BoardsService

September 4, 2026

BoardsService

Kind: Service

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

`BoardsService` encapsulates backend business logic for managing boards and their columns in the Atloria API. It provides CRUD operations, restore support, and column-specific workflows such as adding, updating, reordering, and deleting columns while coordinating persistence and validation for board-related endpoints.

Methods

Method	Signature	Returns	Description
`list`	`list(projectId: string, user: JwtPayload)`	`unknown`	List all (non-deleted) boards for a project
`create`	`create(projectId: string, dto: CreateBoardDto, user: JwtPayload)`	`unknown`	Create a new board (optionally with template columns + seed cards)
`getOne`	`getOne(projectId: string, boardId: string, user: JwtPayload, includeCards: unknown)`	`unknown`	Get a board with all columns and (optionally) cards
`update`	`update(projectId: string, boardId: string, dto: UpdateBoardDto, user: JwtPayload)`	`unknown`	Update board metadata

Method	Signature	Returns	Description
`remove`	`remove(projectId: string, boardId: string, user: JwtPayload)`	`unknown`	Soft-delete a board (default board cannot be deleted)
`restore`	`restore(projectId: string, boardId: string, user: JwtPayload)`	`unknown`	Restore a soft-deleted board (within the 30-day window)
`addColumn`	`addColumn(projectId: string, boardId: string, dto: CreateColumnDto, user: JwtPayload)`	`unknown`	
`updateColumn`	`updateColumn(projectId: string, boardId: string, columnId: string, dto: UpdateColumnDto, user: JwtPayload)`	`unknown`	
`reorderColumns`	`reorderColumns(projectId: string, boardId: string, dto: ReorderColumnsDto, user: JwtPayload)`	`unknown`	
`deleteColumn`	`deleteColumn(projectId: string, boardId: string, columnId: string, moveCardsTo: string | undefined, user: JwtPayload)`	`unknown`	
`listCards`	`listCards(projectId: string, boardId: string, since: Date | undefined, user: JwtPayload)`	`unknown`	
`addCards`	`addCards(projectId: string, boardId: string, dto: AddCardsDto, user: JwtPayload)`	`unknown`	
`moveCard`	`moveCard(projectId: string, boardId: string, cardId: string, dto: MoveCardDto, user: JwtPayload)`	`unknown`	
`removeCard`	`removeCard(projectId: string, boardId: string, cardId: string, user: JwtPayload)`	`unknown`	

Method	Signature	Returns	Description
`getBoardsForIssue`	`getBoardsForIssue(projectId: string, issueId: string, user: JwtPayload)`	`unknown`	Find boards an issue is on (used by issue detail page sidebar)

Dependencies

- `PrismaService`
- `IssuesService`

Where it refuses work

- `BoardsService` stops the work with `NotFoundException` when `!board` — “Board not found”, in 3 places.
- `BoardsService` stops the work with `NotFoundException` when `!card` — “Card not found”, in 2 places.
- `BoardsService` stops the work with `NotFoundException` when `!project` — “Project not found”.
- `BoardsService` stops the work with `ForbiddenException` when `project.organizationId !== user.organizationId` — “You do not have access to this project”.
- `BoardsService` stops the work with `NotFoundException` when `!board` — “Deleted board not found”.
- `BoardsService` stops the work with `BadRequestException` when `dto.columnIds.length !== cols.length || !dto.columnIds.every((id) => existingIds.has(id))` — “columnIds must list every existing column id exactly once”.

Diagram

```
sequenceDiagram
  participant Client
  participant Controller as BoardsController
  participant Service as BoardsService
  participant Database as Persistence Layer

  Client->>Controller: Create or update board request
  Controller->>Service: create() / update()
  Service->>Database: Validate and persist board data
  Database-->>Service: Board result
  Service-->>Controller: Board response
```

```
  Controller-->>Client: HTTP response

  Client->>Controller: Reorder board columns
  Controller->>Service: reorderColumns()
  Service->>Database: Update column ordering
  Database-->>Service: Updated board columns
  Service-->>Controller: Reordered columns
  Controller-->>Client: HTTP response
```

Usage

```
import { Injectable } from '@nestjs/common';
import { BoardsService } from './boards.service';

@Injectable()
export class BoardWorkflowService {
  constructor(private readonly boardsService: BoardsService) {}

  async createBoardWithFirstColumn(userId: string) {
    const board = await this.boardsService.create({
      name: 'Product Roadmap',
      ownerId: userId,
    });

    await this.boardsService.addColumn(board.id, {
      name: 'Backlog',
      order: 0,
    });

    return this.boardsService.getOne(board.id);
  }

  async moveColumns(boardId: string, columnIds: string[]) {
    return this.boardsService.reorderColumns(boardId, columnIds);
  }
}
```

AI Coding Instructions

- Keep board and column mutations in `BoardsService`; controllers should only handle request parsing, guards, and response mapping.
- Use `getOne()` before dependent operations when the workflow requires verifying that a board exists or is accessible.
- Preserve column ordering invariants when implementing `addColumn()`, `updateColumn()`, `reorderColumns()`, or `deleteColumn()`.

- Treat `remove()` and `restore()` as lifecycle operations; do not permanently delete related data unless the intended deletion policy explicitly requires it.
- Ensure authorization and tenant/workspace ownership checks are consistently applied to list, read, update, and column-management operations.

Relationships

- DEPENDS_ON → `PrismaService`
- DEPENDS_ON → `IssuesService`

Referenced By

- `BoardsController` (DEPENDS_ON)
- `BoardsModule` (MODULE_PROVIDES)
- `BoardsModule` (MODULE_EXPORTS)