# Artala

Webhook Reference · Developer Documentation

## Artala Webhooks

Artala can push real-time events to your systems as things happen on a board — tasks created, statuses changed, comments posted, sprints completed, and more. When a subscribed event occurs, Artala sends an HTTP `POST` to your endpoint with a JSON body describing what happened.

This document covers the request format, how to verify authenticity, delivery behaviour, and the payload of every event.

## Registering a webhook

Create and manage webhooks in **Admin → Webhooks** in the app, or via the API. Each webhook has a target URL, a signing secret (generated for you), an active flag, and the list of events it subscribes to.

To list every event type Artala can send:

```
GET /api/webhooks/events
```

returns the array of supported event names shown in the Event reference.

### Managing webhooks via the API

Webhooks can also be managed programmatically (Owner or Admin role required):

| Method | Path | Purpose |
|---|---|---|
| `GET` | `/api/webhooks` | List the workspace's webhooks. |
| `POST` | `/api/webhooks` | Create a webhook (or update one when an `id` is supplied). |
| `DELETE` | `/api/webhooks/{id}` | Delete a webhook. |
| `POST` | `/api/webhooks/{id}/test` | Send a test ping to the webhook. |
| `GET` | `/api/webhooks/{id}/log` | Delivery log for one webhook. |
| `GET` | `/api/webhooks/log` | Delivery log across all webhooks. |

See the REST API reference for request and response details.

## Request format

Every delivery is an HTTP `POST` with `Content-Type: application/json; charset=utf-8` and this envelope:

```
{
  "event": "task.created",
  "timestamp": "2026-07-06T14:32:10.8127344Z",
  "tenantId": "9d1f8a2c-1b3e-4c5a-8f21-0a1b2c3d4e5f",
  "data": { }
}
```

| Field | Description |
|---|---|
| `event` | The event type (e.g. `task.created`). |
| `timestamp` | When the event was dispatched, ISO-8601 UTC. |
| `tenantId` | Your Artala tenant (workspace) id. |
| `data` | The event-specific payload — see the Event reference. |

Field names are **camelCase** throughout.

### Headers

| Header | Description |
|---|---|
| `X-Artala-Event` | The event type, matching `event` in the body. |
| `X-Artala-Signature` | `sha256=<hex>` — an HMAC of the raw body (see below). |
| `X-Artala-Delivery` | A unique id for this delivery attempt. Use it to de-duplicate. |

Respond with any `2xx` status to acknowledge receipt. Any non-`2xx`, a timeout, or a connection error is treated as a failed delivery.

## Verifying signatures

Every request is signed so you can confirm it came from Artala and wasn't altered. The `X-Artala-Signature` header is `sha256=` followed by the lowercase hex **HMAC-SHA256** of the **raw request body**, keyed by your webhook's signing secret.

Compute the same HMAC over the raw body you received and compare. Always verify against the raw bytes **before** parsing JSON — re-serializing can change the bytes and break the comparison. Use a constant-time comparison.

**Node.js**

```
const crypto = require("crypto");

function isValid(rawBody, signatureHeader, secret) {
  const expected =
    "sha256=" +
    crypto.createHmac("sha256", secret).update(rawBody, "utf8").digest("hex");
  const a = Buffer.from(signatureHeader);
  const b = Buffer.from(expected);
  return a.length === b.length && crypto.timingSafeEqual(a, b);
}
```

**Python**

```
import hmac, hashlib

def is_valid(raw_body: bytes, signature_header: str, secret: str) -> bool:
    expected = "sha256=" + hmac.new(
        secret.encode("utf-8"), raw_body, hashlib.sha256
    ).hexdigest()
    return hmac.compare_digest(expected, signature_header)
```

---

## Delivery, retries, and guarantees

- **Timeout:** each attempt waits up to 10 seconds for your `2xx`.
- **Retries:** failed deliveries (HTTP `429` or `5xx`) are retried up to 3 times with exponential backoff and jitter. Other failures (timeouts, connection errors, `4xx`) are not retried.
- **Delivery log:** every attempt — payload, response code, response body, and outcome — is recorded and visible in **Admin → Webhooks**.

