Entwickler-Portal

SDKs, REST-Rezepte und Integrationsmuster.

10 Seiten in dieser Sammlung.

Developer Portal Übersicht

Developer Portal Übersicht

Das aYOUne Developer Portal bündelt alles, was du brauchst, um die Plattform aus deinen eigenen Anwendungen heraus anzusprechen — von simplen REST-Calls über Webhooks bis zur Integration via offiziellen TypeScript- oder PHP-SDKs.

Wer dieses Portal liest

aYOUne ist eine multi-tenant Business-as-a-Service-Plattform mit ~330 Microservices, die hinter einem einheitlichen Routing-Pattern https://{module}-api.ayoune.app zusammenfließen. Wer Drittsysteme anbinden möchte (Shop, ERP, Custom-Frontend, Mobile-App, Zapier-ähnliche Flows) findet hier sämtliche Bausteine. Im Gegensatz zum Admin-Handbook (wiki:admin-handbook/overview), das für Customer-Admins gedacht ist, richtet sich dieses Portal an Entwickler.

Schnellnavigation

| Thema | Inhalt | |---|---| | Erste Schritte (wiki:developer-portal/getting-started) | Account, JWT-Token, erster API-Call | | REST-API Basics (wiki:developer-portal/rest-api-basics) | Auth-Header, JSON-Envelope, Pagination | | Webhooks (wiki:developer-portal/webhooks) | Outbound-Webhooks abonnieren | | SDKs (wiki:developer-portal/sdks) | TypeScript- und PHP-SDK | | CORS & Auth (wiki:developer-portal/cors-and-auth) | Browser-Calls, Token-Lifecycle | | Error Codes (wiki:developer-portal/error-codes) | HTTP-Statuscodes & payload.error |

Architekturüberblick

Jedes Domänen-Modul (crm, cms, marketing, automation, config, ...) exponiert eine eigene Express-API. Alle Routen folgen demselben Schema:

```
GET    https://crm-api.ayoune.app/consumers
GET    https://crm-api.ayoune.app/consumers/:id
POST   https://crm-api.ayoune.app/consumers
PATCH  https://crm-api.ayoune.app/consumers/:id
DELETE https://crm-api.ayoune.app/consumers/:id
```

Hinter den Kulissen verwendet jede Route den createAPIHandler-Faktor und antwortet mit dem standardisierten defaultAPIAnswer-Envelope:

```
{
  "payload": { "data": [...], "entries": 42 },
  "meta": { "pageInfo": { "totalEntries": 42 } }
}
```

Welche Module gibt es?

Eine vollständige Modul-Liste mit Routing-Hosts und Highlight-Endpoints findest du in der API-Reference (wiki:api-reference/overview). Über 25 Domänen — von crm über cms, marketing, automation, ai, chat, config bis zu hub — stehen dir mit demselben Auth-Token offen, sofern dein Tenant das jeweilige Paket gebucht hat.

Nächster Schritt

Erste Schritte (wiki:developer-portal/getting-started) — Token holen, ersten Call machen.

Erste Schritte

Erste Schritte

In diesem Guide holst du dir einen JWT-Token, machst deinen ersten authentifizierten API-Call und verstehst, wie das Multi-Tenant-Modell funktioniert.

1. Account & Customer

aYOUne ist multi-tenant: Jeder Datensatz lebt in einem Customer-Scope (entspricht dem Tenant). Dein Login-Account (aYOUneUsers) gehört zu einem oder mehreren Customers (aYOUneCustomers); zwischen ihnen wechselst du via auth.ayoune.app/changecustomer oder im Frontend per Customer-Switcher.

