EventsService

September 4, 2026

EventsService

Kind: Service

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

`EventsService` provides a backend integration point for recording application events. Use `record()` when the caller needs the persisted event identifier, and use `recordSafe()` for best-effort event recording that does not return a value to the caller.

Methods

Method	Signature	Returns	Description
`record`	`record(input: {		

```
type: string;
projectId?: string | null;
organizationId?: string | null;
sessionId?: string | null;
userId?: string | null;
payload?: Record<string, unknown> | null;
```

}) | Promise<{ id: string }> | `Record an event.` | | recordSafe | recordSafe(input: Parameters<EventsService['record']>[0]) | void` | Server-side auto-recording — swallow failures (never break the observed feature). |

Dependencies

- `PrismaService`

Where it refuses work

- `EventsService` stops the work with `BadRequestException` when `!EVENT_TYPES.includes(input.type as EventType)`.

- `EventsService` stops the work with `BadRequestException` when `payload && Buffer.byteLength(JSON.stringify(payload), 'utf8') > MAX_PAYLOAD_BYTES` — “Event payload too large (max 4KB).”.

Diagram

```
sequenceDiagram
  participant C as API Consumer
  participant S as EventsService
  participant E as Event Storage

  C->>S: record()
  S->>E: Persist event
  E-->>S: Event ID
  S-->>C: Promise<{ id }>

  C->>S: recordSafe()
  S->>E: Persist event safely
  S-->>C: void
```

Usage

```
import { Injectable } from '@nestjs/common';
import { EventsService } from './events/events.service';

@Injectable()
export class OrdersService {
  constructor(private readonly eventsService: EventsService) {}

  async createOrder() {
    // Create the order first...

    const event = await this.eventsService.record();

    return {
      eventId: event.id,
    };
  }

  trackBackgroundActivity(): void {
    // Use for best-effort recording when no result is needed.
    this.eventsService.recordSafe();
  }
}
```

AI Coding Instructions

- Inject `EventsService` through NestJS constructor injection rather than creating service instances manually.
- Use `await record()` when downstream logic requires the returned event `id`.
- Use `recordSafe()` only when callers do not need a result and event-recording failures should not affect the primary workflow.
- Keep event recording close to the business action it represents, after the action has completed successfully.

Relationships

- DEPENDS_ON → `PrismaService`

Referenced By

- `EventsController` (DEPENDS_ON)
- `EventsModule` (MODULE_PROVIDES)
- `EventsModule` (MODULE_EXPORTS)
- `IssuesService` (DEPENDS_ON)
- `PublicProjectController` (DEPENDS_ON)
- `TechDocsChatService` (DEPENDS_ON)
- `PublicToursController` (DEPENDS_ON)
- `ToursHealingService` (DEPENDS_ON)