FlowReplayService

September 4, 2026

FlowReplayService

Kind: Service

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

FlowReplayService (WS-A) — the sequential single-context walk.

Replays a recorded flow in ONE browser context / ONE page, shooting a screenshot after each action (plus one for the initial state). The first shot is uploaded immediately ("first shot fast") so the reader sees something while the rest of the walk runs. Every step is best-effort: a failed step is logged and the walk continues, mirroring the batch service's resilience.

All heavy lifting reuses the Phase-0 foundation (AuthContextService, PageSettleService, ActionExecutorService, AnnotationEmitterService, crop helpers, StorageService, ScreenshotMetadataService) — nothing is forked.

`FlowReplayService` replays a recorded user flow sequentially in a single browser context and single page, capturing a screenshot for the initial state and after each action. It prioritizes uploading the first screenshot immediately (“first shot fast”) so consumers see early feedback while the remaining steps continue. Each step is executed best-effort: failures are logged and the replay continues, reusing the shared Phase-0 services (auth, settle, action execution, annotations, cropping, storage, metadata).

Methods

Method	Signature	Returns	Description
`createFlowJob`	`createFlowJob(dto: FlowReplayRequestDto)`	`Promise<BatchScreenshotJobResponseDto>`	Create the ScreenshotJob row for a flow replay (synchronous — returns fast).
`replayFlow`	`replayFlow(jobId: string, dto: FlowReplayRequestDto)`	`Promise<void>`	Replay the flow.

Method	Signature	Returns	Description
`getFlowSteps`	`getFlowSteps(jobId: string)`	`Promise<FlowStepResult[]	undefined>`
`reshootStep`	`reshootStep(dto: FlowReshootRequestDto)`	`Promise<FlowStepResult	null>`
`replayForLocales`	`replayForLocales(dto: FlowLocalesRequestDto)`	`Promise<LocaleFlowResult[]>`	SUPERIOR (F2): replay the WHOLE recipe once per locale in one call, emitting a localized screenshot set per locale under a locale-namespaced blob key (flows/…
`validateFlow`	`validateFlow(dto: FlowValidateRequestDto)`	`Promise<FlowValidationReport>`	Walk the recipe WITHOUT screenshots and report, per step, whether its selector still resolves on the live app.

Dependencies

- `PlaywrightService`
- `StorageService`
- `ScreenshotMetadataService`
- `PrismaService`
- `RedisService`
- `ActionExecutorService`
- `AuthContextService`
- `PageSettleService`
- `AnnotationEmitterService`
- `ConfigService`

Where it refuses work

- `FlowReplayService` stops the work with an early return when `!this.redis.isAvailable()`, in 2 places.
- `FlowReplayService` stops the work with an early return when `unique(s)`, in 2 places.
- `FlowReplayService` stops the work with an early return when `!redactions || redactions.length === 0`.
- `FlowReplayService` stops the work with an early return when `dto.auth`.
- `FlowReplayService` stops the work with an early return when `step.value`.
- `FlowReplayService` stops the work with an early return when `step.selector`.

When something fails

- `FlowReplayService` handles failure in 13 places: it logs it and continues in 10, discards it silently in 2, and turns it into a return value in 1. A failure discarded silently leaves no trace for whoever debugs this later.

Diagram

```
sequenceDiagram
  autonumber
  participant Caller
  participant FlowReplayService as FlowReplayService
  participant Auth as AuthContextService
  participant PageSettle as PageSettleService
  participant Exec as ActionExecutorService
  participant Annot as AnnotationEmitterService
  participant Shot as ScreenshotMetadataService
  participant Storage as StorageService

  Caller->>FlowReplayService: replay(flowRecording, options)
  FlowReplayService->>Auth: init single browser context + page
  FlowReplayService->>PageSettle: settle initial state
  FlowReplayService->>Annot: emit annotations (initial)
  FlowReplayService->>Shot: capture + build metadata (initial)
  FlowReplayService->>Storage: upload(initial) (first shot fast)
  FlowReplayService-->>Caller: emit/return initial result (early)

  loop each recorded action
    FlowReplayService->>Exec: execute(action)
    alt action fails
      FlowReplayService->>FlowReplayService: log error, continue
    end
    FlowReplayService->>PageSettle: settle after action
    FlowReplayService->>Annot: emit annotations (step)
    FlowReplayService->>Shot: capture + build metadata (step)
    FlowReplayService->>Storage: upload(step)
  end
```

```
  FlowReplayService->>Auth: dispose/cleanup context
  FlowReplayService-->>Caller: final replay result
```

Usage

```ts
import { Injectable } from '@nestjs/common';
import { FlowReplayService } from './screenshot/services/flow-replay.service';

@Injectable()
export class FlowReplayRunner {
  constructor(private readonly flowReplay: FlowReplayService) {}

  async runRecordedFlow(flowRecording: any) {
    // Typical call shape: pass the recorded flow + any replay options.
    // (Exact DTOs/options depend on your app; this illustrates usage.)
    const result = await this.flowReplay.replay(flowRecording, {
      // e.g. storagePrefix, viewport, crop, annotations, etc.
      firstShotFast: true,
    });

    // result usually includes per-step screenshot refs/metadata
    // and any logged step errors (best-effort execution).
    return result;
  }
}
```

AI Coding Instructions

- Keep replays strictly single-context / single-page; do not introduce parallel pages here—use the batch/parallel service for that.
- Preserve the “first shot fast” behavior: upload/emit the initial screenshot as early as possible before executing the rest of the steps.
- Treat each action as best-effort: catch and log step failures, then continue; avoid failing the entire replay unless setup/teardown is broken.
- Reuse Phase-0 integration points (AuthContextService, PageSettleService, ActionExecutorService, AnnotationEmitterService, crop helpers, StorageService, ScreenshotMetadataService) instead of duplicating logic.
- Always run settle + annotation emission before capturing screenshots to avoid flakey, mid-transition images and inconsistent overlays.

Relationships

- DEPENDS_ON → `PlaywrightService`
- DEPENDS_ON → `StorageService`
- DEPENDS_ON → `ScreenshotMetadataService`

- DEPENDS_ON → `PrismaService`
- DEPENDS_ON → `RedisService`
- DEPENDS_ON → `ActionExecutorService`
- DEPENDS_ON → `AuthContextService`
- DEPENDS_ON → `PageSettleService`
- DEPENDS_ON → `AnnotationEmitterService`
- DEPENDS_ON → `configservice`

Referenced By

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