CaptureService

September 4, 2026

CaptureService

Kind: Service

Source: `atloria-monorepo/apps/api/src/capture/capture.service.ts`

`CaptureService` manages capture flows and their associated steps within the API. It supports creating, retrieving, listing, updating, deleting, and publishing flows, including reshooting individual steps or localized capture content. The service is used by controllers and other backend components to coordinate flow lifecycle operations.

Methods

Method	Signature	Returns	Description
`createFlow`	`createFlow(projectId: string, user: JwtPayload, dto: CreateFlowDto)`	`unknown`	Store a recorded flow.
`listFlows`	`listFlows(projectId: string, user: JwtPayload)`	`unknown`	List flows for a project (lightweight summary).
`deleteFlow`	`deleteFlow(projectId: string, user: JwtPayload, id: string)`	`unknown`	Delete a flow.
`getFlow`	`getFlow(projectId: string, user: JwtPayload, id: string)`	`unknown`	Full captured flow for the owner's step-list editor (F2).
`updateSteps`	`updateSteps(projectId: string, user: JwtPayload, id: string, dto: UpdateFlowStepsDto)`	`unknown`	Persist owner edits to a flow's step list (reorder / delete / caption / redaction boxes).
`reshootStep`	`reshootStep(projectId: string, user: JwtPayload, id: string, stepIndex: number, opts: ReshootStepDto)`	`unknown`	Re-shoot ONE step of a stored flow (F2).

Method	Signature	Returns	Description
`reshootLocales`	`reshootLocales(projectId: string, user: JwtPayload, id: string, dto: ReshootLocalesDto)`	`unknown`	SUPERIOR (F2): re-shoot the whole recipe under N locales in one run, returning a localized screenshot set per locale.
`createDraft`	`createDraft(projectId: string, user: JwtPayload, id: string)`	`unknown`	Turn a captured flow into a DRAFT manual Document.
`getPublicFlow`	`getPublicFlow(id: string)`	`unknown`	Full CapturedFlow for the tour player on the customer's site.

Dependencies

- `PrismaService`
- `DraftDocumentService`
- `FlowDraftStitcherService`

Where it refuses work

- `CaptureService` stops the work with `NotFoundException` when `!flow` — “Flow not found”, in 6 places.
- `CaptureService` stops the work with `BadRequestException` when `dto.steps.length > MAX_STEPS`, in 2 places.
- `CaptureService` stops the work with `BadRequestException` when `!Array.isArray(dto.steps) || dto.steps.length === 0` — “A flow must contain at least one step.”.
- `CaptureService` stops the work with `NotFoundException` when `!existing` — “Flow not found”.
- `CaptureService` stops the work with `BadRequestException` when `!Array.isArray(dto.steps) || dto.steps.length === 0` — “A flow must keep at least one step.”.
- `CaptureService` stops the work with `BadRequestException` when `!Number.isInteger(stepIndex) || stepIndex < 0 || stepIndex > steps.length`.

When something fails

- `CaptureService` handles failure in 3 places: it logs it and continues in 2, and turns it into a return value in 1.

Diagram

```
sequenceDiagram
  participant Client
  participant Controller as CaptureController
  participant Service as CaptureService
  participant Store as Persistence Layer

  Client->>Controller: Create or update capture flow
  Controller->>Service: createFlow() / updateSteps()
  Service->>Store: Persist flow and steps
  Store-->>Service: Saved flow
  Service-->>Controller: Flow result
  Controller-->>Client: API response

  Client->>Controller: Request public flow
  Controller->>Service: getPublicFlow()
  Service->>Store: Load published flow
  Store-->>Service: Public flow data
  Service-->>Controller: Public flow
  Controller-->>Client: API response
```

Usage

```
import { CaptureService } from './capture.service';

export class CaptureController {
  constructor(private readonly captureService: CaptureService) {}

  async createCaptureFlow() {
    const flow = await this.captureService.createFlow();

    return flow;
  }

  async updateCaptureSteps() {
    const updatedFlow = await this.captureService.updateSteps();

    return updatedFlow;
  }

  async reshootStep() {
    return this.captureService.reshootStep();
  }
```

```
  async getPublicCaptureFlow() {
    return this.captureService.getPublicFlow();
  }
}
```

AI Coding Instructions

- Keep capture-flow lifecycle logic inside `CaptureService`; controllers should validate input and delegate to the service.
- Use `createFlow`, `getFlow`, and `listFlows` for authenticated or internal flow management, and use `getPublicFlow` only for published/public access paths.
- Update capture content through `updateSteps`; use `reshootStep` or `reshootLocales` when replacing previously captured assets rather than creating duplicate steps.
- Ensure deletion through `deleteFlow` also handles related step, draft, and asset cleanup according to persistence-layer rules.
- Preserve authorization and ownership checks when adding new flow operations, especially for public-flow and reshoot functionality.

Relationships

- DEPENDS_ON → `PrismaService`
- DEPENDS_ON → `DraftDocumentService`
- DEPENDS_ON → `FlowDraftStitcherService`

Referenced By

- `CaptureController` (DEPENDS_ON)
- `CaptureModule` (MODULE_PROVIDES)
- `CaptureModule` (MODULE_EXPORTS)
- `PublicCaptureController` (DEPENDS_ON)