📚 LMS Backend Functionality Breakdown

Author: Gift Ndlala

Scope: Instructor & Student Portals – Key Functionality, Implementation, and Logic

🎓 Instructor Portal – Functionality & Implementation

1. 📘 Course Management

```
class CourseViewSet(viewsets.ModelViewSet):

    def perform_create(self, serializer):
        serializer.save(instructor=self.request.user)

    @action(detail=True, methods=['get'])
    def students(self, request, pk=None):
        course = self.get_object()
        enrollments = CourseEnrollment.objects.filter(course=course)
        return Response(CourseEnrollmentSerializer(enrollments,
many=True).data)

    @action(detail=True, methods=['get'])
    def progress(self, request, pk=None):
        progress = StudentProgress.objects.filter(
            student__course_enrollments__course=course
        )
        return Response(StudentProgressSerializer(progress,
many=True).data)
```

☑ *Explanation:*

- **Create Course**: Ensures the instructor creating the course is automatically assigned as the owner.
- **List Students**: Retrieves all students enrolled in the specified course using the course ID (pk).
- **Track Progress**: Fetches performance records of all students linked to this course.

2. 📚 Module Management

```
class ModuleViewSet(viewsets.ModelViewSet):

    def perform_create(self, serializer):
        course = serializer.validated_data['course']
        if not course.instructor == self.request.user:
            raise permissions.PermissionDenied()
        serializer.save()

    @action(detail=True, methods=['get'])
    def students(self, request, pk=None):
        enrollments = CourseEnrollment.objects.filter(
            course=module.course,
            status='enrolled'
        )
        return Response(CourseEnrollmentSerializer(enrollments,
many=True).data)
```

☑ *Explanation:*

- **Permission Check**: Ensures instructors can only add modules to their own courses.
- **Student List by Module**: Returns enrolled students in a specific module context.

3. 📝 Assessment Management

```
class AssessmentViewSet(viewsets.ModelViewSet):

    def perform_create(self, serializer):
        serializer.save(instructor=self.request.user)

    def get_queryset(self):
        return Assessment.objects.filter(instructor=self.request.user)
```

✅ *Explanation:*

- Instructors create assessments tied to themselves.
- Returns assessments scoped to the authenticated instructor.

4. 📄 Assignment Management

```
class AssignmentViewSet(viewsets.ModelViewSet):

    def perform_create(self, serializer):
        serializer.save()

    def get_queryset(self):
        return Assignment.objects.filter(
            module__course__instructor=self.request.user
        )
```

✅ *Explanation:*

- Creates assignments under the instructor’s course modules.
- Fetches only those relevant to the instructor's managed modules.

🎒 Student Portal – Functionality & Implementation

1. 📘 Module Access

```
class StudentModuleViewSet(viewsets.ReadOnlyModelViewSet):

    def get_queryset(self):
        return Module.objects.filter(
            course__enrollments__student__user=self.request.user,
            course__enrollments__status='enrolled',
            is_active=True
        )
```

✅ *Explanation:*

- Shows students only the modules they are officially enrolled in.
- Includes checks for `enrolled` status and `active` modules only.

2. 📤 Assignment Submission

```
class StudentAssignmentViewSet(viewsets.ReadOnlyModelViewSet):

    @action(detail=True, methods=['post'])
    def submit(self, request, pk=None):
        assignment = self.get_object()
        if assignment.due_date < timezone.now():
            return Response(
                {'error': 'Assignment submission deadline has
passed'},
                status=status.HTTP_400_BAD_REQUEST
            )

        submission = AssignmentSubmission.objects.create(
```

```
            assignment=assignment,
            student=request.user.student,
            submission_file=request.data.get('file'),
            comments=request.data.get('comments', ''),
            status='submitted'
        )
        return
Response(AssignmentSubmissionSerializer(submission).data)

    @action(detail=True)
    def my_submission(self, request, pk=None):
        submission = get_object_or_404(
            AssignmentSubmission,
            assignment=assignment,
            student__user=request.user
        )
        return
Response(AssignmentSubmissionSerializer(submission).data)
```

Explanation:

- Submits assignments (with deadline checks).
- View previously submitted work for a specific assignment.

3. Assessment Answer Submission

```
class StudentAnswerViewSet(viewsets.ModelViewSet):

    def perform_create(self, serializer):
        assignment_id = self.kwargs.get('assignment_pk')
        assignment = get_object_or_404(AssessmentAssignment,
id=assignment_id)
        serializer.save(assessment_assignment=assignment)
```

Explanation:

- Handles submission of assessment answers per student.
- Relates answers to a particular assessment.

4. Progress Tracking

```
class StudentProgressViewSet(viewsets.ModelViewSet):

    def get_queryset(self):
        return
StudentProgress.objects.filter(student__user=self.request.user)
```

Explanation:

- Students can access their learning activity and completion status.

Key Supporting Utility Functions

Assignment Creation

```
def create_assignment(self, module_id, title, description, due_date,
points):
    module = get_object_or_404(Module, id=module_id)
    if not module.course.instructor == self.request.user:
        raise PermissionDenied()

    assignment = Assignment.objects.create(
        module=module,
        title=title,
        description=description,
```

```
        due_date=due_date,
        points=points
    )
    return assignment
```

☑ Assignment Submission (Manual Logic)

```
def submit_assignment(self, assignment_id, file, comments):
    assignment = get_object_or_404(Assignment, id=assignment_id)
    if assignment.due_date < timezone.now():
        raise ValidationError("Submission deadline has passed")

    submission = AssignmentSubmission.objects.create(
        assignment=assignment,
        student=self.request.user.student,
        submission_file=file,
        comments=comments,
        status='submitted'
    )
    return submission
```

☑ Assessment Creation

```
def create_assessment(self, module_id, title, description,
assessment_type, total_marks, passing_marks):
    module = get_object_or_404(Module, id=module_id)
    assessment = Assessment.objects.create(
        module=module,
        title=title,
        description=description,
        assessment_type=assessment_type,
        total_marks=total_marks,
        passing_marks=passing_marks,
        instructor=self.request.user
    )
    return assessment
```

☑️ Student Progress Tracker

```
def track_progress(self, student_id, module_id, lesson_id):
    progress = StudentProgress.objects.create(
        student_id=student_id,
        module_id=module_id,
        lesson_id=lesson_id,
        completed=False,
        last_accessed=timezone.now()
    )
    return progress
```

☑️ Grading Submissions

```
def grade_assessment(self, assessment_id, student_id, marks,
feedback):
    assessment = get_object_or_404(Assessment, id=assessment_id)
    student = get_object_or_404(Student, id=student_id)

    grade = AssessmentGrade.objects.create(
        assessment=assessment,
        student=student,
        marks_obtained=marks,
        feedback=feedback,
        graded_by=self.request.user
    )
    return grade
```

🔍 Key Design Observations

- ✅ Role-based access control is well implemented (`self.request.user` checks).
- ✅ Viewsets follow DRF best practices with `perform_create`, `@action`, and queryset filtering.
- ⚠️ API security could be further enhanced with:

    - Throttling
    - Input sanitization
    - File upload validation (e.g., antivirus or file type check)

🧠 Suggested Additions

- **PDF Download Handling**: Ensure static/media routing is correctly configured.
- **Notification Logic**: Real-time alert logic for new announcements or feedback.
- **eWallet Transactions**: Instructor-managed balance updates.
- **Recent Activities Widget**: Dashboard-level tracking of logins, submissions, messages.