2. JWT-Token holen

Der einfachste Weg führt über auth.ayoune.app:

```
curl -X POST https://auth.ayoune.app/login \
  -H "Content-Type: application/json" \
  -d '{"email":"dev@example.com","password":"<password>"}'
```

Die Antwort enthält ein JWT, das du im weiteren Verlauf als Authorization: Bearer <token> mitschickst:

```
{
  "payload": {
    "token": "eyJhbGciOiJIUzI1NiIsInR...",
    "user": { "_id": "...", "email": "dev@example.com" },
    "customer": { "_id": "...", "name": "Demo GmbH" }
  }
}
```

Alternativ gibt es einen Service-Token-Flow (für Server-zu-Server-Integrationen) — siehe CORS & Auth (wiki:developer-portal/cors-and-auth).

3. Erster API-Call

Hole dir die ersten 10 Consumer aus dem CRM:

```
curl https://crm-api.ayoune.app/consumers?limit=10 \
  -H "Authorization: Bearer $TOKEN"
```

Antwortformat:

```
{
  "payload": {
    "data": [{ "_id": "...", "first_name": "Max", "last_name": "Hanrieder" }],
    "entries": 10
  },
  "meta": { "pageInfo": { "totalEntries": 1247, "page": 1, "limit": 10 } }
}
```

4. Token in der CLI verwenden

Wenn du lieber mit der ay-CLI arbeitest:

```
ay login            # Browser-basiert
ay storage set token <jwt>   # vorhandenes Token persistieren
ay list consumers   # erster Call
```

Nächste Schritte

- REST-API Basics (wiki:developer-portal/rest-api-basics) — Header, Pagination, Filter
- Webhooks (wiki:developer-portal/webhooks) — Push statt Pull
- SDKs (wiki:developer-portal/sdks) — TypeScript & PHP

::: tip Für die ersten Experimente kannst du den Demo-Account verwenden — er ist superuser-frei und hat realistische Test-Daten. Frag bei Tolinax-Support nach den Zugangsdaten. :::

REST-API Basics

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

Webhooks

Webhooks

Webhooks liefern aYOUne-Events in Echtzeit an deinen HTTP-Endpoint — ohne Polling, ohne Cron. Du registrierst eine URL und einen Event-Filter; die Plattform sendet alle Events, die zum Filter passen, als signiertes JSON-POST.

Wann Webhooks sinnvoll sind

- Neuer Lead/Consumer im CRM !’ ERP-System anlegen
- Coupon eingelöst !’ Shop-Bestand reduzieren
- Pipeline grün !’ Slack-Notification
- Custom-Function-Ergebnis !’ Drittsystem füttern

Wenn du Pull statt Push brauchst, sind die REST-API Basics (wiki:developer-portal/rest-api-basics) der richtige Weg.

Webhook anlegen

```
curl -X POST https://automation-api.ayoune.app/webhooks \
  -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json" \
  -d '{
    "name": "Lead -> HubSpot",
    "url": "https://hooks.example.com/incoming/abc123",
    "events": ["consumers.created", "consumers.updated"],
    "secret": "wh_secret_min32chars_xxxxxxxxxxxxxxxx",
    "active": true
  }'
```

| Feld | Pflicht | Beschreibung | |---|---|---| | name | ja | Anzeige-Name | | url | ja | Empfänger (HTTPS empfohlen) | | events | ja | Array aus <plural>.<verb> | | secret | nein | min. 32 Zeichen für HMAC-Signatur | | active | nein | default true | | headers | nein | Zusätzliche Request-Header |

Event-Format

Jeder Webhook-Aufruf ist ein POST mit folgendem Body:

```
{
  "event": "consumers.created",
  "timestamp": "2026-04-25T10:00:00.000Z",
  "customer": "64a1b2c3d4e5f60012345678",
  "data": { "_id": "...", "email": "max@example.com", "first_name": "Max" }
}
```

Signatur prüfen

Die Plattform setzt den Header X-aYOUne-Signature auf sha256=<HMAC-SHA256(secret, body)>. Verifiziere ihn in deinem Endpoint:

```
import crypto from "crypto";

function verify(req) {
```

```
  const expected = "sha256=" + crypto
    .createHmac("sha256", process.env.WH_SECRET!)
    .update(req.rawBody)
    .digest("hex");
  return crypto.timingSafeEqual(
    Buffer.from(expected),
    Buffer.from(req.headers["x-ayoune-signature"]),
  );
}
```

Retries

Antwortet dein Endpoint nicht mit 2xx, wiederholt die Plattform mit exponential backoff (max. 5 Versuche, 1 min !’ 16 min). Nach 5 Fehlversuchen landet das Event in der webhookretries-Collection und kann via Admin-UI oder ay webhooks retry <id> reanimiert werden.