Please design your consumer around these properties:

- **De-duplicate.** A retry can deliver the same event more than once. Treat `X-Artala-Delivery` as an idempotency key and ignore ids you've already processed.
- **Don't rely on ordering.** Deliveries are sent concurrently and may arrive out of order. Use `timestamp` and the entity's own `updatedAt` to resolve sequence.
- **Treat webhooks as best-effort, not a system of record.** If your endpoint is down past the retry window, that event is not redelivered later. For anything you must not miss, reconcile periodically against the REST API rather than relying on webhooks alone.

### Events from scheduled activity

Some events are produced by Artala's own background schedule rather than by someone using the app: recurring task instances ( `task.created` ), and the status, assignee and comment changes made by **due-date** automation rules ( `task.updated` / `task.completed` , `task.assigned` , `comment.created` ).

These behave slightly differently, in your favour:

- **They are queued before they are sent.** The event is recorded at the moment the change happens, so a restart or a deployment on our side cannot lose it.
- **They arrive within a minute**, not instantly — usually within about 15 seconds of the change.
- **They are retried for roughly 80 minutes** (1, 2, 5, 15 and 60 minutes apart) on top of the immediate retries above, so an endpoint that is briefly down or mid-deploy still receives them.
- **Because of that, duplicates are more likely here** than on interactive events. De-duplicating on `X-Artala-Delivery` is what makes this safe.

Everything else — signature, headers, payload shape — is identical to an interactive event. There is no way to tell from the payload whether a change was made by a person or by the schedule; use the entity's own fields if you need to.

### Test ping

Sending a test ping from the webhook settings delivers this body (it bypasses the event subscription filter so it always arrives):

```
{
  "event": "ping",
  "timestamp": "2026-07-06T14:32:10.8127344Z",
  "tenantId": "9d1f8a2c-1b3e-4c5a-8f21-0a1b2c3d4e5f",
  "data": {
    "message": "This is a test ping from Artala.",
    "webhookId": "2a7d1c88-9e04-4b31-a1f2-3c4d5e6f7a8b"
  }
}
```

# Event reference

All examples show only the `data` block; each is wrapped in the standard envelope on delivery.

### The task object

Task events carry a common task object as their `data` (or, where noted, nested under a `task` key). It looks like this:

```
{
  "id": "3f7c2b10-5a44-4e91-bb02-9c1d2e3f4a5b",
  "title": "Draft the Q3 rollout plan",
  "description": "Outline milestones and owners.",
  "status": "InProgress",
  "priority": 2,
  "priorityName": "Medium",
  "assigneeId": "c48e1d20-7b62-4a13-8c9d-2e3f4a5b6c7d",
  "assigneeName": "Dana Okafor",
  "boardId": "b21a7f90-3c14-4d88-9e6a-1f2a3b4c5d6e",
  "dueDate": "2026-07-15",
  "completedAt": null,
  "labelCount": 2,
  "createdAt": "2026-07-01T09:12:44.0000000Z",
  "updatedAt": "2026-07-06T14:32:10.8127344Z"
}
```

`dueDate` is a calendar date ( `yyyy-MM-dd` ) or `null` . `completedAt` is an ISO-8601 timestamp when the task is in a done status, otherwise `null` .

### Task events

| Event | When | `data` |
|---|---|---|
| `task.created` | A task is created (interactive, import, or playbook). | task object |
| `task.updated` | A task's fields change. | task object |
| `task.completed` | A task moves into a done status. | task object |
| `task.deleted` | A task is deleted. | task object |
| `task.archived` | A task is archived. | task object |
| `task.unarchived` | A task is unarchived. | task object |
| `task.assigned` | A task's assignee is set or changed. | task object |
| `task.due_date_changed` | A task's due date changes. | task object |
| `task.backlog.added` | A task moves to the backlog. | task object |
| `task.backlog.removed` | A task moves out of the backlog. | task object |

