

UTM tracking in api.langify

How langify captures where an installation came from (UTM parameters and referrer), links it to a shop and evaluates it on later plan changes.

As of 2026-10-05 · branch `dev` of `api.langify`

Overview

Tracking runs in four steps inside the API and is evaluated in the intranet (step 5). The key that ties everything together is the `install_uid` (a UUID v4). It is created on first contact, stored in the browser as a cookie and sent back to the API during the OAuth callback.

1

Capture – v1 /install

A visitor clicks a campaign link. UTM + referrer are stored in `install_attribution`; the API returns an `install_uid`, which is set as a cookie.

`POST /auth/remote/install-attribution`

2

Link – OAuth callback

After installation from the App Store, the `install_uid` from the cookie is assigned to the shop (new installs and re-installs only).

`POST /auth/remote/oauth/callback`

3

Evaluate – plan changes

`shop_plan_event` rows and GA4 events are written together with the install's UTM data.

`ApplicationChargeRepository` → `PlanEventTracker`, `AnalyticsService`

4

Reporting API

Install attributions and plan events, available via JWT with scope `report`.

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

5

Intranet – intern.lovely-app.com

KPIs, revenue/MRR per source, plan journeys, PDF export.

`/report` , `/report/revenue`

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

Files

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

The v1 page `/install` (the entry page to the App Store; it is not part of this repo) calls this endpoint server-side. The API neither sets a cookie nor redirects; the caller handles both.

Authentication: JWT with scope `remote-auth` , either as `Authorization: Bearer <token>` or as `?token=` (see `ApiTokenAuthTrait::validateApiToken()`).

Request body (JSON), all fields optional

Field	Target column	Max. length
<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>referrer</code>	<code>referrer</code>	700

Note: the request field is called `referrer` (HTTP spelling), the DB column `referrer` .

Processing

- Every value is trimmed and truncated to the column length (`mb_substr`); an empty string becomes `NULL` .

- `referrer_domain` is derived from the referrer: the host without a leading `www.` (e.g. `https://www.google.com/search?... → google.com`).
- The API generates the `install_uid` itself (`random_bytes` , RFC 4122 v4).
- `created_at` is the capture time. `shop_id` and `install_date` remain empty for now.

Response

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

The caller stores the `install_uid` as a cookie and redirects to the Shopify App Store.

Example

```
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": "autumn-2026",
    "landing_url": "https://langify-app.com/install?
utm_source=newsletter&utm_medium=email&utm_campaign=autumn-2026",
    "referrer": "https://www.google.com/"
}'
```

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

Files

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

After installation from the App Store, the Shopify OAuth flow runs. The v1 callback forwards the Shopify query parameters to the API and adds the `install_uid` from the cookie:

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

Flow

1. The API validates the JWT (scope `remote-auth`) and the Shopify HMAC.
2. It loads the shop or creates it.
3. A **new installation** (`new_install = true`) is a shop that does not yet exist locally, or a shop with `uninstalled_at` set (re-install).
4. Only for a new installation with an `install_uid` present does the API call `link()` : it looks up the row with that `install_uid` in `install_attribution` and sets `shop_id` and `install_date = now()` . If no matching row exists, nothing happens (returns `false` , no error).

Logins into an already installed shop therefore never change the attribution.

3 Evaluate: plan changes

Files

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

On every plan change the API loads the shop's attribution (`InstallAttributionRepository::findByShop()`) and writes to two places:

3a. Database: `shop_plan_event`

`PlanEventTracker::recordPlanChange()` inserts a row with:

- `old_plan` , `new_plan` , `price` , `currency` (currently always `USD`) and `billing_interval` (currently always `monthly`)
- `event_type` : `upgrade` , `downgrade` or `cancel` . Upgrade vs. downgrade is determined by comparing plan IDs (`ApplicationCharge::PLAN_IDS`). Switching to the free plan is `cancel` .
- `attribution_install_uid` : the `install_uid` of the shop's attribution, if any

The values `trial_start` and `trial_end` are defined in the enum but currently never written.

3b. Google Analytics 4 (Measurement Protocol)

`AnalyticsService::trackPlanChange()` sends two events to `https://www.google-analytics.com/mp/collect` :

Event	Purpose
<code>plan_change</code>	Main event for analysis
<code>plan_<targetPlan></code>	One per target plan, for simple conversion reports

Parameters

- `source_plan`, `target_plan`, `plan_change` (`<old>_to_<new>`) and `value = 1`
- if an attribution exists, the UTM values prefixed with `install_`: `install_source`, `install_medium`, `install_campaign`, `install_content` and `install_term`. `NULL` values are omitted.

