# Developer Handbook

# Table of contents:

# Documentation

Learn the basics of using epilot developer tools and APIs to unlock the potential of the epilot 360 platform.

## Arhitecture

Understand the basics of the epilot platform and learn core concepts like entities, authentication and automation.

Overview - Tech Stack

## APIs

Explore and learn to use our APIs to build your own solutions on epilot.

Overview - Docs

## SDK

Explore our developer tools and get started with the epilot SDK

Overview

# Introduction to epilot

Epilot is a multi-tenant SaaS platform for complex ecommerce.

Our tenants use epilot to sell complex products online and collaborate with partners to deliver great ecommerce experiences to their end customers.

## Overview

The epilot application consists of:

- The main 360 portal app
- Embeddable customer-facing journey frontends
- End customer portal app
- Serverless microservices exposed via APIs
- Internal admin area

## Tech Stack

The portal frontend is a single-page web application (SPA) consisting of multiple frontend microservices running on single-spa.

The majority of frontend microservices are written in React + Typescript.

The epilot application backend consists of serverless microservices behind APIs.

The majority of backend microservices are written in Typescript using serverless AWS services such as Lambda, Step Functions, API Gateway, S3, DynamoDB, EventBridge.

We still support some customers using the legacy version of epilot built on Axon Ivy. Ivy is a monolith Java application running on AWS EC2.

@epilot360/ivy-embed

@epilot360/login

OAuth 2.0


API Gateway


Authorizer


Cognito

Backend Microservices


Lambda


DynamoDB


EventBridge

document-api

user-api

journey-configuration-api

notifications-api

legacy-search-api

files-api

workflow-definitions-api

workflow-executions-api

automation-api

product-api

order-api

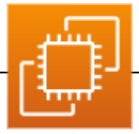
order-api

entity-api

permissions-api

Ivy (legacy)


RDS Aurora
(PostgreSQL)


EC2

epilot-database-hasura

headless-ivy

Java

Ivy Monolith

# API First

As one of our core engineering principles, epilot software is built API first, meaning the design of the software is done by creating concrete API contracts before implementation.

This is done using common, machine readable standards such as:

- OpenAPI specification
- GraphQL schemas
- TypeScript type definitions

Click here to view our API documentation.

# SDK

We provide a Javascript / Typescript SDK to interact with epilot APIs.

## SDK Packages

> TODO: packages not yet public

- @epilot/entity-client
- @epilot/user-client
- @epilot/organization-client
- @epilot/submission-client
- @epilot/pricing-client
- @epilot/file-client
- @epilot/document-client
- @epilot/automation-client
- @epilot/message-client
- @epilot/notification-client
- @epilot/template-variables-client
- @epilot/workflows-definition-client
- @epilot/workflows-execution-client
- @epilot/design-builder-api-client
- @epilot/customer-portal-client

# Serverless

The epilot backend is built using serverless technology. This means we favour the usage of managed 3rd party services to build our features.

We chose this approach to speed up our development by leveraging off-the-shelf components to build our software and avoid having to maintain and operate our own cloud infrastructure.

The majority of backend microservices are built using serverless AWS services such as Lambda, Step Functions, API Gateway, S3, DynamoDB and EventBridge.

## Boilerplate Projects

We offer internal boilerplate (cookie-cutter) projects to bootstrap serverless backend projects on popular frameworks and get started quickly:

- AWS SAM
- AWS CDK
- SLS Framework
- Fargate

## Event Driven Architecture

TODO

# Component Library

[SDK] [Storybook]

We provide a shared React component library, based on Material UI for shared frontend UI elements.

The component library is distributed as a set of npm packages:

- @epilot/base-elements
- @epilot/base-modules

The storybook documentation for the component library can be found under the following links:

- https://base-elements.dev.epilot.io/
- https://base-modules.dev.epilot.io/

## Usage

```
import { Button, ThemeProvider } from '@epilot/base-elements'

function App() {
  return (
    <ThemeProvider>
      <Button>My Button</Button>
    </ThemeProvider>
  )
}
```

# Authentication

The epilot application uses standard OAuth 2.0 for user authentication.

## Quick Start

To obtain OAuth tokens, the user should initiate authentication using their user pool details.

```
# TODO: Provide an example for this using the epilot SDK
```

## Cognito User Pools

Each tenant organisation in epilot has their own Cognito user pool backend and OAuth configuration to provide login and 3rd party Single Sign-On.

## User API

The epilot user API provides functionality to invite and manage users in epilot organisations.

The Cognito sync service part of the User API takes care of managing users in each User Pool.

# Authorization

The epilot application uses standard OAuth 2.0 JWT tokens for authorization.

## API Gateway Authorizer

Requests to non-public epilot APIs are authorized by a custom Lambda authorizer that verifies the passed JWT Token and parses the user's claims contained in the token.