`task.moved` and `task.moved_to_backlog` wrap the task and add the destination board:

```
{
  "task": {
    "id": "3f7c2b10-5a44-4e91-bb02-9c1d2e3f4a5b",
    "title": "Draft the Q3 rollout plan",
    "status": "Todo",
    "boardId": "e90c1a22-4d55-4f66-b7c8-9d0e1f2a3b4c",
    "...": "full task object"
  },
  "targetBoardId": "e90c1a22-4d55-4f66-b7c8-9d0e1f2a3b4c"
}
```

`task.bulk_imported` is an aggregate summary fired once per CSV import. **Each imported task also fires its own `task.created` ,** so subscribe to `task.created` if you want the individual tasks.

```
{
  "boardId": "b21a7f90-3c14-4d88-9e6a-1f2a3b4c5d6e",
  "imported": 42,
  "skipped": 3,
  "importedAt": "2026-07-06T14:32:10.8127344Z"
}
```

### Sprint events

`task.sprint.added`

```
{ "taskId": "3f7c2b10-5a44-4e91-bb02-9c1d2e3f4a5b", "sprintId": "7c9e0b44-1a22-4d33-9e44-5f6a7b8c9d0e" }
```

`task.sprint.removed`

```
{ "taskId": "3f7c2b10-5a44-4e91-bb02-9c1d2e3f4a5b" }
```

`task.sprint.moved`

```
{
  "taskId": "3f7c2b10-5a44-4e91-bb02-9c1d2e3f4a5b",
  "fromSprintId": "7c9e0b44-1a22-4d33-9e44-5f6a7b8c9d0e",
  "toSprintId": "8d0f1c55-2b33-4e44-9f55-6a7b8c9d0e1f"
}
```

`sprint.started`

```
{
  "sprintId": "7c9e0b44-1a22-4d33-9e44-5f6a7b8c9d0e",
  "boardId": "b21a7f90-3c14-4d88-9e6a-1f2a3b4c5d6e",
  "name": "Sprint 12"
}
```

`sprint.completed`

```
{
  "sprintId": "7c9e0b44-1a22-4d33-9e44-5f6a7b8c9d0e",
  "boardId": "b21a7f90-3c14-4d88-9e6a-1f2a3b4c5d6e",
  "name": "Sprint 12",
  "completedTasks": 9,
  "returnedToBacklog": 2
}
```

### Comment events

`comment.created`

```
{
  "commentId": "5b8d3e11-6c44-4a55-b2f3-1c2d3e4f5a6b",
  "content": "Looks good — shipping today.",
  "authorId": "c48e1d20-7b62-4a13-8c9d-2e3f4a5b6c7d",
  "authorName": "Dana Okafor",
  "taskId": "3f7c2b10-5a44-4e91-bb02-9c1d2e3f4a5b",
  "taskTitle": "Draft the Q3 rollout plan",
  "boardId": "b21a7f90-3c14-4d88-9e6a-1f2a3b4c5d6e",
  "createdAt": "2026-07-06T14:32:10.8127344Z"
}
```

`comment.updated`

```
{
  "commentId": "5b8d3e11-6c44-4a55-b2f3-1c2d3e4f5a6b",
  "taskId": "3f7c2b10-5a44-4e91-bb02-9c1d2e3f4a5b",
  "content": "Looks good — shipping tomorrow.",
  "updatedAt": "2026-07-06T15:01:22.4000000Z"
}
```

`comment.deleted`

```
{
  "commentId": "5b8d3e11-6c44-4a55-b2f3-1c2d3e4f5a6b",
  "taskId": "3f7c2b10-5a44-4e91-bb02-9c1d2e3f4a5b",
  "deletedAt": "2026-07-06T15:10:05.1000000Z"
}
```

### Attachment events

`attachment.added`

```
{
  "attachmentId": "6c9e4f22-7d55-4b66-c3a4-2d3e4f5a6b7c",
  "taskId": "3f7c2b10-5a44-4e91-bb02-9c1d2e3f4a5b",
  "fileName": "rollout-plan.pdf",
  "sizeBytes": 284119,
  "addedAt": "2026-07-06T14:40:00.0000000Z"
}
```

