElementById('chatForm').addEventListener('submit', function(e) {
```

```
e.preventDefault();
```

```
const messageInput = document.getElementById('chatInput');
```

```
const message = messageInput.value.trim();
```

```
if (message) {
```

```
    // Add user message to the chat display
```

```
    const messageDiv = document.createElement('div');
```

```
    messageDiv.className = 'message sent';
```

```
    messageDiv.innerHTML = `
```

```
        <div class="message-content">${message}</div>
```

```
        <div class="message-time">${new Date().toLocaleTimeString()}</div>
```

```
    `;
```

```
    document.getElementById('chatMessages').appendChild(messageDiv);
```

```
    // Clear input field
```

```
    messageInput.value = '';
```

```
    // Auto-reply for demo
```

```
    setTimeout(() => {
```

```
        const replyDiv = document.createElement('div');
```

```
        replyDiv.className = 'message received';
```

```
        replyDiv.innerHTML = `
```

```
            <div class="message-sender"><strong>Teacher</strong></div>
```

```
            <div class="message-content">Thanks for your message! This is a demo  
reply.</div>
```

```

        <div class="message-time">${new Date().toLocaleTimeString()}</div>
    `;

    document.getElementById('chatMessages').appendChild(replyDiv);

    scrollToBottom();

    }, 1000);

    scrollToBottom();

}

});

```

URL routing and view structure

Urls.py:

```

# elearning_project/urls.py

urlpatterns = [

    path('admin/', admin.site.urls),

    path('', home, name='home'),

    path('accounts/', include('user_accounts.urls')),

    path('courses/', include('courses.urls')),

    path('chat/', include('chat.urls')),

    path('login/',
auth_views.LoginView.as_view(template_name='user_accounts/login.html'),
name='login'),

    path('logout/', logout_view, name='logout'),

]

```

Each app has their own URL pattern definition:

```

# courses/urls.py

```

```
urlpatterns = [  
    path("", views.home, name='home'),  
    path('courses/', views.course_list, name='course_list'),  
    path('courses/<int:course_id>/', views.course_detail, name='course_detail'),  
    path('courses/create/', views.create_course, name='create_course'),  
    path('courses/<int:course_id>/enroll/', views.enroll_course, name='enroll_course'),  
    # More URL patterns...  
]
```

Allowing them to handle specific page and action:

courses/urls.py