The claims are passed to the API service as context:

```
// Example ID token:
{
  "token_use": "id",
  "sub": "0cd63e9c-42b4-4a38-97b8-1e41e42677e3",
  "cognito:username": "v.kuosmanen@epilot.cloud",
  "email": "v.kuosmanen@epilot.cloud",
  "custom:ivy_org_id": "66",
  "custom:ivy_user_id": "29216",
  "email_verified": true,
  "iss": "https://cognito-idp.eu-central-1.amazonaws.com/eu-central-1_hhz2uIClH",
  "aud": "gj9p0jreihtq00cri6a0fe306",
  "event_id": "cf3df1cd-2aac-433c-8576-d2834c579ebb",
  "auth_time": 1641386601,
  "exp": 1642357470,
  "iat": 1642353870,
}
```

## Permissions API

While the JWT token contains basic information about the identity of the authorized user such as user id and source organization, to check that the user is allowed to perform actions and access resources, we need to check the Permissions API for claims

Example:

```
import { tokenIsPermitted } from '@epilot/permissions';

const isPermitted = await tokenIsPermitted(context.token, 'myaction');
```

Permissions Documentation

## Internal Auth

Sometimes backend microservices need to make internal calls as no specific user.

For this purpose we use a special internal auth service as identity provider, which translates the caller's IAM role to a JWT token accepted by the API Gateway Authorizer.

See documentation for the internal auth service for details.

# Links

- API Gateway Authorizer project: https://gitlab.com/e-pilot/product/auth/custom-authorizer
- Permissions package: https://www.npmjs.com/package/@epilot/permissions
- Internal Auth package: https://www.npmjs.com/package/@epilot/internal-auth

# Permissions

[API Docs] [SDK]

Epilot implements flexible role-based access control using influenced by the design of AWS IAM policies.

The epilot Permissions system consists of these basic ideas:

- **Users** may be assigned **Roles**
- **Roles** are collections of **Grants**
- **Grants** are used to evaluate whether the user has permissions to perform actions and access resources

## Usage

To use the epilot Permissions API, we provide a package that implements both fetching the user's roles and grants as well as evaluating them.

Example:

```
import { tokenIsPermitted } from '@epilot/permissions';

const isPermitted = await tokenIsPermitted(token, 'myaction');
```

Readme link: @epilot/permissions

## Example Role

```
// Example manager role in org 66
{
  "id":"66:manager",
  "name":"Manager",
  "slug":"manager",
  "organization_id":"66",
  "type":"user_role",
  "grants":[
    {
      "action":"entity:*",
      "effect": "allow"
    },
    {
      "action":"users:*",
      "effect": "allow"
    },
    {
      "action":"partners:*",
      "effect": "allow"
```

```
    },
    {
      "action":"legacy_products:*",
      "effect": "allow"
    }
  ]
```

# Grants Evaluation Logic

A lot of this will be familiar to AWS IAM users. This is “heavily inspired” by their design

- **Rule 1**: Tenants are isolated into Organizations. Roles may only grant access to resources within the tenant Organization.
- **Rule 2**: The owner role inherits all grants from the organization. (hardcoded role, present in all orgs)
- **Rule 3**: By default roles have no grants until they are defined. (Principle of least privilege)
- **Rule 4**: When evaluating all role and organization grants are added to the grant pool.
- **Rule 5**: At least one organization role grant and at least one user role grant must be matched to pass the evaluation. Neither must contain a matched explicit deny.
- **Rule 6**: An evaluation will try to match all available grants where a given action and resource matches. Wildcard expressions are supported.

# Organization Root Role

In addition to user roles, each user in the organization also has a mandatory root role when acting in the organization.

The organization root role defines the maximum set of grants any user in that organization may receive.

This role is only accessible to epilot admins and is controlled by the pricing tier of that organization.

# Links

- Permissions package: https://www.npmjs.com/package/@epilot/permissions

# Internal Auth

[API Docs] [SDK]

To facilitate backend microservices calling each other, we provide an internal identity provider called Internal Auth API

The API works by converting the caller's IAM role to a valid JWT token accepted by the API Gateway Authorizer.

API calls must be called using a signed AWS SigV4 request.

## Usage

```
npm install --save @epilot/internal-auth
```

To be able to get internal tokens, your runtime role must have invoke permissions to this API.

```
# SAM example
Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Policies:
      - Statement:
        - Effect: Allow
          Action: execute-api:Invoke
          Resource:
            - arn:aws:execute-api:eu-central-1:*:*/*/GET/v1/internal-auth/auth # 
internal auth api
            - arn:aws:execute-api:eu-central-1:*:*/*/GET/v?/mock-api/* # api you want 
to call
```

Call the API to obtain your token

```
import { getToken } from '@epilot/internal-auth'

const token = await getToken()
```

## Example Token

```
{
  "callerIdentity": "arn:aws:sts::912468240823:assumed-
role/ep_prod_access_admin/awsmfa_20210225T193753",
  "policies": [
    {
```

```
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Action": "*",
          "Resource": "*"
        }
      ]
    }
  ],
  "iss": "https://internal-auth.sls.epilot.io/v1/internal-auth",
  "iat": 1614278397,
  "exp": 1614281997
}
```


# Links

- Internal Auth package: https://www.npmjs.com/package/@epilot/internal-auth

# Microfrontends

The 360 portal application consists of multiple frontend microservices, governed by single-spa.

