

# Take Home Assignment

## Admin CMS + Public Catalog API + Scheduled Publishing

### Goal

Build and deploy a mini CMS used by an internal team to manage **Programs** → **Terms** → **Lessons**, schedule lesson releases, and expose a **public catalog API** for a consumer app.

This must be a **running product**: real DB schema + migrations + deployed backend + deployed frontend + deployed worker.

Important (Flexibility): You are free to change the schema, naming, or add/remove entities if you believe you can make it better. If you do, clearly document your reasoning and the tradeoffs in the README.

Database choice is also yours (Postgres/MySQL/etc.). Use whatever you're most productive with just ensure migrations + deployment work cleanly.

### Deliverables (non-negotiable)

1. Deployed CMS Web App URL (HTTPS)
2. Deployed API URL (HTTPS)
3. Managed database used by the API
4. Migrations included in repo (reproducible from scratch)
5. Worker/cron running in deployment (auto-publishes scheduled lessons)
6. Docker Compose local run: `docker compose up --build` runs web + api + worker + db
7. Seed script creates sample data

### Core Domain Model (must be DB-enforced)

#### Entities

##### Program

- `id (uuid)`
- `title` (required)
- `description`

- `language_primary` (e.g., `te`, `hi`, `en`)
- `languages_available` (array of strings; must include primary)
- `status` enum: `draft` | `published` | `archived`
- `published_at` (nullable)
- `created_at`, `updated_at`

## Topic

- `id`, `name` (unique)
- Many-to-many: Program ↔ Topic

## Term

- `id (uuid)`
- `program_id` (fk)
- `term_number` (int, required)
- `title` (optional)
- `created_at`

## Lesson

- `id (uuid)`
- `term_id` (fk)
- `lesson_number` (int, required)
- `title` (required)
- `content_type` enum: `video` | `article` (*video must be supported*)
- `duration_ms` (required if video)
- `is_paid` (bool, default false)

## Multi-language content

- `content_language_primary` (string)
- `content_languages_available` (array of strings; must include primary)
- `content_urls_by_language` (map of language → URL) (*must include primary language URL*)

## Subtitles

- `subtitle_languages` (array of strings)
- `subtitle_urls_by_language` (map of language → URL) (*optional but recommended; if provided must match languages list*)

## Publishing workflow

- `status` enum: `draft` | `scheduled` | `published` | `archived`
- `publish_at` (nullable)
- `published_at` (nullable)

- `created_at`, `updated_at`
- 

# Media Assets Requirement (CMS-grade)

## Variants

Support multiple asset variants:

- `portrait`, `landscape`, `square`, `banner`

### Program posters (required)

A Program must support **poster assets** per language **and** per variant.

#### Validation

- For `language_primary`, Program must have at least:
  - `portrait` poster URL
  - `landscape` poster URL

### Lesson thumbnails (required)

A Lesson must support **thumbnail assets** per content language **and** per variant.

#### Validation

- For `content_language_primary`, Lesson must have at least:
    - `portrait` thumbnail URL
    - `landscape` thumbnail URL
  - If a Lesson is `published`, these requirements must be satisfied (block publish otherwise).
- 

## Storage approach (candidate chooses one)

### Option A (recommended): normalized tables

#### program\_assets

- `id (uuid)`
- `program_id (fk)`
- `language` (string)
- `variant` enum: `portrait` | `landscape` | `square` | `banner`
- `asset_type` enum: `poster`
- `url` (string, required)
- Unique `(program_id, language, variant, asset_type)`

#### lesson\_assets

- `id` (uuid)
- `lesson_id` (fk)
- `language` (string)
- `variant` enum: `portrait | landscape | square | banner`
- `asset_type` enum: `thumbnail | subtitle` (*subtitle variant can be null if you store subtitles differently*)
- `url` (string, required)
- Unique `(lesson_id, language, variant, asset_type)`

## Option B: JSON columns

Store posters/thumbnails as JSON maps, e.g.

```
{ "te": { "portrait": "url1", "landscape": "url2" }, "en": { "portrait": "url3" } }
```

If using JSON, you must still validate uniqueness + required variants in app logic.

