API Reference



2026, © Approval Studio, v. 1.35

Please find the most recent **API Reference** online here.

Table of contents

- General
- Approval Studio basic concepts
  - REST HTTP Codes
- Authorization/authentication
- Token management
  - POST /api/v1/token/login
  - GET /api/v1/token/validate
- Project management
  - GET /api/v1/project
  - POST /api/v1/project
  - PUT /api/v1/project
  - DELETE /api/v1/project
  - GET /api/v1/project/proofreport
  - PUT /api/v1/project/state
  - GET /api/v1/project/meta
  - POST /api/v1/project/meta
  - PUT /api/v1/project/meta
  - DELETE /api/v1/project/meta
  - GET /api/v1/project/meta/projects
  - POST /api/v1/project/kanban/column
  - POST /api/v1/project/folder
- Asset management
  - GET /api/v1/asset
  - DELETE /api/v1/asset

    - POST /api/v1/asset/upload
    - POST /api/v1/asset/upload_url
    - GET /api/v1/asset/download
    - GET /api/v1/asset/proofreport
- Reference documents management
    - POST /api/v1/refdoc/upload
    - POST /api/v1/refdoc/upload_url
    - POST /api/v1/refdoc/add_url
    - GET /api/v1/refdoc/download
    - DELETE/api/v1/refdoc
- DAL management
    - DAL Concepts
    - DAL data model
    - Endpoints summary
    - GET /api/v1/dal/folders
    - POST /api/v1/dal/folder
    - PUT /api/v1/dal/folder
    - PATCH /api/v1/dal/folder
    - DELETE /api/v1/dal/folder
    - GET /api/v1/dal/asset
    - GET /api/v1/dal/folder/assets
    - PUT /api/v1/dal/asset
    - DELETE /api/v1/dal/asset
    - POST /api/v1/dal/asset/project
    - POST /api/v1/dal/asset/folder
- Task management
    - GET /api/v1/task/all
    - GET /api/v1/task
    - DELETE /api/v1/task
    - POST /api/v1/task/asset_upload
    - POST /api/v1/task/asset_change_upload
    - POST /api/v1/task/refdoc_upload
    - POST /api/v1/task/review_asset
    - POST /api/v1/task/review_asset_ext
    - PUT /api/v1/task/complete
- Annotations management
    - GET /api/v1/annotation/all
    - GET /api/v1/annotation
    - POST /api/v1/annotation
    - DELETE /api/v1/annotation
    - PUT /api/v1/annotation/hide
    - PUT /api/v1/annotation/complete

    - PUT /api/v1/annotation/uncomplete
- Users management
    - GET /api/v1/users
    - GET /api/v1/user/loggedin
- Webhooks
    - Security
    - Retry logic
    - Event object
- Webhooks management
    - GET /api/v1/webhooks
    - POST /api/v1/webhook
    - DELETE /api/v1/webhook
    - PUT /api/v1/webhook/test
- Workflow Management
    - Introduction
    - Triggers
    - Actions
    - Trigger-Reaction dependencies
    - Roles
    - Workflow fields
    - Workflow management
    - GET /api/v1/workflows
    - GET /api/v1/workflow
    - POST /api/v1/workflow

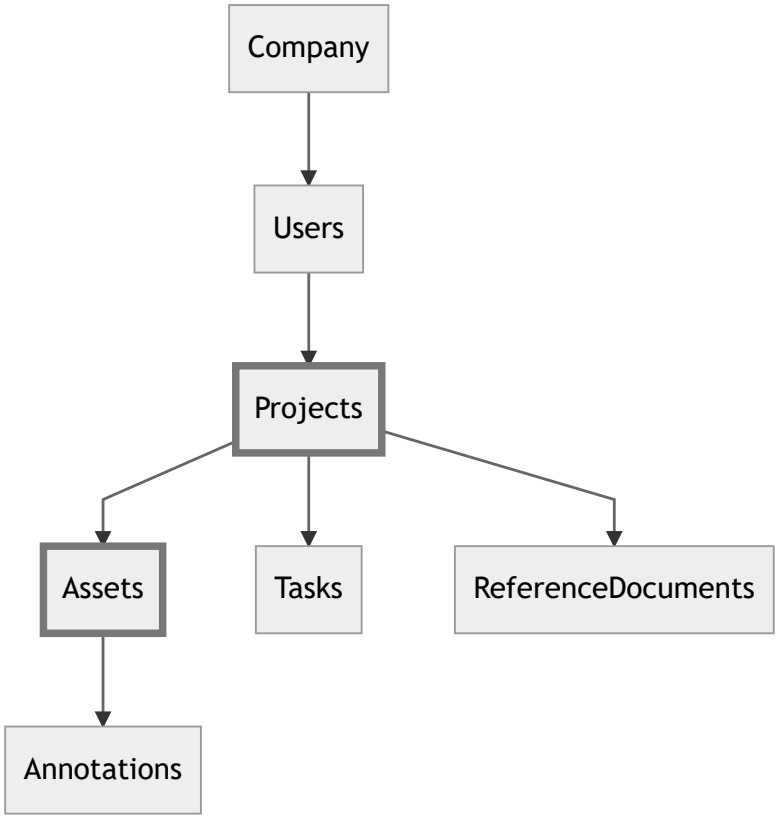
General

This is a REST API to the Approval Studio, design review, and packaging approval SAAS.
API utilizes its flow, authorization, storage, and so on.

Approval Studio basic concepts

From the business point of view, Approval Studio is based on a multitenant concept when a single user may be a part of one or more tenants (called Companies). Please see https://approvalstudio.freshdesk.com/ for detailed instructions on how to work with Approval Studio.

Currently the only point where you may choose a tenant is project creation, please see POST /api/v1/project / Request.



REST HTTP Codes

API may return one of the following HTTP codes:

HTTP Code	Response
200	**Success**. See the method description to get what and how the method returns.
400	**Validation error**. API validates input parameters and when finds an invalid parameter, throws HTTP code 400. Those errors are related to parameters' presence, format, emptiness, etc. Validating against a database, like looking for data by an ID is conducted separately and reflected in case of error, in HTTP codes 404, 406, 412, etc., please see the method's description for detailed explanations.

```
{
  "errors": {
    "parameterName": [
      "'parameterName' must not be empty."
    ]
```

```
  },
  "type": "https://tools.ietf.org/html/rfc7231#section-6.5.1",
  "title": "One or more validation errors occurred.",
  "status": 400,
  "traceId": "|a4a45224-4a7c03bbd5172369."
}
```

HTTP Code	Response
404, **406**, **412**	**Resource not found**. Generally, it means that requested resources are either not found or somehow restricted to proceed. For example, if a given project id is non-existent, the project is not found so gets HTTP code 404.
429	**Rate Limit Reached**. API host calculates the number of calls per sec, minute, and hour, and when the rate of requests reaches the limit, throws HTTP code 429, which means that the API host is overloaded and the client needs to wait before retrying.

If the request gets blocked then the client receives a text response like this:

```
Status Code: 429
Retry-After: 58
Content: API calls quota exceeded! maximum admitted 2 per 1m.
```

`Retry-After` header value is expressed in seconds. And `X-Rate-Limit-XXX` HTTP headers are injected in the response:

```
X-Rate-Limit-Limit: the rate limit period (eg. 1m, 12h, 1d)
X-Rate-Limit-Remaining: number of requests remaining
X-Rate-Limit-Reset: UTC date-time (ISO 8601) when the limits reset
```

HTTP Code	Response
500	**Infrastructure failure**. This means critical unrecoverable technical error generally related to database connections, external services availability, hardware failure, etc. Depending on the client application flow you can retry the calling method or halt processing and call our technical support.

