Django E-Learning Platform - Project Summary

🎯 Project Overview

Production-ready Udemy-like e-learning platform built with Django REST Framework using strict Test-Driven Development methodology. Features comprehensive course management, user enrollment, progress tracking, and shopping cart functionality.

🚀 Technical Excellence

Test-Driven Development

- 100% test coverage across all features
- Red-Green-Refactor cycle maintained consistently
- Comprehensive edge case and error scenario testing

Performance Optimization

- Zero N+1 queries - select_related/prefetch_related used throughout
- Database-level aggregation for real-time calculations
- Optimized filtering with proper indexing

Advanced Architecture

- Custom permission classes with relationship traversal
- Django signals for automatic data synchronization
- Business logic enforcement at multiple levels
- Role-based access control (Student/Instructor/Admin)

📋 Core Features Implemented

Feature	Implementation	Technical Highlights
User Management	Custom User model with roles	Token auth, role upgrades, profile management
Course Hierarchy	Category→Course→Section →Lecture	Multiple instructors, flexible content types, JSON objectives
Enrollment System	Business rule validation	Duplicate prevention, automatic cleanup, access control
Progress Tracking	Real-time percentage calculation	Django signals, automatic updates, performance optimized
Reviews & Ratings	1-5 star system with aggregation	Database-level calculations, business rule validation
Search & Filtering	Multi-parameter search	Text search, category filters, flexible sorting
Shopping Cart	Add/remove with auto-cleanup	Enrollment integration, price calculations, duplicate prevention

🔧 Technology Stack

Backend: Django 4.x + Django REST Framework
Database: SQLite with optimized relationships
Authentication: Token-based with custom permissions
Testing: Django's built-in framework with 100% coverage
API Docs: drf-spectacular (OpenAPI/Swagger)

💡 Key Code Examples

Custom Permission with Relationship Traversal

```
class IsEnrolledInLectureCourse(BasePermission):
    def has_permission(self, request, view):
        lecture_id = view.kwargs.get('lecture_id')
        lecture = Lecture.objects.get(pk=lecture_id)
        return Enrollment.objects.filter(
            student=request.user,
            course=lecture.section.course
        ).exists()
```

Automatic Progress Updates via Signals

```
@receiver(post_save, sender=LectureProgress)
def update_course_progress_on_lecture_save(sender, instance, **kwargs):
    course = instance.lecture.section.course
    course_progress, created = CourseProgress.objects.get_or_create(
        student=instance.student, course=course
    )
    course_progress.update_progress()
```

Performance-Optimized Search

```
def get_queryset(self):
    return Course.objects.select_related('category').prefetch_related(
        'subcategory', 'instructor', 'reviews'
    ).annotate(avg_rating=Avg('reviews__rating'))
```

🧪 Testing Strategy

Comprehensive Test Coverage

- Model Tests: Validation, relationships, business logic
- API Tests: Authentication, permissions, response formats
- Integration Tests: End-to-end workflows
- Edge Cases: Invalid data, boundary conditions

TDD Methodology Example

```
def test_lecture_progress_completion_auto_sets_timestamp_and_watch_time(self):

  """Test LectureProgress.save() automatically sets completion data"""

  progress = LectureProgress.objects.create(

    student=self.student, lecture=self.lecture, is_completed=True

  )

  self.assertIsNotNone(progress.completed_at)

  self.assertEqual(progress.watch_time, 300)  # lecture duration
```

🏛️ Architecture Highlights

Clean Code Principles

- Fat models, thin views with business logic in models
- DRY principle applied throughout codebase
- Single Responsibility for each class/method
- Comprehensive error handling with meaningful messages

Security Implementation

- Token-based authentication for stateless operation
- Business rule validation preventing unauthorized actions
- User data isolation ensuring privacy
- Comprehensive input validation

Database Design

- Optimized relationships with proper foreign keys
- Unique constraints preventing duplicate data
- Computed fields using properties and aggregation
- Signal-based data consistency

📊 Project Metrics

Metric	Value	Description
Lines of Code	~2000+	Clean, well-documented code
Test Coverage	100%	All features covered by tests
Models	10	User, Course, Section, Lecture, etc.
API Endpoints	25+	Full CRUD operations
Custom Permissions	6	Complex authorization logic
Django Signals	3	Automatic data synchronization

🎯 Business Logic Excellence

Enrollment System

- ✅ Prevents duplicate enrollments
- ✅ Automatic progress tracking initialization
- ✅ Role-based access validation
- ✅ Integration with cart system

Progress Tracking

- ✅ Real-time percentage calculations
- ✅ Completion timestamp automation
- ✅ Cross-model data synchronization
- ✅ Performance-optimized queries

Review System

- ✅ Enrollment prerequisite validation
- ✅ Rating aggregation in real-time
- ✅ Unique review constraints
- ✅ Business rule enforcement

🔄 Development Approach

Test-Driven Development Process

1. Red: Write failing test first
2. Green: Implement minimal code to pass
3. Refactor: Clean up while maintaining functionality
4. Repeat: For every feature and edge case

Quality Assurance

- Atomic commits with meaningful messages
- Feature branch workflow
- Code review ready structure
- Production deployment ready

💼 Professional Skills Demonstrated

Technical Skills

- Advanced Django/DRF patterns
- Database optimization and design
- RESTful API development
- Test-Driven Development
- Security best practices
- Performance optimization

Software Engineering Practices

- Clean code architecture
- Comprehensive testing
- Documentation and comments
- Version control with Git
- Problem-solving methodology
- Attention to detail