Lokale Tests

Für lokale Entwicklung empfiehlt sich ngrok (https://ngrok.com/) oder Cloudflare-Tunnels:

```
ngrok http 3000
# Public URL als webhook.url eintragen
```

Siehe auch: Automation-Module (wiki:api-reference/automation-module).

CORS & Auth

CORS & Auth

Wenn du aus einer Browser-App heraus die aYOUne-API ansprichst, brauchst du sowohl korrektes CORS-Setup als auch ein durchdachtes Token-Lifecycle-Modell. Beides ist hier zusammengefasst.

CORS-Verhalten

Per Default akzeptieren die Module-APIs nur Origins, die in der Customer-Konfiguration unter aYOUneCustomers.allowedOrigins[] registriert sind. Trag deinen Frontend-Origin dort ein:

```
ay update ayounecustomers <customerId> \
  --set allowedOrigins='["https://app.firma.tld","https://staging.firma.tld"]'
```

Wildcards (*.firma.tld) sind unterstützt. Auf Self-Hosted-Instanzen kannst du via Umgebungsvariable CORSORIGINOVERRIDE zusätzlich global öffnen — siehe Environment Variables (wiki:self-hosting/environment-variables).

JWT-Lifecycle

aYOUne arbeitet mit zwei Tokens:

Token	TTL	Zweck
Access-Token	15 min	Bei jedem API-Call mitgeschickt
Refresh-Token	30 Tage	Holt neue Access-Tokens

Access-Token erneuern

Erkennst du an einem Response mit 401, dass der Access-Token abgelaufen ist, hol dir den neuen via:

```
curl -X POST https://auth.ayoune.app/refresh \
  -H "Content-Type: application/json" \
  -d '{"refreshToken":"<refresh>"}'
```

Antwort: neues payload.token + neuer payload.refreshToken (Token-Rotation).

Logout & Revocation

```
curl -X POST https://auth.ayoune.app/logout \
  -H "Authorization: Bearer $TOKEN"
```

Logout setzt den Refresh-Token in die Blacklist. Access-Tokens laufen ohnehin in 15 min aus — wer höhere Sicherheit braucht, kann sie via ay storage set token schon vorher invalidieren.

Service-Token (Server-zu-Server)

Für Backend-Integrationen (z.B. dein ERP holt nachts Stündliche), nutze Service-Tokens:

```
ay create credentials "ERP-Sync" \
  --type service-account \
  --rights '["consumers.view","consumers.create"]'
```

Service-Tokens haben kein Refresh — du rotierst sie aktiv via ay credentials refresh <id>. Sie sind im Audit-Log als Aktor service:<name> zu erkennen.

SameSite & Cookies

aYOUne speichert Tokens nicht in Cookies — die Plattform vertraut auf Authorization-Header. Wenn du in einer SPA arbeitest, leg den Access-Token im Speicher (z.B. sessionStorage) ab, nie im localStorage. Refresh-Tokens gehören ausschließlich in HttpOnly + Secure + SameSite=Strict Cookies, falls du sie browserseitig hältst.

Multi-Tenant Customer-Switch

Ein User kann mehreren Customers angehören. Wechsle innerhalb derselben JWT-Session via:

```
curl -X GET "https://auth.ayoune.app/changecustomer?customerId=<id>" \
  -H "Authorization: Bearer $TOKEN"
```

Antwort: ein neues JWT mit dem gewählten Customer-Scope. Alte Tokens bleiben gültig, zeigen aber auf den vorherigen Customer.

Siehe auch: Audit-Trail (wiki:admin-handbook/audit-trail).