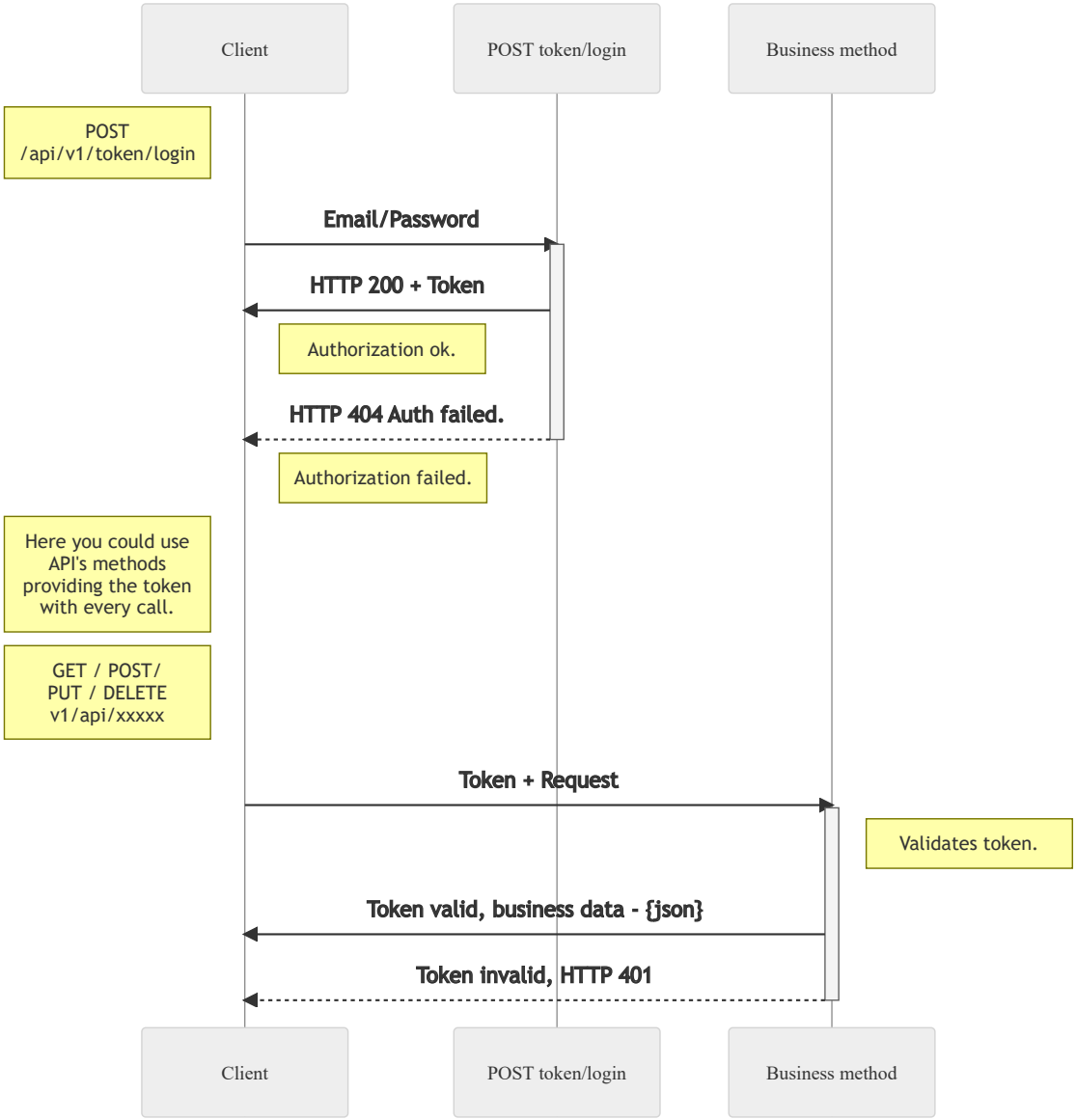
Authorization/authentication

The API is based on an **authorization token** which should be requested prior to using any available API methods.
The flow is the following: request auth token by calling **POST/api/v1/token/login** (see) providing Approval Studio's username/password. If ok, the method returns an authorization token that you should **provide with any other API call** as an **HTTP header** like this: `Authorization: Bearer XXXXXXXXXX.`

The token is valid for a limited amount of time, **600 minutes** by default. When the token is expired and still used you will have the response `HTTP code 401, Unauthorized :`

```
{
  "isError": true,
  "type": "https://httpstatuses.com/401",
  "title": "Unauthorized",
  "status": 401,
  "instance": "/api/v1/annotation"
}
```

If so, you need to **obtain a new token.**
If the token is invalid or not provided, it gets the same response **HTTP code 401**.



Token management

POST /api/v1/token/login

Authorizes a user. When successfully authorized is returns an **authorization token** which must be supplied to every API as a header, like `"Authorization: Bearer YYYYYYYYYYYYYYYYY..."`.

Request

```
{
  "userName": "john.smith@gmail.com",
  "password": "Q@fG%^18_A",
  "keepAliveTime": 12                  // Optional, minutes.
}
```

`keepAliveTime` is an optional parameter that defines how much time the token will be valid. Response's `expirationDate` provides an exact time of the token expiration (see below).
Note: When `keepAliveTime` is zero, the default value is used (600 min/10 hours).

Curl:

```
curl -X POST "https://api.approval.studio/api/v1/token/login" \
     -H "accept: text/plain" \
     -H "Content-Type: application/json-patch+json" \
     -d "{\"userName\":\"joshn.smith@pepsico.com\",\"password\":\"Q@fG%^18_A~\",
```

Responses

HTTP Code	Response
200	**Success**. Login successful, AUTH token provided.

```
{
  "version": "1.0",
  "statusCode": 200,
  "message": "POST Request successful.",
  "result": {
    "token": "eyJhbGciOiJIUzI1NiIsI[...]", // This is the auth token.
    "status": "Success",
    "expirationDate": "2022-02-15T18:42:15.8398451Z", // A date/time when the tok
    "user": {
      "userUID": "XXXXXXXXXXXXXXXXXXXXXXXXXXX",
      "fullName": "John Smith",
      "email": "john.smith@gmail.com",
      "role": "RegularUser|Administrator"
    }
  }
}
```

HTTP Code	Response
400	**Error**: Parameters' validation failed. See **HTTP code 400 description**.
404	**Error**: User with given email and password not found. Either the provided credentials are wrong or the user is locked and not able to login anymore.


```
{
  "version": "1.0",
  "statusCode": 404,
  "message": "Authorization failed.",
  "result": {
    "status": "WrongCredentials"
  }
}
```


HTTP Code	Response
412	**Error**: The user uses **Lite** payment plan option. API is available only for **Pro** users. Please consult our support on this: https://approval.studio/contact/

GET /api/v1/token/validate

Validates an auth token. This method is an kind of a health checker, that validates an authentication token, if it's still valid. You may use it to ensure that the your authentication process completed successfully or that the token is still valid in case of a long-running scenarious like external workflow engines where tokens are saved for further use.

Request

Curl:

```
curl -X 'GET' \
  'https://api.approval.studio/api/v1/token/validate' \
  -H "accept: text/plain" \
  -H "Content-Type: application/json-patch+json" \
```

Responses

HTTP Code	Response
200	**Success**. authentication token provided in the **Auth** header is valid.

```
{
  "version": "1.0",
  "statusCode": 200,
  "message": "POST Request successful.",
  "result": {}
}
```

HTTP Code	Response
401	**Error**: Parameters' validation failed.

Project management

Project attributes:

Name		Explanation
`projectUID`	**r/o**	System-wide unique project identifier, GUID
`projectState`		Projects are in one of the following states: `Active`,`OnHold`,`Completed`,`InTransit`,`Archived`.
`name`		Free-form project name, string, max-length 50 chars, mandatory.
`customer`		Project's customer name, max-length 50 chars, optional.
`project`		Business project or product name this Approval Studio project related for, max-length 50 chars, optional.
`design`		Point to which type of design this is, max-length 50 chars, optional.
`revision`		Ideally, a sequential number or version number; generally a free form string max-length 50 chars, optional.
`description`		Any kind of additional information, descriptions, comment etc., max-length 200 chars, optional.

Name		Explanation
`tags`		Array of free-form single-word tags associated with the project, optional, max 20 elements of 20-chars tags.
`dueDate`		Optional date, UTC, that points to when the project is desired to be completed.
`reviewStatus`	**r/o**	`pendingCount` - count of non-completed review tasks assigned on the project
		`approvedCount` - count of approves made on the project's assets
		`rejectedCount` - count of rejected made on the project's assets
`created`	**r/o**	Refers to when the project was created, UTC.
`templateUID`		It is **a new concept**, introduced in API v.1.15. **Project template** is a set of some data, usually a JSON, that administrator can provide when creating a new project. That could be a list of users, emails, etc any other business-specific attributes that could be used in building a business flow and should be specific for every project. For example, you might need to create a review task just after the project creation and you use API to implement this. In that case, you may provide UID of a user to create a review task for. The template itself doesn't make any changes in the default application flow, it's just a customizable attribute.
`kanbanColumnUID`		It is an unique ID of the Kanban column this project is placed to on the dashboard.
`folderUID`		It is an unique ID of the folder this project is placed to on the dashboard.

Note: Kanban columns and folders are ways to group projects based on some your business needs, probably, based on a kind of state or position on flow. This is out of scope of this document.

Method/Path	Description
`GET /api/v1/project`	Gets a list of projects.

Method/Path	Description
`POST /api/v1/project`	Creates new project.
`PUT /api/v1/project`	Edits existing project.
`DELETE /api/v1/project`	Deletes a project.
`GET /api/v1/project/proofreport`	Gets a link to a proof report for a project.
`PUT /api/v1/project/state`	Changes project's state.