```
urlpatterns = [  
    path("", views.home, name='home'),  
    path('courses/', views.course_list, name='course_list'),  
    path('courses/<int:course_id>/', views.course_detail, name='course_detail'),  
    path('courses/create/', views.create_course, name='create_course'),  
    path('courses/<int:course_id>/enroll/', views.enroll_course, name='enroll_course'),  
    # More URL patterns...  
]
```

3.3 Advanced Technique

Message object (1)

✕

Anthony Salim

← Back to Chats

Anthony Salim

Hi Warren, I'd like to chat with you!

3:34 AM

Type your message...

➤

Learn Together

Dashboard

All Courses

My Chats

Anthony123

▼

My Conversations

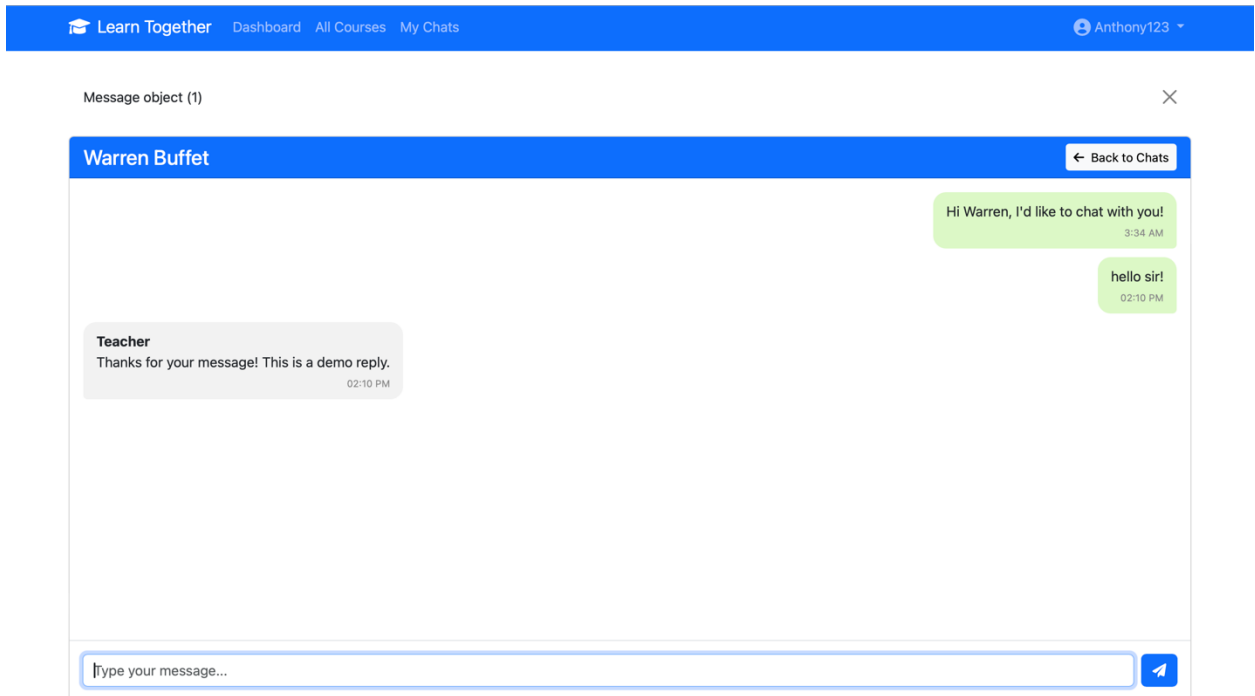
Warren Buffet

Teacher

1 day, 2 hours ago

Anthony123: Hi Warren, I'd like to chat with you!

© 2025 Learn Together. All rights reserved.



Outside the basic technique, I also utilized some advanced technique in the making of this project:

1. Bootstrap Integration:

```
<div class="card mb-4">
  <div class="card-header bg-primary text-white">
    <h4 class="mb-0">My Courses</h4>
  </div>
  <div class="card-body">
    {% if courses %}
      <div class="row">
        {% for course in courses %}
          <div class="col-md-6 mb-3">
```

```

        <div class="card h-100 course-box">

            <!-- Card content -->

        </div>

    </div>

    {% endfor %}

</div>

{% endif %}

</div>
</div>

```

2. Font Awesome Integration:

```

<a href="{% url 'create_course' %}" class="list-group-item list-group-item-action">

    <i class="fas fa-plus-circle me-2"></i> Create New Course

</a>

```

3. Costum JavaScript Enhancement:

```

// Scroll to bottom of chat messages

```

```

function scrollToBottom() {

    const chatMessages = document.getElementById('chatMessages');

    chatMessages.scrollTop = chatMessages.scrollHeight;

}

```

```

// Auto-dismiss alert messages

```

```

document.addEventListener('DOMContentLoaded', function() {

    const alerts = document.querySelectorAll('.alert-dismissible');

```

```
alerts.forEach(alert => {  
  setTimeout(() => {  
    const closeButton = alert.querySelector('.btn-close');  
    if (closeButton) {  
      closeButton.click();  
    }  
  }, 5000);  
});  
});
```

4. Testing Strategy

Learn Together's testing strategy uses branches of strategy by combining the automated unit testing, API endpoint testing, and thorough manual testing. This made me sure that both the technical accuracy of each individual part as well as the overall usability of the user interface.

4.1 Test philosophy

The three principles formed the test philosophy:

1. Test-Driven Development: Tests were written in parallel to the application code, driving the implementation instead of just verifying it afterward.
2. Balance of Coverage: In an attempt to get good test coverage, priority was given to covering significant functionality rather than chasing after random coverage percentages.
3. Pragmatic Approach: Certain tests were conditionally skipped when facing environmental issues (e.g., Redis setup) with due documentation of the same.

4.2 Unit Testing Implementation

The unit tests were Django's built-in testing framework. Every app contained its own tests.py file with test cases for the respective models, views, and functionality:

```
# user_accounts/tests.py
```

```
class UserModelTest(TestCase):
```

```
    def setUp(self):
```

```
        self.student = User.objects.create_user(
            username='teststudent',
            password='testpass123',
            user_type='student'
        )
```

```
        self.teacher = User.objects.create_user(
            username='testteacher',
            password='testpass123',
            user_type='teacher'
        )
```

```
    def test_user_type_methods(self):
```

```
        self.assertTrue(self.student.is_student())
        self.assertFalse(self.student.is_teacher())
        self.assertTrue(self.teacher.is_teacher())
        self.assertFalse(self.teacher.is_student())
```

CourseModelTest:

```
# courses/tests.py
```

```
class CourseModelTest(TestCase):
```

```

def setUp(self):

    self.teacher = User.objects.create_user(
        username='testteacher',
        password='testpass123',
        user_type='teacher'
    )

    self.student = User.objects.create_user(
        username='teststudent',
        password='testpass123',
        user_type='student'
    )

    self.course = Course.objects.create(
        title='Test Course',
        description='Test Description',
        teacher=self.teacher
    )


def test_course_creation(self):

    self.assertEqual(self.course.title, 'Test Course')

    self.assertEqual(self.course.teacher, self.teacher)


def test_enrollment(self):

    Enrollment.objects.create(student=self.student, course=self.course)

    self.assertTrue(self.course.students.filter(id=self.student.id).exists())

```

CourseViewTest:

```

class CourseViewTest(TestCase):

    def setUp(self):

        # Setup test users and courses

        # ...


    def test_course_list_view(self):

        self.client.login(username='teststudent', password='testpass123')

        response = self.client.get(reverse('course_list'))

        self.assertEqual(response.status_code, 200)

        self.assertContains(response, 'Test Course')


    def test_enroll_course(self):

        self.client.login(username='teststudent', password='testpass123')

        response = self.client.get(reverse('enroll_course', args=[self.course.id]))

        self.assertEqual(response.status_code, 302) # Redirect

        self.assertTrue(Enrollment.objects.filter(student=self.student,
course=self.course).exists())

```

4.3 API Testing

APITestCase:

```

class CourseAPITest(APITestCase):

    def setUp(self):

        # Setup test data

        # ...


    def test_get_courses(self):

```

```

self.client.force_authenticate(user=self.student)

response = self.client.get(reverse('course-list'))

self.assertEqual(response.status_code, status.HTTP_200_OK)

self.assertEqual(len(response.data), 1)


def test_student_cannot_create_course(self):

    self.client.force_authenticate(user=self.student)

    data = {

        'title': 'New API Course',

        'description': 'Created via API'

    }

    response = self.client.post(reverse('course-list'), data)