SDKs

SDKs

Statt direkt mit fetch oder curl zu arbeiten, kannst du die offiziellen SDKs verwenden. Sie kapseln Auth, Retries, Pagination, Typing und Error-Handling.

TypeScript-SDK

@tolinax/ayoune-sdk ist auf npm verfügbar und voll typisiert. Es teilt sich die Interfaces mit der Plattform-internen @tolinax/ayoune-interfaces-Library — ein Feld wird also nicht "ein bisschen anders" heißen.

```
npm install @tolinax/ayoune-sdk
```

```
import { AyouneClient } from "@tolinax/ayoune-sdk";

const client = new AyouneClient({
  token: process.env.AYOUNE_TOKEN!,
  // optional: customerId, baseUrl-Override für Self-Hosted-Instanzen
});

const consumers = await client.crm.consumers.list({
  filter: { _status: "active" },
  limit: 50,
  sort: { createdAt: -1 },
});

console.log(`Gefunden: ${consumers.totalEntries} Konsumenten`);

const created = await client.crm.consumers.create({
  email: "neu@example.com",
  first_name: "Max",
  last_name: "Mustermann",
});
```

Streaming-Responses

Für große Listen unterstützt das SDK NDJSON-Streaming via async iterator:

```
for await (const consumer of client.crm.consumers.stream({ limit: 5000 })) {
  console.log(consumer.email);
}
```

Self-Hosting

Bei einer Self-Hosted-Instanz reicht ein baseUrl-Override:

```
const client = new AyouneClient({
  token,
  baseUrl: "https://api.intern.firma.tld",
});
```

PHP-SDK

Für PHP-basierte Shops (Shopware, Magento, eigene Apps) gibt es tolinax/ayoune-php-sdk via Composer:

```
composer require tolinax/ayoune-php-sdk
```

```
use Tolinax\Ayoune\Client;

$client = new Client([
  'token'    => getenv('AYOUNE_TOKEN'),
  'baseUrl'  => 'https://crm-api.ayoune.app',
]);

$result = $client->module('crm')->collection('consumers')->list([
```

```
  'filter' => ['_status' => 'active'],
  'limit'  => 50,
]);

foreach ($result->data as $consumer) {
  echo $consumer->email . PHP_EOL;
}
```

Generated Clients

Für andere Sprachen (Go, Python, Java, …) gibt es einen OpenAPI-Generator. Die OpenAPI-Spezifikation lebt unter https://hub-api.ayoune.app/openapi.json und wird automatisch aus den createAPIHandler-Definitionen abgeleitet.

```
npx @openapitools/openapi-generator-cli generate \
  -i https://hub-api.ayoune.app/openapi.json \
  -g go \
  -o ./ayoune-go-client
```

Versions-Strategie

Die SDKs folgen Calendar-Versioning (2026.x.y) — gleicher Rhythmus wie die Plattform. Bei größeren Änderungen findest du sie in den Release Notes (wiki:release-notes/overview).

::: warning Pinne in Production keine *.0.0 Major-Releases ungeprüft. Lies vorher Breaking Changes (wiki:release-notes/breaking-changes). :::

Error Codes

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).

Developer Portal Übersicht

Developer Portal Übersicht

Das aYOUne Developer Portal bündelt alles, was du brauchst, um die Plattform aus deinen eigenen

Anwendungen heraus anzusprechen — von simplen REST-Calls über Webhooks bis zur Integration via offiziellen TypeScript- oder PHP-SDKs.

Wer dieses Portal liest

aYOUne ist eine multi-tenant Business-as-a-Service-Plattform mit ~330 Microservices, die hinter einem einheitlichen Routing-Pattern https://{module}-api.ayoune.app zusammenfließen. Wer Drittsysteme anbinden möchte (Shop, ERP, Custom-Frontend, Mobile-App, Zapier-ähnliche Flows) findet hier sämtliche Bausteine. Im Gegensatz zum Admin-Handbook (wiki:admin-handbook/overview), das für Customer-Admins gedacht ist, richtet sich dieses Portal an Entwickler.

