Tech Stack Overview

Frontend

Framework

- **React + TypeScript**

UI & Styling

- **Tailwind CSS** – Utility-first styling
- **shadcn/ui** – Prebuilt components (Button, Card, etc.)
- **framer-motion** – Animations
- **lucide-react** – Icons

Backend

Architecture

- **AWS Serverless** (fully managed, event-driven)

Compute : Lambda Functions

- **Video Processing Pipeline**
  - lambda1_generate_job.py — Extracts metadata & student list
  - lambda2_run_rekognition.py — Runs Face Search & writes results
  - lambda3_process_and_store.py — Computes metrics & stores to DynamoDB
- **Chatbot / API Lambda**
  - lambda_chatbot_advanced.py — Bedrock Claude 3 Haiku with tool-calling

Orchestration

- **AWS Step Functions** – Manages the video → processing → storage workflow

Eventing / Triggers

- **Amazon EventBridge Rule**
    - Fires when an S3 Object is created in videos/

Storage

- **Amazon S3**
    - Bucket: hackathon-attendance-media
    - Folders: photos/, videos/, rekognition-results/
- **Amazon DynamoDB**
    - Table: StudentlyticsData
    - PK: record_id = "{session}#{student_id}"

AI/ML Services

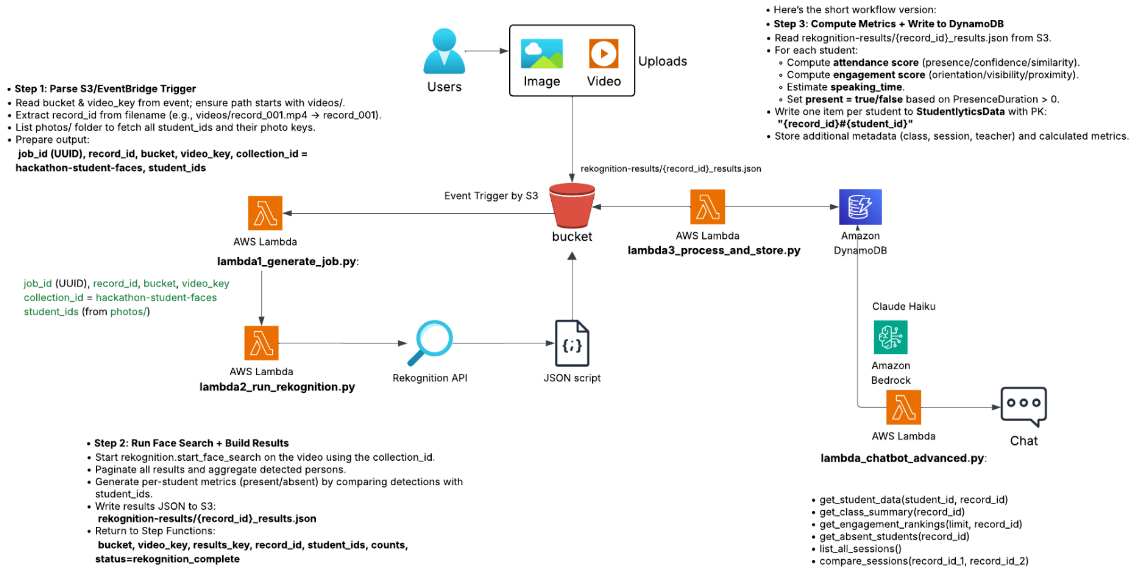
- **Amazon Rekognition**
    - Face collection: hackathon-student-faces
- **Amazon Bedrock**
    - Model: Claude 3 Haiku (via Bedrock Runtime)

API Layer

- **Amazon API Gateway (REST)**
    - Handles chatbot requests + students API
    - CORS enabled
    - API key required for students API

IAM / Observability

- IAM roles for S3, Rekognition, DynamoDB, Logs
- CloudWatch Logs for all Lambdas + Step Functions


Users
Image
Video
Uploads
rekognition-results/{record_id}_results.json
Event Trigger by S3
bucket
AWS Lambda
lambda1_generate_job.py:
AWS Lambda
lambda3_process_and_store.py
Amazon DynamoDB
job_id (UUID), record_id, bucket, video_key
collection_id = hackathon-student-faces
student_ids (from photos/)
AWS Lambda
lambda2_run_rekognition.py
Rekognition API
JSON script
Claude Haiku
Amazon Bedrock
AWS Lambda
lambda_chatbot_advanced.py:
Chat

