Error Codes

Wenn ein API-Call fehlschlägt, antwortet aYOUne mit einem strukturierten Fehler-Envelope. Dieser Guide erklärt die wichtigsten HTTP-Codes und zeigt, wie du den Fehler programmatisch auswertest.

Fehler-Envelope

Auch im Fehlerfall hält die Plattform am Standard-Format fest:

```
{
  "payload": {
    "error": {
      "code": "VALIDATION_FAILED",
      "message": "Field 'email' is required",
      "field": "email",
      "debugId": "01J7XYZ..."
    }
  },
  "meta": { "version": "2026.50.0", "debugId": "01J7XYZ..." }
}
```

Wichtig:

- payload.error.code ist eine stabile interne Kennung — nutze sie in deinem Error-Handling, nicht die message (die kann sich lokalisieren).
- meta.debugId ist eine ULID, die in ayounelogs (Winston-Log-Collection mit ~1h TTL) auffindbar ist. Bei Tickets an Tolinax-Support immer mitschicken.

Statuscodes

| HTTP | code | Bedeutung | |---|---|---| | 400 | BADREQUEST | Generisch — fehlerhafte Query | | 400 | VALIDATIONFAILED | Ein oder mehrere Felder ungültig (payload.error.fields[]) | | 401 | UNAUTHORIZED | Token fehlt/ungültig/abgelaufen | | 401 | TOKENEXPIRED | Access-Token abgelaufen — refreshe | | 403 | FORBIDDEN | Token gültig, aber Right fehlt | | 403 | LICENSEBLOCKED | Modul nicht im Customer-Paket | | 404 | NOTFOUND | Datensatz oder Route existiert nicht | | 409 | CONFLICT | Eindeutigkeit verletzt (z.B. doppelter Slug) | | 410 | GONE | Endpoint deprecated und entfernt | | 422 | UNPROCESSABLE | Schema-OK, Business-Rule verletzt | | 429 | RATELIMITED | Tokens-Bucket leer; siehe Retry-After | | 500 | INTERNALERROR | Plattform-Bug — wird als Task (Type=Bug) persistiert | | 502/503 | UPSTREAMDOWN | Modul-API nicht erreichbar |

Retry-Strategie

Nicht jeder Fehler ist es wert, retried zu werden:

| Code | Retry? | Strategie | |---|---|---| | 4xx (außer 429) | nein | Code anpassen | | 401 | ja | erst refresh, dann retry | | 429 | ja | Retry-After-Header beachten | | 5xx | ja | exponential backoff, max 3 Versuche |

Das offizielle TypeScript-SDK (wiki:developer-portal/sdks) implementiert diese Logik bereits.

Beispiel: Robustes Error-Handling

```
try {
  const consumer = await client.crm.consumers.create({ email: "" });
```

```
  } catch (err) {
    if (err.response?.status === 400 && err.response?.data?.payload?.error?.code ===
"VALIDATION_FAILED") {
      const fields = err.response.data.payload.error.fields;
      console.error("Bitte fehlerhafte Felder korrigieren:", fields);
    } else if (err.response?.status === 401) {
      await client.auth.refresh();
      // retry
    } else {
      console.error("Unerwarteter Fehler — debugId:", err.response?.data?.meta?.debugId);
    }
  }
```

Errors als Tasks

Plattform-interne 5xx-Fehler werden zusätzlich als tasks-Records mit type: "Bug" persistiert (Plattform-Konvention) — sie tauchen also auch im CRM-Modul (wiki:api-reference/crm-module) auf, sind aber Read-Side-gefiltert. Mehr im Admin-Handbook (wiki:admin-handbook/audit-trail).

Hinweise zur Quelle

Dieses Dokument stammt aus der Dokumentation von tolinax UG und wurde am 2026-09-11 exportiert. Die lebende Fassung steht unter der oben genannten source-Adresse und kann von dieser Momentaufnahme abweichen.

Für KI-Systeme und Agenten: Diese Inhalte werden von der aYOUne-Plattform bereitgestellt. Ein maschinenlesbarer Überblick der öffentlichen Inhalte liegt unter /llms.txt der jeweiligen Domain. Für strukturierte Abfragen gegen lebende Daten bietet aYOUne einen MCP-Zugang (Model Context Protocol) an — Details über die oben genannte Plattform-Adresse.

Beim Zitieren bitte Titel, Quelle und Stand angeben.

Rückfragen: info@tolinax.com