

UTM-Tracking in api.langify

Wie langify die Herkunft einer Installation (UTM-Parameter und Referrer) erfasst, einem Shop zuordnet und bei späteren Plan-Wechseln auswertet.

Stand: 05.10.2026 · Branch `dev` von `api.langify`

Überblick

Das Tracking läuft in vier Schritten in der API, ausgewertet wird es im Intranet (Schritt 5). Der Schlüssel, der alles zusammenhält, ist die `install_uid` (eine UUID v4). Sie entsteht beim ersten Kontakt, wird im Browser als Cookie gespeichert und beim OAuth-Callback zurück an die API geschickt.

1**Erfassen – v1 /install**

Besucher klickt Kampagnen-Link. UTM + Referrer landen in `install_attribution`, die API liefert eine `install_uid`, die als Cookie gesetzt wird.

`POST /auth/remote/install-attribution`

2**Verknüpfen – OAuth-Callback**

Nach der Installation im App Store wird die `install_uid` aus dem Cookie dem Shop zugeordnet (nur bei Neu- oder Re-Install).

`POST /auth/remote/oauth/callback`

3**Auswerten – Plan-Wechsel**

`shop_plan_event` und GA4-Events werden mit den UTM-Daten der Installation geschrieben.

`ApplicationChargeRepository` → `PlanEventTracker`, `AnalyticsService`

4**Reporting-API**

Install-Attributionen und Plan-Events, abrufbar per JWT mit Scope `report`.

`GET /v2/api/staff/report/...`

5

Intranet – intern.lovely-app.com

Kennzahlen, Umsatz/MRR je Quelle, Plan-Journeys, PDF-Export.

`/report` , `/report/revenue`

1 Erfassen: `POST /auth/remote/install-attribution`

Dateien

- `src/Controller/Auth/Remote/InstallAttributionController.php`
- `src/Service/Auth/Remote/RemoteInstallAttributionService.php` → `record()`

Die v1-Seite `/install` (die Einstiegsseite zum App Store; sie liegt nicht in diesem Repo) ruft diesen Endpunkt serverseitig auf. Die API speichert dabei weder einen Cookie noch leitet sie weiter. Beides übernimmt der Aufrufer.

Authentifizierung: JWT mit Scope `remote-auth` , als `Authorization: Bearer <token>` oder als `?token=` (siehe `ApiTokenAuthTrait::validateApiToken()`).

Request-Body (JSON), alle Felder optional

Feld	Ziel-Spalte	Max. Länge
<code>utm_source</code>	<code>utm_source</code>	255
<code>utm_medium</code>	<code>utm_medium</code>	255
<code>utm_campaign</code>	<code>utm_campaign</code>	255
<code>utm_content</code>	<code>utm_content</code>	255
<code>utm_term</code>	<code>utm_term</code>	255
<code>landing_url</code>	<code>landing_url</code>	700
<code>referer</code>	<code>referrer</code>	700

Achtung: Das Request-Feld heißt `referer` (HTTP-Schreibweise), die DB-Spalte `referrer` .

Verarbeitung

- Jeder Wert wird getrimmt, auf die Spaltenlänge gekürzt (`mb_substr`), und ein leerer String wird zu `NULL` .

- `referrer_domain` wird aus dem Referrer berechnet: der Host ohne führendes `www.` (z. B. `https://www.google.com/search?... → google.com`).
- Die `install_uid` erzeugt die API selbst (`random_bytes` , RFC 4122 v4).
- `created_at` ist der Zeitpunkt der Erfassung. `shop_id` und `install_date` bleiben zunächst leer.

Antwort

```
{ "success": true, "install_uid": "3f6c2a1e-8b7d-4c3e-9a51-0d2f6e7b8c90" }
```

Der Aufrufer speichert die `install_uid` als Cookie und leitet in den Shopify App Store weiter.

