Final Term eLearning Project - Strategic Implementation Plan

Project Overview

Course: CM3035 - Advanced Web Development
Project: eLearning Application with Real-time Features
Weight: 50% of total module mark

Critical Requirements Analysis

R1 - Core Functionality (HIGHEST PRIORITY - Must Complete All)

User Authentication & Management

- [ ] User registration with password security
- [ ] Login/logout functionality
- [ ] Two user types: Students and Teachers with different permissions
- [ ] User profiles with comprehensive info (username, real name, photo, etc.)
- [ ] Home pages for each user (discoverable and visible to other users)
- [ ] Status updates for students on their home page
- [ ] Teacher search functionality for students and other teachers

Course Management System

- [ ] Teachers can create new courses
- [ ] Teachers can upload course materials (images, PDFs, etc.)
- [ ] Teachers can view their courses and see enrolled students
- [ ] Students can view list of available courses
- [ ] Students can enroll in courses of their choice
- [ ] Students can leave feedback for courses
- [ ] Teachers can remove/block students from courses

Real-time Communication (CRITICAL - Explicitly Required)

- [ ] WebSockets implementation using Django Channels
- [ ] Real-time chat between students and teachers
- [ ] Alternative: Shared whiteboard, audio streaming, or file transfers

Notification System

- [ ] Teachers notified when students enroll in their courses
- [ ] Students notified when new course materials are added

R2 - Technical Implementation (HIGH PRIORITY)

- [ ] Correct use of Django models and migrations

- [ ] Proper forms, validators, and serialization
- [ ] Django REST Framework implementation
- [ ] Appropriate URL routing
- [ ] Comprehensive unit testing

R3-R5 - Architecture Requirements (HIGH PRIORITY)

- [ ] Appropriate database model design with proper relationships
- [ ] REST interface for user data access
- [ ] Server-side code testing

Strategic Implementation Phases

Phase 1: Foundation (Week 1)

Goal: Establish core user system and authentication

Tasks:

1. Set up Django project with proper structure
2. Create User models (Student/Teacher with different permissions)
3. Implement user registration and authentication
4. Create user profiles and home pages
5. Implement basic user search for teachers

Deliverables:

- Working user registration/login system
- User profiles with basic information
- Teacher search functionality
- Basic home pages for users

Phase 2: Course Management (Week 2)

Goal: Complete course creation and enrollment system

Tasks:

1. Create Course model with proper relationships
2. Implement course CRUD operations for teachers
3. Set up file upload system for course materials
4. Create student enrollment system
5. Implement course feedback system
6. Add student blocking/removal functionality

Deliverables:

- Teachers can create and manage courses
- File upload for course materials
- Student enrollment system
- Course feedback system

- Student management for teachers

Phase 3: Real-time Features (Week 3)

Goal: Implement WebSockets and real-time communication

Tasks:

1. Set up Django Channels and Redis
2. Implement real-time chat system
3. Create notification system
4. Test real-time functionality

Deliverables:

- Working WebSockets implementation
- Real-time chat between users
- Notification system for enrollments and new materials

Phase 4: API & Testing (Week 4)

Goal: Complete REST API and comprehensive testing

Tasks:

1. Implement REST API endpoints for user data
2. Write comprehensive unit tests
3. Create documentation
4. Prepare deployment and demo materials

Deliverables:

- Complete REST API
- Unit test suite
- Project documentation
- Deployment-ready application

Success Factors for Maximum Marks

Critical Success Factors

1. **WebSockets MUST work** - This is explicitly required and heavily weighted
2. **All R1 requirements functional** - Core features are the priority
3. **Proper database design** - Normalized and well-structured models
4. **Comprehensive testing** - Unit tests for all major functionality
5. **Clear documentation** - Setup instructions, login credentials, test instructions

What NOT to Do (Time Wasters)

- ❌ Over-engineer the UI - Focus on functionality over aesthetics
- ❌ Add features not in requirements - Stick to the specification

- ❌ Spend excessive time on styling - Functional is better than pretty but incomplete
- ❌ Build complex features without completing basics first

