myprogress

Openbadges.me API Guide

What you can do with an API key

Version 1.0
September 2026

MyKnowledgeMap Ltd

Table of Contents

1. What the API lets you do

An API key turns your Openbadges.me account into something your other systems can talk to. Instead of a member of staff logging in and awarding badges by hand, your learning platform, HR system, event booking tool or student record system can report achievements as they happen, and Openbadges.me decides what to award.

> **The central idea:**
>
> You do not tell the API "issue this badge to this person". You tell it "this person did this thing, at this time". Openbadges.me then applies the rules you have configured and issues badges automatically. This keeps the awarding logic in one place — configured by your badge administrators, not hard-coded into every integration.

1.1 Capabilities at a glance

Capability	What it is for	Direction
Award a badge	Report a single achievement for one person, which triggers badge issuing	You send in
Bulk award	Report many achievements in one call — a whole cohort, a day's attendance, a nightly sync	You send in
List rules	See every automated awarding rule in your organisation, whether it is enabled, and how many people have met it	You read out
Inspect a rule	Retrieve the full condition logic and the outcomes (badge, email) attached to one rule	You read out
Rule completions	Get the list of people who have satisfied a given rule, and when	You read out
Activity reporting	Retrieve everything your key has submitted, filtered by date range or by a single badge	You read out
Expire a badge	End a badge's validity, individually or as a scheduled sweep of everything due	You send in
Revoke a badge	Withdraw a badge that should no longer stand	You send in
Look up issued badges	Every badge held by one person, or every badge your organisation has issued, with current status	You read out
Badge templates	Read the badges defined in your space, their criteria, expiry settings and issue totals	You read out
Certificates	Push formal certificates in, and read the ones already issued	Both
Badge display	Embed a person's earned badges into your own website or portal using the JavaScript integration package	You read out

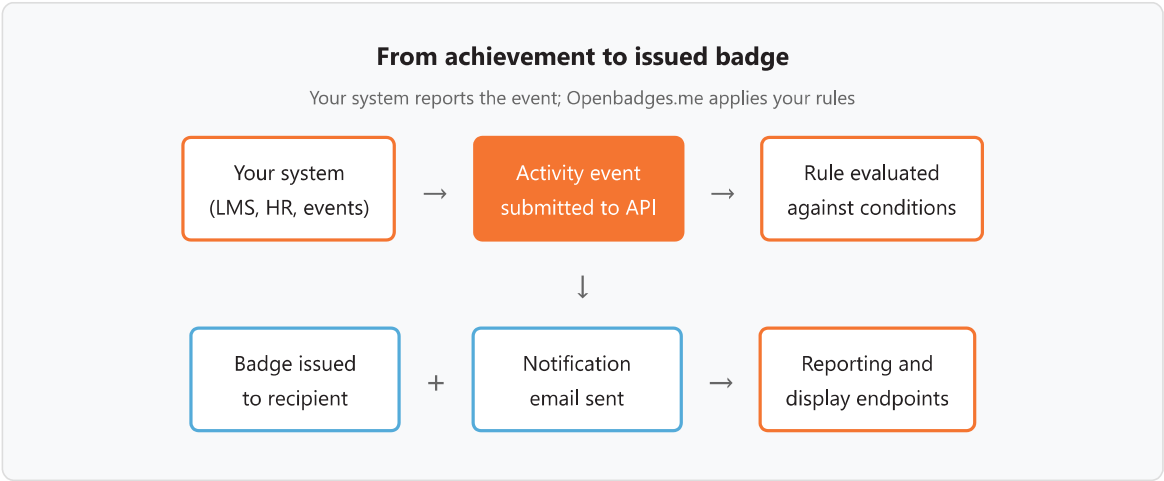
Two services, one set of credentials

Your API key and token work against two separate addresses. Knowing which is which saves a great deal of confusion:

Address	Known as	Covers
`events.openbadges.me`	Events API	Submitting achievements, rules, and reporting on what you submitted
`badges.openbadges.me`	MyBadges API	Issued badges, badge templates, expiring and revoking, certificates

The second is the **System URL** shown on the API screen in the platform. Both accept the same two authentication headers, described in chapter 3.