## @epilot360/root-config

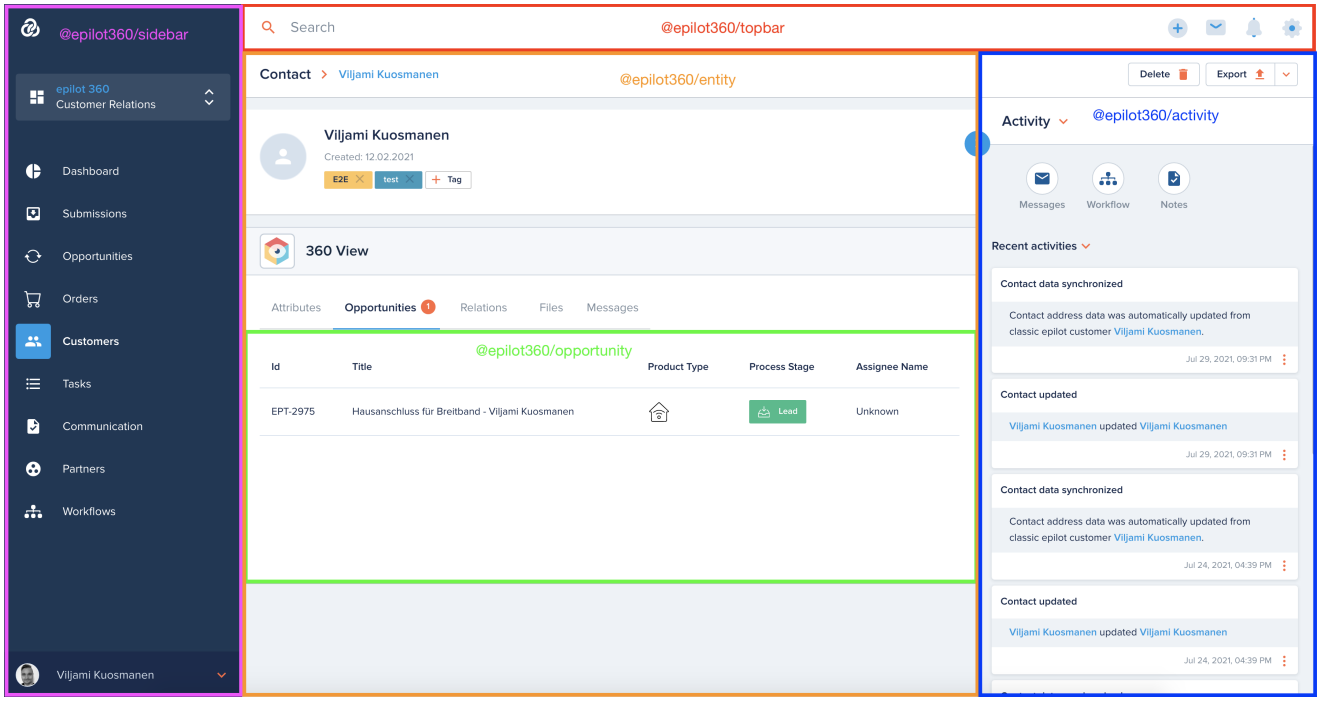
This is the root project defining the microfrontend layout and import maps containing references to the correct bundle for each microfrontend application.

The project also contains the centralised localisation files.

## Layout

The basic layout consists of the following parts:

- @epilot360/login
  - Login overlay
  - Manages the user login state of the entire application
- @epilot360/topbar
  - Top bar menus
  - Global search
  - Notifications
- @epilot360/sidebar
  - Main left sidebar navigation



# Links

- Full list of 360 microfrontends: https://gitlab.com/e-pilot/product/360-portal/epilot360-root-config#microfrontends

# @epilot360/i18n

```
yarn add @epilot360/i18n
```

Translation and localisation library for the epilot360 portal.

Uses i18next.

## Usage

```
// Component.jsx
import { useTranslation } from '@epilot360/i18n'

export const Component = () => {
  const { t } = useTranslation('my-namespace')

  return <h1>{t('hello_world_header', 'Hello World!')}</h1>
}
```

Translations are loaded asynchronously, so make sure to wrap your app inside `<React.Suspense>`.

## Adding Translations

Translations are defined in static JSON locale files in epilot360-root-config.

The easiest way to translate epilot 360 is to run the root-config project locally to see the changes immediately.

While running locally, missing translations will be automatically added to the locale files under `locales/{locale}/{namespace}.json`.

# @epilot360/auth-service

```
yarn add @epilot360/auth-service
```

Access current user's auth information in epilot 360 portal.

## Usage

```
// TODO
```

# @epilot360/feature-flags-service

```
yarn add @epilot360/feature-flag-service
```

Use feature flags in epilot 360 portal.

## Usage

```
// TODO
```

# @epilot360/snackbar-service

```
yarn add @epilot360/snackbar-service
```

Create snackbar alerts in epilot 360 portal

## Usage

```
import { SnackbarUtils } from '@epilot360/snackbar-service'

SnackbarUtils.success({
  title: 'Success',
  message: 'Contact Saved Successfully'
})

SnackbarUtils.info({
  message: 'You got new notifications',
  title: 'Info'
})

SnackbarUtils.warning({
  message: 'Memory Leak Found',
  title: 'Warning'
}

SnackbarUtils.error({
  message: 'Some error has occurred',
  title: 'Error'
})

SnackbarUtils.custom(<MyChildComponent/>)
```

# Flexible Entities

[API Docs] [SDK]

The epilot application uses a flexible *Entities* data layer to model business data on the platform.