- **Step 1: Parse S3/EventBridge Trigger**
- Read bucket & video_key from event; ensure path starts with videos/.
- Extract record_id from filename (e.g., videos/record_001.mp4 → record_001).
- List photos/ folder to fetch all student_ids and their photo keys.
- Prepare output:
  job_id (UUID), record_id, bucket, video_key, collection_id = hackathon-student-faces, student_ids

- Here's the short workflow version:
- **Step 3: Compute Metrics + Write to DynamoDB**
- Read rekognition-results/{record_id}_results.json from S3.
- For each student:
  - Compute **attendance score** (presence/confidence/similarity).
  - Compute **engagement score** (orientation/visibility/proximity).
  - Estimate **speaking_time**.
  - Set **present = true/false** based on PresenceDuration > 0.
- Write one item per student to **StudentlyticsData** with PK:
  "{record_id}#{student_id}"
- Store additional metadata (class, session, teacher) and calculated metrics.

- **Step 2: Run Face Search + Build Results**
- Start rekognition.start_face_search on the video using the collection_id.
- Paginate all results and aggregate detected persons.
- Generate per-student metrics (present/absent) by comparing detections with student_ids.
- Write results JSON to S3:
  rekognition-results/{record_id}_results.json
- Return to Step Functions:
  bucket, video_key, results_key, record_id, student_ids, counts, status=rekognition_complete

- get_student_data(student_id, record_id)
- get_class_summary(record_id)
- get_engagement_rankings(limit, record_id)
- get_absent_students(record_id)
- list_all_sessions()
- compare_sessions(record_id_1, record_id_2)

Trigger

An object is created in S3 under videos/.

- EventBridge rule (S3 Object Created for videos/) targets the Step Function VideoProcessingPipeline.
- Defined in backend/step-functions/state-machine.json.

Step 1 — Generate Job

(GenerateJobForVideo → lambda1_generate_job.py)

- Input: EventBridge event (or direct S3 event).
- Extracts bucket and video_key. Validates it's under videos/.

- Derives record_id from filename (e.g., videos/record_001.mp4 → record_001).
- Lists photos/ in the bucket to collect student_ids and mapping of student_id → photo_key.
- Output passed to next step:
  - job_id (UUID), record_id, bucket, video_key
  - collection_id = hackathon-student-faces
  - student_ids (from photos/)

Step 2 — Run Rekognition

(RunRekognition → lambda2_run_rekognition.py)

- Starts rekognition.start_face_search on the video with the above collection_id.
- Polls rekognition.get_face_search for completion (with timeout).
- Paginates results and aggregates all Persons.
- Computes structured per-student metrics including present/absent by cross-referencing student_ids.
- Writes a JSON artifact to S3:
  - Key: rekognition-results/{record_id}_results.json
  - Content includes student_data keyed by student_id and summary counts.
- Output to Step Functions includes:
  - bucket, video_key, results_key, record_id, student_ids, status = rekognition_complete, counts

---

Step 3 — Process & Store

(ProcessAndStore → lambda3_process_and_store.py)

- Reads rekognition-results/{record_id}_results.json from S3.
- For each student:
  - Calculates attendance score (weighting presence/confidence/similarity).
  - Calculates engagement score (orientation/visibility/proximity).
  - Estimates speaking_time.
  - Sets status boolean: present if PresenceDuration(sec) > 0.
- Writes one DynamoDB item per student to StudentlyticsData.
  - Primary key record_id is a composite string: "{record_id}#{student_id}".
  - Also stores metadata (class, session, teacher) and metrics.

✅ Completion

Step Functions transitions to Success if prior steps succeed; otherwise HandleError.

💬 Chatbot/API Flow

- **Lambda**: lambda_chatbot_advanced.py
- **Intended integration**: Exposed via API Gateway (CORS handled for OPTIONS, expects POST with body.message).
- **Model**: Calls Bedrock claude-3-haiku with tool/function-calling.
- **Tools provided**:
  - get_student_data(student_id, record_id?)
  - get_class_summary(record_id?)
  - get_engagement_rankings(limit, record_id?)

  - get_absent_students(record_id?)
  - list_all_sessions()
  - compare_sessions(record_id_1, record_id_2)

🗃️ Data Model and Artifacts

- **S3 inputs**:
  - photos/student_<id>.jpg|png → indexed to Rekognition with ExternalImageId = <id>
  - videos/<...>.mp4 → triggers processing
- **S3 outputs**:
  - rekognition-results/{record_id}_results.json → structured detection and summary
- **DynamoDB: StudentlyticsData**
  - PK: record_id = "{record_id}#{student_id}"
  - Attributes include:
    - student_id, student_name, student_email, photo_url
    - class_id, class_name, department, room, schedule, topic
    - session_id, session_date, start_time, end_time
    - teacher_*
    - status (boolean present/absent)
    - time_inside_class, speaking_time
    - attendance (Decimal), engagement (Decimal)
    - timestamp

🧰 Notable Utilities and Alternative Path

- **index_student_photos.py**
    - Bulk indexes `photos/` into Rekognition collection (`hackathon-student-faces`), setting `ExternalImageId` to the numeric student ID.
    - Also lists faces present in the collection.
- **rekognition_video_processor.py** (prototype/alternate)
    - Single Lambda approach that handles S3 event, starts Rekognition, aggregates results, and writes to a separate table `Attendance`.
    - Not referenced by the Step Function pipeline. Likely an earlier prototype.