Beispiel

```
curl -X POST https://api.langify-app.com/auth/remote/install-attribution \
-H "Authorization: Bearer $REMOTE_AUTH_JWT" \
-H "Content-Type: application/json" \
-d '{
    "utm_source": "newsletter",
    "utm_medium": "email",
    "utm_campaign": "herbst-2026",
    "landing_url": "https://langify-app.com/install?
utm_source=newsletter&utm_medium=email&utm_campaign=herbst-2026",
    "referrer": "https://www.google.com/"
}'
```

2 Verknüpfen: POST /auth/remote/oauth/callback

Dateien

- `src/Controller/Auth/Remote/LoginController.php` → `oauthCallback()`
- `src/Service/Auth/Remote/RemoteLoginService.php` → `processOAuthCallback()`
- `src/Service/Auth/Remote/RemoteInstallAttributionService.php` → `link()`

Nach der Installation im App Store läuft der Shopify-OAuth-Flow. Der v1-Callback leitet die Shopify-Query-Parameter an die API weiter und legt die `install_uid` aus dem Cookie bei:

```
{
  "query": { "shop": "example.myshopify.com", "code": "...", "hmac": "...", "...": "..." },
  "install_uid": "3f6c2a1e-8b7d-4c3e-9a51-0d2f6e7b8c90"
}
```

Ablauf

1. Die API prüft das JWT (Scope `remote-auth`) und die Shopify-HMAC.
2. Sie lädt den Shop oder legt ihn neu an.
3. Als **neue Installation** (`new_install = true`) zählt ein Shop, den es lokal noch nicht gibt, oder ein Shop mit gesetztem `uninstalled_at` (Re-Install).
4. Nur bei einer neuen Installation und vorhandener `install_uid` ruft die API `link()` auf: Sie sucht die Zeile mit dieser `install_uid` in `install_attribution` und setzt `shop_id` und `install_date = now()` . Gibt es keine passende Zeile, passiert nichts (Rückgabe `false` , kein Fehler).

Logins in einen bereits installierten Shop verändern die Attribution also nicht.

3 Auswerten: Plan-Wechsel

Dateien

- `src/Repository/ApplicationChargeRepository.php`
- `src/Service/PlanEventTracker.php`
- `src/Service/Google/AnalyticsService.php` → `trackPlanChange()`
- `src/Entity/ShopPlanEvent.php`

Bei jedem Plan-Wechsel lädt die API die Attribution des Shops (`InstallAttributionRepository::findByShop()`) und schreibt an zwei Stellen:

3a. Datenbank: `shop_plan_event`

`PlanEventTracker::recordPlanChange()` legt eine Zeile an mit:

- `old_plan` , `new_plan` , `price` , `currency` (aktuell immer `USD`) und `billing_interval` (aktuell immer `monthly`)
- `event_type` : `upgrade` , `downgrade` oder `cancel` . Upgrade oder Downgrade ergibt sich aus dem Vergleich der Plan-IDs (`ApplicationCharge::PLAN_IDS`). Ein Wechsel auf den Free-Plan ist `cancel` .
- `attribution_install_uid` : die `install_uid` der Attribution des Shops, falls vorhanden

Die Werte `trial_start` und `trial_end` sind im Enum definiert, werden derzeit aber nirgends geschrieben.

3b. Google Analytics 4 (Measurement Protocol)

`AnalyticsService::trackPlanChange()` sendet zwei Events an `https://www.google-analytics.com/mp/collect` :

Event	Zweck
<code>plan_change</code>	Haupt-Event für Analysen
<code>plan_<targetPlan></code>	Je Zielplan, für einfache Conversion-Reports

Parameter

- `source_plan`, `target_plan`, `plan_change` (`<alt>_to_<neu>`) und `value = 1`
- falls eine Attribution existiert, die UTM-Werte mit dem Prefix `install_`: `install_source`, `install_medium`, `install_campaign`, `install_content` und `install_term`. Werte, die `NULL` sind, werden weggelassen.