## Entities

Entities are business objects in epilot with flexible user-defineable schemas.

Each entity can contain arbitrary JSON data, which is made accessible via the Entity API.

## Schemas

Schemas represent different types of entities in epilot.

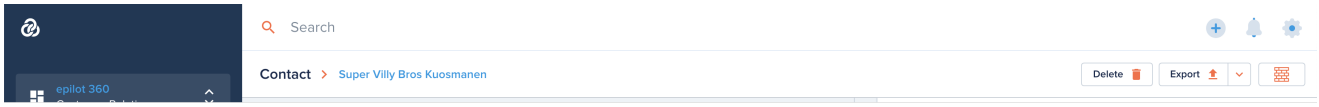
Examples of Entity Schemas:

- Contact
- Account
- Product
- Submission
- Order
- Opportunity
- File
- Message

The primary purpose of schemas is to control how the flexible entities are represented in the epilot 360 portal UI.

## Entity Builder

The epilot 360 potal provides a builder interface to modify the schemas defined in the organization.




# Attributes

Schemas define a list of Attributes, which are fields that the entity can have.

These Attributes are rendered in Tables and Details views for all entities of the Schema.

Examples of Attributes:

- First Name
- Last Name
- VAT ID
- Product Name
- Product SKU
- Order ID

# Capabilities

Entity Schemas may also define capabilities for the entities, which define extra functionality for that entity.

Examples of capabilities:

- File Attachments
- Comments

# Relations

Entities also natively support relations, meaning entities can be linked with each other.

Related entities appear in each-others detail views as previewable links.

# Activity

Any events related to the entity are stored in an append-only Activity feed.

Each activity item contains a description of the activity, the caller and details about any operations touching the data of the entity.

# Entity API

[API Docs] [SDK]

# Schema API

[API Docs] [SDK]

# Search API

[API Docs] [SDK]

# Activity API

[API Docs] [SDK]

# Journey Frontend

```
// TODO
```

# Journey Builder

```
// TODO
```

# Designs

```
// TODO
```

# Submissions

[API Docs] [SDK]

Submissions are entities that store raw data collected from Journeys or other outside data sources into epilot.

Submissions are created via the public Submission API.