1.2 How the pieces fit together



Three concepts are worth getting straight before you build anything:

- **Activity** — a thing a person can do that you care about. Each activity has an identifier that you use when reporting it. In the simplest set-up this identifier is the badge's own reference, so reporting the activity awards that badge directly.
- **Activity event** — one occurrence of an activity by one person at one point in time. This is what your integration sends in.
- **Rule** — the logic that turns events into outcomes. A rule can be as simple as "the event happened once" or as involved as "attended five sessions and scored 70 or more, between January and June".

1.3 What the API does not do

> **Note:**
>
> Badge *design* — creating the badge image, writing the criteria and description, setting the expiry, and building the awarding rules — is done in the Openbadges.me web interface, not through the API. The API can read badge templates and rules back (chapters 5 and 9), but it cannot author them. Plan for your badge administrators to define badges and rules in the platform first, then hand the resulting identifiers to your development team.

2. Getting your API key and token

API credentials live in the **API** area, reached from **Issuing badges** → **API** in the left-hand menu. The screen explains its own purpose in the platform's words: the APIs "support the direct issuing of a specific badge to a specific recipient or the automatic event rules-based issuing of badges".

Three things appear on that screen:

On screen	What it is	Where you use it
System URL (the box above the table)	Your badge system's address, typically `https://badges.openbadges.me`	The `systemUrl` setting in the badge display component (chapter 7)
API Key Id	A long opaque string identifying this integration, with the date it was created	The `ApiKey` request header (chapter 3)
Actions — the key icon ("Refresh authentication token")	Generates an authentication token for the key beside it	The `Authorization: Bearer` header, and the display component's `authToken`

So the sequence for a new integration is:

1. **Generate the API key**, if your organisation does not already have one. It appears in the table with its creation date.
2. **Generate an authentication token** using the key icon under *Actions*. Copy the token as soon as it is shown and store it safely.
3. **Note the System URL** from the box above the table if you intend to display badges on your own site.

> **Tip:** If a token is lost or you suspect it has been shared too widely, you do not have to start again. Refresh the token against the same API key; the key itself stays as it is, and any configuration tied to it is preserved.

> **Important — refreshing is not read-only:**
>
> The key icon *replaces* the current authentication token. Any live integration still using the old token will start receiving `401 Unauthorised` the moment you refresh. Only use it when you are ready to update the token in every system that holds it, and let those system owners know first.

> **Treat both values as passwords:**
>
> The key and token together allow anything described in this guide, including awarding badges in your organisation's name and reading the personal details of everyone who has earned one. Store them in your application's secrets store — never in front-end JavaScript, a shared spreadsheet, a support ticket, or source control.

3. Connecting and authenticating

Calls go to one of two addresses, depending on what you are doing:

```
https://events.openbadges.me  —  issuing, rules, activity reporting
```

```
https://badges.openbadges.me  —  issued badges, templates, expiry and revocation
```

Every request must carry your credentials as HTTP headers. Two header-based schemes are recognised, and a request needs to satisfy one of them:

Header	Value	Notes
`ApiKey`	Your API key, exactly as shown in System settings	The straightforward option for server-to-server integrations
`Authorization`	`Bearer <your token>`	Note the word `Bearer` and a single space before the token

Requests and responses are JSON throughout, so send `Content-Type: application/json` on anything with a body.

```
curl -X POST "https://events.openbadges.me/api/ActivityEvents" \
  -H "ApiKey: YOUR_API_KEY" \
  -H "Content-Type: application/json" \
  -d '{ ... }'
```

> **Tip:** Interactive documentation — where you can paste your credentials in and try each call against your own data — is published for both services: the Events API and the MyBadges API. Listing your rules is a harmless first call that proves the credentials work before you write any code.

4. Awarding badges

Awarding is done by submitting activity events. There is one call for a single event and one for a batch.