Details

- `client_id = sha1(shopify_shop_id)`. Events are therefore tied to a stable, pseudonymous ID per shop, not to the visitor's browser session.
- Configured via `GOOGLE_ANALYTICS_MEASUREMENT_ID` and `GOOGLE_ANALYTICS_API_SECRET`. If either is empty, nothing is sent.
- The API currently always sends `debug_mode=true`.
- For testing there is `src/Controller/Debug/GoogleTestController.php`.

When it fires (`ApplicationChargeRepository`)

Method	Situation
<code>changePlanGetConfirmationUrl()</code>	Direct plan change without merchant confirmation
<code>checkCharge()</code> , active charge	The merchant confirmed the charge
<code>checkCharge()</code> , declined, expired or notfound	Charge declined or expired; the API records a switch to Free (<code>cancel</code>)

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

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

Authentication: JWT with scope `report`

Endpoint	Content	Filters
GET /install-attributions	Rows from <code>install_attribution</code>	<code>from</code> , <code>to</code> (on <code>created_at</code>), <code>limit</code> (max. 500, default 100), <code>offset</code>
GET /plan-events	Rows from <code>shop_plan_event</code>	as above, plus <code>event_type</code>

Both endpoints sort by `created_at DESC`.

5 Reading it in the intranet: `intern.lovely-app.com/report`

The intranet (repo `intern.lovely-app`, PHP/Phalcon) reads the two report endpoints from section 4 and evaluates them. It has no database or copy of its own: every report is fetched live from the API and cached in Redis for one hour.

Files (repo `intern.lovely-app`)

- `app/engines/ReportApiService.php` : API client (Guzzle), pagination, Redis cache
- `app/controllers/ReportController.php` : evaluation (KPIs, linking, revenue, plan journeys), PDF
- `app/views/report/index.phtml`, `tab.phtml`, `revenue.phtml` : UI (tabs are loaded via htmx)

Access

Page	URL	Role
Attribution Report	<code>/report</code>	<code>reporting</code> or <code>admin</code>
Revenue Report	<code>/report/revenue</code>	<code>reporting</code> or <code>admin</code>

Both are in the menu under **Reports** (alongside “Plan Overview” and “Plan Flow”, which do not use UTM data).

Configuration (intranet environment variables, CapRover or `.env.local`)

Variable	Meaning
<code>LANGIFY_REPORT_API_URL</code>	API base URL, live <code>https://api.langify-app.com</code>
<code>LANGIFY_REPORT_API_TOKEN</code>	JWT with scope <code>report</code> , sent as <code>Authorization: Bearer</code>
<code>LANGIFY_REPORT_RESOLVE</code>	optional, local only: a <code>CURLOPT_RESOLVE</code> entry (<code>host:port:ip</code>), e.g. to point <code>api.langify.dvlbox.de</code> at a Docker IP

- If URL or token is missing, the page shows “API not configured”.
- If the API rejects the token (HTTP 401/403), the page shows this along with the HTTP code.
- **Token expiry:** the intranet reads `exp` from the JWT and warns **14 days** before expiry at the top of the page. Then create a new token with scope `report` and replace `LANGIFY_REPORT_API_TOKEN` .

Data retrieval

- **Pagination:** `ReportApiService::fetchAll()` fetches pages of 500 rows per endpoint (`limit / offset`) until a page is shorter.
- **Cache:** results are kept in Redis for one hour under `report:<md5(endpoint+parameters)>` . The **“Reload data”** button (`/report/reload`) deletes all `report:*` keys.
- **Install attributions** are always loaded from `2020-01-01` up to the end date. Revenue attribution needs the full history, because a shop may have installed before the selected period and upgraded within it. Filtering to the period happens afterwards in the intranet, by `installDate` (or `createdAt` if missing).
- **Plan events** are loaded for the selected period, optionally filtered by `event_type` .
- **End date:** the API reads a plain date as `00:00:00` . The intranet therefore appends `23:59:59` ; otherwise entries from the last day would be missing.