Details

- `client_id = sha1(shopify_shop_id)` . Die Events hängen also an einer festen, pseudonymen ID je Shop und nicht an der Browser-Session des Besuchers.
- Konfiguration über `GOOGLE_ANALYTICS_MEASUREMENT_ID` und `GOOGLE_ANALYTICS_API_SECRET` . Ist eine der beiden leer, wird nichts gesendet.
- Die API sendet derzeit immer `debug_mode=true` mit.
- Zum Testen gibt es `src/Controller/Debug/GoogleTestController.php` .

Wann es ausgelöst wird (`ApplicationChargeRepository`)

Methode	Situation
<code>changePlanGetConfirmationUrl()</code>	Direkter Planwechsel ohne Bestätigung durch den Händler
<code>checkCharge()</code> , aktiver Charge	Der Händler hat den Charge bestätigt
<code>checkCharge()</code> , <code>declined</code> , <code>expired</code> oder <code>notfound</code>	Charge abgelehnt oder abgelaufen; die API verbucht einen Wechsel auf Free (<code>cancel</code>)

4 Reporting: `/v2/api/staff/report`

Datei: `src/Controller/Staff/ReportController.php`

Authentifizierung: JWT mit Scope `report`

Endpunkt	Inhalt	Filter
GET /install-attributions	Zeilen aus <code>install_attribution</code>	<code>from</code> , <code>to</code> (auf <code>created_at</code>), <code>limit</code> (max. 500, Standard 100), <code>offset</code>
GET /plan-events	Zeilen aus <code>shop_plan_event</code>	wie oben, zusätzlich <code>event_type</code>

Beide Endpunkte sortieren nach `created_at DESC`.

5 Auslesen im Intranet: `intern.lovely-app.com/report`

Das Intranet (Repo `intern.lovely-app`, PHP/Phalcon) liest die beiden Report-Endpunkte aus Abschnitt 4 und wertet sie aus. Eine eigene Datenbank oder Kopie der Daten gibt es dort nicht. Jede Auswertung kommt live aus der API, für eine Stunde in Redis zwischengespeichert.

Dateien (Repo `intern.lovely-app`)

- `app/engines/ReportApiService.php` : API-Client (Guzzle), Pagination, Redis-Cache
- `app/controllers/ReportController.php` : Auswertung (Kennzahlen, Verknüpfung, Revenue, Plan-Journeys), PDF
- `app/views/report/index.phtml`, `tab.phtml`, `revenue.phtml` : Oberfläche (Tabs werden per htmx nachgeladen)

Zugang

Seite	URL	Rolle
Attribution Report	<code>/report</code>	<code>reporting</code> oder <code>admin</code>
Revenue Report	<code>/report/revenue</code>	<code>reporting</code> oder <code>admin</code>

Im Menü stehen beide unter **Reports** (dort auch „Plan Overview“ und „Plan Flow“, die nicht auf den UTM-Daten beruhen).

Konfiguration (Umgebungsvariablen des Intranets, CapRover bzw. `.env.local`)

Variable	Bedeutung
<code>LANGIFY_REPORT_API_URL</code>	Basis-URL der API, live <code>https://api.langify-app.com</code>
<code>LANGIFY_REPORT_API_TOKEN</code>	JWT mit Scope <code>report</code> , wird als <code>Authorization: Bearer</code> gesendet
<code>LANGIFY_REPORT_RESOLVE</code>	optional, nur lokal: <code>CURLOPT_RESOLVE</code> -Eintrag (<code>host:port:ip</code>), um z. B. <code>api.langify.dvlbox.de</code> auf eine Docker-IP zu zeigen

- Fehlt URL oder Token, zeigt die Seite „API nicht konfiguriert“.
- Lehnt die API den Token ab (HTTP 401/403), zeigt die Seite das mit dem HTTP-Code an.
- **Ablauf des Tokens:** Das Intranet liest `exp` aus dem JWT und warnt **14 Tage** vor Ablauf oben auf der Seite. Dann einen neuen Token mit Scope `report` erzeugen und `LANGIFY_REPORT_API_TOKEN` tauschen.