4.1 Recording a single achievement

```
POST /api/ActivityEvents
```

Use this when your system learns about an achievement as it happens — a course marked complete, a session attended, an assessment passed.

```
POST /api/ActivityEvents
ApiKey: YOUR_API_KEY
Content-Type: application/json

{
  "ActivityId": "OB-60ff00b6-aca0-4860-aa1e-3112251739bf",
  "ActivityTime": "2026-09-09T14:30:00Z",
  "UserId": "staff-104872",
  "OwnerOrganisation": "your-organisation-guid",
  "Email": "jane.smith@example.org",
  "FirstName": "Jane",
  "LastName": "Smith",
  "Int1": 82,
  "Text1": "Infection Control - Module 3",
  "Date1": "2026-09-09T00:00:00Z",
  "Testimonial": "Completed with distinction.",
  "IssuedBy": "Learning and Development",
  "Process": true
}
```

4.2 Recording achievements in bulk

```
POST /api/ActivityEvents/bulk
```

The bulk call takes an array of exactly the same objects. Use it for a whole cohort at once, for an overnight synchronisation from your student record system, or to backfill history when you first go live. Events in a single batch do not have to relate to the same badge or the same person.

```
POST /api/ActivityEvents/bulk
ApiKey: YOUR_API_KEY
Content-Type: application/json

[
  {
    "ActivityId": "OB-60ff00b6-aca0-4860-aa1e-3112251739bf",
    "ActivityTime": "2026-09-09T09:00:00Z",
    "UserId": "staff-104872",
    "OwnerOrganisation": "your-organisation-guid",
    "Email": "jane.smith@example.org",
    "FirstName": "Jane",
    "LastName": "Smith"
  },
  {
    "ActivityId": "OB-60ff00b6-aca0-4860-aa1e-3112251739bf",
    "ActivityTime": "2026-09-09T09:00:00Z",
    "UserId": "staff-104873",
    "OwnerOrganisation": "your-organisation-guid",
    "Email": "raj.patel@example.org",
    "FirstName": "Raj",
    "LastName": "Patel"
  }
]
```

> **Tip:** Send your own stable identifier in `UserId` — a payroll number, student reference or account ID — rather than relying on the email address alone. People change email addresses; a stable identifier keeps a person's achievement history joined up, and keeps your reconciliation reports honest.

4.3 Field reference

Required

Field	Type	What to send
`ActivityId`	Text, up to 100 characters	The identifier of the activity or badge being reported, taken from the platform. Openbadges.me badge references look like `OB-60ff00b6-aca0-4860-aa1e-3112251739bf`.
`ActivityTime`	Date and time	When the achievement actually happened — not when you got round to sending it. Send it in ISO 8601 form, e.g. `2026-09-09T14:30:00Z`.
`UserId`	Text, up to 255 characters	Your unique identifier for the recipient. An email address is acceptable, but a stable internal reference is better.
`OwnerOrganisation`	GUID	The organisation the event belongs to, as shown against your API configuration.

Recipient details

Field	Type	What to send
`Email`	Text	Where the badge notification is sent, and how the recipient claims the badge into their own collection.
`FirstName`	Text	Recipient's first name, used in the notification and on the badge record.
`LastName`	Text	Recipient's surname.

> **Note:** These three are optional in the strict sense that the API will accept an event without them, but a badge cannot reach anybody without an email address. Send all three for any event that should result in an award; omit them only when you are reporting activity for a person the platform already knows by `UserId`.

Optional data fields

These carry the detail your rules test against, and the detail you may want to see in reporting later. What each one means is entirely up to you — decide the convention once and document it for your team.

Field	Type	Typical use
`Int1`, `Int2`	Whole numbers	Scores, percentages, hours, counts, attempt numbers
`Date1`	Date and time	A second meaningful date — expiry, assessment date, placement start
`Text1`, `Text2`	Text, up to 255 characters each	Module names, cohort or site codes, grades, assessor names
`Testimonial`	Text, up to 1,000 characters	A comment or endorsement to carry alongside the achievement
`IssuedBy`	Text	Who is awarding it — a department, team or named assessor
`Meta`	Text	Additional information you want to travel with the event
`Process`	True or false	Whether the event should be evaluated for awarding straight away

> **Plan your field mapping before you build:**
>
> Rules are written against these field names, so the meaning you assign to `Int1` or `Text1` becomes part of your badge configuration. Agree the mapping between your badge administrators and your developers up front — for example "`Int1` is always the percentage score, `Text1` is always the module name" — and keep it consistent across every integration that reports into the same badges.