`attachment.deleted`

```
{
  "attachmentId": "6c9e4f22-7d55-4b66-c3a4-2d3e4f5a6b7c",
  "taskId": "3f7c2b10-5a44-4e91-bb02-9c1d2e3f4a5b",
  "deletedAt": "2026-07-06T16:00:00.0000000Z"
}
```

### Board events

`board.created`

```
{
  "boardId": "b21a7f90-3c14-4d88-9e6a-1f2a3b4c5d6e",
  "name": "Product Roadmap",
  "isPrivate": false,
  "createdAt": "2026-07-01T08:00:00.0000000Z"
}
```

`board.updated`

```
{ "boardId": "b21a7f90-3c14-4d88-9e6a-1f2a3b4c5d6e", "updatedAt": "2026-07-06T14:32:10.8127344Z" }
```

`board.archived`

```
{ "boardId": "b21a7f90-3c14-4d88-9e6a-1f2a3b4c5d6e", "archivedAt": "2026-07-06T14:32:10.8127344Z" }
```

`board.unarchived`

```
{ "boardId": "b21a7f90-3c14-4d88-9e6a-1f2a3b4c5d6e", "unarchivedAt": "2026-07-06T14:32:10.8127344Z" }
```

`board.member.added`

```
{
  "boardId": "b21a7f90-3c14-4d88-9e6a-1f2a3b4c5d6e",
  "userId": "c48e1d20-7b62-4a13-8c9d-2e3f4a5b6c7d",
  "role": "Member",
  "addedAt": "2026-07-06T14:32:10.8127344Z"
}
```

`board.member.removed`

```
{
  "boardId": "b21a7f90-3c14-4d88-9e6a-1f2a3b4c5d6e",
  "userId": "c48e1d20-7b62-4a13-8c9d-2e3f4a5b6c7d",
  "removedAt": "2026-07-06T14:32:10.8127344Z"
}
```

### Member events

`member.invited`

```
{
  "userId": "d59f2e30-8c73-4b24-9daf-3e4f5a6b7c8d",
  "email": "sam@example.com",
  "role": "Member",
  "invitedAt": "2026-07-06T14:32:10.8127344Z"
}
```

`member.role_changed`

```
{
  "userId": "d59f2e30-8c73-4b24-9daf-3e4f5a6b7c8d",
  "newRole": "Admin",
  "changedAt": "2026-07-06T14:32:10.8127344Z"
}
```

`member.deactivated`

```
{
  "userId": "d59f2e30-8c73-4b24-9daf-3e4f5a6b7c8d",
  "deactivatedAt": "2026-07-06T14:32:10.8127344Z"
}
```

### Timesheet events

`timesheet.week.submitted`

```
{
  "weekId": "a13c5f80-9d84-4c35-8e6b-4f5a6b7c8d9e",
  "userId": "c48e1d20-7b62-4a13-8c9d-2e3f4a5b6c7d",
  "weekStart": "2026-06-29",
  "weekEnd": "2026-07-05",
  "status": "Submitted",
  "totalHours": 38.5,
  "submittedBy": "Dana Okafor"
}
```

`timesheet.week.approved`, `timesheet.week.rejected`, and `timesheet.week.reopened` share this shape (`status` reflects the new state; `rejectNote` is present on rejection, otherwise `null`):

```
{
  "weekId": "a13c5f80-9d84-4c35-8e6b-4f5a6b7c8d9e",
  "userId": "c48e1d20-7b62-4a13-8c9d-2e3f4a5b6c7d",
  "weekStart": "2026-06-29",
  "weekEnd": "2026-07-05",
  "status": "Approved",
  "rejectNote": null,
  "actor": "Priya Menon"
}
```

---

## Current limitations

- **At-least-once delivery is not yet guaranteed for events triggered inside the app.** Interactive changes are delivered immediately and retried within the attempt, but if your endpoint is down past that window the event is not redelivered later. Events from scheduled background jobs are queued durably and retried for about 80 minutes (see Delivery); extending the same durability to interactive events is planned. Reconcile via the API for anything critical.

This page will be updated as these limitations are resolved.