ManualVideoService

September 4, 2026

ManualVideoService

Kind: Service

Source: `atloria-monorepo/apps/api/src/documentation/manual-video.service.ts`

ManualVideo service (D1) — the read/edit surface behind the reader's "Video" tab.

Every query is org-scoped (fail-closed): a ManualVideo carries organizationId and is only ever returned/updated when it matches the caller's org — a missing/foreign row is a 404, never a cross-tenant read. Editing the narration `script` marks the row `stale`; requesting a regen enqueues a videoOnly re-render (TTS + assembly from
the cached clips) that patches the ACTIVE version in place — the video mirror of the screenshot asset-update loop.

`ManualVideoService` provides the organization-scoped read and edit surface for a manual’s Video tab. It lists and retrieves manual videos, updates narration scripts, and requests video-only regeneration using cached clips; script edits mark the video as stale, while regeneration patches the active version in place.

Methods

Method	Signature	Returns	Description
`listForProject`	`listForProject(projectId: string, user: JwtPayload)`	`unknown`	All page videos for a project (Video tab list), newest first.
`get`	`get(id: string, user: JwtPayload)`	`unknown`	One video, org-scoped (404 if missing or foreign).
`getPublicForEmbed`	`getPublicForEmbed(id: string)`	`unknown`	PUBLIC render payload for the standalone `/embed/video/:id` page + iframe embeds.

Method	Signature	Returns	Description
`updateScript`	`updateScript(id: string, script: ManualVideoScript, user: JwtPayload)`	`unknown`	Save an edited narration script.
`requestRegen`	`requestRegen(id: string, user: JwtPayload)`	`unknown`	Request a videoOnly re-render from the edited script + cached clips.

Dependencies

- `PrismaService`
- `Queue`

Where it refuses work

- `ManualVideoService` stops the work with `NotFoundException` when `!video` — “Manual video not found”.
- `ManualVideoService` stops the work with `NotFoundException` when `!video || !video.videoUrl || !video.documentId` — “Video not found”.
- `ManualVideoService` stops the work with `NotFoundException` when `!doc` — “Video not found”.
- `ManualVideoService` stops the work with `BadRequestException` when `!script || typeof script !== 'object' || typeof script.steps !== 'object'` — “script must be { intro?, steps: {n: text}, outro? }”.
- `ManualVideoService` stops the work with `BadRequestException` when `!sourceJobId` — “No source generation run found for this video — run a full generation first”.
- `ManualVideoService` stops the work with `BadRequestException` when `!botPassword` — “ATLORIX_BOT_PASSWORD env not set — cannot enqueue atlorix jobs without a service account …”.

When something fails

- `ManualVideoService` handles failure in 1 place: it lets it reach the caller in all 1.

Diagram

```
sequenceDiagram
  participant Client
```

```
  participant Service as ManualVideoService
  participant DB as Database
  participant Renderer as Video Render Queue

  Client->>Service: updateScript(orgId, videoId, script)
  Service->>DB: Find video by id + organizationId
  alt Video missing or belongs to another org
    DB-->>Service: No matching row
    Service-->>Client: 404 Not Found
  else Authorized video
    DB-->>Service: ManualVideo
    Service->>DB: Update script and set stale = true
    Service-->>Client: Updated stale video
  end

  Client->>Service: requestRegen(orgId, videoId)
  Service->>DB: Find video by id + organizationId
  Service->>Renderer: Enqueue videoOnly render (TTS + cached clips)
  Renderer->>DB: Patch ACTIVE version in place
  Service-->>Client: Regeneration requested
```

Usage

```
import { Injectable } from '@nestjs/common';
import { ManualVideoService } from './documentation/manual-video.service';

@Injectable()
export class ManualVideoController {
  constructor(private readonly manualVideoService: ManualVideoService) {}

  async updateNarration(
    organizationId: string,
    videoId: string,
    script: string,
  ) {
    const video = await this.manualVideoService.updateScript(
      organizationId,
      videoId,
      script,
    );

    // The updated video is now stale and can be regenerated.
    await this.manualVideoService.requestRegen(organizationId, video.id);

    return video;
  }

  async listProjectVideos(organizationId: string, projectId: string) {
    return this.manualVideoService.listForProject(organizationId, projectId);
```

```
  }
}
```

AI Coding Instructions

- Always pass and enforce `organizationId` for authenticated reads and mutations; missing or foreign video records must resolve as `404`, not authorization-specific responses.
- When changing a narration `script`, mark the associated video as `stale` so clients can indicate that the rendered output is outdated.
- Use `requestRegen()` for video-only rendering flows; it should reuse cached clips and trigger TTS plus video assembly rather than a full screenshot/render pipeline.
- Preserve the active-version patching behavior: completed regeneration updates the existing ACTIVE version in place.
- Use `getPublicForEmbed()` only for intentionally public embed access; do not substitute it for organization-scoped internal retrieval.

Relationships

- DEPENDS_ON → `PrismaService`
- DEPENDS_ON → `queue`

Referenced By

- `DocumentationModule` (MODULE_PROVIDES)
- `ManualVideoController` (DEPENDS_ON)
- `PublicManualVideoController` (DEPENDS_ON)