5. Rules — automated awarding

Rules are what make the system more than a pass-through. They are built in the web interface under **Issuing badges** → **Rules**, where each rule is listed with its name, short name, description, who last updated it, and a running **Completions** count. The API exposes that same information, plus the underlying logic, so your own dashboards can show progress towards a badge without anyone logging into Openbadges.me.

> **Note:** The API reads rules; it does not create or edit them. Use *Create new rule* in the platform to define the logic, then use the calls below to monitor it.

5.1 Listing your rules

```
GET /api/rule
```

Returns every rule for your organisation — no parameters needed, because your credentials already identify you. Each entry includes:

Field	Meaning
`Id`	The rule's identifier, used in the two calls below
`Name`, `ShortName`, `Description`	How the rule is labelled and described in the platform
`Completions`	How many people have satisfied the rule — useful as a headline figure
`Enabled`	Whether the rule is currently active
`ReadOnly`	Whether the rule is locked against editing
`StartDate`, `EndDate`	The window in which the rule applies, where one is set
`LastModified`, `LastModifiedBy`	When the rule last changed, and who changed it
`LinkedId`	The related item the rule is attached to

> **Tip:** A quiet badge is usually a disabled rule or an expired date window rather than a broken integration. When achievements are going in but nothing is being awarded, check `Enabled`, `StartDate` and `EndDate` on this list before investigating anything else.

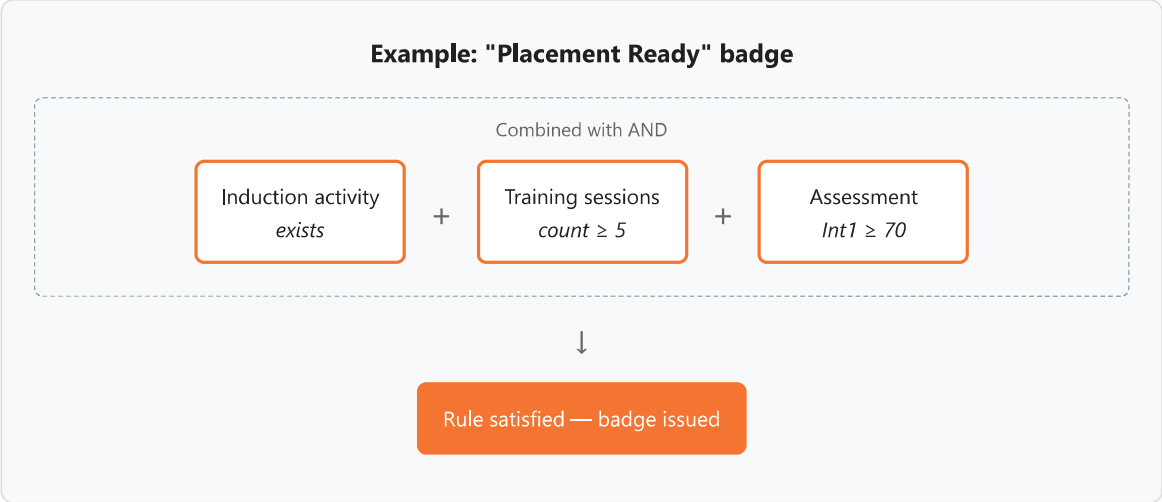
5.2 Inspecting a single rule

```
GET /api/rule/{id}
```

Returns everything from the list above, plus the two things that explain the rule's behaviour: the **condition logic** it evaluates, and the **outcomes** it produces. Each outcome has a type — a *badge* to issue or an *email* to send — and an order, so a single rule can both award a badge and notify someone.

5.3 How rule logic is expressed

You do not need to write this logic yourself, but understanding its shape helps enormously when you are deciding what to put in your event fields. A rule is a tree. At the leaves are tests against a single activity; above them, `And` and `Or` combine those tests.



