BatchScreenshotController

September 4, 2026

BatchScreenshotController

Kind: Controller

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

Controller for batch screenshot operations

Endpoints:

- POST /screenshot/batch - Create and start a batch job
- GET /screenshot/batch/:jobId/status - Get job status and results
- POST /screenshot/validate-selectors - Validate selectors exist on a page

`BatchScreenshotController` exposes HTTP endpoints for creating and running screenshot batch jobs, querying their status/results, and validating CSS/XPath selectors against a target page. It acts as the API boundary for the screenshot-worker service, delegating orchestration to underlying services/queues and returning job progress and artifacts to callers.

Diagram

```
graph LR
  Client[Client / API Consumer] -->|POST /screenshot/batch| C[BatchScreenshotController]
  Client -->|GET /screenshot/batch/:jobId/status| C
  Client -->|POST /screenshot/validate-selectors| C

  C --> S[Batch Screenshot Service]
  S --> Q[(Job Queue / Worker)]
  Q --> W[Browser Runner (Playwright/Puppeteer)]
  W --> R[(Storage: screenshots & metadata)]
  S --> R
  C -->|status/results| Client
```

Usage

```
// Example: calling the controller endpoints from a Node client (e.g., another service)
import axios from "axios";

const baseURL = process.env.SCREENSHOT_WORKER_URL ?? "http://localhost:3000";

async function runBatch() {
  // 1) Create/start a batch screenshot job
  const createRes = await axios.post(`${baseURL}/screenshot/batch`, {
    // Shape depends on your DTOs; adjust to match your API contract.
    pages: [
      { url: "https://example.com", selectors: ["header", "#main"] },
      { url: "https://example.com/about", selectors: ["h1", ".content"] },
    ],
    viewport: { width: 1280, height: 720 },
    fullPage: true,
  });

  const jobId: string = createRes.data.jobId;
  console.log("Created job:", jobId);

  // 2) Poll for job status/results
  while (true) {
    const statusRes = await axios.get(
      `${baseURL}/screenshot/batch/${encodeURIComponent(jobId)}/status`,
    );

    const { status, progress, results, error } = statusRes.data;
    console.log({ status, progress });

    if (status === "completed") {
      // results commonly include screenshot URLs/paths and per-page metadata
      console.log("Batch results:", results);
      break;
    }

    if (status === "failed") {
      throw new Error(error ?? "Batch screenshot job failed");
    }

    await new Promise((r) => setTimeout(r, 1000));
  }
}

async function validateSelectors() {
  const res = await axios.post(`${baseURL}/screenshot/validate-selectors`, {
    url: "https://example.com",
    selectors: ["#main", ".does-not-exist"],
  });
```

```
  // Example response: { valid: ["#main"], invalid: [".does-not-exist"] }
  console.log("Selector validation:", res.data);
}

runBatch()
  .then(validateSelectors)
  .catch((e) => {
    console.error(e);
    process.exit(1);
  });
```

AI Coding Instructions

- Keep controller methods thin: perform input validation/DTO parsing and delegate orchestration to a service/queue layer; avoid embedding browser/screenshot logic in the controller.
- Preserve idempotency and clarity for `GET /screenshot/batch/:jobId/status`: never mutate job state on reads; return a stable schema for `status`, `progress`, and `results`.
- Normalize and validate all external inputs (URLs, selectors, viewport/fullPage flags) to prevent SSRF-like issues and to avoid expensive browser launches for obviously invalid requests.
- Treat job IDs as opaque: always `encodeURIComponent` in routing/clients and avoid assumptions about format when adding logging or storage keys.
- When adding new endpoints, align with existing NestJS patterns (DTOs, validation pipes, consistent HTTP status codes) and ensure integration points (queue, storage, runner) are injected and testable.

Relationships

- MODULE_DECLARES → `createBatchJob`
- MODULE_DECLARES → `createFlowJob`
- MODULE_DECLARES → `reshootFlowStep`
- MODULE_DECLARES → `reshootFlowLocales`
- MODULE_DECLARES → `validateFlow`
- MODULE_DECLARES → `getJobStatus`
- MODULE_DECLARES → `getPageHtml`
- MODULE_DECLARES → `validateSelectors`
- DEPENDS_ON → `BatchScreenshotService`

- DEPENDS_ON → `FlowReplayService`
- DEPENDS_ON → `PlaywrightService`

Referenced By

- `ScreenshotModule` (MODULE_DECLARES)