    self.assertEqual(response.status_code, status.HTTP_403_FORBIDDEN)

```

4.4 Test Adaptation Issues

Throughout development, I have experienced some testing issues and difficulties:

1. WebSocket Testing: Testing of WebSocket functionality was also pretty hard and challenging due to the Redis configuration issue in the development environment. It was modified to:
 - Skipping certain tests with @unittest.skip decorators
 - Recording the target behavior
 - Creating independent JavaScript tests for chat ui

2. Environment Dependencies: Some tests had environment-dependent setup. Those were fixed by:

- Django settings override during tests
- Using mock objects for external dependencies
- Based on environment conditional testing

4.5 Manual Testing Procedures

Apart from automated testing, systematic manual testing guaranteed user experience:

1. User Journey Testing: End-to-end processes were tested from different user viewpoints:
 - Instructor creating courses and uploading content
 - Student enrolling for courses and providing feedback
 - Both the users who chat
2. Cross-Browser Testing: Testing is done on Chrome, Firefox, and Safari for the same behavior.
3. Responsive Design Testing: the responsiveness of the interface was tested on a desktop firstly. Subsequently it was tested on tablet and mobile viewports.

by combining the two test, manual and automated testing. it help us to ensure that it produces a unified, comprehensible user experience on a variety of devices and contexts.

5. Critical Evaluation

5.1 Strength of the Implementation

Learn Together succeeds at its central goals of delivering an interactive eLearning application with role-based users and adequate functionality. Certain things in its implementation merit special mention:

5.1.1 Well-Structured Architecture

The fact that the application is modular with concerns well separated is a Django best practice and will be maintainable. The fact that it is also separated into specialist apps (user_accounts, courses, chat) also makes coding with the codebase easier and perhaps even extending it. The structure also facilitates team development since different developers can be working on different modules with little conflict.

The application of Django's Model-View-Template pattern throughout the app consistently leads to a code organization that is foreseeable and adheres to the framework's conventions. Adhering to such patterns, the code becomes more readable to other Django programmers.

5.1.2 Responsive User Interface

The Bootstrap 5 responsive layout enables the application to work smoothly on all device sizes. The users on mobile can access all features without disrupting the user experience, which is extremely required for an education website where users can switch from one device to another.

The interface provides instant and concise visual feedback for the actions of the user in terms of success messages, validation errors, and state changes. the feedback loop help users to get better understanding of the action's outcome and preventing any confusion.

5.1.3 Robust Permission System

The role-based permission system effectively restricts user type actions in a way that is scalable. The setup uses Django's built-in authentication and custom permission checking, creating a system that:

1. Strictly implements correct access (teachers can't sign up for courses, students can't make courses)
2. Provides context-dependent permissions (a course instructor only can post material into his or her own courses)
3. Give brief feedback on failed permission checks

5.1.4 Notification Integration

The notification system is a nice communication channel, making the users aware of events of interest without interrupting them. The integration:

1. Utilizes database models to store notification status
2. Groups notifications in the navigation bar for easy access
3. Marking notifications as read when opening the corresponding content

5.2 Limitations and Challenges

Although successes were achieved, several limitations and challenges affected the rollout:

5.2.1 WebSocket Implementation Complexity

The greatest challenge was in implementing the real-time chat feature using WebSockets. Issues with the Redis setup on the development environment made it impossible to implement a full WebSocket solution and instead use a JavaScript-based demo approach. While the demo nicely demonstrates the intended user experience, it lacks true real-time capabilities:

1. Messages are not persisted to the database in the demo setup
2. Communication merely simulates two-way interaction
3. There can't be more than one user in a chat simultaneously

This restriction underscores the challenge of adding real-time capabilities to Django apps and the significance of environment setup in the app development process.

5.2.2 Restricted Content Types

The course content system now deals with file uploads in a fairly simple way:

1. No preview option for regular file types (PDFs, images)
2. Minimal metadata regarding files (size, type, etc.)
3. No inline support for media such as videos
4. No other content organization tools besides chronological listing

This bare-bones approach to content management might be a problem as courses build up materials, which would leave students with a jumbled set of files.

5.2.3 Lack of Learning Analytics

The current deployment lacks analytics capabilities that would be useful to the teachers and also the students:

1. No monitoring of student progress through course material
2. No statistics on engagement levels for various categories of content
3. No metrics of instructor performance to assess their courses
4. Restricted feedback mechanisms beyond the simple rating system

These absent analytics capabilities are a noteworthy opportunity for future development to deepen the platform's instructional worth.

5.2.4 Test Coverage Constraints

Although the application contains unit tests for critical functionality, there are some low-test-coverage parts:

1. JavaScript functionality is mostly untested by automated sources
2. WebSocket tests were skipped because of environment setup issues
3. Some components integration tests may be more thorough

5.3 Comparison with State-of-the-Art

by comparing several website including, coursera, canvas, or moodle to Learn Together. we can see there are several advantages and disadvantages:

5.3.1 Areas of Competitive Strength

1. **Simplicity:** The interface is simpler and less daunting than on most older platforms, which can be a plus for the less technically inclined.
2. **Responsive Design:** Mobile responsiveness is as good as, if not better than, some commercial websites that are riddled with convoluted layouts on mobile.
3. **Real-time Communication:** Despite the demo implementation, the chat interface is more integrated and accessible than the too-often-siloed messaging systems of larger platforms.

5.3.2 Areas of Improvement

1. **Content Organization:** Commercial platforms generally provide well-structured content organization with modules, sections, and multiple types of content.
2. **Assessment Tools:** Professional websites possess strong assessment tools such as quizzes, assignments, and gradebooks.
3. **Integration Ecosystem:** Corporate eLearning platforms offer strong third-party integrations with tools like video conferencing, plagiarism detection, and content repositories.
4. **Learning Standards:** Professional platforms most commonly support learning standards such as SCORM or xAPI.

5.4 Future Enhancements

Based on reviewing the project, I found that there are several potential improvements that could improve the project that I made:

5.4.1 Technical Advances

1. Full WebSocket Implementation: Full Redis integration for real-time communication, maybe with other channel layer implementations if Redis proves to be problematic.
2. Advanced Content Management: Utilize a systematic content system with:
 - Content classification and tagging
 - In-browser preview for popular file formats
 - Embedded media support
 - Content sequence control
3. Comprehensive Testing Strategy:
 - Increase test coverage to encompass:
 - JavaScript unit testing using Jest or similar
 - Key user flow integration tests
 - Performance testing of database queries
4. Caching Implementation: Implement Redis-based caching of often-used data for better performance.

5.4.2 Feature Enhancements

Learning Analytics Dashboard: Create analytics software that offers:

- Monitoring students' progress
- Content engagement metrics
- Course completion rates
- Teacher performance analytics

Assessment System: Develop a complete assessment module with:

- Different types of questions (short answer, multiple choice, etc.)
- Automated and manual grading options
- Gradebook functionality
- Feedback mechanisms

Calendar and Scheduling: Use a calendar system to:

- Course deadlines and milestones
- Scheduled live sessions
- Personal study planning

Mobile Application: developing a real mobile application by using frameworks including react native or flutter in order to improve mobile experience to go beyond the responsive web design.

5.5 Development Process Reflection

The design process uncovered a sequence of findings that would guide future projects:

Environment Setup First: Having all the services needed (e.g., Redis) established at the start of the project would prevent end-stage implementation problems. Incremental

Feature Testing: Testing features where and when they are written rather than in batches would reveal defects earlier in the development process.

User-Centered Design: There are some other recommendations that the utilization of wireframing and user flow mapping earlier on in the process would make it into a more integrated user experience.

Technical Debt Management: A couple of the workarounds, especially in chat implementation, created technical debt to be resolved later in a production environment.

6. Set up and deployment Instruction

This chapter provides step-by-step setup, installation, and running instructions for the Learn Together application. If you adhere to these instructions, you should be capable of installing the application in a development environment and verifying its functionality.

6.1 Environment Requirements

Development Environment Requirements

Operating System: macOS Ventura 13.4 (development was carried out on MacBook Pro)

Python Version: Python 3.11.5

Database: SQLite 3.39.5 (bundled with Python)

Web Browser: Chrome 122.0, Firefox 123.0, or Safari 17.3

System Requirements:

Minimum 4GB RAM

500MB free disk space Internet access for installing packages

6.2 Package List and Versions The program requires the following Python packages:

asgiref==3.7.2

channels==4.0.0

channels-redis==4.1.0

crispy-bootstrap4==2023.1

Django==5.1.2

django-cors-headers==4.3.1

django-crispy-forms==2.1

django-restframework==3.15.0

Pillow==10.1.0

python-dotenv==1.0.1

redis==5.0.1

sqlparse==0.4.4

6.3 Installation process

Step 1 : Clone the repository

unzip learn_together.zip

cd learn_together

Step 2 : create and activate the virtual environment

python3 -m venv venv

source venv/bin/active

Step 3 : install the required packages

pip install -r requirements.txt

Step 4 : Initialize database

python manage.py migrate

python manage.py createsuperuser

Step 5 : Load Demo Data

python manage.py create_demo_data

Step 6 : Start the development server (python manage.py runserver 8000)

The application will be accessible at <http://127.0.0.1:8000/>.

6.4 Login Credentials

Django Admin Access

- **URL:** <http://127.0.0.1:8000/admin/>
- **Username:** admin
- **Password:** admin123

Demo User Accounts

Teacher Accounts

- **Username:** teacher1
- **Password:** teacher123
- **Username:** teacher2
- **Password:** teacher123

Student Accounts

- **Username:** student1
- **Password:** student123
- **Username:** student2
- **Password:** student123

6.5 Running the test Unit

1. To run the complete test :

