🎯Project Overview

An online class platform with role-based access: Superuser manages teachers; teachers manage courses, materials, assignments, and students; students enroll and learn. A real-time Attention Model (`model.h5`) estimates student attentiveness/distractedness per course/session for teacher dashboards.

Core Domain Model (ERD Summary)

- **User** *(custom auth)* —< 1:1 >— **TeacherProfile** / **StudentProfile**
- **Course** —< many-to-one >— **TeacherProfile** (owner)
- **Enrollment** —< many-to-one >— **Course**, **StudentProfile** (unique pair)
- **Module** —< many-to-one >— **Course**
- **CourseMaterial** —< many-to-one >— **Module** (or Course if no modules)
- **Assignment** —< many-to-one >— **Course**
- **Submission** —< many-to-one >— **Assignment**, **StudentProfile** (unique pair)
- **ClassSession** —< many-to-one >— **Course** (live/recorded learning session)
- **AttentionRecord** —< many-to-one >— **StudentProfile**, **Course**, **ClassSession**
- **Grade** —< many-to-one >— **Submission** (optional if you want separate table)
- **Announcement** —< many-to-one >— **Course**
- **Notification** —< many-to-one >— **User** (optional)

Optional tables you can skip initially: `Quiz`, `Question`, `Choice`, `Attempt`, `Payment`, `Comment`.

🏹Database Schema (Django/SQL Outline)

1) Auth & Profiles

User (custom `AbstractUser` with `role`) - `id` (PK) - `username`, `email` (unique), `password` - `first_name`, `last_name` - `role` (enum: `SUPERADMIN`, `TEACHER`, `STUDENT`) - `date_joined`, `last_login`, `is_active`, `is_staff`, `is_superuser`

TeacherProfile - `id` (PK), `user` (OneToOne -> User) - `bio`, `title`, `avatar`, `phone`

StudentProfile - `id` (PK), `user` (OneToOne -> User) - `roll_no`, `avatar`, `guardian_email`

2) Courses & Content

Course - `id` (PK), `title`, `slug` (unique), `description` - `owner` (FK -> TeacherProfile) - `thumbnail`, `category`, `level` (`BEGINNER|INTERMEDIATE|ADVANCED`) - `published` (bool), `created_at`, `updated_at`

Module *(optional but recommended)* - `id`, `course` (FK), `title`, `order`

CourseMaterial - `id`, `module` (FK, null=True, course fallback) - `title`, `material_type` (`VIDEO|PDF|NOTE|LINK`) - `file` (for PDF/notes) or `video_url` (HLS/VOD) or `external_url` - `duration_seconds` (nullable), `order`, `is_free_preview` (bool) - `created_at`

3) Enrollment & Learning Flow

Enrollment *(unique together: student, course)* - `id`, `student` (FK -> StudentProfile), `course` (FK) - `status` (`ACTIVE|COMPLETED|DROPPED`) - `enrolled_at`, `completed_at` (nullable)

ClassSession *(a discrete class period/live stream/recording)* - `id`, `course` (FK), `title` - `session_type` (`LIVE|RECORDED`) - `start_time`, `end_time` (nullable for recorded) - `recording_url` (nullable)

4) Assignments & Submissions

Assignment - `id`, `course` (FK), `title`, `description` - `due_at`, `max_score` (default: 100), `attachment` (nullable) - `created_at`

Submission *(unique together: assignment, student)* - `id`, `assignment` (FK), `student` (FK -> StudentProfile) - `submitted_at`, `file`, `text_answer` (nullable) - `score` (nullable), `feedback` (text, nullable), `graded_at` (nullable)

5) Attention & Attendance

AttentionRecord - `id`, `student` (FK -> StudentProfile), `course` (FK), `session` (FK -> ClassSession) - `timestamp` (indexed) - `attentive_prob` (float 0–1), `distracted_prob` (float 0–1) - `model_version` (string), `source` (`WEBCAM|UPLOAD`) - `extra` (JSON: fps, face_confidence, etc.)

(Optional) **Attendance** - `id`, `student` (FK), `session` (FK), `status` (`PRESENT|ABSENT|LATE`), `marked_at`

6) Communication

Announcement - `id`, `course` (FK), `title`, `message`, `created_at`

Notification *(optional MVP+)* - `id`, `user` (FK), `message`, `read` (bool), `created_at`

🌴 Authentication & RBAC

- **Django**: Custom `User` model from day 1.
- **DRF**: `djangorestframework-simplejwt` for JWT (access + refresh).
- **Permissions**:
- `IsTeacher` → only teachers can create courses, assignments, materials, announce.
- `IsStudent` → only students can submit assignments and view own data.
- `IsOwnerOrReadOnly` → teachers can edit only their courses.
- Enrollment checks for protected endpoints.
- **Superuser** creates teachers (or teacher self-signup + admin approve). When a `User.role=TEACHER`, auto-create `TeacherProfile` via signal; same for students.