GET /api/v1/project

Returns one or more projects and projects' assets, tasks, and reference documents – depending on the parameters passed.
All the parameters are optional; the method always uses **AND** combination of all the parameters.

Parameter	Type & Explanation
`ProjectUID`	`string` Unique project UID or comma-separated list of project UIDs. **Note**: behavior is changed in v.1.15. Now if `ProjectUID` is provided and this project belongs to an appropriate client, you will get project data regardless of the owner list and whether you are an admin or not. In other words, if this project belongs to your client (tenant), you will get it. **Note**: validation changes in v.1.29. If multiple projects UIDs provided, the result will contain only those projects that are found. Wrong, deleted etc will be silently omitted.
`States`	`string` One or more project states, comma-separated, see *ProjectStates* above. Default value is `Active,OnHold,Completed`.
`Query`	`string` A free-form text to case-sensitive search in projects' attributes: `Project name`, `Customer`, `Project`, `Design`, `Revision`, `Description`, `Tags`.

Parameter	Type & Explanation
`IsLoadAssets`	`boolean`, `false` by default. If it is set, returns assets for every project as a child collection.
`IsLoadLastVerAssets`	`boolean`, `true` by default. If set, only the most recent version of every asset be returned. Ignored, if `IsLoadAssets` is not set.
`IsLoadTasks`	`boolean`, `false` by default. If set, a list of active tasks for every project would be returned as a child collection.
`IsLoadRefDocs`	`boolean`, `false` by default. If set, a list of uploaded reference documents for every project would be returned as a child collection.
`IsLoadMetadata`	`boolean`, `false` by default. If set, metadata for every project would be returned as a list of key-value entries.

Request URL

```
https://api.approval.studio/api/v1/project?ProjectUID=XXXXX&
States=Active%2COnHold%2CCompleted%2CArchived%2CInTransit&Query=Some%20Free%20Tex
IsLoadAssets=true&IsLoadLastVerAssets=true&IsLoadTasks=true&IsLoadRefDocs=true
```

Curl:

```
curl -X GET "https://api.approval.studio/api/v1/project?
             ProjectUID=XXXXXXXXXXXXXXXXXXX&States=Active%2COnHold%2CCompleted&
             Query=Some%20Free%20Text&IsLoadAssets=true&
             IsLoadLastVerAssets=true&IsLoadTasks=true&IsLoadRefDocs=true" \
    -H  "accept: text/plain" \
    -H  "Authorization: Bearer YYYYYYYYYYYY"
```

- Default `States` is `Active,OnHold,Completed`.
- If `ProjectUID` is provided and no project is found, an empty list returns an HTTP code 200.

Response

```
[
  {
    "projectUID": "GUID",
    "projectState": "Active|OnHold|Completed|InTransit|Archived",
    "name": "string",
```

```
"customer": "string",
"project": "string",
"design": "string",
"revision": "string",
"description": "string",
"tags": [
  "tag 1", "tag 2"...
],
"dueDate": "2020-11-22T23:00:53.425Z",
"reviewStatus": {                    // Project-level proof review status
                                     // (see proofing flow explanations).
          "pendingCount": 1,         // The asset has 1 uncompleted review ta
          "approvedCount": 2,        // The asset has been approved 2 times
          "rejectedCount": 3         // The asset has been rejected 3 times
      },
"created": "2020-11-22T23:00:53.425Z",
"template": {
    "templateUID": "XXXXXXXXXXXXXXX",// Template's unique ID.
    "name": "string",                // Template name, presumable unique for a
    "data": {                        // Optional data, associated with the pro
      "key": "value",                // Json or free-form string.
      "key1": "value1"
    }
  }
"assets": { // Optional,
  "asset_one.jpeg": // Unique asset name
  [                 // List of assets' versions, one or more
    {
      "assetUID": "GUID",
      "version": 0,
      "status": "Pending|Processed|Failed", // Or integer, 0,1,2
      "reviewStatus": {                    // Proof review status (see proofi
          "pendingCount": 1,               // The asset has 1 uncompleted rev
          "approvedCount": 2,              // The asset has been approved 2 t
          "rejectedCount": 3               // The asset has been rejected 3 t
      },
      "pagesCount": int, >=1,
      "created": "2020-11-22T23:00:53.425Z",
      "fileSize": int, bytes,
      "reviewUrl": string,                 // URL to a prooftool to view/proc
      "thumbnailUrl": string               // URL to asset's thumnail image
      "fullSizeBitmapUrl": string          // URL to converted asset's image
    }
  ],
  "asset_two.pdf": [
    {
      "assetUID": "GUID",
      "version": 0,
      "status": "Pending|Processed|Failed",
```

```
      "reviewStatus": {
          "pendingCount": 1,
          "approvedCount": 2,
          "rejectedCount": 3
      },
      "pagesCount": 0,
      "created": "2020-11-22T23:00:53.425Z",
      "fileSize": 0,
      "reviewUrl": "https://app.approval.studio/xxx",
      "thumbnailUrl": "https://app.approval.studio/yyy"
      "fullSizeBitmapUrl": "https://app.approval.studio/zzz"
    }
  ],
  ...
  ]
},
"tasks": [
  {
    "taskUID": "GUID",
    "type": "UploadAssets|UploadRefDocs|ReviewAssets|ExternalReviewAssets|Upl
    "status": "Pending|Closed|Approved|Rejected",
    "comment": "string",
    "dueDate": "2020-11-22T23:00:53.425Z", // Optional.
    "created": "2020-11-22T23:00:53.425Z",
    "closed": "2020-11-22T23:00:53.425Z",
    "user": {
      "userUID": "GUID",
      "fullName": "string",
      "email": "string"
    },
    "assets": [
      "Asset GUID", "Asset GUID 2"...
    ],
    "reviewUrl": string                    // URL to launch a prooftool for th
                                           // Appears only for the ReviewAsset
  }
],
"refDocs": [
  {
    "refDocGUID": "GUID",
    "created": "2020-11-22T23:00:53.425Z",
    "name": "filename.ext",
    "fileSize": int, bytes.
  }
],
"metadata": [
    "orderNumber": "A19KQ64A",           // Key-Value pair(s) of metadata
    "contectEmail": "contact@mail.com"
],
```

```
    "client": {
        "clientUID": "YYYYYYYYYYYY",        // Client's uniquie ID.
        "name": "Yourcompany Ltd"           // Client name.
    },
    "kanbanColumnUID": "j2ax",              // Optional kanban column UID for this
    "folderUID": "dv5t",                    // Optional folder UID for this proje
    "workflow": {
        "workflowUID": "WWWWWWWW",          // Optional workflow UID.
        "name": "Primary Workflow"          // Optional workflow's name.
    }
  }
]
```

POST /api/v1/project

Creates a new project taking mandatory project name and a list of owners and a set of optional attributes.

Request

Field	Type & Explanation
`clientUID`	`string[50]` Optional client UID. See GET /api/v1/users / Responses. If no client ID is provided, the first client will be chosen by default.
`projectName`	`string[200]` Mandatory project name, free-form text.
`customer`	`string[200]` Optional customer name.
`project`	`string[50]` Optional (sub)project name.
`design`	`string[50]` Optional design type/name, like “package” or “banner” etc.
`revision`	`string[50]` Optional revision number, sequential or free-form.
`description`	`string[1000]` Optional project description, free-form text.
`tags`	`string array` Optional tag list, max 20 tags of max length of 25 chars each.
`dueDate`	`ISO date` Optional UTC date (or date-time) that point to a date when project supposed to be completed. The date affects the project’s status and sort order on the application dashboard.

Field	Type & Explanation
`projectOwnersUIDS`	`string array` Mandatory list of the project owner(s)'s UIDs. At least one owner must be provided, max number of owners is 20.

```
{
  "clientUID": "string",        // Optional tenant(client) ID.
  "projectName": "string",  // Project name, mandatory.
  "customer": "string",
  "project": "string",
  "design": "string",
  "revision": "string",
  "description": "string",
  "tags": [
    "string", "string"
  ],
  "dueDate": "2020-12-07T20:56:38.818Z",
  "projectOwnersUIDS": [    // Project owner(s), at least one owner must be p
    "XXXXXXXX"
  ],
   "templateUID": "string"  // Optional project template UID. See client list
   "folderUID": "string"    // Optional folder UID. See client list to get an

}
```

```
curl -X POST "https://api.approval.studio/api/v1/project"
     -H "accept: text/plain"
     -H "Authorization: Bearer YYYYYYYY"
     -H "Content-Type: application/json-patch+json"
     -d "{\"projectName\":\"string\",\"customer\":\"string\",\"project\":\"string
```

Responses

HTTP Code	Response
200	Project created and instance returned.

```
{
  "projectUID": "string",
  "projectState": 0,
  "name": "string",
  "customer": "string",
  "project": "string",
  "design": "string",
  "revision": "string",
```

```
    "description": "string",
    "tags": [
      "string"
    ],
    "dueDate": "2020-11-27T13:37:07.534Z",
    "created": "2020-11-27T13:37:07.534Z",
    "template": {
      "templateUID": "string",
      "name": "string",
      "data": "string or object"
    },
    "client": {
      "clientUID": "string",
      "name": "string"
    },
    "kanbanColumnUID": "string",
    "folderUID": "string,
    "workflow": {
      "workflowUID": "string",
      "name": "string"
    }
  }
```

HTTP Code	Response
400	**Error**: Bad Request. One of the pre-requisites failed to validate

PUT /api/v1/project

Changes project's attribute(s) including name, due date, and list of owners (edit project).