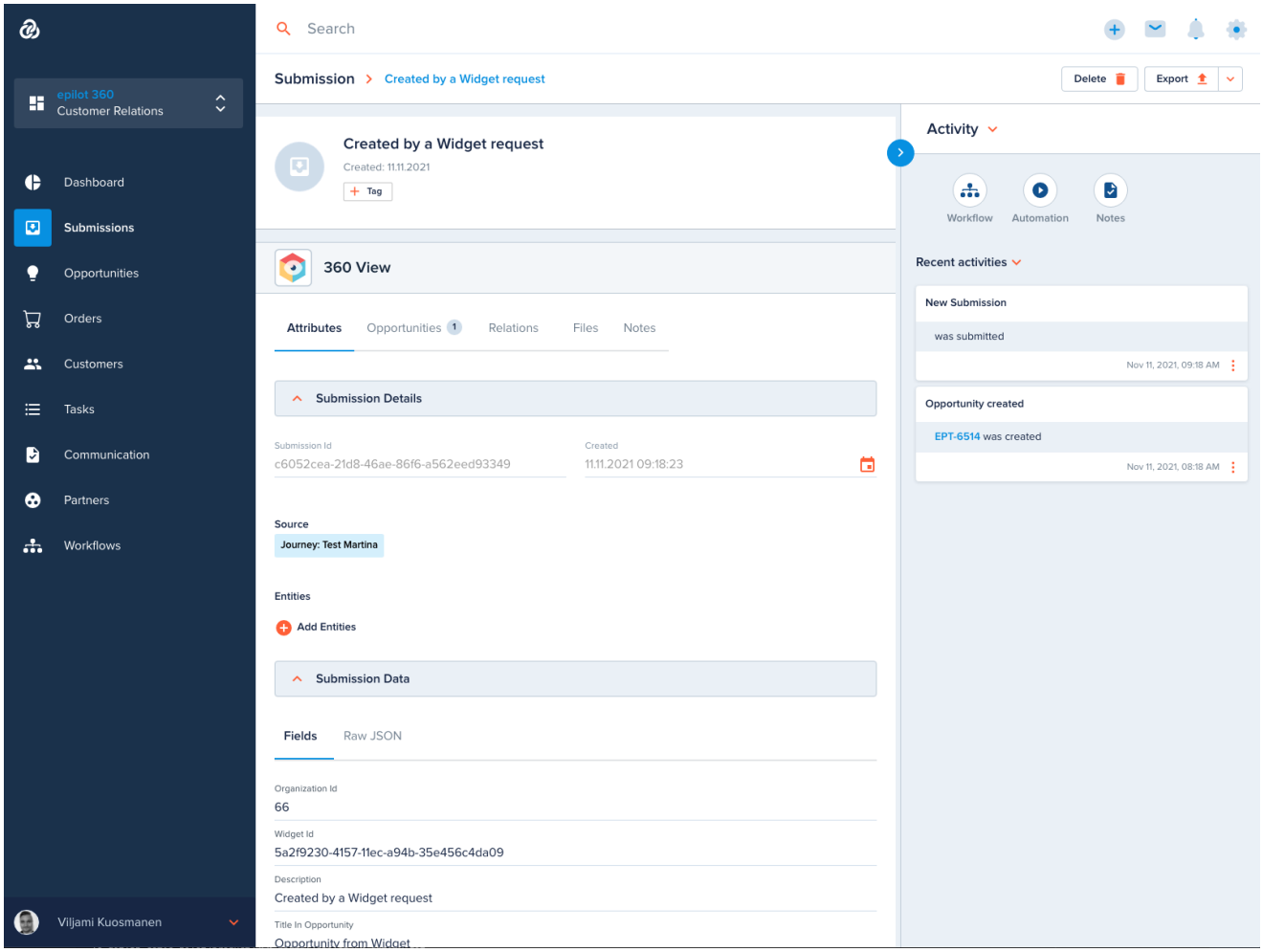
## Submission Schema

A Submission entity does not have a fixed schema for all its data like other business entities, but rather is designed to collect the raw JSON payload to be further processed in other entities.

A typical use of submissions is to map the incoming submission payload into further business entities like Contacts, Opportunities and Orders using Automation.

A standard Submission has the following two Attributes:

- **Source**
  - Links to the source of the submission
- **Entities**
  - Mapped entity relations




# Example Submission Payload

```
// example submission from a journey
{
  "steps": [
    {
      "Produktauswahl": {
        "product": {
          "selectedProductId": "a457da80-7ef1-4b4b-8373-f2baf2731317",
          "selectedPriceId": "d091655d-a241-42d7-9adc-2195b9b1de04",
          "selectionMetadata": {
            "selectedProduct": {
              "_id": "a457da80-7ef1-4b4b-8373-f2baf2731317",
              "_schema": "product",
              "name": "Walbox New",
              "code": "WN",
              "vendor": "WN",
              "labels": [
                "sale"
              ],
              "priceOptions": {
                "$relation": []
              },
```

```
            "_org": "66",
            "_updated_at": "2022-01-04T20:26:00.571Z",
            "_title": "Walbox New",
            "price_options": {
              "$relation": [
                {
                  "entity_id": "d091655d-a241-42d7-9adc-2195b9b1de04"
                }
              ]
            }
          },
          "selectedPrice": {
            "_id": "d091655d-a241-42d7-9adc-2195b9b1de04",
            "unit_amount": 100000,
            "unit_amount_currency": "EUR",
            "unit_amount_decimal": "1000",
            "sales_tax": "standard",
            "tax_behavior": "inclusive",
            "price_display_in_journeys": "show_price",
            "type": "one_time",
            "description": "test price",
            "active": true,
            "_schema": "price",
            "_org": "66",
            "_created_at": "2022-01-04T20:25:54.389Z",
            "_updated_at": "2022-01-04T20:25:54.389Z",
            "billing_period": "weekly",
            "billing_duration_unit": "months",
            "notice_time_unit": "months",
            "termination_time_unit": "months",
            "renewal_duration_unit": "months",
            "_title": "test price"
          }
        },
        "productName": "Walbox New",
        "prices": [
          {
            "price": 1000,
            "priceCurrency": "€",
            "title": "Einmalig",
            "interval": "one_time"
          }
        ],
        "productFeatures": [],
        "collapsedDetails": true,
        "id": "a457da80-7ef1-4b4b-8373-f2baf2731317||d091655d-a241-42d7-9adc-
2195b9b1de04"
      },
      "quantity": 1
    }
  },
  {
    "Persönliche Informationen": {
      "customerType": "Private",
      "salutation": "Herr",
      "title": "Dr.",
```

```
        "firstName": "Viljami",
        "lastName": "Kuosmanen",
        "email": "v.kuosmanen@epilot.cloud",
        "telephone": "0101010101"
      },
      "Adresse": {
        "countryCode": "de",
        "zipCode": "50668",
        "city": "Köln",
        "streetName": "Example Str.",
        "houseNumber": "11"
      }
    },
    {
      "Zusätzliche Dokumente": [
        {
          "file_name": "cool-cat.jpeg",
          "file_size": 80430,
          "original_name": "cool-cat.jpeg",
          "file_type": "jpeg",
          "s3ref": {
            "bucket": "epilot-prod-user-content",
            "key": "66/temp/2ec73b47-fe8b-4cc2-a0f4-81d2fb549f38/cool-cat.jpeg"
          }
        }
      ],
      "Zahlungsmethoden": {
        "type": "BankTransfer"
      }
    },
    {
      "Einwilligungen & Bestellung": {
        "first_consent": true,
        "second_consent": true
      }
    },
    {}
  ],
  "order_number": "ORN5513gA",
  "order_id": "4e357016-83b0-43a8-a64e-c47cf8367d34",
  "source_type": "journey",
  "source_id": "d32ee890-73be-11ec-ab85-6950938ffb7d"
}
```

# Message API

[API Docs] [SDK]

The Message API is the central email service for epilot.