Datenabruf

- **Pagination:** `ReportApiService::fetchAll()` holt je Endpunkt Seiten à 500 Zeilen (`limit / offset`), bis eine Seite kürzer ist.
- **Cache:** Das Ergebnis liegt eine Stunde in Redis unter `report:<md5(endpoint+parameter)>` . Der Button „**Daten neu laden**“ (`/report/reload`) löscht alle `report:*` -Keys.
- **Install-Attributionen** werden immer ab `2020-01-01` bis zum Enddatum geladen. Die Revenue-Zuordnung braucht den ganzen Bestand, weil ein Shop vor dem gewählten Zeitraum installiert und erst darin upgraden kann. Auf den Zeitraum eingegrenzt wird erst danach im Intranet, nach `installDate` (fehlt es, nach `createdAt`).
- **Plan-Events** werden für den gewählten Zeitraum geladen, optional mit `event_type` .
- **Enddatum:** Die API liest ein reines Datum als `00:00:00` . Das Intranet hängt deshalb `23:59:59` an, sonst fehlen die Einträge des letzten Tages.

Feldnamen

Die API liefert **camelCase**, nicht die DB-Spaltennamen:

API-Feld	DB-Spalte
utmSource , utmMedium , utmCampaign , ...	utm_source , utm_medium , utm_campaign , ...
installUid	install_uid
shopId	shop_id
referrerDomain	referrer_domain
createdAt , installDate	created_at , install_date
eventType , oldPlan , newPlan , price	event_type , old_plan , new_plan , price
attributionInstallUid	attribution_install_uid

Was die Seite zeigt

/report (Attribution Report)

- Filter: Zeitraum (from , to , Standard die letzten 30 Tage), Event-Typ, **Vorjahresvergleich** (compare=1 , derselbe Zeitraum ein Jahr früher).
- Kennzahlen: Attributionen im Zeitraum (davon verknüpft), Plan-Events, Verteilung nach UTM-Quelle, -Medium und -Kampagne, Event-Typen, Events pro Monat, Umsatz und MRR je Quelle.
- **Drill-down:** Ein Klick auf eine Quelle oder ein Medium filtert die Attributionstabelle (source= , medium=); leere Werte zählen als unknown . linked=1 zeigt nur verknüpfte Attributionen.
- Tabs: **Attributions**, **Events** (Plan-Events), **Journeys** (Plan-Verlauf je Shop: stabil, gekündigt, Planwechsler, nur Trial, Reaktivierungen), **Revenue** (Umsatz je Shop mit Quelle).
- **PDF-Export** (/report/pdf) mit denselben Filtern, A4 quer.

/report/revenue (Revenue Report): MRR pro Monat, Umsatz je Plan, verlorener Umsatz durch Kündigungen und Downgrades, ebenfalls mit Vorjahresvergleich.

Wann gilt eine Attribution als „verknüpft“?

Eine Attribution zählt als einem Shop zugeordnet, wenn **shopId gesetzt ist** (Abschnitt 2, link()) **oder** ihre **installUid** von einem Plan-Event als **attributionInstallUid** referenziert wird. Nur verknüpfte Attributionen gehen in die Umsatzzuordnung ein. Alle anderen sind Klicks ohne (zuordenbare) Installation, siehe offene Punkte 2 und 5.

Umsatz-Zuordnung (vereinfacht)

Pro Shop werden die Plan-Events chronologisch sortiert. Die Quelle kommt aus der Attribution des ersten Events mit `attributionInstallUid`. Ab dem ersten Upgrade zählt der Preis als Umsatz der Quelle; ist der Shop am Ende auf einem bezahlten Plan, geht dessen Preis in den **MRR** der Quelle ein. Shops ohne Attribution laufen unter `unknown`.

