PLATFORM API GUIDE

Getting started with the Crazy Egg Platform API

Latest version: https://platform-api-docs.crazyegg.com/guides/getting-started

Source revision a14dd9f5cd

Schema dated 2026-11-18, SHA-256 6fd8bff02a72

Generated 2026-11-18

Contents

- Before you begin
- Endpoint
- Personal access tokens
  - Create a token
  - Read-only and read & write tokens
  - What a token can access
  - Revoke and delete tokens
  - Keep tokens secret
- Send a request
  - Headers
  - Request body
  - Check your connection
- Schema and reference
- Responses and errors
  - HTTP status codes
  - Error format
  - Error codes
  - Authentication and permission errors
- Rate limits
  - What counts
  - Limits
  - When you reach a limit
- Batching
  - Several fields in one document
  - JSON array batches
- Subscriptions
- Page editor operations
- Getting help

The Crazy Egg Platform API is a GraphQL API that gives your own tools access to the same data and actions as the Crazy Egg app. This guide explains how to connect: where to send requests, how to authenticate with a personal access token, how to read responses and errors, and how rate limits work.

It does not describe individual queries and mutations. The API reference (https://platform-api-docs.crazyegg.com/reference) lists every operation and type, generated from the schema and its descriptions.

Before you begin

You need:

- A Crazy Egg account.
- A personal access token for that account. Create a personal access token explains how.
- An HTTP client that can send JSON, such as `curl` or the HTTP library of your programming language. No GraphQL-specific library is required.

Endpoint

Send every request to:

```
https://api.crazyegg.com/api
```

Use `POST` with a JSON body over HTTPS. Keep each request body under 8 MB.

The Crazy Egg app uses a different path, `/gql`. That endpoint rejects personal access tokens, so integrations must always use `/api`.

Personal access tokens

A personal access token authenticates your requests as your Crazy Egg account. Every token starts with `ce_pat_`.

Create a token

1. Sign in to Crazy Egg and open your account settings at `https://auth.app.crazyegg.com/settings`.

2. Under **Personal Access Tokens**, choose **Manage tokens**.
3. Enter a **Token name** that tells you where the token is used, for example the name of your integration.
4. Select **Read-only token** if the integration only needs to read data.
5. Choose **Create token**, then copy the token.

The token is shown only once. If you lose it, revoke it and create a new one.

Read-only and read & write tokens

Each token has one of two access levels, shown in the token list:

ACCESS	QUERIES	MUTATIONS
Read-only	Allowed, except those needing edit access	Rejected
Read & write	Allowed	Allowed

A read-only token receives this error for every mutation:

```
{
  "message": "Personal access token is read-only",
  "type": "authorization"
}
```

A read-only token also counts as read-only access in permission checks, so a query that needs edit access is denied with a `permission` error. Read-only access does not make every query free of side effects either: check the operation's description in the reference before calling it from an automated job.

What a token can access

A token belongs to the user who created it and to the account it was created in. Requests made with it have that user's role and site access in that account, and some features also depend on the account's subscription. A read & write token never grants more than the user has; a read-only token additionally blocks mutations and edit access.

When the user lacks the access an operation needs, the error has a `type` of `authorization`, `permission` or `subscription`, as described in Authentication and permission errors.

Revoke and delete tokens

The settings page lists your tokens with their access level and when they were created, last used and expire. Choose **Revoke** to stop a token from working immediately. A revoked token stays in the list until you choose **Delete**.

Tokens created on the settings page do not expire; the list shows **Expires: Never**. A token with an expiry date is rejected with the message `Expired token` once that date passes.

Keep tokens secret

A token can read and, unless it is read-only, change your account's data. Treat it like a password:

- Store it in a secret manager or an environment variable, not in source code.
- Send it only from your servers. Never include it in a web page or a mobile app; the API does not accept browser requests from other websites.
- Create a separate token for each integration, so you can revoke one without affecting the others.

Send a request

Headers

Send these headers with every request:

```
POST /api HTTP/1.1
Host: api.crazyegg.com
Authorization: Bearer ce_pat_...
Content-Type: application/json
```

Without `Content-Type: application/json`, the body is not read as JSON and the request fails with a syntax error.