Technical Stack

Core Technologies

- **Django** - Main web framework
- **Django REST Framework** - API development
- **Django Channels** - WebSockets for real-time features
- **Redis** - Backend for WebSockets
- **SQLite/PostgreSQL** - Database
- **Pillow** - Image handling for user photos

Key Dependencies

```
Django>=4.2.0
djangorestframework>=3.14.0
channels>=4.0.0
channels-redis>=4.1.0
Pillow>=10.0.0
django-crispy-forms>=2.0
```

Database Design Overview

Core Models

1. **User** (Abstract base)
    - Student (inherits from User)
    - Teacher (inherits from User)
2. **Course**
    - Title, description, teacher (FK)
    - Materials (FileField)
    - Students (ManyToMany)
3. **CourseEnrollment**
    - Student, Course, enrollment_date
4. **CourseFeedback**
    - Student, Course, feedback_text, rating
5. **StatusUpdate**
    - User, content, timestamp

6. **ChatMessage**
    - Sender, receiver, content, timestamp

Testing Strategy

Unit Tests Required

- [ ] User registration and authentication
- [ ] Course creation and management
- [ ] Student enrollment
- [ ] File upload functionality
- [ ] Real-time chat functionality
- [ ] Notification system
- [ ] REST API endpoints

Test Coverage Goals

- Minimum 80% code coverage
- All critical user flows tested
- API endpoints fully tested
- WebSocket functionality tested

Deliverables Checklist

D1: Django Application

- [ ] Complete eLearning application
- [ ] Demo users (students and teachers)
- [ ] All R1 requirements implemented
- [ ] WebSockets functionality working
- [ ] File upload system functional

D2: Report (4000-6000 words)

- [ ] Application description and design reasoning
- [ ] How requirements R1-R5 are met
- [ ] Code organization and logic explanation
- [ ] Critical evaluation of application
- [ ] Unit test running instructions
- [ ] Installation and setup instructions
- [ ] Package versions and development environment
- [ ] Login credentials for admin, teachers, and students

D3: Video Demo (≤10 minutes)

- [ ] App installation using requirements.txt
- [ ] Database design discussion
- [ ] Unit test execution
- [ ] App launch and login demonstration

- [ ] Course enrollment and feedback features
- [ ] Real-time chat demonstration
- [ ] Redis server setup and second user login

D4: Bonus - Deployment (Optional)

- [ ] AWS/Digital Ocean deployment
- [ ] App URL and login details in report

Risk Mitigation

High-Risk Areas

1. **WebSockets Implementation** - Start early, use Django Channels documentation
2. **File Uploads** - Test thoroughly with different file types
3. **Real-time Features** - Ensure Redis is properly configured
4. **User Permissions** - Implement proper role-based access control

Contingency Plans

- If WebSockets prove difficult, focus on basic chat functionality first
- If file uploads fail, use simple text-based materials initially
- If time runs short, prioritize R1 requirements over R2-R5

Weekly Milestones

Week 1 Milestone

- [ ] User authentication system working
- [ ] Basic user profiles functional
- [ ] Teacher search implemented

Week 2 Milestone

- [ ] Course creation and management working
- [ ] Student enrollment system functional
- [ ] File upload system operational

Week 3 Milestone

- [ ] WebSockets chat working
- [ ] Notification system implemented
- [ ] Real-time features tested

Week 4 Milestone

- [ ] REST API complete
- [ ] All tests passing
- [ ] Documentation finished
- [ ] Video demo recorded

Success Metrics

Functional Requirements

- [ ] All 12 R1 requirements implemented and working
- [ ] WebSockets functionality demonstrated
- [ ] File upload system operational
- [ ] User permissions working correctly

Technical Requirements

- [ ] Proper Django project structure
- [ ] Clean, commented code following PEP8
- [ ] Comprehensive unit test suite
- [ ] REST API with proper serialization

Documentation Requirements

- [ ] Clear setup instructions
- [ ] Login credentials provided
- [ ] Test running instructions
- [ ] Video demonstration of all features

Remember: Focus on completing ALL R1 requirements first. A fully functional basic system is better than an incomplete advanced system. WebSockets implementation is critical for high marks.