The kinds of test available against your event data are:

Condition	What it checks	Example
Exists	The activity has been reported at all	Completed the induction
Count	How many times it has been reported	Attended five sessions
Distinct count	How many *different* values appeared in a chosen field	Worked in three different placement settings (distinct `Text1`)
Value	A field compared against a threshold	Scored 70 or above (`Int1`)
Running total	A field summed across all events	Accumulated 40 hours (sum of `Int1`)

Comparisons available include equal to, not equal to, greater than, greater than or equal to, less than, less than or equal to, *like* for text matching, and *between* for ranges. Tests can target the numeric fields (`Int1`, `Int2`), the text fields (`Text1`, `Text2`) or the date field (`Date1`).

> **Why this matters to your integration:**
>
> Every one of these conditions reads the optional fields described in section 4.3. If your events do not populate `Int1` with the score, no rule can test the score. Send the data even if no rule uses it yet — retro-fitting it across historical events is far more work than including it from day one.

6. Reporting on activity

Three read calls let you reconcile what you sent, audit what was awarded, and feed your own dashboards.

6.1 All activity in a date range

```
POST /api/ActivityEvents/entries
```

Returns every event submitted using your API key, between two dates. The date range goes in the request body:

```
POST /api/ActivityEvents/entries
ApiKey: YOUR_API_KEY
Content-Type: application/json

{
  "From": "2026-09-01T00:00:00Z",
  "To": "2026-09-30T23:59:59Z"
}
```

Each returned event carries its own record identifier, the `ActivityId`, the `UserId`, the `ActivityTime`, the `CreatedOn` timestamp recording when it reached the platform, and whichever optional fields you populated.

> **Note:** This call reports on events submitted *by your key*. Badges awarded manually in the web interface, or through a different integration with its own key, will not appear here. For a whole-organisation view, use the platform's own reporting.

6.2 Activity for one badge

```
POST /api/ActivityEvents/{id}/entries
```

The same data, narrowed to a single activity or badge. Put the identifier in the path:

```
POST /api/ActivityEvents/OB-60ff00b6-aca0-4860-aa1e-3112251739bf/entries
ApiKey: YOUR_API_KEY
```

This is the call to use for a "who has earned this badge" view in your own system.

6.3 Completions for one rule

```
GET /api/rule/{id}/completions
```

Returns everyone who has satisfied a given rule. Each entry gives the `UserId`, the `CompletedDate`, and the version of the rule that was in force at the time.

> **Tip:** The rule version on each completion is more useful than it looks. If a rule is tightened part-way through a year, it tells you which cohort earned the badge under which criteria — worth capturing if your badges support any kind of audit or regulatory reporting.

Choosing between the reporting calls

Question you are answering	Use
Did everything I sent last night arrive?	Activity in a date range
Who has been put forward for this badge?	Activity for one badge
Who has actually *earned* the badge?	Completions for the rule behind it
How many people are on track overall?	The `Completions` figure on the rule list

7. Expiring and revoking badges

Awarding is not the end of a badge's life. A certification reaches its renewal date; a badge is awarded in error; a holder no longer meets the standard. These calls live on the **MyBadges** service and let your own systems close a badge off without anyone working through the interface.

7.1 Expired or revoked — choosing correctly

> **They are not interchangeable:**
>
> **Expired** means the badge ran its course. A renewal date passed, a licence lapsed. It is an ordinary, expected ending and the achievement behind it still happened.
>
> **Revoked** means the badge is withdrawn — issued in error, or the holder no longer meets the standard it represents. Both states are recorded with a date so the history stays auditable, but they say very different things about the person holding the badge. Choose deliberately, and make sure whoever operates your integration understands the difference.

7.2 Expiring or revoking one badge

```
POST /api/SetExpired/{functionkey}/{id}
```

```
POST /api/SetRevoked/{functionkey}/{id}
```

The direct calls. `{id}` is the issued badge's identifier — chapter 8 explains how to look one up from a person and an activity. `{functionkey}` is an additional value carried in the path alongside your normal header credentials; if you do not have it, ask your Openbadges.me contact rather than guessing.