Request body

The body is a JSON object with these fields:

FIELD	REQUIRED	CONTENTS
`query`	Yes	The GraphQL document as a string.
`variables`	No	A JSON object with values for the document's variables.
`operationName`	When the document has several operations	The name of the operation to run.

Pass changing values through `variables` instead of building them into the `query` string. This avoids escaping problems and keeps documents reusable.

Check your connection

This request checks that you reach the endpoint and that it accepts your request format:

```
curl https://api.crazyegg.com/api \
  -H "Authorization: Bearer $CRAZYEGG_PAT" \
  -H "Content-Type: application/json" \
  --data '{"query": "query { __typename }"}'
```

The response names the schema's query type:

```
{
  "data": {
    "__typename": "RootQueryType"
  }
}
```

`__typename` is part of every GraphQL schema and needs no authentication, so this response does not show that your token is valid. To check the token, run a query from the API reference (https://platform-api-docs.crazyegg.com/reference) that your user can read. With an invalid, revoked or expired token, that field returns `null` and the response has an `authentication` error.

Schema and reference

The API reference (https://platform-api-docs.crazyegg.com/reference) documents every operation and type. You can also download the complete schema in GraphQL SDL format from /schema.graphql (https://platform-api-docs.crazyegg.com/schema.graphql) to generate client code or validate documents before sending them.

When reading the schema:

- A type ending in `!` is never null. Other fields can return `null`.
- Custom scalars, such as `JSON`, are described on their own reference pages.
- Deprecated fields still work. Read the deprecation notice: some name a replacement, others explain why the field is no longer useful.

The reference shows the source revision it was generated from. It can be briefly ahead of or behind the deployed API. If a live response disagrees with the reference, trust the response.

Responses and errors

A response is a JSON object with `data`, `errors`, or both. A request can partly succeed: some fields return data while others return `null` with a matching entry in `errors`. Always check `errors`, even when `data` is present.

HTTP status codes

STATUS	WHEN
`200`	The request was read as GraphQL. Errors in the document or in its fields, including authentication errors, use `200`.
`400`	The request could not be read: invalid JSON, no `query`, or a malformed batch. Also a document over the size limit described in Rate limits.
`429`	The rate limit was reached. See When you reach a limit.
`5xx`	A server failure. Retry later.

A `400` caused by invalid JSON, and a server failure, can use a different body: `errors` is then an object rather than a list, and `request_id` sits at the top level. Check the status before parsing the body as a GraphQL response.

Error format

Each error has a human-readable `message`. When the server can classify the failure, it adds a machine-readable `extensions.code`:

```
{
  "errors": [
    {
      "message": "`limit` must be an integer of 1 or more; got 0.",
      "path": ["exampleField"],
      "locations": [{ "line": 2, "column": 3 }],
      "extensions": { "code": "INVALID_ARGUMENT", "argument": "limit" }
    }
  ]
}
```

Branch your code on the codes and types described below. Show `message` to people; its wording can change.

Error codes

CODE	MEANING	WHAT TO DO
`NOT_FOUND`	The ID or lookup matched nothing visible to you.	Check the ID and the selected site.
`INVALID_ARGUMENT`	An argument is outside what the operation accepts.	Fix the argument; `extensions.argument` names it when present.
`INVALID_JSON`	A string argument that must contain JSON is invalid.	Re-encode the value.
`INVALID_STATE`	The resource's current state does not allow this.	Read the resource and change its state first.
`VALIDATION`	Input failed validation rules.	Fix the fields listed in the error.
`FORBIDDEN`	Your account may not act on this resource.	Do not retry; the account needs access.
`RATE_LIMITED`	A rate limit was reached.	Wait and retry; see Rate limits.
`TIMEOUT`	The operation did not finish in time.	Retry with a narrower request.
`OPERATION_FAILED`	A failure with no more specific code.	Read the message to decide whether a retry can help.
`INTERNAL`	An unexpected server error.	Report the `request_id` from `extensions`.

Some operations document additional codes in their reference descriptions.

A document that does not match the schema, such as one with an unknown field, a wrong type or a missing argument, is rejected before it runs. Its errors carry `"code": "GRAPHQL_VALIDATION_FAILED"` next to `message`, not inside `extensions`, and the message names the problem. A document that is not valid GraphQL syntax returns a `syntax error` message with no code.

