IssuesService

September 4, 2026

IssuesService

Kind: Service

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

`IssuesService` encapsulates the backend workflow for customer support issues and tickets. It handles public ticket creation, chat-escalation intake, customer replies, CSAT recording, resolution notifications, and administrative ticket retrieval, updates, statistics, and listing.

Methods

Method	Signature	Returns	Description
`createPublic`	`createPublic(dto: CreateIssueDto)`	`unknown`	Create a new issue report from a reader
`createFromChatEscalation`	`createFromChatEscalation(input: {		

```
projectId: string;
chatSessionId: string;
question: string;
transcript: Array<{ role: string; content: string }>;
reporterEmail?: string;
pageUrl?: string;
```

}) | unknown | Chat → ticket bridge. | | getPublicTicket | getPublicTicket(projectSlugId: string, ticketNumber: number, reporterSessionId: string) | unknown | Get public ticket tracking data (no auth — the reporterSessionId minted when the ticket was filed proves ownership, exactly like addCustomerReply). | | addCustomerReply | addCustomerReply(projectSlugId: string, ticketNumber: number, input: { body: string; reporterSessionId: string }) | unknown | Customer reply from the public tracking page (no auth — the reporter proves ownership with the

reporterSessionId minted when the ticket was filed). |
| recordTicketCsat | recordTicketCsat(projectSlugId: string, ticketNumber: number, score: number) | unknown | Ticket-level CSAT from the public tracking page. |
| notifyReporterResolved | notifyReporterResolved(projectId: string, issue: {
ticketNumber: number;
reporterEmail: string | null;
pageTitle: string | null;
category: string;
priority: string;
description: string;
closedReason?: string | null;
}) | unknown | Fire the 'resolved' reporter email (with its CSAT ask) for a ticket. |
| list | list(projectId: string, user: JwtPayload, params: {
status?: string;
category?: string;
ticketType?: string;
priority?: string;
assigneeId?: string;
search?: string;
sortBy?: string;
sortOrder?: string;
page?: number;
limit?: number;
}) | unknown | List issues with filters + pagination | | getStats | getStats(projectId: string, user: JwtPayload) | unknown | Aggregate stats for the dashboard header |
getOne	getOne(projectId: string, issueId: string, user: JwtPayload)	unknown	Get a single issue with full detail		update	update(projectId: string, issueId: string, dto: UpdateIssueDto, user: JwtPayload)	unknown	Update a ticket (status, priority, assignee, close reason)		bulkUpdate	bulkUpdate(projectId: string, dto: BulkUpdateIssuesDto, user: JwtPayload)	unknown	Bulk update multiple issues
addComment	addComment(projectId: string, issueId: string, dto: CreateIssueCommentDto, user: JwtPayload, opts: { suppressAutoTransition?: boolean })	unknown	Add a comment to a ticket.
getDuplicates	getDuplicates(projectId: string, issueId: string, user: JwtPayload)	unknown`	Find potential duplicates for a ticket (same page within 30 days)

Dependencies

- `PrismaService`
- `EmailService`
- `EventsService`

Where it refuses work

- `IssuesService` stops the work with `NotFoundException` when `!project` — “Project not found”, in 5 places.
- `IssuesService` stops the work with `NotFoundException` when `!issue` — “Ticket not found”, in 3 places.
- `IssuesService` stops the work with `NotFoundException` when `!issue` — “Issue not found”, in 3 places.
- `IssuesService` stops the work with `ConflictException` when `existing` — “You already reported an issue on this page recently. Please wait a few minutes.”.
- `IssuesService` stops the work with `ForbiddenException` when `issue.reporterSessionId !== reporterSessionId` — “This ticket belongs to a different session”.
- `IssuesService` stops the work with `ForbiddenException` when `issue.reporterSessionId !== input.reporterSessionId` — “This ticket belongs to a different session”.

Diagram

```
sequenceDiagram
  participant Client as Public Client / Admin UI
  participant Controller as Issues Controller
  participant Service as IssuesService
  participant Store as Ticket Repository
  participant Notify as Notification Service

  Client->>Controller: Create issue or submit reply
  Controller->>Service: createPublic() / addCustomerReply()
  Service->>Store: Create or update ticket data
  Store-->>Service: Ticket result
  Service-->>Controller: Ticket response
  Controller-->>Client: Ticket details

  Client->>Controller: Resolve ticket
  Controller->>Service: update()
  Service->>Store: Persist resolved status
  Service->>Notify: notifyReporterResolved()
  Notify-->>Client: Resolution notification
```

Usage

```
import { Injectable } from '@nestjs/common';
import { IssuesService } from './issues.service';

@Injectable()
export class SupportWorkflowService {
  constructor(private readonly issuesService: IssuesService) {}

  async submitPublicIssue() {
    // Use the DTOs and payload shape defined by the issues module.
    const ticket = await this.issuesService.createPublic({
      reporterEmail: 'customer@example.com',
      subject: 'Unable to access my account',
      message: 'I receive an error after signing in.',
    });

    return ticket;
  }

  async replyToTicket(ticketReference: string) {
    return this.issuesService.addCustomerReply(ticketReference, {
      message: 'The issue is still happening after clearing my cache.',
    });
  }

  async loadAdminDashboard() {
    const [tickets, stats] = await Promise.all([
      this.issuesService.list(),
      this.issuesService.getStats(),
    ]);

    return { tickets, stats };
  }
}
```

AI Coding Instructions

- Keep ticket lifecycle operations inside `IssuesService`; controllers should validate input and delegate workflow logic to the service.
- Use `createPublic()` for externally submitted issues and `createFromChatEscalation()` when converting a chat/support interaction into a ticket.
- Preserve ticket identity and access controls when using `getPublicTicket()` or accepting replies through `addCustomerReply()`.
- When adding status transitions in `update()`, ensure resolved-ticket flows continue to call `notifyReporterResolved()` where appropriate.

- Treat `getStats()` and `list()` as administrative/reporting integration points; keep filtering, pagination, and aggregation behavior consistent with the surrounding API conventions.

Relationships

- DEPENDS_ON → `PrismaService`
- DEPENDS_ON → `EmailService`
- DEPENDS_ON → `EventsService`

Referenced By

- `BoardsService` (DEPENDS_ON)
- `DeskService` (DEPENDS_ON)
- `IssuesController` (DEPENDS_ON)
- `IssuesModule` (MODULE_PROVIDES)
- `IssuesModule` (MODULE_EXPORTS)
- `PublicIssuesController` (DEPENDS_ON)
- `PublicProjectController` (DEPENDS_ON)
- `TechDocsChatService` (DEPENDS_ON)