Request

```
{
    "projectUID": "ProjectUID", // ID of project to edit
    "projectName": "string",
    "customer": "string",
    "project": "string",
    "design": "string",
    "revision": "string",
    "description": "string",
    "tags": [
      "string", "string", "string"
    ],
```

```
    "dueDate": "2020-12-02",
    "projectOwnersUIDS": [
      "UserUID","UserUID" ...
    ],
    "folderUID": "string"
  }
```

Omit those properties you want to stay untouched; so if you provide a request like this below, only the project name will be updated:

```
{
    "projectUID": "XXXXXXXXXX", // Project ID to edit
    "projectName": "New name"
}
```

Curl:

```
curl -X PUT "http://api.approval.studio/api/v1/project"
     -H "accept: text/plain"
     -H "Authorization: Bearer YYYYYYYYYYYYYYYYY..."
     -H "Content-Type: application/json"
     -d "{\"projectUID\":\"XXXXXXXXXX\",\"projectName\":\"New Name\"}"
```

Responses

HTTP Code	Response
200	**Success**. The project's attributes changed.

```
{
  "projectUID": "XXXXXXXXXX",
  "projectState": "Active|OnHold|Completed|InTransit|Archived",
  "name": "string",
  "customer": "string",
  "project": "string",
  "design": "string",
  "revision": "string",
  "description": "string",
  "tags": [
    "string", "string", ...
  ],
  "dueDate": "2020-11-30",
  "created": "2020-11-30T12:09:36.426Z",
  "kanbanColumnUID": "string",
  "folderUID": "string,
  "workflow": {
```

```
    "workflowUID": "string",
    "name": "string"
  }
}
```

HTTP Code	Response
400	**Error**: Parameters' validation failed.
	See HTTP code 400 description.

HTTP Code	Response
404	**Error**: Project with the given ID not found or it has been already deleted.
	Error: Folder with the given UID not available for this project.
	Error: Project owner with the UID [XXXXXXXXXX] not available.

```
{
  "version": "1.0",
  "statusCode": 404,
  "message": "Project not found or already deleted."
}
```

HTTP Code	Response
406	**Error**: Project editing is possible only when the project is `Active` or `OnHold`. `Completed`, `Archived` or `InTransit` projects are not mutable. Please see PUT /api/v1/project/state.

```
{
  "version": "1.0",
  "statusCode": 406,
  "message": "Can't edit completed or archived projects."
}
```

HTTP Code	Response
412	**Error**: Project must be in state **Completed** or **Archived**. If not, the code 412 returns.

```
{
  "version": "1.0",
```

```
  "statusCode": 412,
  "message": "Only archived or on-hold project can be deleted."
}
```

DELETE /api/v1/project

Deletes a project.

This is undoable; once the project is deleted, it disappears from a list of projects; assets and uploaded reference documents are deleted as well.

The project should have the status **Completed** or **Archived** to be deleted; an error will be thrown elsewhere,

Request

```
{
  "projectUID": "XXXXXXXXXXXX"
}
```

Curl:

```
curl -X DELETE "https://api.approval.studio/api/v1/project"  \
     -H  "accept: text/plain" \
     -H  "Authorization: Bearer YYYYYYYYYYYYYYYYY..." \
     -H  "Content-Type: application/json" \
     -d "{\"projectUID\":\"XXXXXXXXXXXXX\"}"
```

Responses

HTTP Code	Response
200	**Success**. Project deleted.

```
{
  "version": "1.0",
  "statusCode": 200,
  "message": "Project deleted."
}
```

HTTP Code	Response
400	**Error**: Parameters' validation failed.
	See HTTP code 400 description.

HTTP Code	Response
404	**Error**: Project with the given ID not found, and, therefore project deletion failed.

```
{
  "version": "1.0",
  "statusCode": 404,
  "message": "Project not found."
}
```

HTTP Code	Response
412	**Error**: Project must be either **Completed** or **Archived** to be deleted.

GET /api/v1/project/proofreport

Returns URL to get a printable **proof report** for the given project. The method does not allow direct downloading.
Read `result/downloadURL` and navigate it to get the report body. The report is a plain single-layer PDF 1.4 file.

Request

The request is a set of **URL GET** parameters separated by **&**:

```
curl -X GET "https://api.approval.studio/api/v1/project/proofreport?ProjectUID=XX
     -H  "accept: text/plain" \
     -H  "Authorization: Bearer YYYYYY"
```

Parameter	Type & Explanation
`ProjectUID`	`string[50]` Mandatory project identifier.

Responses

HTTP Code	Response
200	**Success**. Proof report URL generated.

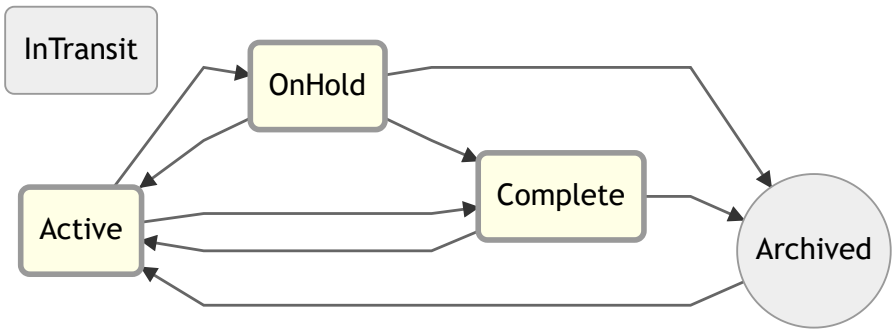

```
{
  "version": "1.0",
  "statusCode": 200,
  "message": "GET Request successful.",
  "result": {
    "downloadURL": "https://approval.studio/ProofApi/GetProofReport/DDDDDDDDDDDDDI
    "project": {
      "projectUID": "3AB3A5F58951467B975378198C7265D1",
      "projectState": "Complete",
      "name": "Project for Pepsi Co",
      "tags": ["tag1", "tag2"],
      "created": "2020-12-02T13:57:15.568992"
    }
  }
}
```


HTTP Code	Response
400	**Error**: Parameters' validation failed.
	See HTTP code 400 description.
HTTP Code	Response
---------- --	----------
404	**Error**: Project with the given ID not found and therefore proof report generation failed.


```
{
  "version": "1.0",
  "statusCode": 404,
  "message": "Project not found."
}
```


PUT /api/v1/project/state

Changes project's state according to the state rules:



The diagram above explains the pre-requisite check for changing the project's state.
Note: The diagram above **is a recommended way** to change project's state, **it is not mandatory to follow it**, but highly recommended.
Note: Detailed description of the project management is out of scope of this document.

States

Those are the states:

State	Explanation
Active	Work on a project is undergoing. This is where all the review work is going on.
OnHold	Due to some reason you want to postpone working on a project. Setting state to **OnHold** moves the project to a separate lane. Tasks are still intact, the only difference from Active is a position on a separate lane on the dashboard.
Completed	When setting the state to **Completed** all project tasks are deleted and the project itself moves from the dashboard to a separate screen where all completes are stored separately. All the assets and reference documents are preserved as well as history etc. The project goes read-only.
Archived	This is generally the same as **Completed** but assets and reference documents are moved to a remote, slow storage. The process of moving files to other storage could take a while, so when it is going on, the system marks a project as **InTransit** (see below).
InTransit	Project is locked to **InTransit** state by API itself when it is switching to **Archive** state or, vice versa, from **Archived** to **Active**. You can't set this state forcedly.

Request

```
{
  "projectUID": "XXXXXXXXXXX",
  "projectState": "Active|OnHold|Completed|Archived"
}
```

Responses

HTTP Code	Response
200	**Success**. The project's state changed.

```
{
  "projectUID": "XXXXXXXXXX",
  "projectState": "Active|OnHold|Completed|Archived",
  "name": "string",
  "customer": "string",
  "project": "string",
  "design": "string",
  "revision": "string",
  "description": "string",
  "tags": [
    "string", "string", ...
  ],
  "dueDate": "2020-11-30",
  "created": "2020-11-30T12:09:36.426Z"
}
```

HTTP Code	Response
400	**Error**: Parameters' validation failed.
	See HTTP code 400 description.
HTTP Code	**Response**
404	**Error**: Project with the given ID not found or it has been already deleted.

```
{
  "version": "1.0",
  "statusCode": 404,
  "message": "Project not found."
}
```

HTTP Code	Response
406	**Error**: The project with the given ID already has the requested state; no updating made.

```
{
  "version": "1.0",
  "statusCode": 412,
  "message": "Project already has state XXXX."
}
```

GET /api/v1/project/meta

This methods **gets a key-value list of the metadata** for the given project.