Authentication and permission errors

Authentication and permission errors carry a `type` field next to `message`:

`TYPE`	EXAMPLE `MESSAGE`	CAUSE
`authentication`	`Invalid token`	The token is wrong, revoked, or sent to `/gql`.
`authentication`	`Expired token`	The token has expired.

`TYPE`	EXAMPLE `MESSAGE`	CAUSE
`authentication`	`No user session`	The `Authorization` header is missing.
`authentication`	`Invalid or missing token`	A page editor operation got no valid credential.
`authorization`	`Personal access token is read-only`	A read-only token tried to run a mutation.
`authorization`	`User is not authorized`	The token's user lacks the access needed.
`permission`	`No permission for: <feature> - <access>`	The user's role or a read-only token lacks that access.
`subscription`	`No active subscription: <feature>`	The account's plan does not include the feature.

Some authorization errors also have the `FORBIDDEN` code.

Rate limits

Rate limits apply to requests made with personal access tokens.

What counts

The limit counts **root operations**, not HTTP requests. Each field selected at the top level of a document counts as one operation, including each alias. Nested fields are free.

All personal access tokens of an account share one allowance. Creating more tokens does not raise the limit.

Limits

Every request must fit all three limits:

- 100 operations per 5 seconds
- 500 operations per 5 minutes
- 1,000 operations per hour

Each limit uses a fixed window: it starts with the first operation after the previous window ended and resets when its period is over. Around a reset, a short burst can reach about twice a limit.

The limiter admits or rejects a whole document before any of it runs. When a document needs slightly more than the remaining allowance, it may exceed it by up to 10% of its own cost. A single document can contain at most 111 root operations.

When you reach a limit

A request that exceeds the allowance returns HTTP `429` with a `Retry-After` header, in seconds. The body has no `data`:

```
{
  "errors": [
    {
      "message": "Rate limit exceeded",
      "extensions": {
        "code": "RATE_LIMITED",
        "request_cost": 35,
        "limit": 1000,
        "window_seconds": 3600,
        "retry_after_seconds": 2741
      }
    }
  ]
}
```

Wait at least `Retry-After` seconds before sending the request again. Nothing in the rejected document ran.

A document with more than 111 root operations returns HTTP `400` without `Retry-After`:

```
{
  "errors": [
    {
      "message": "Document exceeds rate limit maximum",
      "extensions": {
        "code": "RATE_LIMIT_DOCUMENT_TOO_LARGE",
        "request_cost": 112,
        "max_request_cost": 111
      }
    }
  ]
}
```

Retrying the same document always fails. Split it into documents of 111 root operations or fewer.

Batching

There are two ways to send several operations at once. They count against the same allowance but behave differently when a limit is reached.

Several fields in one document

A single document can select several root fields, with aliases if you need the same field more than once. Each root field counts as one operation. The whole document is admitted or rejected together, and a rejection uses the HTTP status codes described in When you reach a limit. An admitted document can still partly succeed.

JSON array batches

The endpoint also accepts a JSON array of request objects. Each object can carry an `id`, which the response repeats. In this example, replace `__typename` with operations from the reference:

```
[
  { "id": "1", "query": "query { __typename }" },
  { "id": "2", "query": "query Second { __typename }" }
]
```

The response is an array in the same order. Each entry wraps its result in `payload`:

```
[
  { "id": "1", "payload": { "data": { "__typename": "RootQueryType" } } },
  { "id": "2", "payload": { "data": { "__typename": "RootQueryType" } } }
]
```

Each entry is counted and admitted on its own, so some entries can succeed while others are rejected. Once the batch is read, it returns HTTP `200` without `Retry-After`, even when entries were rate limited. Check `payload.errors` in every entry to decide what to retry. A malformed batch, or an entry without a `query`, rejects the whole request with HTTP `400`.

Subscriptions

The schema includes subscriptions, but they are not supported for personal access token integrations: the GraphQL WebSocket transport that delivers them does not accept personal access tokens. Use queries to read current data instead.

Page editor operations

A few operations serve the Crazy Egg page editor; their reference pages say so. Besides personal access tokens, they accept the page editor's own credential, and they check access to the editor, page or site involved. Their descriptions explain the access each one needs.

Getting help

When you contact Crazy Egg support about a failed request, include the time of the request, the HTTP status, the error `message` and code, and the `request_id` if the response has one. Never send your personal access token.