# ToursService

September 4, 2026

## ToursService

**Kind:** Service

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

ToursService (D2) — CRUD + publishing + the server-side evaluated public config the embedded widget consumes. Tenancy: routes are guarded by ResourceOrgGuard on :projectId; the service ALSO re-verifies (belt and braces, mirroring CaptureService.verifyAccess) and pins every child lookup to the projectId so a foreign experience id can never resolve.

`ToursService` manages the lifecycle of tours/experiences within a project: creation, retrieval, updates, publishing, pausing, archiving, deletion, and funnel reporting. It enforces tenant isolation by re-verifying project access and scoping every child experience lookup to `projectId`, and it generates the server-evaluated public configuration consumed by the embedded widget.

### Methods

| Method | Signature | Returns | Description |
|---|---|---|---|
| `create` | `create(projectId: string, user: JwtPayload, dto: CreateExperienceDto)` | `unknown` | |
| `list` | `list(projectId: string, user: JwtPayload)` | `unknown` | |
| `get` | `get(projectId: string, user: JwtPayload, id: string)` | `unknown` | |
| `update` | `update(projectId: string, user: JwtPayload, id: string, dto: UpdateExperienceDto)` | `unknown` | |
| `publish` | `publish(projectId: string, user: JwtPayload, id: string)` | `unknown` | draft → live. |
| `pause` | `pause(projectId: string, user: JwtPayload, id: string)` | `unknown` | live → draft (paused). |

| Method | Signature | Returns | Description |
|---|---|---|---|
| `archive` | `archive(projectId: string, user: JwtPayload, id: string)` | `unknown` | Soft retire — stays queryable for analytics, never served. |
| `remove` | `remove(projectId: string, user: JwtPayload, id: string)` | `unknown` | |
| `funnel` | `funnel(projectId: string, user: JwtPayload, id: string, days: unknown)` | `unknown` | Per-experience funnel from the interaction_events spine: started → per-step viewers → completed, plus dismissals, missing-element counts (self-healing signal… |
| `getPublicConfig` | `getPublicConfig(projectParam: string, url: string, hostSegments: string[], readerToken: string)` | `unknown` | Everything the widget may run on `url` for this project. |
| `resolveProjectId` | `resolveProjectId(projectParam: string)` | `Promise<string>` | Resolve the widget's `data-project` to a real project id. |
| `getVisitorChecklist` | `getVisitorChecklist(projectParam: string, experienceId: string, visitorId: string)` | `Promise<{ items: Record<string, boolean> }>` | Rehydrate a visitor's ticked checklist items ({ "": true }). |
| `setVisitorChecklistItem` | `setVisitorChecklistItem(projectParam: string, experienceId: string, visitorId: string, itemKey: string, done: boolean)` | `Promise<{ ok: true }>` | Persist one checklist item's done-state for a visitor (upsert). |
| `resolveProjectOrg` | `resolveProjectOrg(projectId: string)` | `Promise<string | null>` | |
| `findPublicExperience` | `findPublicExperience(projectId: string, experienceId: string)` | `unknown` | Public lookup used by the events endpoint: experience must belong to the project. |
| `invalidateConfig` | `invalidateConfig(projectId: string)` | `Promise<void>` | |

## Dependencies

- `PrismaService`
- `RedisService`
- `ReaderTokenService`

## Where it refuses work

- `ToursService` stops the work with `BadRequestException` when `!TOUR_KINDS.includes(dto.kind)`.
- `ToursService` stops the work with `BadRequestException` when `!dto.flowId` — “A tour experience needs a flowId.”.
- `ToursService` stops the work with `BadRequestException` when `row.status === 'archived'` — “Un-archive (edit) the experience before publishing.”.
- `ToursService` stops the work with `BadRequestException` when `steps.length === 0` — “This tour has no steps to play — record a flow first.”.
- `ToursService` stops the work with `BadRequestException` when `!projectParam` — “projectId is required”.
- `ToursService` stops the work with `NotFoundException` when `!row` — “Experience not found”.

## When something fails

- `ToursService` handles failure in 1 place: it discards it silently in all 1.

## Diagram

```
sequenceDiagram
    participant Client
    participant Controller as ToursController
    participant Guard as ResourceOrgGuard
    participant Service as ToursService
    participant DB as Database
    participant Widget as Embedded Widget

    Client->>Controller: Request for /projects/:projectId/tours
    Controller->>Guard: Validate project organization access
    Guard-->>Controller: Access granted
    Controller->>Service: create/list/get/update(..., projectId)

    Service->>Service: Re-verify project access
    Service->>DB: Query tour scoped by projectId
    DB-->>Service: Project-scoped result
    Service-->>Controller: Tour response
    Controller-->>Client: API response

    Widget->>Controller: Request public tour config
    Controller->>Service: getPublicConfig(projectId, context)
    Service->>DB: Load published, project-scoped tours
    DB-->>Service: Eligible experiences
```

```
    Service->>Service: Evaluate public targeting/configuration
    Service-->>Widget: Public widget configuration
```

## Usage

```
import { Injectable } from '@nestjs/common';
import { ToursService } from './tours.service';

@Injectable()
export class TourAdminService {
  constructor(private readonly toursService: ToursService) {}

  async publishOnboardingTour(projectId: string, tourId: string) {
    // ToursService must receive the project scope for every operation.
    const tour = await this.toursService.get(projectId, tourId);

    await this.toursService.update(projectId, tourId, {
      name: 'Updated onboarding tour',
      // Additional tour configuration...
    });

    return this.toursService.publish(projectId, tour.id);
  }

  async getWidgetConfig(projectId: string) {
    // Used by the public/embed delivery path to retrieve evaluated config.
    return this.toursService.getPublicConfig(projectId);
  }
}
```

## AI Coding Instructions

- Always pass and enforce `projectId` for tour and experience operations; never load a child resource by its ID alone.
- Preserve the service-level access verification even when routes are protected by `ResourceOrgGuard`; this is intentional defense in depth.
- Use lifecycle methods ( `publish` , `pause` , `archive` ) rather than directly mutating publication or status fields.
- Keep `getPublicConfig()` limited to safe, published, server-evaluated data intended for the embedded widget; do not expose internal admin configuration.
- When adding related-resource queries, scope them to both the resource identifier and `projectId` to prevent cross-tenant resolution.

## Relationships

- DEPENDS_ON → `PrismaService`
- DEPENDS_ON → `RedisService`

- DEPENDS_ON → `ReaderTokenService`

## Referenced By

- `PublicToursController` (DEPENDS_ON)
- `ToursController` (DEPENDS_ON)
- `ToursModule` (MODULE_PROVIDES)
- `ToursModule` (MODULE_EXPORTS)