# Email Templates

```
// TODO
```

# Template Variables

[API Docs] [SDK]

The Template Variables API provides variable discovery and substitution email and document templates using Handlebars.

## Template Variables API

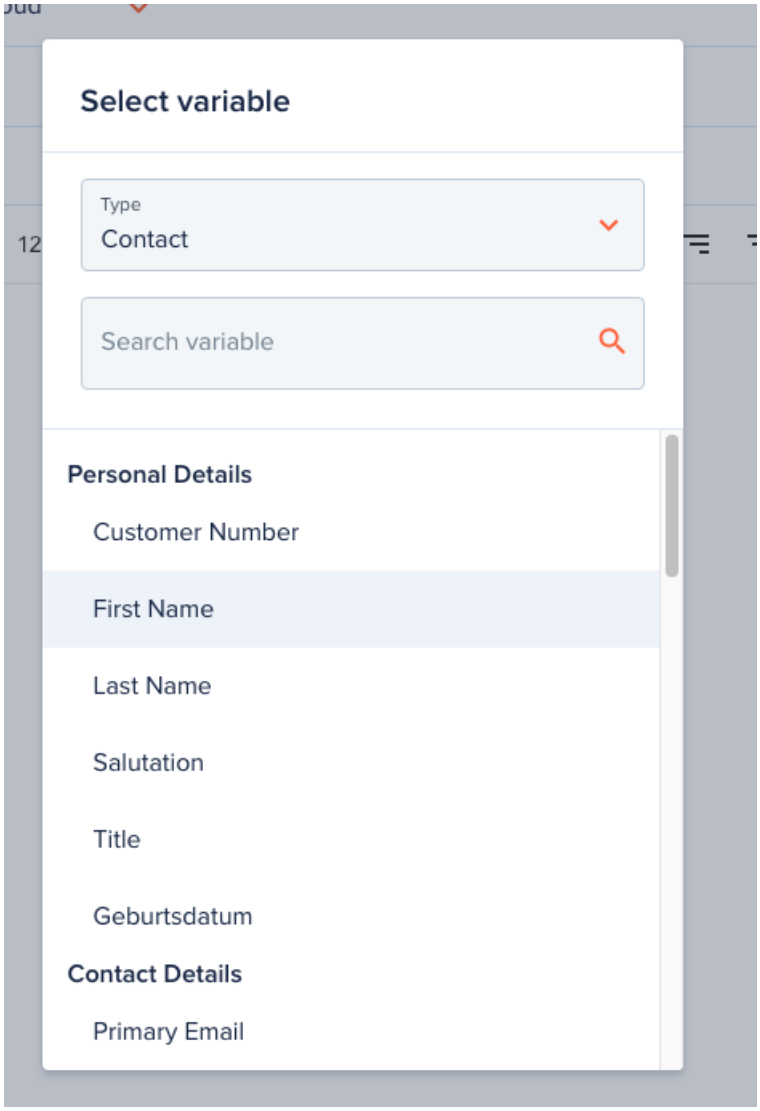
This API is called to both discover available variables as well as execute the variable substitution using handlebars.

Each time an email or document template is used, the Template Variable API is called with the appropriate standardised parameters.

The Template Variable API uses the Entity API and others to fetch the correct values for each variable when compiling the template.

## Variable Picker

We provide a picker UI for users to search and explore available variables.


Select variable
Type
Contact
Search variable
Personal Details
Customer Number
First Name
Last Name
Salutation
Title
Geburtsdatum
Contact Details
Primary Email

# Consent API

[API Docs] [SDK]

The Consent API stores consent events such as marketing opt-ins in epilot.

## Consent Events

```
// TODO
```

# File Entity

[API Docs] [SDK]

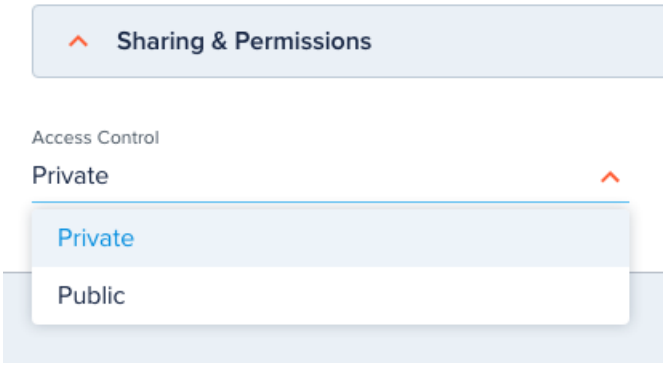
All files uploaded to epilot are represented as File Entities and available via the Entity API.

## Access Control

The access control of the file is defined via the File entity.

The `access_control` attribute of a file can be set as either `private` (default) or `public-read`.

Whenever a File entity is created, updated or deleted, the underlying file's access control S3 is updated accordingly.




## File Relations

When a file is attached to an entity, a new File Entity is created and stored as a relation on the parent entity.

