STUDENTS' UNION
TECHNICAL TEAM

Back-End Developer Recruitments

TASK 3

StudyDeck Forum

DEADLINE

5 Jan 2026

Overview

StudyDeck is the premier academic platform maintained by the Students' Union Technical Team. It serves the student body by providing easy access to lecture videos, notes, PYQs, and course handouts.
You are required to build a **community-driven Forum Module** for this application using **Django**. The goal is to create a space where students can discuss courses, ask doubts, and share resources efficiently using Django's built-in templating engine.
You are allowed to use pre-built components (Bootstrap, Tailwind Components, etc.) or templates you find online for the frontend part.
Here's an example of a community forum and a thread

Task Scope

Guidelines :

The task is divided into three phases:

- **Phase 0: Base Architecture** - Setup the foundational models and auth.
- **Phase 1: Core Forum Implementation** - Building the discussion logic, permissions and interactivity.
- **Phase 2: Deployment** - Hosting the application.

Technical Stack Requirements :

- Framework: Django (Use standard Django Views and Templates).
- Database: SQLite is acceptable, but PostgreSQL is preferred.
- Deployment: Public URL required (PythonAnywhere, Render, AWS, etc.).
- Submission: GitHub Repository link + Deployed Website URL.

Phase 0: Base Architecture

Before building the forum, you must establish the ”Base Models” that represent the existing data on StudyDeck.

1.1 Authentication & User Profiles

Implement a robust user system. The forum relies on knowing who is posting content.

- **Registration & Login**: Allow users to login using BITS Email only (Google OAuth). You are required to use Django’s built-in authentication system in conjunction with Django AllAuth for facilitating Google login. This article and this one will help you in the process.
- **User Metadata**: Extract basic details such as the user’s Full Name, Image and Mail.

Phase 0: Base Architecture

1.2 Foundation Models

Create basic courses and resources. You do not need to build pages for them, but they must exist in the database to be referenced later. Write a script to populate these models.

1. **Course**
   - `Code`: e.g., "CS F111"
   - `Title`: e.g., "Computer Programming"
   - `Department`: e.g., "Computer Science"
2. **Resource**
   - `Title`: Name of the file.
   - `Type`: PDF, Video, or Link.
   - `Link`: A URL field pointing to the content.

Phase 1: Core Forum Features

In this phase, you will build the heart of the module.

2.1 User Roles & Permissions

The system must distinguish between regular users and administrative staff.

- **Student User**: The default role. Students can create threads, reply to discussions, and like posts. They can only edit or delete their own content.
- **Moderator**: A privileged role. Moderators must have the ability to delete *any* post, lock threads to prevent further replies, and resolve reported content.

2.2 Categories

Discussions should be organized into specific topics to keep the forum clean.

- Create a bunch of **Categories** (e.g., ”General Queries”, ”Exam Prep”).
- Each category should have a name and a unique slug for URL generation.
- Threads must belong to a specific category.

Phase 1: Core Forum Features

2.3 Threads (Discussions)

A Thread is the starting point of a conversation.

- **Core Fields**: Title, Content, Author, and Created Timestamp.
- **Contextual Linking**: Tag Courses or Documents
- **Pagination**: Ensure your Views paginate threads (e.g., 10 per page)

Phase 1: Core Forum Features

3.1 Replies (Posts)

Threads contain replies from the community.

- Permissions: Enforce proper user permissions
- Moderation: Moderators bypass ownership checks and can delete any reply.
- Soft Delete (Bonus): Instead of permanently deleting a row from the database, add an is_deleted boolean field.

3.2 Interactivity

Make the platform engaging.

- Upvotes/Likes: Users should be able to ”Like” a thread or a post.
- Tags: Implement a Tagging system. This allows flexible filtering (e.g., #midsem, #quiz1, #urgent).

Phase 1: Core Forum Features

3.3 Reporting System

- Reasoning: Include a text field for the user to explain why they are reporting the content.
- Resolution: Add a status field (Pending, Resolved). Provide a View for Moderators to see all pending reports and update their status.

Phase 2: Deployment

4.1 Deployment (Mandatory)

Code residing only on localhost will not be evaluated. You must prove your application works in a production environment.

- Deploy your backend to a public URL using services like **Render, PythonAnywhere, AWS, Azure,** etc.

- **Documentation**: Your repository must include a README.md containing:

1. Setup Instructions (Virtualenv, pip install, migrations).
2. Feature Walkthrough (How to use the platform).
3. Design Decisions (Why you chose specific database structures).

Phase 2: Brownie

4.2 Bonus Challenges

- Use **PostgreSQL** instead of SQLite.
- Implement **Markdown Support** for rendering rich text in posts.
- Add **Rate Limiting** to specific forms to prevent spam.
- Create an email **Notification** System for replies, mentions, thread changes.
- Implement **Fuzzy Search** for finding threads by title similarity.
- **Sort** threads and replies by popular/latest

Resources

- Starting with Python
- Git Tutorial
- Corey Schafer Django Tutorial

If you're comfortable with reading docs, congrats!! here are some excellent docs to start off with Django

- Django Polls App(Getting Started)
- Django MDN Docs
- Django Girls Blog

NOTE: The suggested model structure is only a suggestion, feel free to make changes as and when necessary

HAPPY CODING!