The concept of project's metadata is the following: you can assign any kind of textual invisible for end-users data to a chosen project. Technically this is a list of key-value pairs.
Project's metadata might be used to provide data necessary for integration with other software, like order number, linked with a project, email of contact person etc.

Note: Apart from this method, you may use GET /api/v1/project with flag `IsLoadMetadata` set to `true`.
Curl:

```
curl -X 'GET' \
  'https://api.approval.studio/api/v1/project/meta?ProjectUID=XXXXXXXXXXXXXXXXXX
  -H 'accept: text/plain'
  -H 'Authorization: Bearer ZZZZZZZZZZZZZZZZZZZZ'
```

Parameter	Type & Explanation
`ProjectUID`	`string` Unique project identifier.

Responses

HTTP Code	Response
200	**Success**. List of metadata entries returned.

```
{
  "version": "1.0",
  "statusCode": 200,
  "message": "GET Request successful.",
  "result": {                                    // Metadata group
    "orderNumber": "A19KQ64A",                   // Key-Value pair(s) of metadata
    "contectEmail": "contact@username.com"
  }
}
```

HTTP Code	Response
400	**Error**: Parameters' validation failed.
	See HTTP code 400 description.

HTTP Code	Response
404	**Error**: Project with the given ID not found.

```
{
  "version": "1.0",
  "statusCode": 404,
  "message": "Project not found."
}
```

POST /api/v1/project/meta

This methods **adds one or more metadata entry** to the given project.

Request:

```
{
  "projectUID": "D37683CE23E24079951DC48C96114094",
  "metadata": {
    "orderNumber": "A19KQ64A",
    "contectEmail": "contact@username.com"
  }
}
```

Curl:

```
curl -X 'POST' \
  'https://api.approval.studio/api/v1/project/meta' \
  -H 'accept: text/plain' \
  -H 'Authorization: Bearer YYYYYYYY' \
```

```
  -H 'Content-Type: application/json-patch+json' \
  -d '{
  "projectUID": "D37683CE23E24079951DC48C96114094",
  "metadata": {
    "orderNumber": "A19KQ64A",
    "contectEmail": "contact@username.com"
  }
}'
```

Responses

HTTP Code	Response
200	**Success**. Metadata added successfully.

```
{
  "version": "1.0",
  "statusCode": 200,
  "message": "Metadata added."
}
```

HTTP Code	Response
400	**Error**: Parameters' validation failed.
	See HTTP code 400 description.

HTTP Code	Response
404	**Error**: Project with the given ID not found.

```
{
  "version": "1.0",
  "statusCode": 404,
  "message": "Project not found."
}
```

HTTP Code	Response
406	**Error**: Completed and archived projects can't be altered. You can add metadata to only project with statuses `Active` or `OnHold`.

PUT /api/v1/project/meta

This methods **edits an existing metadata entry** of the given project.

Request:

```
{
  "projectUID": "XXXXXXXXXXXXXX",
  "key": "orderNumber",
  "value": "A19KQ64A"
}
```

Curl:

```
curl -X 'PUT' \
  'https://api.approval.studio/api/v1/project/meta' \
  -H 'accept: text/plain' \
  -H 'Authorization: Bearer YYYYYYYY' \
  -H 'Content-Type: application/json-patch+json' \
  -d '{
  "projectUID": "XXXXXXXXXXXXXX",
  "key": "orderNumber",
  "value": "A19KQ64A"
}'
```

Responses

HTTP Code	Response
200	**Success**. Metadata changed successfully.

```
{
  "version": "1.0",
  "statusCode": 200,
  "message": "Metadata edited."
}
```

HTTP Code	Response
400	**Error**: Parameters' validation failed.
	See HTTP code 400 description.

HTTP Code	Response
404	**Error**: Project with the given ID not found.

```
{
  "version": "1.0",
  "statusCode": 404,
  "message": "Project not found."
}
```

HTTP Code	Response
406	**Error**: Completed and archived projects can't be altered. You can add metadata to only project with statuses `Active` or `OnHold`.

DELETE /api/v1/project/meta

This methods **deleted an existing metadata entry** of the given project.

Request:

{

“projectUID”: “XXXXXXXXXXXXXX”,

“key”: “orderNumber”

}

Curl:

```
curl -X 'DELETE' \
  'https://api.approval.studio/api/v1/project/meta' \
  -H 'accept: text/plain' \
  -H 'Authorization: Bearer YYYYYYYY' \
  -H 'Content-Type: application/json-patch+json' \
  -d '{
  "projectUID": "XXXXXXXXXXXXXX",
  "key": "orderNumber"
}'
```

Responses

HTTP Code	Response
200	**Success**. Metadata entry deleted successfully.

```
{
  "version": "1.0",
  "statusCode": 200,
```

```
  "message": "Metadata deleted."
}
```

HTTP Code	Response
400	**Error**: Parameters' validation failed.
	See HTTP code 400 description.

HTTP Code	Response
404	**Error**: Project with the given ID not found.

```
{
  "version": "1.0",
  "statusCode": 404,
  "message": "Project not found."
}
```

HTTP Code	Response
406	**Error**: Completed and archived projects can't be altered. You can add metadata to only project with statuses `Active` or `OnHold`.

GET /api/v1/project/meta/projects

This methods gets one or more projects that contain metadata with the given key and optional value.

Curl:

```
curl -X 'GET' \
  'https://api.approval.studio/api/v1/project/meta/projects?Key=orderNumber&Value
  -H 'accept: text/plain' \
  -H 'Authorization: Bearer YYYYYYYY'
```

Responses

HTTP Code	Response
200	**Success**. Search successful. Actually, zero or more projects found.

```
{
  "version": "1.0",
  "statusCode": 200,
  "message": "GET Request successful.",
  "result": [
    {
      "projectUID": "XXXXXXXXXXXXX",
      "projectState": "Active",
      "name": "Integratiion Design",
      "tags": [
        "Design"
      ],
      "reviewStatus": {
        "pendingCount": 5,
        "approvedCount": 4,
        "rejectedCount": 0
      },
      "created": "2022-10-17T16:30:14.971521",
      "owners": [
        "DDDDDDDDDDDDDD"
      ],
      "client": {
        "clientUID": "CCCCCCCCCCCCCC",
        "name": "Enterprize Company"
      },
      "metadata": [
        "orderNumber": "A19KQ64A",
        "contectEmail": "contact@mail.com"
      ]
    },
   [...]
  ]
}
```


HTTP Code	Response
400	**Error**: Parameters' validation failed.
	See HTTP code 400 description.

POST /api/v1/project/kanban/column

This methods **moves a given project to a given kanban column** on the dashboard.
Curl:

```
curl -X 'POST' \
  'https://api.approval.studio/api/v1/project/kanban/column' \
  -H 'accept: text/plain' \
  -H 'Content-Type: application/json-patch+json' \
  -d '{
  "projectUID": "XXXXXXXXXX", // ProjectUID as is throughout the system.
  "kanbanColumnUID": "asdFxQ" // A valid kanban column UID.
}'
```

Note: You may retrieve a list of Kanban column UIDs from the `GET /api/v1/users` method.

HTTP Code	Response
400	**Error**: Parameters' validation failed.
	See HTTP code 400 description.
404	**Error**: Project with the given UID not found.
404	**Error**: Kanban column with the given UID not available for this project.
	Please read kanban column UIDs using the `GET /api/v1/users` method
406	**Error**: Can't edit completed or archived projects.
	You may move a project to a given kanban column only when this project is active or on hold.

POST /api/v1/project/folder

This methods **moves a given project to a given folder** on the dashboard.
Curl:

```
curl -X 'POST' \
  'https://api.approval.studio/api/v1/project/folder' \
  -H 'accept: text/plain' \
  -H 'Content-Type: application/json-patch+json' \
  -d '{
  "projectUID": "XXXXXXXXXX", // ProjectUID as is throughout the system.
  "folderUID": "49lu"         // A valid folder UID.
}'
```

Note: You may retrieve a list of folder UIDs from the `GET /api/v1/users` method.

