ScreenshotMetadataService

September 4, 2026

ScreenshotMetadataService

Kind: Service

Source: `atloria-monorepo/apps/screenshot-worker/src/app/screenshot/screenshot-metadata.service.ts`

`ScreenshotMetadataService` manages persisted metadata for generated screenshots in the screenshot worker. It calculates content hashes, detects existing screenshots, determines whether regeneration is needed, stores versioned records, and manages screenshot history and baseline selection.

Methods

Method	Signature	Returns	Description
`calculateContentHash`	`calculateContentHash(buffer: Buffer)`	`string`	Calculate content hash for screenshot
`findExisting`	`findExisting(metadata: ScreenshotMetadata)`	`Promise<ExistingScreenshot | null>`	
`shouldRegenerate`	`shouldRegenerate(existing: ExistingScreenshot, newContentHash: string)`	`boolean`	Check if screenshot needs regeneration Compares content hash to determine if visual content has changed.
`save`	`save(buffer: Buffer, metadata: ScreenshotMetadata, assetUrl: string, organizationId: string,`	`Promise<{ id: string; version: number }>`	Save screenshot metadata to database

Method	Signature	Returns	Description
	`capturedBy: string, jobId: string)`		
`getHistory`	`getHistory(projectId: string, flowStepId: string)`	`Promise<ExistingScreenshot[]>`	Get screenshot history for a flow step
`markAsBaseline`	`markAsBaseline(screenshotId: string)`	`Promise<void>`	Mark screenshot as baseline (for visual regression testing)
`getBaseline`	`getBaseline(projectId: string, viewport: string, stateType: string, flowStepId: string)`	`Promise<ExistingScreenshot | null>`	

Dependencies

- `PrismaService`

Where it refuses work

- `ScreenshotMetadataService` stops the work with an early return when `!screenshot`, in 2 places.
- `ScreenshotMetadataService` stops the work with an early return when `!existing.contentHash`.
- `ScreenshotMetadataService` stops the work with an early return when `width < 768`.
- `ScreenshotMetadataService` stops the work with an early return when `width < 1024`.

When something fails

- `ScreenshotMetadataService` handles failure in 5 places: it turns it into a return value in 3, and lets it reach the caller in 2.

Diagram

```
sequenceDiagram
  participant Worker as Screenshot Worker
  participant Service as ScreenshotMetadataService
  participant DB as Metadata Store

  Worker->>Service: calculateContentHash()
```

```
  Service-->>Worker: content hash

  Worker->>Service: findExisting()
  Service->>DB: Query by screenshot identity/hash
  DB-->>Service: ExistingScreenshot | null
  Service-->>Worker: existing record

  Worker->>Service: shouldRegenerate()
  Service-->>Worker: boolean

  alt Screenshot must be generated
    Worker->>Service: save()
    Service->>DB: Create/update versioned metadata
    DB-->>Service: id, version
    Service-->>Worker: { id, version }
  end

  Worker->>Service: getBaseline()
  Service->>DB: Fetch baseline screenshot
  DB-->>Service: ExistingScreenshot | null
  Service-->>Worker: baseline record
```

Usage

```typescript
import { Injectable } from '@nestjs/common';
import { ScreenshotMetadataService } from './screenshot-metadata.service';

@Injectable()
export class ScreenshotJobService {
  constructor(
    private readonly screenshotMetadataService: ScreenshotMetadataService,
  ) {}

  async processScreenshot(): Promise<void> {
    const contentHash = this.screenshotMetadataService.calculateContentHash();
    const existing = await this.screenshotMetadataService.findExisting();

    if (!this.screenshotMetadataService.shouldRegenerate()) {
      console.log(`Skipping unchanged screenshot (${contentHash})`);
      return;
    }

    // Generate and upload the screenshot before saving its metadata.
    const saved = await this.screenshotMetadataService.save();

    console.log(
      `Saved screenshot metadata: ${saved.id} (version ${saved.version})`,
    );

    const baseline = await this.screenshotMetadataService.getBaseline();

    if (!baseline) {
      await this.screenshotMetadataService.markAsBaseline();
```

```
    }
  }
}
```

AI Coding Instructions

- Use `findExisting()` and `shouldRegenerate()` before triggering expensive screenshot rendering or upload work.
- Treat `calculateContentHash()` as the canonical change-detection value; keep its inputs stable when adding screenshot-affecting configuration.
- Call `save()` only after the screenshot artifact has been successfully generated and stored, so metadata does not reference a missing asset.
- Use `getHistory()` for version-aware workflows and `getBaseline()` / `markAsBaseline()` for visual regression comparison flows.
- Preserve the service's NestJS dependency-injection usage; avoid constructing `ScreenshotMetadataService` manually outside the Nest container.

Relationships

- DEPENDS_ON → `PrismaService`

Referenced By

- `ScreenshotModule` (MODULE_PROVIDES)
- `ScreenshotModule` (MODULE_EXPORTS)
- `ScreenshotService` (DEPENDS_ON)
- `BatchScreenshotService` (DEPENDS_ON)
- `FlowReplayService` (DEPENDS_ON)