Attention Model Integration (Keras .h5)

Ingestion options

1. **Realtime (recommended)**
2. **Frontend**: Student page captures webcam via WebRTC; send frames (downsampled, say 1 fps) over **WebSocket**.
3. **Backend**: **Django Channels** + Redis; consumer batches frames per student/session, runs inference with TensorFlow/Keras (`model.h5`), emits live attentiveness % back to the client and writes `AttentionRecord` rows every N seconds.
4. **Post-hoc**
5. Allow uploading a short recording; backend extracts frames and logs attention timeline.

Performance & privacy notes

- Downscale frames to 224×224 (or model input size), jpeg quality ~60.
- Throttle to 0.5–2 fps per student.
- Store only **scores + timestamps**; **do not store raw images** by default.
- Expose `model_version` string; create a feature switch to disable model.

Minimal Channels consumer flow (pseudocode)

```
class AttentionConsumer(AsyncWebsocketConsumer):
    async def connect(self):
        self.user = self.scope['user']
        # check role=STUDENT and enrollment
        await self.accept()

    async def receive(self, text_data=None, bytes_data=None):
        # text_data: {course_id, session_id, frame_b64}
        # decode frame, preprocess, model.predict()
        # save AttentionRecord every k frames; send {attentive_prob} back
        pass
```

REST API (DRF) — Endpoint Sketch

Base path: `/api/`

Auth

- `POST /auth/register/teacher` *(admin only or open with approval)*
- `POST /auth/register/student`
- `POST /auth/login` (JWT)
- `POST /auth/token/refresh`

Users

- `GET /me` — profile + role

Courses

- `GET /courses` (list + filters)
- `POST /courses` *(teacher)*
- `GET /courses/{id}`
- `PATCH /courses/{id}` *(owner teacher)*
- `DELETE /courses/{id}` *(owner teacher/admin)*

Modules & Materials

- `POST /courses/{id}/modules` *(teacher)*
- `POST /modules/{id}/materials` *(teacher)*
- `GET /courses/{id}/materials` *(enrolled students + owner)*

Enrollment

- `POST /courses/{id}/enroll` *(teacher enrolls students or student self-enroll if enabled)*
- `GET /courses/{id}/enrollments` *(teacher)*
- `GET /me/enrollments` *(student)*

Assignments & Submissions

- `POST /courses/{id}/assignments` *(teacher)*
- `GET /courses/{id}/assignments`
- `POST /assignments/{id}/submit` *(student)*
- `GET /assignments/{id}/submissions` *(teacher)*
- `PATCH /submissions/{id}` *(teacher grading)*

Attention & Sessions

- `POST /courses/{id}/sessions` *(teacher)*
- `GET /courses/{id}/sessions`
- `WS /ws/attention` *(student stream)*
- `GET /courses/{id}/attention/summary?from=&to=` *(teacher)*
- `GET /students/{id}/attention?course_id=&from=&to=` *(teacher)*

Announcements

- `POST /courses/{id}/announcements` *(teacher)*
- `GET /courses/{id}/announcements`

🧩 Django App Structure (backend)

```
backend/
  manage.py
  config/
```

```
    settings.py  # split: base.py, dev.py, prod.py
    urls.py
    asgi.py      # Channels entry (ASGI)
    wsgi.py
  users/
    models.py (User, TeacherProfile, StudentProfile)
    signals.py (create profiles)
    serializers.py, views.py, permissions.py
    urls.py
  courses/
    models.py (Course, Module, CourseMaterial, Enrollment, ClassSession,
Announcement)
    serializers.py, views.py, permissions.py
    urls.py
  assignments/
    models.py (Assignment, Submission)
    serializers.py, views.py, urls.py
  attention/
    models.py (AttentionRecord)
    consumers.py (Channels)
    services.py (keras loader & predictor)
    urls.py
  common/
    storage.py, utils.py
  requirements.txt
```

Tech Choices & Services

- **DB**: Postgres
- **Cache/Broker**: Redis (Channels + Celery)
- **Background jobs**: Celery (batch analytics, daily summaries)
- **Media**: S3-compatible (MinIO in dev, S3/GCS in prod). For video, use HLS (e.g., Mux or self-hosted Nginx RTMP + transcoder).
- **Auth**: DRF + SimpleJWT (rotate + blacklist)
- **Realtime**: Django Channels
- **CORS**: `django-cors-headers`
- **Testing**: pytest + pytest-django + factory_boy

