RetryHandlerService

September 4, 2026

RetryHandlerService

Kind: Service

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

`RetryHandlerService` centralizes retry behavior for screenshot-worker operations that may fail transiently, such as network requests or temporary browser failures. It classifies errors as retryable, calculates exponential backoff delays, and reruns asynchronous work until it succeeds or the retry limit is reached.

Methods

Method	Signature	Returns	Description
`isRetryableError`	`isRetryableError(error: Error	string	null
`calculateBackoffDelay`	`calculateBackoffDelay(attempt: number, config: RetryConfig)`	`number`	Calculates the backoff delay for a given attempt Formula: min(initialDelayMs * (backoffMultiplier ^ (attempt - 1)), maxDelayMs)
`retryWithBackoff`	`retryWithBackoff(fn: () => Promise<T>, config: RetryConfig)`	`Promise<T>`	Executes a function with automatic retry and exponential backoff

Where it refuses work

- `RetryHandlerService` stops the work with an early return when `regex.test(errorMessage) || regex.test(errorName)`, in 2 places.
- `RetryHandlerService` stops the work with an early return when `!error`.
- `RetryHandlerService` stops the work with an early return when `NON_RETRYABLE_ERROR_NAMES.includes(errorName)`.
- `RetryHandlerService` stops the work with an early return when `attempt <= 0`.

When something fails

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

Diagram

```
sequenceDiagram
    participant Caller
    participant RetryHandlerService
    participant Operation

    Caller->>RetryHandlerService: retryWithBackoff(operation)
    RetryHandlerService->>Operation: execute()

    alt Operation succeeds
        Operation-->>RetryHandlerService: result
        RetryHandlerService-->>Caller: result
    else Operation fails
        Operation-->>RetryHandlerService: error
        RetryHandlerService->>RetryHandlerService: isRetryableError(error)

        alt Retryable and attempts remain
            RetryHandlerService->>RetryHandlerService: calculateBackoffDelay(attempt)
            RetryHandlerService->>RetryHandlerService: wait(delay)
            RetryHandlerService->>Operation: execute again
        else Not retryable or limit reached
            RetryHandlerService-->>Caller: throw error
        end
    end
```

Usage

```
import { Injectable } from '@nestjs/common';
import { RetryHandlerService } from './retry-handler.service';

@Injectable()
export class ScreenshotGenerationService {
  constructor(
```

```
    private readonly retryHandlerService: RetryHandlerService,
  ) {}

  async captureScreenshot(url: string): Promise<Buffer> {
    return this.retryHandlerService.retryWithBackoff(async () => {
      const response = await fetch(url);

      if (!response.ok) {
        throw new Error(`Failed to load page: ${response.status}`);
      }

      return Buffer.from(await response.arrayBuffer());
    });
  }
}
```

AI Coding Instructions

- Use `retryWithBackoff` for asynchronous operations that can safely be repeated without creating duplicate side effects.
- Keep retry classification in `isRetryableError`; add new transient error types there rather than duplicating retry checks in callers.
- Preserve the backoff calculation in `calculateBackoffDelay` so retry timing remains consistent across screenshot-worker services.
- Do not retry validation errors, malformed input, authorization failures, or other deterministic failures.
- Ensure callers log enough context about the failed operation, but avoid logging sensitive URLs, credentials, or request payloads.

Referenced By

- `ScreenshotModule` (MODULE_PROVIDES)
- `BatchScreenshotService` (DEPENDS_ON)