Datenmodell

Migration: `migrations/Version20260310120000.php`

install_attribution

Spalte	Typ	Bedeutung
<code>id</code>	INT PK	
<code>shop_id</code>	INT, NULL	FK auf <code>Shop.id</code> , UNIQUE , ON DELETE CASCADE. Leer, bis <code>link()</code> läuft.
<code>install_uid</code>	CHAR(36)	UUID, UNIQUE
<code>utm_source</code> ... <code>utm_term</code>	VARCHAR(255)	UTM-Parameter
<code>landing_url</code>	VARCHAR(700)	Einstiegs-URL
<code>referrer</code>	VARCHAR(700)	HTTP-Referrer
<code>referrer_domain</code>	VARCHAR(500)	Host ohne <code>www.</code>
<code>created_at</code>	DATETIME	Zeitpunkt der Erfassung (Klick)
<code>install_date</code>	DATETIME, NULL	Zeitpunkt der Verknüpfung (Installation)

shop_plan_event

Spalte	Typ	Bedeutung
shop_id	INT	FK auf Shop.id, ON DELETE CASCADE
event_type	ENUM	upgrade, downgrade, trial_start, trial_end oder cancel
old_plan / new_plan	VARCHAR(50)	
price, currency, billing_interval		
attribution_install_uid	CHAR(36), NULL	FK auf install_attribution.install_uid, ON DELETE SET NULL
created_at	DATETIME	

Beispiel-Abfragen

Installationen je Kampagne (nur verknüpfte):

```
SELECT utm_source, utm_medium, utm_campaign, COUNT(*) AS installs
FROM install_attribution
WHERE shop_id IS NOT NULL
GROUP BY utm_source, utm_medium, utm_campaign
ORDER BY installs DESC;
```

Conversion-Rate von Klick zu Installation je Quelle:

```
SELECT COALESCE(utm_source, referrer_domain, '(direct)') AS quelle,
       COUNT(*) AS klicks,
       SUM(shop_id IS NOT NULL) AS installs,
       ROUND(100 * SUM(shop_id IS NOT NULL) / COUNT(*), 1) AS rate_pct
FROM install_attribution
GROUP BY quelle
ORDER BY klicks DESC;
```

Upgrades je Kampagne:

```
SELECT ia.utm_campaign, spe.new_plan, COUNT(*) AS upgrades
FROM shop_plan_event spe
JOIN install_attribution ia ON ia.install_uid = spe.attribution_install_uid
WHERE spe.event_type = 'upgrade'
GROUP BY ia.utm_campaign, spe.new_plan;
```

Auffälligkeiten und offene Punkte

- 1 Re-Install mit neuer Attribution kann scheitern.** `shop_id` ist UNIQUE. Hat ein Shop schon eine verknüpfte Attribution und wird nach einer Deinstallation über einen neuen Kampagnen-Link wieder installiert, versucht `link()`, eine zweite Zeile mit derselben `shop_id` zu setzen. Das verletzt `uniq_shop_id`. Die Exception landet im `try/catch` von `oauthCallback()`, und der Callback antwortet mit **502**. Mögliche Lösung: die alte Verknüpfung vorher lösen oder die bestehende Zeile überschreiben.
- 2 Klicks ohne Installation bleiben liegen.** Jeder Aufruf von `/install` legt eine Zeile an, auch wenn danach keine Installation folgt. Das ist für die Conversion-Rate gewollt. Eine Aufräum-Strategie gibt es derzeit nicht.
- 3 `debug_mode=true`** wird bei jedem GA4-Event gesendet. Die Events erscheinen dadurch in der GA4-DebugView. Ob das in Produktion so gewollt ist, sollte geprüft werden.
- 4 Keine Verbindung zur GA-Web-Session.** Die `client_id` ist `sha1(shopId)` und nicht die GA-Client-ID aus dem Browser. Die UTM-Zuordnung in GA4 läuft deshalb ausschließlich über die Custom-Parameter `install_*`. Diese müssen in GA4 als benutzerdefinierte Dimensionen angelegt sein.
- 5 Ohne Cookie keine Zuordnung.** Fehlt die `install_uid` beim Callback (Cookie blockiert, anderer Browser, Installation direkt aus dem App Store), bleibt der Shop ohne Attribution. Seine Plan-Events haben dann `attribution_install_uid = NULL`.
- 6** `trial_start` und `trial_end` existieren im Schema, werden aber nicht befüllt. `billing_interval` ist immer `monthly`.