Expiring with a specific date

```
POST /api/SetExpired/{functionkey}
```

This variant takes a body, so you can set the expiry date explicitly rather than accepting the moment of the call.

Field	Required	What it is
`OpenBadgeId`	Yes	The Open Badge identifier
`IssueBadgeId`	Yes	The identifier of the issued badge being expired
`ExpiryDate`	Yes	The date on which validity ends
`OpenBadgeUrl`	No	The badge's public address
`BakedBadgeUrl`	No	The address of the baked badge image
`Process` , `System`	No	Processing and originating-system markers

7.3 Running expiry as a scheduled sweep

```
GET /api/ExpireRevoke/GetBadgesToExpire/{functionkey}/{system}
```

```
POST /api/ExpireRevoke
```

The first returns the badges currently due to expire for a given system — the natural input to a nightly or weekly job. The second processes expiry and revocation using the default notification email. Together they let renewal housekeeping run on your own schedule rather than one badge at a time.

> **These calls change other people's records:**
>
> Expiring or revoking a badge alters something a person may already have shared on a CV or a professional profile, and it can trigger a notification to them. Put a confirmation step in front of anything staff-initiated, record who asked for it, and do your testing against a badge you created for the purpose — never against a live holder's.

8. Looking up issued badges

Chapter 6 reports on what you *submitted*. These calls report on what was actually *issued*, including each badge's current status — which makes them the right place to look when you need to reconcile your records or find the identifier for an expiry or revocation.

8.1 By recipient and by organisation

```
GET /api/issuedbadgesexternal/email?email=...
```

Every badge issued to one person by your developer key. This is the call behind a "what does this learner hold" view.

```
GET /api/issuedbadgesexternal/organisation
```

Every badge your organisation has issued — the basis for your own reporting or a data warehouse feed.

```
GET /api/issuedbadgesexternal/BadgeId?email=...&ActivityId=...
```

Resolves a person plus an activity to a single badge identifier. Use it when you need the identifier for one of the expiry or revocation calls; it returns `404` when nothing matches.

8.2 What comes back

Each issued badge carries its identifier and status, the recipient's email, the badge's name and description, who issued it, the address of the baked badge image, timestamps for creation, baking and issue, any testimonial, and the expiry, expired and revoked flags with their dates. The status is one of six values:

Status	Meaning
`InProgress`	Being prepared
`NotBaked`	Created, but the badge image has not been generated yet
`Baked`	Image generated, not yet issued
`Issued`	With the recipient — the normal healthy state
`Expired`	Validity has ended
`Revoked`	Withdrawn

> **Tip:** Because the expired and revoked flags come back with their dates, this is the cleanest way to keep your own records honest. Pull your organisation's issued badges on a schedule and compare them against what your system believes is currently valid.

8.3 Scoped tokens for portals

```
POST /api/issuedbadgesexternal/AuthKey/{developerKey}/{organisationId}
```

```
GET /api/issuedbadgesexternal/scoped
```

The first takes a list of email addresses and returns a token scoped to just those people. The second then returns badges for whoever that token covers, with no email parameter to tamper with.

> **The right pattern for a staff or learner portal:**
>
> This pairing exists so that a browser never has to hold your organisation-wide credentials. Your own server establishes who is signed in, mints a token scoped to that person, and hands only that to the page. The page can then only ever see that person's badges. Prefer this over passing an email address to the general endpoint from anywhere a user could alter it.

Badge instructions

```
POST /api/issuedbadgesexternal/instruct
```

Sends an instruction against a badge, identified by its badge identifier, with an optional unique reference, the instruction itself and a timestamp. Confirm which instructions your space supports with your Openbadges.me contact before building against it.

9. Badge templates and certificates

9.1 Badge templates

```
GET /api/issuedbadgesexternal/badgeTemplate
```

```
GET /api/issuedbadgesexternal/badgeTemplate/{id}
```

A badge template is a badge *as designed* — the definition, rather than an award of it. Reading templates gives you each badge's name and description, its issuer and graphic, its criteria (either a block of text or a link), whether it expires and after how long, its language and public address, and running totals for how many times it has been issued, clicked and claimed.