Schnellnavigation

| Thema | Inhalt | |---|---| | Erste Schritte (wiki:developer-portal/getting-started) | Account, JWT-Token, erster API-Call | | REST-API Basics (wiki:developer-portal/rest-api-basics) | Auth-Header, JSON-Envelope, Pagination | | Webhooks (wiki:developer-portal/webhooks) | Outbound-Webhooks abonnieren | | SDKs (wiki:developer-portal/sdks) | TypeScript- und PHP-SDK | | CORS & Auth (wiki:developer-portal/cors-and-auth) | Browser-Calls, Token-Lifecycle | | Error Codes (wiki:developer-portal/error-codes) | HTTP-Statuscodes & payload.error |

Architekturüberblick

Jedes Domänen-Modul (crm, cms, marketing, automation, config, ...) exponiert eine eigene Express-API. Alle Routen folgen demselben Schema:

```
GET    https://crm-api.ayoune.app/consumers
GET    https://crm-api.ayoune.app/consumers/:id
POST   https://crm-api.ayoune.app/consumers
PATCH  https://crm-api.ayoune.app/consumers/:id
DELETE https://crm-api.ayoune.app/consumers/:id
```

Hinter den Kulissen verwendet jede Route den createAPIHandler-Faktor und antwortet mit dem standardisierten defaultAPIAnswer-Envelope:

```
{
  "payload": { "data": [...], "entries": 42 },
  "meta": { "pageInfo": { "totalEntries": 42 } }
}
```

Welche Module gibt es?

Eine vollständige Modul-Liste mit Routing-Hosts und Highlight-Endpoints findest du in der API-Reference (wiki:api-reference/overview). Über 25 Domänen — von crm über cms, marketing, automation, ai, chat, config bis zu hub — stehen dir mit demselben Auth-Token offen, sofern dein Tenant das jeweilige Paket gebucht hat.

Nächster Schritt

Erste Schritte (wiki:developer-portal/getting-started) — Token holen, ersten Call machen.

Erste Schritte

Erste Schritte

In diesem Guide holst du dir einen JWT-Token, machst deinen ersten authentifizierten API-Call und verstehst, wie das Multi-Tenant-Modell funktioniert.

1. Account & Customer

aYOUne ist multi-tenant: Jeder Datensatz lebt in einem Customer-Scope (entspricht dem Tenant). Dein Login-Account (aYOUneUsers) gehört zu einem oder mehreren Customers (aYOUneCustomers); zwischen ihnen wechselst du via auth.ayoune.app/changecustomer oder im Frontend per Customer-Switcher.

2. JWT-Token holen

Der einfachste Weg führt über auth.ayoune.app:

```
curl -X POST https://auth.ayoune.app/login \
  -H "Content-Type: application/json" \
  -d '{"email":"dev@example.com","password":"<password>"}'
```

Die Antwort enthält ein JWT, das du im weiteren Verlauf als Authorization: Bearer <token> mitschickst:

```
{
  "payload": {
    "token": "eyJhbGciOiJIUzI1NiIsInR...",
    "user": { "_id": "...", "email": "dev@example.com" },
    "customer": { "_id": "...", "name": "Demo GmbH" }
  }
}
```

Alternativ gibt es einen Service-Token-Flow (für Server-zu-Server-Integrationen) — siehe CORS & Auth (wiki:developer-portal/cors-and-auth).