A file can be attached to an entity via:

- Upload through Files Tab
- File Manager through Files Tab
- Document Generation
- File Attribute
- Image Attribute

360 View

Attributes | Relations 5 | Files | Notes

Drag a file here or:

Browse | File Manager | Templates

Max file size: 500 MB

2-20-2-21-23-39-29m.jpeg
38.8 kB - image/jpeg

a077Rpv_700bwp.webp
97.4 kB - image/webp

a077Rpv_700bwp.webp
97.4 kB - image/webp

2023-tesla-roadster-front-view-carbuzz-847781.jpeg
86.7 kB - image/jpeg

0a3ad697-afec-4e0d-8dd1-c5a1a9ee2425_How-to-get-started-with-Google-AdWords-for-your-marketplace.-.jpg
625 kB - image/jpeg

Add Price Option

## Product Images



## Product Downloads

Drag a file here or:

Browse | File Manager | Templates

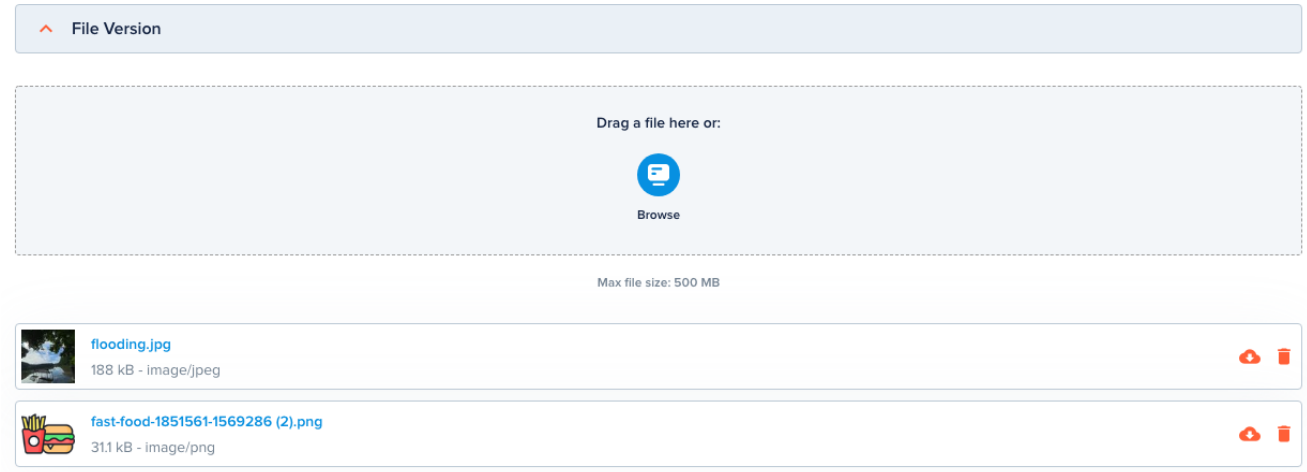
Max file size: 500 MB

2023-tesla-roadster-front-view-carbuzz-847781.jpeg
86.7 kB - image/jpeg


# Versions

File entities can be updated with new versions.

By default, the latest version is used when the File entity is referred.



# File API

[API Docs] [SDK]

Files in epilot are uploaded and managed through the File API.

## Uploading Files

The `uploadFile` operation returns a temporary presigned S3 URL, which the client uses to upload a file using the `PUT` or `POST` method.

After uploading, the client should call the `saveFile` operation to save the uploaded file as an entity make it permanent.

Files that are uploaded but not saved expire and are deleted within 24 hours.

## Updating Files

Modifying or saving new versions of File entities happens via the `saveFile` operation.

## Deleting Files

Deleting files is done using the `deleteFile` operation. When the file entity is deleted, the underyling S3 object is deleted permanently.

# Document Generation

[API Docs] [SDK]

Epilot supports generating PDF documents from template files containing template variables.

The Document Generation API uses references compatible with the File API.

# Pricing API

[API Docs] [SDK]

The Pricing API manages everything related to Products, Pricing and Checkout in epilot.

# Products

```
// TODO
```

# Opportunities

```
// TODO
```

# Taxes

```
// TODO
```

# Orders

```
// TODO
```

# Automation Flows

[API Docs] [SDK]

```
// TODO
```

# Automation Executions

[API Docs] [SDK]

## Actions

## Workers

# Workflows

[API Docs]

```
// TODO
```

# Webhooks

[API Docs] [SDK]

The epilot Webhooks API provides the possibility to subscribe to epilot public events. This will allow you to receive notifications with payload to your configured webhook URL every time events happen in your account.

This document describes the steps how to configure hooks, subscribe to events and how to manage those configurations. Service is reachable using https connection to ensure encryption between client and service.

Webhooks can be comfortably configured and managed by admin users in epilot portal.

Webhooks API Documentation

## Available events

For an overview about all events you can subscribe to with Webhooks you can call following endpoint `/webhooks/configured-events`. The response will contain a list of event names and their labels in form:

- **eventName**: Name for identifying the event.
- **eventLabel**: Either a user friendly label, or the eventName itself. When using the UI, you have the list of the available events in the drop down menu in webhook management form. Following Events are available for subscription:
- **customer request**: generated on incoming requests from JB Journeys

In Events Schemas section you find the schemas of available events ready to register.