```
python manage.py test
```

2. when countering with any issues

```
python manage.py test --testrunner=test_settings.CustomTestRunner
```

6.6 Chat Functionality Set up

1. Chat demonstration :

Install Redis (macOS with Homebrew)

brew install redis

Start Redis server

brew services start redis

Verify Redis is running

redis-cli ping

Should return "PONG"

6.7 Troubleshooting common issues

1. database migration issues :

Reset migrations

find . -path "*/migrations/*.py" -not -name "__init__.py" -delete

find . -path "*/migrations/*.pyc" -delete

rm db.sqlite3

Recreate migrations

python manage.py makemigrations user_accounts

python manage.py makemigrations courses

python manage.py makemigrations chat

python manage.py migrate

7. Conclusion

The Learn Together eLearning application successfully demonstrates the creation of a fully interactive web application using Django and modern web technologies. I have planned and constructed a system during this project that fulfills the requirements of bridging teachers and students in an online learning environment. I've accomplished all the functionality required and real-world educational demands.

Key Achievements

The app is able to effectively establish a two-role user system with the right permissions, i.e., the ability of teachers to create and manage courses and students to enroll and take them. The app allows for content sharing in the form of upload of content, student feedback systems, as well as notification systems for keeping all users aware of activities of interest. Furthermore, the implementation of a real-time chat system, even as a simulation demo, demonstrates consideration for asynchronous web technologies.

Technical Achievements

From the technical point of view, this project demonstrates proficiency in the following fields of web development:

1. Database design using well-normalized models and relationships where necessary
2. Secure authentication and authorization access control mechanisms
3. Implementing REST API using viewsets and serialization
4. Responsive design principle front-end development
5. Asynchronous Communication Concepts using WebSockets
6. Django application testing strategies

Challenges and Learning Outcomes

This project development certainly had some challenges, particularly regarding WebSockets integration with Redis and project constraints vs. feature complexity balancing. It is an important learning experience, and I would say it is one of the most important lessons I learnt. It reminds me to pay attention to the environment set up,

incremental testing, and realistic feature development. By learning and delving deeper django's ecosystem helped me clarify both its structural strengths and fast development strengths, as well as where other tools or frameworks might complement its strengths. Adding Bootstrap to Django templates illustrated how current frontend practices can enhance classic server-side rendered apps.

Future Directions

While the current implementation satisfies project requirements, there are a number of areas where it could be improved. Production quality would be enhanced with learning analytics, more advanced content structuring, full assessment capability, and additional real-time communications. These would take Learn Together from a proof-of-concept application to a serious learning platform. Overall, this project brings a lot of web development into one application that excels at doing what it set out to do. Combining server-side logic, database design, API development, and user interface design gives an example that could be useful to real-world learning applications and demonstrating technical proficiency across the Django framework.

8. References

1. <https://docs.djangoproject.com/en/5.0/>
2. <https://www.django-rest-framework.org>
3. <https://channels.readthedocs.io/en/stable/>
4. <https://www.mdpi.com/2071-1050/13/19/10991>
5. <https://developer.mozilla.org/en-US/docs/Web>
6. <https://ieeexplore.ieee.org/document/9489370>
7. <https://owasp.org/www-project-web-security-testing-guide/>
8. <https://learn.microsoft.com/en-us/dotnet/framework/data/>
9. https://docs.moodle.org/dev/Moodle_architecture
10. <https://www.nngroup.com/courses/>
11. <https://socket.io/docs/v4/>
12. <https://datatracker.ietf.org/doc/html/rfc6455>