HTTP Code	Response
400	**Error**: Parameters' validation failed.
	See HTTP code 400 description.
404	**Error**: Project with the given UID not found.
404	**Error**: Kanban column with the given UID not available for this project.
	Please read kanban column UIDs using the `GET /api/v1/users` method
406	**Error**: Can't edit completed or archived projects.
	You may move a project to a given folder only when this project is active or on hold.

Asset management

Method/Path	Description
`GET /api/v1/assets`	Gets an asset instance.
`DELETE /api/v1/assets`	Deletes an asset.
`POST /api/v1/assets/upload`	Uploads an asset from file.
`POST /api/v1/assets/upload_url`	Uploads an asset from the Internet.
`GET /api/v1/assets/download`	Downloads an asset.
`GET /api/v1/assets/proofreport`	Gets a link to a proof report for an asset.

GET /api/v1/asset

Gets an asset instance by a given asset id.

Parameter	Type & Explanation
`AssetUID`	`string` Unique asset GUID.

Curl:

```
curl -X GET "http://api.approval.studio/api/v1/asset?AssetUID=XXXXXXX" -H "accept
```

Responses

HTTP Code	Response
200	**Success**. Asset instance returned.

```
  "version": "1.0",
  "statusCode": 200,
  "message": "GET Request successful.",
  "result": {
    "assetUID": "xxxxxxxxxx",
    "version": 1,
    "name": "drawing.pdf",
    "status": "Processed",
    "reviewStatus": {                                  // Proof review status (see
       "pendingCount": 1,                              // The asset has 1 uncomple
       "approvedCount": 2,                             // The asset has been appro
       "rejectedCount": 3                              // The asset has been rejec
    },
    "pagesCount": 3,
    "created": "2020-10-22T08:09:05.778512",
    "fileSize": 1063800,
    "reviewUrl": "https://app.approval.studio/xxx",    // URL to launch prooftool
    "thumbnailUrl": "https://app.approval.studio/yyy" // URL to asset's thumbnail
    "fullSizeBitmapUrl": "https://app.approval.studio/zzz/pageNum" // URL to conv
  }
}
```

Note: `fullSizeBitmapUrl` is provided in form `"https://baseurl/assettoken/0"` .
– "**0**" here is a zero-based page number, zero is used by default, but you can change it to whatever you need.
Please remember that some image asset formats are multipage (.pdf, .tiff, office formats etc.).
– You will get **http code 404** if provide wrong page number, bigger than asset's page count or if you provide negative value.

HTTP Code	Response
400	**Error**: Parameters' validation failed.
	See HTTP code 400 description.

HTTP Code	Response
404	**Error**: Asset with the given ID not found.

```
{
  "version": "1.0",
  "statusCode": 404,
  "message": "Asset not found."
}
```

DELETE /api/v1/asset

Deletes an asset by a given asset id.
Asset deletion is an unrecoverable operation that leads to removing the asset from the project and deleting a file from storage.
It is no way to restore an asset after it has been deleted.

Request

```
{
  "AssetUID": "XXXXXXXXXXXX"
}
```

Curl:

```
curl -X DELETE "https://api.approval.studio/api/v1/asset" \
     -H  "accept: text/plain" \
     -H  "Authorization: Bearer YYYYYYYYYY" \
     -H  "Content-Type: application/json-patch+json" \
     -d "{\"assetUID\":\"XXXXXXXXXXXX\"}"
```

Responses

HTTP Code	Response
200	**Success**. Asset deleted.

```
{
  "version": "1.0",
  "statusCode": 200,
```

```
  "message": "Asset deleted."
}
```

HTTP Code	Response
400	**Error**: Parameters' validation failed.
	See HTTP code 400 description.

HTTP Code	Response
404	**Error**: Asset with the given ID not found.

```
{
  "version": "1.0",
  "statusCode": 404,
  "message": "Asset not found."
}
```

POST /api/v1/asset/upload

Uploads an asset and **initiates asset processing**.
It validates initial input and **adds the asset to the asset processing queue**, which runs asynchronously. After uploading you **need to pool asset status** to get to know when it is processed or failed to process.
You can **upload assets directly** to a selected project or **point an upload task** as an upload initiator.

- Time required to process assets is highly dependent on asset file size, dimensions, number of pages, and on how much processing node(s) are loaded.
- Asset processing may fail or be rejected due to a number of reasons, business and technical. Those could be asset type (file extension), resolution or physical size (in case of vector images - PDF/AI, etc), payment plan, and a number of other options. Please refer to the company's site or/and support to learn more.

Request

Parameter	Type & Explanation
ProjectUID	`string` Unique project ID, **mandatory**.
FileName	`form file` Asset file name, **mandatory**. Note: It's just a name, not a path.

Curl:

```
curl -X POST "http://api.approval.studio/api/v1/asset/upload?ProjectUID=XXXXXXXXX
     -H "accept: text/plain"
     -H "Authorization: Bearer YYYYYYYYYYYYYY"
     -H "Content-Type: multipart/form-data"
     -F "uploadedFile=assetfilename.jpg;type=image/jpeg"
```

Responses

HTTP Code	Response
200	**Success**. "Asset uploaded successfully and is pending to process.

```
{
  "version": "1.0",
  "statusCode": 200,
  "message": "Asset uploaded successfully and is pending to process.
              Track its status to catch when it is ready to use.",
    "result": {
    "assetUID": "AAAAAAAAAAAAAAAA" // newly generated asset id.
  }
}
```

HTTP Code	Response
400	**Error**: Parameters' validation failed.
	See HTTP code 400 description.
403	**Error**: You have reached the storage limit for your plan (XXX Gb) and therefore not allowed to upload new assets.
	This code means you have no free storage space to upload a new asset; asset is not uploaded and will not be processed any further. Please buy more storage or contact support to assist in dealing with this.
404	**Error**: Project UID is either invalid or points to a non-existing or inactive project.
404	**Error**: Task UID is either invalid or points to a non-existing or inactive task.

```
{
  "version": "1.0",
  "statusCode": 404,
```

```
  "message": "Project UID provided is either invalid or points to a non-existing
             "Task UID provided is either invalid or points to a non-existing or
}
```

HTTP Code	Response
412	**Error**: Only task types **UploadAssets** and **UploadChangedAsset** are allowed.
412	**Error**: In case when it is **UploadChangedAsset** task: the file to upload must be the same type as the original file.
	Note: **UploadChangedAsset** means that the project owner(s) requested a new version of the asset to upload. It is mandatory that all asset versions must be the same type, i.e. all versions of the same asset are PDF or JPEG or PNG, etc. If you provide a different file type, you get this error.

```
{
  "version": "1.0",
  "statusCode": 412,
  "message": "Only task types UploadAssets and UploadChangedAsset are allowed." -
             "File to upload must be the same type as the original file (.pdf)."
}
```

POST /api/v1/asset/upload_url

This method works, generally, in the same way as `/api/v1/asset/upload` with the only difference that the file to upload is taken from the publicly accessible http(s) server.

A mandatory `URL` parameter points to the file to upload and process as a project's asset.

Request

Parameter	Type & Explanation
ProjectUID	`string` Unique project ID, **mandatory**.
FileName	`form file` Asset file name, **optional**. Note: It's just a name, not a path. If not provided, the filename would be assigned automatically using filename taken form http header `Content-Disposition` . If this header is not provided, filename would be `"asset.extension"` where extension is taken from http content type.

Parameter	Type & Explanation
URL	`string` URL to download a file from, **mandatory**.

Curl:

```
curl -X POST "http://api.approval.studio/api/v1/asset/upload_url"
     -H "accept: text/plain"
     -H "Authorization: Bearer YYYYYYYYYYYYYY"
     -H "Content-Type: application/json-patch+json"
     -d '{
     "projectUID": "XXXXXXXXX",
     "fileName": "filename.jpeg",
     "url": "https://cdn.images.com/ZZZZZZZZZZZ"
     }'
```

Responses

HTTP Code	Response
200	**Success**. "Asset uploaded successfully and is pending to process.


```
{
  "version": "1.0",
  "statusCode": 200,
  "message": "Asset uploaded successfully and is pending to process. Track it's s
  "result": {
    "assetUID": "XXXXXXXXX"
  }
}
```


HTTP Code	Response
400	**Error**: Parameters' validation failed.
	See HTTP code 400 description.
403	**Error**: You have reached the storage limit for your plan (XXX Gb) and therefore not allowed to upload new assets.
	This code means you have no free storage space to upload a new asset; asset is not uploaded and will not be processed any further. Please buy more storage or contact support to assist in dealing with this.

HTTP Code	Response
404	**Error**: Project UID is either invalid or points to a non-existing or inactive project.


```
{
  "version": "1.0",
  "statusCode": 404,
  "message": "Project UID provided is either invalid or points to a non-existing (
}
```


GET /api/v1/asset/download

Returns URL to **download asset**.

Request

Parameter	Type & Explanation
AssetUID	`string` Unique project ID, **mandatory**.