3. Erster API-Call

Hole dir die ersten 10 Consumer aus dem CRM:

```
curl https://crm-api.ayoune.app/consumers?limit=10 \
  -H "Authorization: Bearer $TOKEN"
```

Antwortformat:

```
{
  "payload": {
    "data": [{ "_id": "...", "first_name": "Max", "last_name": "Hanrieder" }],
    "entries": 10
  },
  "meta": { "pageInfo": { "totalEntries": 1247, "page": 1, "limit": 10 } }
}
```

4. Token in der CLI verwenden

Wenn du lieber mit der ay-CLI arbeitest:

```
ay login            # Browser-basiert
```

```
ay storage set token <jwt>   # vorhandenes Token persistieren
ay list consumers   # erster Call
```

Nächste Schritte

- REST-API Basics (wiki:developer-portal/rest-api-basics) — Header, Pagination, Filter
- Webhooks (wiki:developer-portal/webhooks) — Push statt Pull
- SDKs (wiki:developer-portal/sdks) — TypeScript & PHP

::: tip Für die ersten Experimente kannst du den Demo-Account verwenden — er ist superuser-frei und hat realistische Test-Daten. Frag bei Tolinax-Support nach den Zugangsdaten. :::

CORS & Auth

CORS & Auth

Wenn du aus einer Browser-App heraus die aYOUne-API ansprichst, brauchst du sowohl korrektes CORS-Setup als auch ein durchdachtes Token-Lifecycle-Modell. Beides ist hier zusammengefasst.

CORS-Verhalten

Per Default akzeptieren die Module-APIs nur Origins, die in der Customer-Konfiguration unter aYOUneCustomers.allowedOrigins[] registriert sind. Trag deinen Frontend-Origin dort ein:

```
ay update ayounecustomers <customerId> \
  --set allowedOrigins='["https://app.firma.tld","https://staging.firma.tld"]'
```

Wildcards (*.firma.tld) sind unterstützt. Auf Self-Hosted-Instanzen kannst du via Umgebungsvariable CORSORIGINOVERRIDE zusätzlich global öffnen — siehe Environment Variables (wiki:self-hosting/environment-variables).

JWT-Lifecycle

aYOUne arbeitet mit zwei Tokens:

| Token | TTL | Zweck | |---|---|---| | Access-Token | 15 min | Bei jedem API-Call mitgeschickt | | Refresh-Token | 30 Tage | Holt neue Access-Tokens |

Access-Token erneuern

Erkennst du an einem Response mit 401, dass der Access-Token abgelaufen ist, hol dir den neuen via:

```
curl -X POST https://auth.ayoune.app/refresh \
  -H "Content-Type: application/json" \
  -d '{"refreshToken":"<refresh>"}'
```

Antwort: neues payload.token + neuer payload.refreshToken (Token-Rotation).

Logout & Revocation

```
curl -X POST https://auth.ayoune.app/logout \
  -H "Authorization: Bearer $TOKEN"
```

Logout setzt den Refresh-Token in die Blacklist. Access-Tokens laufen ohnehin in 15 min aus — wer

höhere Sicherheit braucht, kann sie via ay storage set token schon vorher invalidieren.

Service-Token (Server-zu-Server)

Für Backend-Integrationen (z.B. dein ERP holt nachts Stündliche), nutze Service-Tokens:

```
ay create credentials "ERP-Sync" \
  --type service-account \
  --rights '["consumers.view","consumers.create"]'
```

Service-Tokens haben kein Refresh — du rotierst sie aktiv via ay credentials refresh <id>. Sie sind im Audit-Log als Aktor service:<name> zu erkennen.

SameSite & Cookies

aYOUne speichert Tokens nicht in Cookies — die Plattform vertraut auf Authorization-Header. Wenn du in einer SPA arbeitest, leg den Access-Token im Speicher (z.B. sessionStorage) ab, nie im localStorage. Refresh-Tokens gehören ausschließlich in HttpOnly + Secure + SameSite=Strict Cookies, falls du sie browserseitig hältst.

Multi-Tenant Customer-Switch

Ein User kann mehreren Customers angehören. Wechsle innerhalb derselben JWT-Session via:

```
curl -X GET "https://auth.ayoune.app/changecustomer?customerId=<id>" \
  -H "Authorization: Bearer $TOKEN"
```

Antwort: ein neues JWT mit dem gewählten Customer-Scope. Alte Tokens bleiben gültig, zeigen aber auf den vorherigen Customer.

Siehe auch: Audit-Trail (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