StorageService

September 4, 2026

StorageService

Kind: Service

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

Azure Blob Storage service for storing screenshots

Features:

- Upload screenshots to Azure Blob Storage
- Generate public URLs
- Support for metadata
- Auto-create containers

`StorageService` is a NestJS backend service responsible for persisting screenshot files into Azure Blob Storage and returning accessible URLs for later retrieval. It encapsulates container management (including auto-creation), upload operations, and optional blob metadata handling so the screenshot-worker can store outputs reliably without duplicating storage logic.

Methods

Method	Signature	Returns	Description
`upload`	`upload(options: UploadOptions)`	`Promise<UploadResult>`	Upload a screenshot to Azure Blob Storage
`uploadMultiple`	`uploadMultiple(uploads: UploadOptions[])`	`Promise<UploadResult[]>`	Upload multiple screenshots
`delete`	`delete(filename: string)`	`Promise<void>`	Delete a screenshot
`isAvailable`	`isAvailable()`	`boolean`	Check if storage is

Method	Signature	Returns	Description
			available

Dependencies

- `ConfigService`

Where it refuses work

- `StorageService` stops the work with `Error` when `!this.containerClient` — “Azure Blob Storage not initialized”, in 2 places.

When something fails

- `StorageService` handles failure in 4 places: it logs it and continues in 2, and lets it reach the caller in 2.

Diagram

```
sequenceDiagram
  autonumber
  participant Worker as Screenshot Worker
  participant Storage as StorageService
  participant Azure as Azure Blob Storage

  Worker->>Storage: uploadScreenshot(buffer, filename, container, metadata?)
  Storage->>Azure: ensureContainerExists(container)
  Azure-->>Storage: container ready
  Storage->>Azure: uploadBlob(container, filename, buffer, metadata)
  Azure-->>Storage: upload result (etag/version)
  Storage->>Azure: generatePublicUrl(container, filename)
  Azure-->>Storage: public URL
  Storage-->>Worker: { url, blobName, container, etag? }
```

Usage

```
import { Injectable } from '@nestjs/common';
import { StorageService } from './storage/storage.service';

@Injectable()
export class ScreenshotJobService {
  constructor(private readonly storage: StorageService) {}
```

```
  async persistScreenshot(pngBuffer: Buffer, jobId: string) {
    const container = 'screenshots';
    const blobName = `jobs/${jobId}.png`;

    const result = await this.storage.uploadScreenshot(pngBuffer, blobName, {
      container,
      contentType: 'image/png',
      metadata: {
        jobId,
        source: 'screenshot-worker',
      },
    });

    // Store result.url in DB / emit event, etc.
    return result.url;
  }
}
```

AI Coding Instructions

- Keep Azure-specific logic (container creation, blob client setup, URL generation) inside `StorageService`; call it from jobs/workers/controllers rather than re-implementing storage concerns.
- Always pass a deterministic `blobName` (include job IDs / timestamps) to avoid accidental overwrites; if overwrites are intended, document it explicitly.
- When attaching metadata, ensure values are strings and keep keys stable (Azure metadata constraints); avoid putting sensitive data in metadata since it can be exposed.
- Handle container creation idempotently (create-if-not-exists) and surface meaningful errors (auth, missing connection string, permissions) to the caller for retry/backoff logic.
- Prefer returning the generated public URL from the service (single source of truth for URL format/SAS/public access strategy) and avoid hardcoding URL templates elsewhere.

Relationships

- DEPENDS_ON → `configservice`

Referenced By

- `AppService` (DEPENDS_ON)

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