## Create Webhook

To secure your client server connection, please setup your webhook using endpoints supporting encryption, preferably TLS. So we ensure encrypted data transfer to your server. An easy way to create webhook is to use the webhook ui in the epilot portal. You can set up your webhook configuration using the management form, and fill in there details about the webhook endpoint, event types and authorization information. However you still can use our API in the same way. To subscribe to an event using the API please use the `/webhooks/{yourOrganizationId}/configs` endpoint and post your receiver endpoint settings:

| field | Required | Description |
|---|---|---|
| eventName | required | epilot event you want to subscribe. see Which events are available |
| url | required | your endpoint where you want to receive the payload |
| httpMethod | required | http method |

| field | Required | Description |
| --- | --- | --- |
| enabled | optional | boolean whether the webhook is active or not |
| auth | required | your endpoint authentication information. See Authentication |
| filter | optional | filter options, here you have the possibility to filter events, by product categories for example |

# Authentication

We assume that your event handler endpoint is secured, we support currently following authentication types:

- Basic authentication
- Oauth
- API key

Further details on how to set up your authentication information using the API:

## Basic Authentication

If you are using basic authentication you can set up the `auth` field

| field | Required | Description |
| --- | --- | --- |
| authType | required | "BASIC" |
| basicAuthConfig | required | object only if authType is BASIC |

**basicAuthConfig**:

| field | Required | Description |
| --- | --- | --- |
| username | required | valid username for your endpoint |
| password | required | password |

## OAuth

In case your endpoint is secured using Oauth your `Auth` should have following structure:

| field | Required | Description |
| --- | --- | --- |

| field | Required | Description |
|---|---|---|
| authType | required | "OAUTH_CLIENT_CREDENTIALS" |
| oauthConfig | required | object only if authType is OAUTH_CLIENT_CREDENTIALS |

**oauthConfig**:

| field | Required | Description |
|---|---|---|
| clientId | required | your app Oauth client Id |
| clientSecret | required | Oauth client secret |
| endpoint | required | HTTPS endpoint for authentication |
| httpMethod | required | HTTP methods like GET, POST... |

### API key

We support also endpoints secured using API keys. In this case your webhook set up could be configured this way:

| field | Required | Description |
|---|---|---|
| authType * | "API_KEY" | |
| apiKeyConfig * | object only if authType is API_KEY | |

**apiKeyConfig**:

| field | Required | Description |
|---|---|---|
| keyName | required | used key name |
| keyValue | required | value of the used key |

## Enable Filtering

For better software integration you have the possibility to set up more granular filter for your subscribed events. For this please make use of the filter option when you create a webhook. Please note that the entire event will be sent when the filter matches.

Using our ui in epilot portal, you can enable the filter option, and select items to be filtered.

Using the webhook API, here more details about the possible and required fields.

| field | Required | Description |
| --- | --- | --- |
| keyToFilter | required | field of payload you want to filter |
| supportedValues | required | list of values you want to receive |

The subscribed events are simply a json structure, to have a filter in place for specific field in the event, you just set the `keyToFilter` value to be the field attribute structure of the json.

For example you have the following event structure and you want your filter to apply only auth type :

```
keyToFilter: {
  id,
  name,
  security {
    auth {
      type // basic|apikey|oauth|none
    }
  }
}
```

the `keyToFilter` should be `keyToFilter.security.auth`. And possible values should be set in `supportedValues` field.

Please note that the `supportedValues` are **case sensitive**.

Following example will filter this sample event to send only events having auth types basic and oauth.

```
filter: {
  keyToFilter: 'keyToFilter.security.auth',
  supportedValues: ['basic', 'oauth']
}
```

**Sample filter:**

For `Customer request` event you can filter events and receive only events related to specific product category:

```
filter: {
  keyToFilter: 'customer_request.request_items.product_category',
  supportedValues: ['solar', 'electricity']
}
```

# List configured webhooks

The `/webhooks/{yourOrganizationId}/configs` endpoint provides the list of the configured webhooks by your organization in following structure:

| field | Description |
| --- | --- |
| id | webhook id |
| eventName | subscribed event name |
| url | configured client endpoint Url |
| creationTime | webhook creation time |
| httpMethod | configured http method |
| enabled | boolean whether the webhook is enabled or not |
| auth | Auth settings if set |
| filter | filter settings if set |

# Delete webhook

To delete configured webhook using the ui, just hit the delete button for the wanted webhook configuration. To delete a webhook configuration using the API please use the `/webhooks/{yourOrganizationId}/configs/{WebhookId}` endpoint with `DELETE` http call.

After deleting a webhook configuration you are still able to fetch failed and successfully sent events related to the deleted configuration.

To retrieve webhook id you can query the configured webhooks, see List configured Webhooks.

# Edit Webhook

Using the ui in the epilot portal you can very easily edit a webhook config. You will be asked to edit the pre filled form. To update a webhook configuration using the API please use following endpoint `/webhooks/{yourOrganizationId}/configs/{WebhookId}` using `PUT` http method. To retrieve webhook id you can query the configured webhooks, see List configured Webhooks.

Additional to this path parameters, the payload to update the webhook configuration is the same we use for creating new webhooks. You can also refer to Which events are available for more details.

To deactivate or reactivate a webhook configuration, you can make use of this endpoint, providing the organization id and webhook id as path parameter, and your payload should contain same configuration saved except the field

`enabled` should either `false` if you want to deactivate the webhook otherwise `true` if the webhook should be active.