Curl:

```
curl -X GET "http://api.approval.studio/api/v1/asset/download?AssetUID=XXXXXXXXXX
     -H  "accept: text/plain"
     -H  "Authorization: Bearer YYYYYYYYYYYYYYYYYY"
```

Responses

HTTP Code	Response
200	**Success**. URL generated.


```
{
  "version": "1.0",
  "statusCode": 200,
  "message": "GET Request successful.",
  "result": {
    "downloadURL": "https://approval.studio/ProofApi/DownloadAsset/ZZZZZZZZZZZ[..
    "asset": {
```

```
      "assetUID": "XXXXXXXXXXXX",
      "version": 1,
      "name": "FileName.jpg",
      "status": "Processed",
      "pagesCount": 1,
      "created": "2020-12-03T23:54:53.40858",
      "fileSize": 20743
    }
  }
}
```


HTTP Code	Response
400	**Error**: Parameters' validation failed.
	See HTTP code 400 description.

HTTP Code	Response
404	**Error**: Asset with the given ID not found.


```
{
  "version": "1.0",
  "statusCode": 404,
  "message": "Asset not found."
}
```


GET /api/v1/asset/proofreport

Returns **URL** to download **asset's proof report**.

Request

Parameter	Type & Explanation
AssetUID	`string` Unique project ID, **mandatory**.

Curl:

```
curl -X GET "http://api.approval.studio/api/v1/asset/proofreport?AssetUID=XXXXXXX
     -H  "accept: text/plain"
     -H  "Authorization: Bearer YYYYYYYYYYYYYYYYYY"
```

Responses

HTTP Code	Response
200	**Success**. URL generated.


```
{
  "version": "1.0",
  "statusCode": 200,
  "message": "GET Request successful.",
  "result": {
    "downloadURL": "https://approval.studio/ProofApi/GetProofReport/ZZZZZZZZZZZ[.
    "asset": {
      "assetUID": "XXXXXXXXXXXXX",
      "version": 1,
      "name": "FileName.jpg",
      "status": "Processed",
      "pagesCount": 1,
      "created": "2020-12-03T23:54:53.40858",
      "fileSize": 20743
    }
  }
}
```


HTTP Code	Response
400	**Error**: Parameters' validation failed.
	See HTTP code 400 description.

HTTP Code	Response
404	**Error**: Asset with the given ID not found.


```
{
  "version": "1.0",
  "statusCode": 404,
  "message": "Asset not found."
}
```


Reference documents management

Method/Path	Description
`POST /api/v1/refdoc/upload`	Uploads a reference document and initiates document's processing.
`POST /api/v1/refdoc/upload_url`	Uploads an document taken from the publicly accessible URL and initiates its processing.
`POST /api/v1/refdoc/add_url`	Adds a URL as a reference document.
`GET /api/v1/refdoc/download`	Returns a URL to download the reference document.
`DELETE/api/v1/refdoc`	Deletes a reference document.

POST /api/v1/refdoc/upload

Uploads a document as a local file and assigns it to the given project.
generally, works like asset uploading,

Request

Parameter	Type & Explanation
ProjectUID	`string` Unique project ID, **mandatory**.
FileName	`form file` Document file name, **mandatory**. Note: It's just a name, not a path.

Curl:

```
curl -X POST "http://api.approval.studio/api/v1/refdoc/upload?ProjectUID=XXXXXXXX
     -H "accept: text/plain"
     -H "Authorization: Bearer YYYYYYYYYYYYYY"
     -H "Content-Type: multipart/form-data"
     -F "uploadedFile=refdocfilename.jpg;type=image/jpeg"
```

Responses

HTTP Code	Response
200	**Success**. "Asset uploaded successfully and is pending to process.

```
{
  "version": "1.0",
  "statusCode": 200,
  "message": "Document uploaded successfully and is pending to process.
             Track its status to catch when it is ready to use.",
    "result": {
    "refDocUID": "AAAAAAAAAAAAAAAA" // newly generated document id.
  }
}
```

HTTP Code	Response
400	**Error**: Parameters' validation failed.
	See HTTP code 400 description.

HTTP Code	Response
404	**Error**: Project UID is either invalid or points to a non-existing or inactive project.

```
{
  "version": "1.0",
  "statusCode": 404,
  "message": "Project UID provided is either invalid or points to a non-existing
}
```

POST /api/v1/refdoc/upload_url

This method uploads reference document by taking it from the publicly accessible http(s) server.

A mandatory `URL` parameter points to the file to upload and process as a project's asset.

Request

Parameter	Type & Explanation
ProjectUID	`string` Unique project ID, **mandatory**.
FileName	`form file` Document file name, **optional**. Note: It's just a name, not a path. If not provided, the filename would be assigned automatically using filename taken form http header `Content-Disposition` . If this header is not

Parameter	Type & Explanation
	provided, filename would be `"asset.extension"` where extension is taken from http content type.
URL	`string` URL to download a file from.

Curl:

```
curl -X POST "http://api.approval.studio/api/v1/refdoc/upload_url"
     -H "accept: text/plain"
     -H "Authorization: Bearer YYYYYYYYYYYYYY"
     -H "Content-Type: application/json-patch+json"
     -d '{
     "projectUID": "XXXXXXXXX",
     "fileName": "filename.jpeg",
     "url": "https://cdn.images.com/ZZZZZZZZZZZ"
     }'
```

Responses

HTTP Code	Response
200	**Success**. "Document uploaded successfully and is pending to process.

```
{
  "version": "1.0",
  "statusCode": 200,
  "message": "Document uploaded successfully and is pending to process. Track it'
  "result": {
    "refDocUID": "XXXXXXXXX"
  }
}
```

HTTP Code	Response
400	**Error**: Parameters' validation failed.
	See HTTP code 400 description.

HTTP Code	Response
404	**Error**: Project UID is either invalid or points to a non-existing or inactive project.

```
{
  "version": "1.0",
  "statusCode": 404,
  "message": "Project UID provided is either invalid or points to a non-existing
}
```


POST /api/v1/refdoc/add_url

Plain text URL is a kind of a reference document in the Approval Studio. You can either upload a file or add a URL to the list of reference documents in a project.

This method adds a URL to the document list for a given project.

Curl:

```
curl -X POST "http://api.approval.studio/api/v1/refdoc/add_url"
     -H "accept: text/plain"
     -H "Authorization: Bearer YYYYYYYYYYYYYY"
     -H "Content-Type: application/json-patch+json"
     -d '{
     "projectUID": "XXXXXXXXX",
     "url": "https://cdn.images.com/ZZZZZZZZZZZ"
     }'
```

Responses

HTTP Code	Response
200	**Success**. "Document uploaded successfully and is pending to process.


```
{
  "version": "1.0",
  "statusCode": 200,
  "message": "Reference document's URL added to the list.",
  "result": {
    "refDocUID": "XXXXXXXXX"
  }
}
```


HTTP Code	Response
400	**Error**: Parameters' validation failed.

HTTP Code	Response
	See HTTP code 400 description.

HTTP Code	Response
404	**Error**: Project UID is either invalid or points to a non-existing or inactive project.


```
{
  "version": "1.0",
  "statusCode": 404,
  "message": "Project UID provided is either invalid or points to a non-existing (
}
```


GET /api/v1/refdoc/download

This method returns URL to download a given reference document.

If you need to download the document itself, please use any available suitable technology to do this using provided URL.

Curl:

```
curl -X POST "http://api.approval.studio/api/v1/refdoc/download?refDocUID=XXXXXXX
     -H "accept: text/plain"
     -H "Authorization: Bearer YYYYYYYYYYYYYY"
```

Responses

HTTP Code	Response
200	**Success**. URL returned in the response JSON.


```
{
  "version": "1.0",
  "statusCode": 200,
  "message": "GET Request successful.",
  "result": {
    "downloadURL": "https://ZZZZZZZZZZ"
```

```
  }
}
```

HTTP Code	Response
400	**Error**: Parameters' validation failed.
	See HTTP code 400 description.

HTTP Code	Response
404	**Error**: Reference document UID is either invalid or points to a non-existing or deleted document.

```
{
  "version": "1.0",
  "statusCode": 404,
  "message": "Reference document not found."
}
```

DELETE/api/v1/refdoc

This method deletes a reference document or reference URL.

Curl:

```
curl -X DELETE "http://api.approval.studio/api/v1/refdoc"
     -H "accept: text/plain"
     -H "Authorization: Bearer YYYYYYYYYYYYYY"
```

Responses

HTTP Code	Response
200	**Success**. URL returned in the response JSON.

```
{
  "version": "1.0",
  "statusCode": 200,
  "message": "Document deleted."
}
```

HTTP Code	Response
400	**Error**: Parameters' validation failed.
	See HTTP code 400 description.

