AuthContextService

September 4, 2026

AuthContextService

Kind: Service

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

Service for authenticated browser-context lifecycle.

Extracted verbatim from BatchScreenshotService so that batch capture,
flow-replay (WS-A) and clip capture (WS-B) can all share ONE implementation
of login / session-loss detection / re-auth without forking it.

`AuthContextService` owns the lifecycle of authenticated browser contexts for screenshot-worker capture jobs. It centralizes login, session validation, session-loss detection, and re-authentication so batch screenshots, flow replay, and clip capture use the same authentication behavior without duplicating browser-session logic.

Methods

Method	Signature	Returns	Description
`createAuthenticatedContext`	`createAuthenticatedContext(browser: Browser, auth: BatchAuthConfigDto, config: Required<BatchConfigDto>, locale: string)`	`Promise<BrowserContext>`	Create authenticated browser context For SPA logins, we keep the login page open and navigate from it instead of closing it.
`isSessionLost`	`isSessionLost(page: Page, auth: BatchAuthConfigDto, targetUrl: string)`	`Promise<boolean>`	Check if session is lost (page redirected to login) Simple check: if URL contains login patterns, session is

Method	Signature	Returns	Description
			lost BUT: if the target URL itself was a login p…
`reAuthenticate`	`reAuthenticate(context: BrowserContext, auth: BatchAuthConfigDto, config: Required<BatchConfigDto>)`	`Promise<void>`	Re-authenticate when session is lost Creates a new page in the context, performs login, then closes it.

Where it refuses work

- `AuthContextService` stops the work with `Error` when `auth.useSmartDetection` — “Smart detection is not available. Please provide explicit login selectors (usernameSelect…”, in 2 places.
- `AuthContextService` stops the work with `Error` when `!usernameSelector || !passwordSelector || !submitSelector` — “Missing login selectors (username, password, submit)”.
- `AuthContextService` stops the work with `Error` when `!usernameSelector || !passwordSelector || !submitSelector` — “Missing login selectors for re-authentication”.
- `AuthContextService` stops the work with an early return when `!href || href === url`.
- `AuthContextService` stops the work with an early return when `visible`.

When something fails

- `AuthContextService` handles failure in 7 places: it discards it silently in 4, lets it reach the caller in 2, and logs it and continues in 1.

Diagram

```
sequenceDiagram
    participant Caller as Capture Service
    participant Auth as AuthContextService
    participant Browser as Browser Context
    participant App as Target Application

    Caller->>Auth: acquireAuthenticatedContext(credentials)
    Auth->>Browser: Create or reuse browser context
    Auth->>App: Check existing session

    alt Session is valid
        App-->>Auth: Authenticated session detected
```

```
    else Session missing or expired
        Auth->>App: Perform login flow
        App-->>Auth: Session established
    end

    Auth-->>Caller: Authenticated browser context
    Caller->>App: Run screenshot / replay / clip capture

    alt Session loss detected during capture
        Caller->>Auth: Re-authenticate context
        Auth->>App: Perform login flow again
        Auth-->>Caller: Refreshed authenticated context
    end
```

Usage

```
import { Injectable } from '@nestjs/common';
import { AuthContextService } from './services/auth-context.service';

@Injectable()
export class BatchCaptureService {
  constructor(
    private readonly authContextService: AuthContextService,
  ) {}

  async captureScreenshot(job: ScreenshotJob) {
    const context = await this.authContextService.acquireAuthenticatedContext({
      loginUrl: job.loginUrl,
      username: job.username,
      password: job.password,
    });

    try {
      const page = await context.newPage();

      await page.goto(job.targetUrl, { waitUntil: 'networkidle' });

      // Re-authenticate when the target application redirects to login
      // or otherwise indicates that the session has expired.
      if (await this.authContextService.isSessionLost(page)) {
        await this.authContextService.reauthenticate(context, job.credentials);
        await page.goto(job.targetUrl, { waitUntil: 'networkidle' });
      }

      return await page.screenshot({ fullPage: true });
    } finally {
      await context.close();
    }
  }
}
```

AI Coding Instructions

- Use `AuthContextService` for every authenticated browser workflow; do not implement login or session-expiry handling independently in batch, flow-replay, or clip services.
- Check for session loss after navigation and before capture-sensitive actions, especially when target applications can redirect to a login page.
- Keep browser-context ownership clear: callers should close contexts/pages they acquire unless the service explicitly manages pooled or reused contexts.
- Preserve the shared authentication contract when adding new capture modes so login credentials, session checks, and re-authentication remain consistent.
- Avoid treating a successful page load as proof of authentication; use the service's session-detection logic for application-specific validation.

Referenced By

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