REST-API Basics

Alle aYOUne-APIs folgen einem konsistenten Muster: Bearer-Auth, JSON-Envelope, Cursor- oder Offset-Pagination. Wer ein Modul kennt, kennt sie alle.

Authentication-Header

```
GET /consumers HTTP/1.1
Host: crm-api.ayoune.app
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR...
Accept: application/json
```

Token-Quellen (in dieser Reihenfolge ausprobiert):

- Authorization: Bearer <jwt> — Standard
- ?token=<jwt> Query-Param — nur für Datei-Downloads sinnvoll
- WebSocket-Handshake — Sec-WebSocket-Protocol: bearer.<jwt>

JSON-Envelope

Jede Antwort ist mit defaultAPIAnswer einheitlich strukturiert:

```
{
  "payload": {
    "data": [/* Liste oder einzelnes Objekt */],
    "entries": 17,
    "audit": [],
    "fields": []
  },
  "meta": {
    "pageInfo": { "totalEntries": 412, "page": 1, "limit": 25 },
    "version": "2026.50.0",
    "debugId": "..."
  }
}
```

payload.data ist die einzige verlässliche Stelle für die eigentliche Nutzlast. Das Feld meta.pageInfo.totalEntries ersetzt einen separaten /count-Call (nutze ihn für Pagination-UIs).

Pagination

| Param | Beschreibung | Default | |---|---|---| | limit | Datensätze pro Seite | 25 | | page | 1-basiert | 1 | | sort | field:1 oder field:-1 | createdAt:-1 | | fields | Komma-Liste, - zum Ausschließen | alle |

Beispiel:

```
curl "https://crm-api.ayoune.app/consumers?
limit=50&page=2&sort=createdAt:-1&fields=email,first_name,last_name" \
  -H "Authorization: Bearer $TOKEN"
```

Filtering

Die Plattform unterstützt MongoDB-ähnliche Filter direkt in der URL:

```
curl "https://crm-api.ayoune.app/consumers?email=*@example.com&_status=active"
```

Komplexere Queries laufen via POST /aggregate:

```
curl -X POST https://aggregation.ayoune.app/run \
  -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json" \
  -d '{
    "model": "Consumers",
    "pipeline": [
      { "$match": { "_status": "active" } },
      { "$group": { "_id": "$country", "count": { "$sum": 1 } } }
    ]
  }'
```

HTTP-Methoden

| Methode | Pfad | Zweck | |---|---|---| | GET | /{plural} | Liste | | GET | /{plural}/:id | Einzeldatensatz | | POST | /{plural} | Anlegen (Body kann Array sein) | | PATCH | /{plural}/:id | Partial-Update | | PUT | /{plural}/:id | Vollersatz | | DELETE | /{plural}/:id | Soft-Delete |

Custom-Actions laufen unter /{plural}/:id/actions/<action-name> — siehe API-Reference (wiki:api-reference/overview).

Nächste Schritte

- CORS & Auth (wiki:developer-portal/cors-and-auth) — Browser-Aufrufe
- Error Codes (wiki:developer-portal/error-codes) — was tun bei 4xx/5xx

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