BatchScreenshotService

September 4, 2026

BatchScreenshotService

Kind: Service

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

Service for batch screenshot processing

Orchestrates:

- Parallel task execution with concurrency limits
- Authentication session management
- Retry logic with exponential backoff
- Progress tracking and caching
- Content deduplication

`BatchScreenshotService` coordinates batch screenshot generation in the screenshot worker. It manages authenticated browser sessions, executes screenshot tasks with bounded concurrency, retries transient failures using exponential backoff, and records cached progress while deduplicating equivalent content.

Methods

Method	Signature	Returns	Description
`createJob`	`createJob(request: TakeBatchScreenshotsDto)`	`Promise<BatchScreenshotJobResponseDto>`	Create a new batch job (synchronous - returns immediately)
`processBatch`	`processBatch(request: TakeBatchScreenshotsDto)`	`Promise<BatchScreenshotJobResponseDto>`	Process a batch of screenshots (main orchestration

Method	Signature	Returns	Description
			method) Flow: 1.
`getJobStatus`	`getJobStatus(jobId: string, organizationId: string)`	`Promise<BatchScreenshotStatusDto | null>`	

Dependencies

- `PlaywrightService`
- `StorageService`
- `ScreenshotMetadataService`
- `PrismaService`
- `RedisService`
- `RetryHandlerService`
- `ActionExecutorService`
- `StateResolverService`
- `AuthContextService`
- `PageSettleService`
- `ConfigService`
- `DiscoveryService`

Where it refuses work

- `BatchScreenshotService` stops the work with `Error` when `status >= 400`.
- `BatchScreenshotService` stops the work with `Error` when `!hasContent`.
- `BatchScreenshotService` stops the work with `Error` when `!reauthed`.
- `BatchScreenshotService` stops the work with an early return when `intended.origin !== actual.origin`.
- `BatchScreenshotService` stops the work with an early return when `intendedPath === actualPath`.
- `BatchScreenshotService` stops the work with an early return when `normalizePath(intended.pathname) === '/web' && normalizePath(actual.pathname) === '/web'`.

When something fails

- `BatchScreenshotService` handles failure in 7 places: it logs it and continues in 4, and turns it into a return value in 3.

Diagram

```
sequenceDiagram
  participant Client
  participant Service as BatchScreenshotService
  participant Cache as Progress/Cache Store
  participant Auth as Authentication Session
  participant Worker as Screenshot Worker
  participant Browser as Browser/Page

  Client->>Service: processBatch(tasks)
  Service->>Cache: Load batch progress and cached results
  Service->>Service: Deduplicate content/tasks
  Service->>Auth: Create or reuse authenticated session

  loop Until all unique tasks are processed
    Service->>Worker: Start task (within concurrency limit)
    Worker->>Browser: Navigate and capture screenshot

    alt Screenshot succeeds
      Browser-->>Worker: Screenshot result
      Worker-->>Service: Completed result
      Service->>Cache: Store result and update progress
    else Transient failure
      Browser-->>Worker: Error
      Worker-->>Service: Failed attempt
      Service->>Service: Wait with exponential backoff
      Service->>Worker: Retry task
    else Retries exhausted
      Worker-->>Service: Final failure
      Service->>Cache: Record failed progress state
    end
  end

  Service-->>Client: Batch processing summary
```

Usage

```
import { Injectable } from '@nestjs/common';
import { BatchScreenshotService } from './screenshot/services/batch-screenshot.service';

@Injectable()
export class ScreenshotJobProcessor {
  constructor(
    private readonly batchScreenshotService: BatchScreenshotService,
  ) {}

  async processScreenshots() {
    const result = await this.batchScreenshotService.processBatch([
      {
        id: 'homepage',
```

```
        url: 'https://example.com',
        viewport: { width: 1440, height: 900 },
      },
      {
        id: 'dashboard',
        url: 'https://example.com/dashboard',
        viewport: { width: 1440, height: 900 },
      },
    ]);

    return {
      completed: result.completed,
      failed: result.failed,
      cached: result.cached,
    };
  }
}
```

AI Coding Instructions

- Preserve bounded concurrency when adding new task types; do not create unbounded `Promise.all()` workloads for large batches.
- Reuse the service's authentication-session lifecycle rather than creating a new browser login session per screenshot.
- Treat retryable browser, navigation, and network errors separately from permanent validation errors; retain exponential backoff behavior.
- Update progress/cache state for successful, cached, and terminally failed tasks so interrupted batches can be resumed safely.
- Ensure deduplication keys include all screenshot-affecting inputs, such as URL, viewport, authentication context, and rendering options.

Relationships

- DEPENDS_ON → `PlaywrightService`
- DEPENDS_ON → `StorageService`
- DEPENDS_ON → `ScreenshotMetadataService`
- DEPENDS_ON → `PrismaService`
- DEPENDS_ON → `RedisService`
- DEPENDS_ON → `RetryHandlerService`
- DEPENDS_ON → `ActionExecutorService`
- DEPENDS_ON → `StateResolverService`
- DEPENDS_ON → `AuthContextService`

- DEPENDS_ON → `PageSettleService`
- DEPENDS_ON → `configservice`
- DEPENDS_ON → `DiscoveryService`

Referenced By

- `BatchScreenshotController` (DEPENDS_ON)
- `ScreenshotModule` (MODULE_PROVIDES)
- `ScreenshotModule` (MODULE_EXPORTS)