Two classifications are particularly useful:

Field	Values	Why it matters
Status	`Draft`, `Published`, `Withdrawn`	Only published badges are live. Check this before presenting a badge as available to award
Category	`Push`, `Certificate`	Distinguishes ordinary badges from certificate-style ones

> **Tip:** Reading templates saves hard-coding badge identifiers into your integration. Pull the list, filter to published badges, and let your own screens offer administrators real badge names — the identifiers then come from the platform rather than a configuration file that quietly goes out of date.

9.2 Certificates

```
GET /api/externalcertificates/email?email=...
```

```
GET /api/externalcertificates/organisation
```

```
GET /api/externalcertificates/BadgeId?email=...&certificateNo=...
```

```
POST /api/externalcertificates
```

The read calls mirror the issued-badge ones but cover certificate-category badges — by recipient, across the organisation, or resolving a person plus a certificate number to a badge. The final call pushes a certificate in, carrying the document's address, the certificate's identifier, name, type, number, date and reprint flag, and the recipient's identifier, email, name and training-organisation details.

> **Note:** This is a specialised route for organisations pushing formal certificates from a training or awarding system. If you are issuing ordinary badges, the activity events in chapter 4 are the route you want, not this one.

10. Displaying badges on your own site

Alongside the events API there is a JavaScript integration package for showing people the badges they have earned, inside your own portal, intranet or website — so learners do not have to visit a separate system to see their achievements.

You drop the supplied stylesheet and scripts onto the page and call the badge container function, telling it whose badges to show and how to present them. The options available are:

Option	What it controls
`emails`	The recipients whose badges should be displayed
`display`	Presentation as cards or as a list
`orderBy`	Sort by date issued, date created, issuer name, badge name or status
`isAscending`	Sort direction
`searchQuery`	Filters the badges shown
`buttons`	Lets you supply your own action buttons
`authToken`, `systemUrl`	Your token and the System URL, both from Issuing badges → API

> **Note:** Because this component runs in the browser, give careful thought to how the page decides whose badges to display and where the token comes from. The safe pattern is for your own server to establish who the signed-in user is and render the component for that person only — never to accept an email address from the page's own query string, and never to embed a long-lived token in a public page.

11. Responses and troubleshooting

Response	Meaning	What to do
200 OK	The call succeeded. Read calls return the requested data.	Nothing. Note that a successful submission means the event was accepted, not that a badge was necessarily awarded — that depends on your rules.
401 Unauthorised	Credentials missing, malformed or no longer valid.	Check the header name is exactly `ApiKey`, or that the `Authorization` value starts with `Bearer` and a space. Confirm the token has not been regenerated since your configuration was set.
404 Not Found	The rule identifier does not exist for your organisation.	Re-read the rule list and use an identifier from it.
406 Not Acceptable	The submitted event failed validation.	Check all four required fields are present, that dates are valid ISO 8601, and that no text field exceeds its length limit.

Events accepted but no badge appearing

This is the most common support question, and it is almost never a fault in the API. Work through it in this order:

1. **Is the rule enabled?** Check the rule list (chapter 5).
2. **Is the event inside the rule's date window?** Remember the rule tests `ActivityTime`, not the time you sent it — backfilled history can fall outside the window.
3. **Does the event carry the data the rule tests?** A rule requiring a score of 70 cannot pass if `Int1` is empty.
4. **Is the `ActivityId` the one the rule is looking for?** A typo here produces a perfectly valid event that no rule matches.
5. **Is there an email address?** An award with nowhere to go will not reach anybody.

12. Good practice

> **Report the truth, let the platform decide**
>
> Resist the temptation to work out in your own code whether somebody deserves a badge and then report only the successes. Send every relevant event and let the rules decide. Criteria then change in one place, by the people who own them, without a software release.

