SdkArtifactsService

September 4, 2026

SdkArtifactsService

Kind: Service

Source: `atloria-monorepo/apps/api/src/sdk/sdk-artifacts.service.ts`

C2 core: content-addressed SDK artifact bookkeeping + the enqueue hook.

Dedup is the unique (projectId, language, specHash) row itself: an unchanged spec re-upsert finds a live row for the pinned generator version and enqueues nothing. A changed spec has a new hash → no row → enqueue. A generator-version bump leaves stale-version rows behind → re-enqueue (full regen by design).

`SdkArtifactsService` manages content-addressed SDK artifact records for published API specifications and coordinates generation work through an enqueue hook. It deduplicates generation by `(projectId, language, specHash)` and generator version, while exposing configuration, regeneration, status, public availability, and download resolution APIs.

Methods

Method	Signature	Returns	Description
`onSpecUpserted`	`onSpecUpserted(projectId: string, spec: unknown, options: { force?: boolean; onlyLanguages?: SdkGeneratedLanguage[] })`	`Promise<{ enqueued: string[]; skipped: string[]; reason?: string }>`	Ensure artifacts exist (or are being generated) for every enabled generated language of this project's CURRENT spec.
`getStatus`	`getStatus(projectId: string)`	`Promise<SdkStatusView>`	
`setConfig`	`setConfig(projectId: string, languages: string[])`	`Promise<{ languages: string[] }>`	

Method	Signature	Returns	Description
`regenerate`	`regenerate(projectId: string)`	`Promise<{ enqueued: string[]; skipped: string[]; reason?: string }>`	Owner "Regenerate": force re-enqueue every enabled language for the current spec.
`resolvePublishedProject`	`resolvePublishedProject(slugWithId: string)`	`Promise<{ projectId: string; organizationId: string }>`	Resolve a published project from the base62-suffixed public slug.
`getPublicAvailability`	`getPublicAvailability(projectId: string)`	`Promise<{		

```
languages: Array<{ id: string; label: string; status: 'ready' | 'generating' | 'unavailable' }>;
```

}> `| Languages offered on the reader's "Get the SDK" dropdown, with readiness. |`
`| getPublicDownload | getPublicDownload(projectId: string, language: SdkGeneratedLanguage) | Promise` | Public download resolution for a GENERATED language (typescript is emitted per-request by the controller). |

Dependencies

- `PrismaService`
- `SdkQueue`

Where it refuses work

- `SdkArtifactsService` stops the work with `BadRequestException` when `invalid.length > 0`.
- `SdkArtifactsService` stops the work with `NotFoundException` when `!snapshot?.apiSpecJson` — “No API spec found — run docs generation first.”.
- `SdkArtifactsService` stops the work with `BadRequestException` when `!match` — “Invalid URL format”.
- `SdkArtifactsService` stops the work with `NotFoundException` when `!project || !project.isPublic` — “Project not found”.
- `SdkArtifactsService` stops the work with an early return when `!quality.eligible`, in 2 places.
- `SdkArtifactsService` stops the work with an early return when `artifact?.status === 'READY' && artifact.blobUrl`, in 2 places.

Diagram

```
sequenceDiagram
  participant Caller as API/Controller
  participant Service as SdkArtifactsService
  participant DB as Artifact Store
  participant Queue as SDK Generation Queue
  participant Storage as Artifact Storage

  Caller->>Service: onSpecUpserted(projectId, specHash)
  Service->>DB: Find artifact by project, language, hash, generator version

  alt Live artifact exists
    DB-->>Service: Existing artifact
    Service-->>Caller: skipped language
  else No matching live artifact
    Service->>DB: Create artifact bookkeeping row
    Service->>Queue: Enqueue SDK generation job
    Queue-->>Service: Job accepted
    Service-->>Caller: enqueued language
  end

  Caller->>Service: getPublicDownload()
  Service->>DB: Resolve ready artifact for published project
  Service->>Storage: Resolve downloadable artifact
  Storage-->>Service: Download metadata
  Service-->>Caller: PublicSdkDownload
```

Usage

```
import { Injectable } from '@nestjs/common';
import { SdkArtifactsService } from './sdk-artifacts.service';

@Injectable()
export class SpecPublishService {
  constructor(
    private readonly sdkArtifactsService: SdkArtifactsService,
  ) {}

  async publishSpec(projectId: string, specHash: string) {
    // Trigger SDK generation only for languages without a live artifact
    // matching this spec hash and the pinned generator version.
    const result = await this.sdkArtifactsService.onSpecUpserted();

    return {
      projectId,
      specHash,
      sdkGeneration: {
        enqueued: result.enqueued,
        skipped: result.skipped,
        reason: result.reason,
      },
    };
  }
```

```
  async regenerateSdks() {
    // Explicitly request regeneration for configured SDK languages.
    return this.sdkArtifactsService.regenerate();
  }

  async getPublicSdkDownloads() {
    const availability =
      await this.sdkArtifactsService.getPublicAvailability();

    return availability.languages.filter(
      (language) => language.status === 'ready',
    );
  }
}
```

AI Coding Instructions

- Preserve content-addressed deduplication: artifact identity depends on `projectId`, `language`, and `specHash`, with generator version determining whether an existing row is reusable.
- Call `onSpecUpserted()` after a successful specification upsert; do not enqueue SDK generation directly from controllers or unrelated services.
- Treat a generator-version bump as a full regeneration event: stale-version artifact rows must not suppress new jobs.
- Use `getPublicAvailability()` before exposing public SDK links, and only resolve downloads for artifacts in the `ready` state.
- Keep `setConfig()` and `regenerate()` aligned with the configured language list so disabled languages are not unexpectedly enqueued.

Relationships

- DEPENDS_ON → `PrismaService`
- DEPENDS_ON → `SdkQueue`

Referenced By

- `PublicSdkController` (DEPENDS_ON)
- `SdkController` (DEPENDS_ON)
- `SdkModule` (MODULE_PROVIDES)
- `SdkModule` (MODULE_EXPORTS)
- `TechnicalDocsService` (DEPENDS_ON)