⚙️ Triggers, IAM, and Config

- **Triggers**:
    - EventBridge → Step Function on S3 Object Created in `videos/`
    - API Gateway → lambda_chatbot_advanced.py for chatbot endpoints
- **IAM Role** (README suggests):
    - S3 read/write, Rekognition full access, DynamoDB full access, CloudWatch Logs
- **Config constants**:
    - Bucket: `hackathon-attendance-media`
    - Rekognition collection: `hackathon-student-faces`
    - Table: `StudentlyticsData`
    - Region: `us-east-1`

  - Model: us.anthropic.claude-3-haiku-20240307-v1:0

🧭 Observations and Small Issues

- **lambda1_generate_job.py**
  - Likely typo at the end: video_metadata['ContentLeng'] should be video_metadata['ContentLength']. The file seems truncated after this line; ensure the rest of the metadata extraction and emitted payload are complete.
- **State Machine placeholders**
  - YOUR_ACCOUNT_ID must be replaced in state-machine.json Lambda ARNs.
- **Chatbot deployment**
  - Ensure an API Gateway integration is configured with proxy integration to pass httpMethod and body. CORS headers are implemented.

🧠 How It Works in One Sentence

Upload a classroom video to S3 → EventBridge triggers a Step Function that creates a job, runs Rekognition face search, saves structured results to S3, then computes attendance/engagement and writes per-student records to DynamoDB, which the Bedrock-powered chatbot reads to answer analytics queries.

✅ Recommended Actions

- **Fix minor bug**:
  - **lambda1_generate_job.py**: Replace ContentLeng with ContentLength and complete any missing return payload.
- **Wire up Step Functions**:

    - Replace ARNs in state-machine.json with your real Lambda ARNs and deploy.
- **Provision API**:
    - Expose lambda_chatbot_advanced.py via API Gateway (POST /chat), enable CORS.
- **Verify Rekognition collection**:
    - Run index_student_photos.py once to populate faces, confirm via list_indexed_faces().

# Short answer

Yes. Using GraphQL is a good fit here, especially with AWS AppSync. It can sit between your React frontend and DynamoDB/Lambda, giving you typed queries, a single endpoint, and optional real-time subscriptions.

# How it would work for your project

- **Service**: AWS AppSync (managed GraphQL)

- **Data sources**:

  - DynamoDB table `StudentlyticsData` for student/session records

  - Lambda resolvers for computed analytics (e.g., class summary, absent list) or to reuse your existing Lambdas

- **Auth**: API Key (hackathon/demo), Cognito, or IAM

- **Frontend**: Apollo Client or AWS Amplify API to call GraphQL

## Suggested GraphQL schema

- **Types**

  - [StudentRecord](cci:2://file:///Users/truptaditya/Documents/GitHub/Hackathon/AWS-Hackathon/src/pages/StudentsPage.tsx:7:0-22:1) mapped to your `StudentlyticsData` attributes

  - `ClassSummary` for analytics (present/absent, averages)

- **Queries**

  - `listSessions`: recent session IDs (record_id prefixes)

  - `getSessionStudents(recordId: String!)`: students for a session

  - [getStudent(studentId: Int!, recordId: String)](cci:1://file:///Users/truptaditya/Documents/GitHub/Hackathon/AWS-Hackathon/src/services/api.ts:4:0-20:1): details for a student

  - `getClassSummary(recordId: String!)`: computed summary (can be a Lambda resolver)

- **Mutations** (optional)

  - [addStudent](cci:1://file:///Users/truptaditya/Documents/GitHub/Hackathon/AWS-Hackathon/src/services/api.ts:49:0-68:1), [updateStudent](cci:1://file:///Users/truptaditya/Documents/GitHub/Hackathon/AWS-Hackathon/src/services/api.ts:70:0-89:1), [deleteStudent](cci:1://file:///Users/truptaditya/Documents/GitHub/Hackathon/AWS-Hackathon/src/services/api.ts:91:0-108:1) if you'll manage data from UI