🏓 Frontend (React + Vite) Structure

```
client/
  src/
    app/ (routing, store)
    api/ (axios client, auth interceptors)
    features/
      auth/ (login/register pages)
      courses/ (list, detail)
```

```
    learn/ (player, materials)
    assignments/ (list, submit)
    teacher/
      dashboard/
      course-editor/
      grading/
    attention/
      StudentAttentionStream.tsx (webcam, WS client)
      TeacherAttentionDashboard.tsx (charts)
  components/ (UI kit)
```

UI Flows

- **Teacher**: Dashboard → Create Course → Add Modules/Materials → Enroll Students → Create Assignments → Monitor Attention → Grade Submissions → Announce updates
- **Student**: Browse Enrolled Courses → Watch Materials → Live Session (webcam stream on) → Submit Assignments → View feedback + personal attention stats

---

📷Attention Analytics (Teacher Dashboard)

- **Per session & per student**: line chart of attentive% across time.
- **Per course**: heatmap (students × sessions) average attentive%.
- **Drilldowns**: click student → list of low-attention intervals (timestamps) to review recordings.
- **Aggregates**: `avg attentive`, `avg distracted`, `time attentive (min)`, trend vs last week.

MVP Scope (2–3 weeks)

1) **Week 1** - Backend: custom User, JWT, profiles, courses, enrollment, assignments, submissions models + migrations. - CRUD endpoints + permissions; file uploads to local storage. - Frontend: auth, teacher create course, student dashboard, enrollment flow.

2) **Week 2** - ClassSession + CourseMaterial delivery; basic player for video/PDF. - Attention: Django Channels skeleton; load `model.h5` and test inference on single image. - Realtime student stream @1 fps; save `AttentionRecord` every 5s. - Teacher dashboard (course attention summary, per-student chart).

3) **Week 3 (Polish)** - Grading + feedback, announcements, analytics aggregates (Celery nightly job). - Improve security (rate limits, object-level perms), add tests, refine UI.

🧨Testing Checklist

- Unit tests for permissions (teacher vs student vs other)
- Enrollment rules (unique, only owner teacher can enroll others)
- Assignment deadlines; late submissions flagged

- Attention consumer auth (enrolled students only), fps throttling, invalid frames
- API contract tests (OpenAPI schema via `drf-spectacular`)

Setup & Commands

Backend

```
python -m venv venv && source venv/bin/activate  # Windows:
venv\Scripts\activate
pip install django djangorestframework djangorestframework-simplejwt django-
cors-headers psycopg2-binary channels channels-redis pillow
django-admin startproject config .
python manage.py startapp users
python manage.py startapp courses
python manage.py startapp assignments
python manage.py startapp attention
python manage.py makemigrations && python manage.py migrate
python manage.py createsuperuser
```

settings.py essentials - `AUTH_USER_MODEL = 'users.User'` - `REST_FRAMEWORK` JWT auth - `CORS_ALLOW_ALL_ORIGINS = True` *(dev only)* - `ASGI_APPLICATION = 'config.asgi.application'` - `CHANNEL_LAYERS` with Redis

Load model

```
# attention/services.py
from tensorflow.keras.models import load_model
from functools import lru_cache

@lru_cache(maxsize=1)
def get_model():
    return load_model('model.h5')
```

Frontend

```
npm create vite@latest client -- --template react-ts
cd client && npm i axios react-router-dom zustand socket.io-client
npm run dev
```

Security & Privacy Notes

- Use HTTPS end-to-end; secure cookies for refresh tokens.
- Validate ownership/enrollment on every resource access.
- Do **not** store face images by default; store only scores.

- Add consent UI for webcam use; allow students to opt out if policy requires.

🖌️ Deployment

- Docker compose: web (ASGI), worker (Celery), redis, postgres, nginx.
- Static/media via S3 + CloudFront (or equivalent).
- Rolling deploy; run `migrate` and collectstatic on release.

Suggestions & Corrections

- Add **ClassSession** to anchor attention records (not just course).
- Add **Module** for better content structure.
- Keep **Submission.score/feedback** on the same table for simplicity.
- Start without quizzes/payments; add later.

Next Actions (You Can Do Now)

1. Initialize Django project with custom `User` model and JWT.
2. Create models above; run migrations.
3. Implement Course + Enrollment endpoints with permissions.
4. Set up Channels + a mock attention consumer (random scores) to test the full loop.
5. Replace mock with your `model.h5` inference.
6. Build student webcam page & teacher attention dashboard.

If you want, I can generate **starter Django models, serializer/viewset stubs, and React pages** in separate files for you to paste directly into your repo.