Practical recommendations

- **Send the real achievement time.** Populating `ActivityTime` with "now" instead of the actual date will break any rule with a date window and will make your reporting misleading.
- **Use bulk for volume.** One call for a cohort is faster, easier to retry, and far kinder to both systems than hundreds of individual calls.
- **Make your integration re-runnable.** Decide early how you will avoid double-reporting if a nightly job runs twice — usually by tracking what you have already sent at your end.
- **Reconcile regularly.** A weekly comparison of what you sent against the date-range reporting call catches silent failures long before a learner complains about a missing badge.
- **Test in the interactive documentation first.** Confirming a single call by hand saves a great deal of guesswork before you write any code.
- **Use a separate key per integration** where you can. It makes the reporting calls meaningful per system, and lets you revoke one integration without disturbing the others.
- **Only send the personal data you need.** Name and email are needed to deliver a badge; the optional fields should carry evidence of the achievement, not general personal information. Agree what is sent, and how long it is retained, with whoever owns data protection in your organisation.

13. Reference summary

Every call requires either the `ApiKey` header or an `Authorization: Bearer` token.

13.1 Events API — events.openbadges.me

Purpose	Method and path	Input
Award — single achievement	`POST /api/ActivityEvents`	One activity event object
Award — bulk	`POST /api/ActivityEvents/bulk`	An array of activity event objects
Report — by date range	`POST /api/ActivityEvents/entries`	`From` and `To` dates
Report — by badge	`POST /api/ActivityEvents/{id}/entries`	Activity or badge identifier in the path
Rules — list all	`GET /api/rule`	None
Rules — full detail	`GET /api/rule/{id}`	Rule identifier in the path
Rules — completions	`GET /api/rule/{id}/completions`	Rule identifier in the path

13.2 MyBadges API — badges.openbadges.me

Purpose	Method and path	Input
Expire — one badge	`POST /api/SetExpired/{functionkey}/{id}`	Issued badge identifier in the path
Expire — with a set date	`POST /api/SetExpired/{functionkey}`	Badge identifiers and expiry date in the body
Revoke — one badge	`POST /api/SetRevoked/{functionkey}/{id}`	Issued badge identifier in the path
Badges due to expire	`GET /api/ExpireRevoke/GetBadgesToExpire/{functionkey}/{system}`	System in the path
Process expiry and revocation	`POST /api/ExpireRevoke`	None
Issued badges — one person	`GET /api/issuedbadgesexternal/email`	`email` query parameter
Issued badges — organisation	`GET /api/issuedbadgesexternal/organisation`	None
Resolve a badge identifier	`GET /api/issuedbadgesexternal/BadgeId`	`email` and `ActivityId`
Mint a scoped token	`POST /api/issuedbadgesexternal/AuthKey/{developerKey}/{organisationId}`	Email addresses in the body
Badges for a scoped token	`GET /api/issuedbadgesexternal/scoped`	None
Badge instruction	`POST /api/issuedbadgesexternal/instruct`	Badge identifier and instruction in the body
Badge templates	`GET /api/issuedbadgesexternal/badgeTemplate` `GET /api/issuedbadgesexternal/badgeTemplate/{id}`	None, or template identifier in the path

Certificates — read	`GET /api/externalcertificates/email`
	`GET /api/externalcertificates/organisation`
	`GET /api/externalcertificates/BadgeId`
		`email`, none, or `email` plus `certificateNo`

Certificates — push	`POST /api/externalcertificates`	Certificate details in the body

Where to look next

- Events API — interactive documentation
- MyBadges API — interactive documentation
- Get your API keys — generating the key and token
- **In the platform:** Issuing badges → API for credentials, Issuing badges → Rules for awarding logic, Reporting for the whole-organisation view
- Simple Badge Issuing API — the minimal integration
- Advanced Badge Issuing API — rules and reporting
- Full integration package — the badge display component

> **About this document:**
>
> Compiled September 2026 from three sources: the published Openbadges.me API documentation, the live API specifications for both the Events and MyBadges services (from which the field names, limits, status values and response codes are taken), and the API and Rules screens in the platform itself. Where your own space has been configured differently — additional rules, custom activity identifiers, a different System URL — the platform's own screens are the authority.