Field names

The API returns **camelCase**, not the DB column names:

API field	DB column
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

What the pages show

/report (Attribution Report)

- Filters: period (from , to , default last 30 days), event type, **year-over-year comparison** (compare=1 , same period one year earlier).
- KPIs: attributions in the period (of which linked), plan events, breakdown by UTM source, medium and campaign, event types, events per month, revenue and MRR per source.
- **Drill-down:** clicking a source or medium filters the attribution table (source= , medium=); empty values count as unknown . linked=1 shows linked attributions only.
- Tabs: **Attributions**, **Events** (plan events), **Journeys** (plan history per shop: stable, cancelled, plan switchers, trial only, reactivations), **Revenue** (revenue per shop with source).
- **PDF export** (/report/pdf) with the same filters, A4 landscape.

/report/revenue (Revenue Report): MRR per month, revenue per plan, revenue lost to cancellations and downgrades, also with year-over-year comparison.

When is an attribution “linked”?

An attribution counts as assigned to a shop if **shopId is set** (section 2, link()) **or** its installUid is referenced by a plan event as attributionInstallUid . Only linked attributions feed into revenue attribution. All others are clicks without an (attributable) installation, see open issues 2 and 5.

Revenue attribution (simplified)

Per shop, plan events are sorted chronologically. The source comes from the attribution of the first event with an attributionInstallUid . From the first upgrade on, the price counts as

revenue for that source; if the shop ends up on a paid plan, its price contributes to the source's **MRR**. Shops without attribution are grouped under `unknown` .

Data model

Migration: `migrations/Version20260310120000.php`

install_attribution

Column	Type	Meaning
<code>id</code>	INT PK	
<code>shop_id</code>	INT, NULL	FK to <code>Shop.id</code> , UNIQUE , ON DELETE CASCADE . Empty until <code>link()</code> runs.
<code>install_uid</code>	CHAR(36)	UUID, UNIQUE
<code>utm_source</code> ... <code>utm_term</code>	VARCHAR(255)	UTM parameters
<code>landing_url</code>	VARCHAR(700)	Landing URL
<code>referrer</code>	VARCHAR(700)	HTTP referrer
<code>referrer_domain</code>	VARCHAR(500)	Host without <code>www.</code>
<code>created_at</code>	DATETIME	Capture time (click)
<code>install_date</code>	DATETIME, NULL	Link time (installation)

shop_plan_event

Column	Type	Meaning
shop_id	INT	FK to Shop.id, ON DELETE CASCADE
event_type	ENUM	upgrade, downgrade, trial_start, trial_end or cancel
old_plan / new_plan	VARCHAR(50)	
price, currency, billing_interval		
attribution_install_uid	CHAR(36), NULL	FK to install_attribution.install_uid, ON DELETE SET NULL
created_at	DATETIME	

Example queries

Installs per campaign (linked only):

```
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;
```

Click-to-install conversion rate per source:

```
SELECT COALESCE(utm_source, referrer_domain, '(direct)') AS source,
       COUNT(*) AS clicks,
       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 source
ORDER BY clicks DESC;
```

Upgrades per campaign:

```
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;
```

Observations and open issues

- 1 Re-install with a new attribution can fail.** `shop_id` is UNIQUE. If a shop already has a linked attribution and is re-installed via a new campaign link after uninstalling, `link()` tries to set the same `shop_id` on a second row. This violates `uniq_shop_id`. The exception is caught by the `try/catch` in `oauthCallback()` and the callback responds with **502**. Possible fix: release the old link first or overwrite the existing row.
- 2 Clicks without installation accumulate.** Every call to `/install` creates a row, even if no installation follows. This is intended for the conversion rate. There is currently no cleanup strategy.
- 3 `debug_mode=true`** is sent with every GA4 event, so events show up in the GA4 DebugView. Whether this is intended in production should be checked.
- 4 No connection to the GA web session.** The `client_id` is `sha1(shopId)`, not the GA client ID from the browser. UTM attribution in GA4 therefore relies solely on the custom parameters `install_*`, which must be registered as custom dimensions in GA4.
- 5 No cookie, no attribution.** If the `install_uid` is missing at callback time (cookie blocked, different browser, install directly from the App Store), the shop stays without attribution and its plan events have `attribution_install_uid = NULL`.
- 6** `trial_start` and `trial_end` exist in the schema but are never populated. `billing_interval` is always `monthly`.