## DB Constraints (must implement)

- Unique `(program_id, term_number)`
- Unique `(term_id, lesson_number)`
- Unique `topic.name`
- If `lesson.status='scheduled'` → `publish_at IS NOT NULL`
- If `lesson.status='published'` → `published_at IS NOT NULL`
- Primary language must be included in available languages (Program + Lesson)
- Asset uniqueness constraints (DB constraints if using normalized tables)

Index expectations (you'll be judged on this)

- `lesson(status, publish_at)`
- `lesson(term_id, lesson_number)`
- `program(status, language_primary, published_at)`
- M2M join indexes for topic filters
- Asset lookup indexes if using normalized assets tables

## Publishing Workflow (core backend challenge)

# Allowed Lesson transitions

- `draft → published` (publish now)
- `draft → scheduled` (schedule publish)
- `scheduled → published` (auto via worker when time arrives)
- `published → archived`
- `draft/scheduled → archived`

## Worker/cron (must be deployed + running)

Runs every minute (demo frequency is fine) and:

1. Finds lessons with `status='scheduled' AND publish_at <= now()`
2. Publishes them in a **transaction**:
  - set `status='published'`
  - set `published_at=now()`

## Program publishing rule

A Program automatically becomes `published` when it has  $\geq 1$  published lesson:

- set `program.status='published'`
- set `program.published_at` only once (first publish)

## Hard requirements

- **Idempotent**: rerunning worker doesn't change already-published lessons' timestamps
- **Concurrency-safe**: assume two workers can run simultaneously (use row locks or safe conditional updates)

---

# Authentication + Roles (CMS)

Roles:

- **Admin**: everything + manage users
- **Editor**: manage programs/terms/lessons + schedule/publish/archive
- **Viewer**: read-only CMS access

Must include:

- Login UI
- API protected with role checks (not only frontend hiding)

Auth approach is up to you (JWT/session/API keys), but must be secure and documented.

---

# CMS Web UI (must-have)

## Screens

1. Login
2. Programs list
  - filters: status, primary language, topic
  - show poster previews (at least primary language portrait)
3. Program detail
  - edit: title/description, language\_primary, languages\_available, topics
  - posters manager: per language + variant URLs with previews
  - terms list + create term
  - lessons list with status badges, publish\_at/published\_at, is\_paid
4. Lesson editor
  - edit lesson fields
  - manage thumbnails per language + variant with previews
  - content URLs per language (simple table UI is fine)
  - subtitle languages (+ optional URLs)
  - actions: Publish now / Schedule / Archive
  - clear validation errors

No need for fancy design; must be usable.

---

## Public Catalog API (consumer-facing)

No auth (or simple read-only public token). Must return **published-only** data.

## Endpoints (minimum)

- `GET /catalog/programs?language=&topic=&cursor=&limit=`
  - only programs with  $\geq 1$  published lesson
  - sorted by most recently published
- `GET /catalog/programs/:id`
  - includes terms + **published lessons only**
  - includes multi-language fields and assets
- `GET /catalog/lessons/:id` (published only)

## Requirements

- Pagination (cursor preferred)
- Cache headers on catalog routes ( `Cache-Control` at minimum; ETag optional)
- Consistent error format `{ code, message, details? }`

## Expected asset structure in responses

Program:

```
{ "assets": { "posters": { "te": { "portrait": "...", "landscape": "...", "square": "..." } } } }
```

Lesson:

```
{ "assets": { "thumbnails": { "te": { "portrait": "...", "landscape": "..."} } } }
```

## Operational Requirements

- `GET /health` returns OK + DB connectivity
- Structured logs (request id/correlation id preferred)
- Secrets via env vars (no secrets in repo)

## Local Run Requirement

`docker compose up --build` must run:

- `web`
- `api`
- `worker`
- `db`

## Seed Data Requirement

Seed must create at least:

- 2 Programs
- 2 Terms total
- 6 Lessons total

- Multi-language example:
    - at least 1 Program has 2 languages
    - at least 2 Lessons have multi-language content URLs
  - Assets example:
    - each Program primary language has **portrait + landscape** posters
    - each Lesson primary content language has **portrait + landscape** thumbnails
  - One scheduled lesson with `publish_at` within the next 2 minutes (for demo)
- 

## README must include

- Architecture overview (diagram ok)
  - Local setup steps
  - How migrations run
  - How seed runs
  - Deployed URLs (web + api)
  - Demo flow:
    1. login as editor
    2. create/edit lesson, schedule publish
    3. wait for worker → verify it becomes published
    4. verify public catalog now includes it
- 

## Evaluation Rubric

- Schema + migrations + constraints + indexing (25%)
  - Worker correctness: idempotent + concurrency-safe + transactional (25%)
  - Full-stack usability + RBAC enforcement (20%)
  - Catalog API quality (pagination/filtering/assets) (15%)
  - Deployment + ops (health/logging/config) (15%)
- 

## Heading 2