HTTP Code	Response
404	**Error**: Reference document UID is either invalid or points to a non-existing or deleted document.

```
{
  "version": "1.0",
  "statusCode": 404,
  "message": "Reference document not found or it was already deleted."
}
```

DAL management

The **Digital Asset Library (DAL)** is a centralized repository for storing, organizing, and reusing assets across projects within Approval Studio. It supports a hierarchical folder structure, asset metadata, history tracking, and operations like moving, copying, and deleting assets/folders.

DAL Concepts

- **Folders**: Hierarchical containers (similar to project folders but global per client/tenant).
- **Assets**: Assets that have been explicitly added to the library (with history, tags, description, etc.).
- **Operations**: CRUD for folders and assets, plus copy/move between DAL and projects.

Assets in DAL retain full history and can be copied back into active projects.

DAL data model

Folder

The folder hierarchy is a tree. Every folder may contain nested sub-folders, returned recursively.

Field	Explanation
`folderUID`	System-wide unique folder identifier.
`name`	Folder display name.
`subFolders`	Array of child `DALFolder` objects (same structure, nested recursively).

Asset

Field	Explanation
`assetUID`	System-wide unique DAL asset identifier.
`name`	Asset name.
`description`	Free-form asset description.
`originProjectUID`	UID of the project the asset was originally copied from, if any.
`tags`	Array of free-form tag strings associated with the asset.
`addedDate`	UTC date-time the asset was added to the library.
`history`	Array of history entries describing the asset's lifecycle (see below).

Asset history entry

Field	Explanation
`created`	UTC date-time the event occurred.
`description`	Human-readable description of the event.

Endpoints summary

Method	Description
`GET /api/v1/dal/folders`	Returns the DAL folder hierarchy.

Method	Description
`POST /api/v1/dal/folder`	Creates a new folder under a given parent folder.
`PUT /api/v1/dal/folder`	Renames an existing folder.
`PATCH /api/v1/dal/folder`	Moves a folder under another parent folder.
`DELETE /api/v1/dal/folder`	Deletes a folder.
`GET /api/v1/dal/asset`	Returns a single DAL asset by UID.
`GET /api/v1/dal/folder/assets`	Returns the assets contained in a given folder.
`PUT /api/v1/dal/asset`	Moves a DAL asset to a given folder.
`DELETE /api/v1/dal/asset`	Deletes a DAL asset.
`POST /api/v1/dal/asset/project`	Copies a DAL asset into a project.
`POST /api/v1/dal/asset/folder`	Copies a project's asset into a DAL folder.

GET /api/v1/dal/folders

Returns the DAL folders' hierarchy for the caller's client as a single root folder with its sub-folders nested recursively.

This method takes **no parameters**; the client is resolved from the authenticated user.

Curl:

```
curl -X GET "https://api.approval.studio/api/v1/dal/folders" \
     -H "accept: text/plain" \
     -H "Authorization: Bearer YYYYYYYYYYYYY"
```

Responses

HTTP Code	Response
200	**Success**. The folder hierarchy is returned in `result.folders` (a single root folder with nested `subFolders`).

```
{
  "version": "1.0",
  "statusCode": 200,
  "message": null,
  "isError": false,
  "result": {
    "folders": [
      {
        "folderUID": "XXXXXXXX",          // Root folder.
        "name": "Library",
        "subFolders": [
          {
            "folderUID": "YYYYYYYY",
            "name": "Logos",
            "subFolders": []
          },
          {
            "folderUID": "ZZZZZZZZ",
            "name": "Packaging",
            "subFolders": [
              {
                "folderUID": "QQQQQQQQ",
                "name": "2025",
                "subFolders": []
              }
            ]
          }
        ]
      }
    ]
  }
}
```

HTTP Code	Response
500	**Error**: Retrieving folders failed — no folders could be obtained for the client.

```
{
  "version": "1.0",
  "statusCode": 500,
  "message": "Retrieving folders failed."
}
```

POST /api/v1/dal/folder

Creates a new folder under an existing parent folder.

Request

Field	Type & Explanation
`folderUID`	`string` Mandatory UID of the **parent** folder under which the new folder is created. Must exist in the client's folder hierarchy.
`name`	`string` Mandatory name for the new folder. Must be unique among siblings.

```
{
  "folderUID": "XXXXXXXX",   // Parent folder UID.
  "name": "New folder name"
}
```

Curl:

```
curl -X POST "https://api.approval.studio/api/v1/dal/folder" \
     -H "accept: text/plain" \
     -H "Authorization: Bearer YYYYYYYYYYYYY" \
     -H "Content-Type: application/json" \
     -d "{\"folderUID\":\"XXXXXXXX\",\"name\":\"New folder name\"}"
```

Responses

HTTP Code	Response
200	**Success**. Folder created.

```
{
  "version": "1.0",
  "statusCode": 200,
  "message": "Folder successfully created."
}
```

HTTP Code	Response
400	**Error**: Parameters' validation failed. See HTTP code 400 description.
404	**Error**: Client not found for the authenticated user.

HTTP Code	Response
	Error: Parent folder not found — the client has no folders, or the given `folderUID` does not exist in the hierarchy.
	Error: A folder with the given name already exists under the parent folder.

```
{
  "version": "1.0",
  "statusCode": 404,
  "message": "Folder with the given name already exist."
}
```

HTTP Code	Response
500	**Error**: Folder creation failed.

PUT /api/v1/dal/folder

Renames an existing folder.

Request

Field	Type & Explanation
`folderUID`	`string` Mandatory UID of the folder to rename. Must exist in the client's folder hierarchy.
`name`	`string` Mandatory new folder name.

```
{
  "folderUID": "XXXXXXXX",
  "name": "Renamed folder"
}
```

Curl:

```
curl -X PUT "https://api.approval.studio/api/v1/dal/folder" \
     -H "accept: text/plain" \
     -H "Authorization: Bearer YYYYYYYYYYYY" \
```

```
    -H "Content-Type: application/json" \
    -d "{\"folderUID\":\"XXXXXXXX\",\"name\":\"Renamed folder\"}"
```

Responses

HTTP Code	Response
200	**Success**. Folder renamed.

```
{
  "version": "1.0",
  "statusCode": 200,
  "message": "Folder successfully renamed."
}
```

HTTP Code	Response
400	**Error**: Parameters' validation failed. See HTTP code 400 description.
404	**Error**: Folder with the given `folderUID` not found.
500	**Error**: Folder renaming failed.

PATCH /api/v1/dal/folder

Moves an existing folder so it becomes a child of another (target) folder.

Request

Field	Type & Explanation
`folderUID`	`string` Mandatory UID of the folder to move (the source folder). Must exist in the hierarchy.
`parentFolderUID`	`string` Mandatory UID of the target parent folder the source folder is moved under. Must exist in the hierarchy.

```
{
  "folderUID": "XXXXXXXX",        // Folder to move.
  "parentFolderUID": "YYYYYYYY"   // New parent folder.
}
```

Curl:

```
curl -X PATCH "https://api.approval.studio/api/v1/dal/folder" \
     -H "accept: text/plain" \
     -H "Authorization: Bearer YYYYYYYYYYYYY" \
     -H "Content-Type: application/json" \
     -d "{\"folderUID\":\"XXXXXXXX\",\"parentFolderUID\":\"YYYYYYYY\"}"
```

Responses

HTTP Code	Response
200	**Success**. Folder moved.

```
{
  "version": "1.0",
  "statusCode": 200,
  "message": "Folder successfully moved."
}
```

HTTP Code	Response
400	**Error**: Parameters' validation failed. See HTTP code 400 description.
404	**Error**: Source folder (`folderUID`) not found.
	Error: Target folder (`parentFolderUID`) not found.
500	**Error**: Folder moving failed.

DELETE /api/v1/dal/folder

Deletes an existing folder.

Request

Field	Type & Explanation
`folderUID`	`string` Mandatory UID of the folder to delete. Must exist in the client's folder hierarchy.

```
{
  "folderUID": "XXXXXXXX"
}
```

Curl:

```
curl -X DELETE "https://api.approval.studio/api/v1/dal/folder" \
     -H "accept: text/plain" \
     -H "Authorization: Bearer YYYYYYYYYYYYY" \
     -H "Content-Type: application/json" \
     -d "{\"folderUID\":\"XXXXXXXX\"}"
```

Responses

HTTP Code	Response
200	**Success**. Folder deleted.

```
{
  "version": "1.0",
  "statusCode": 200,
  "message": "Folder successfully deleted."
}
```

HTTP Code	Response
400	**Error**: Parameters' validation failed. See HTTP code 400 description.
404	**Error**: Folder with the given `folderUID` not found.
500	**Error**: Folder deletion failed.

GET /api/v1/dal/asset

Returns a single DAL asset by its UID.

Parameter	Type & Explanation
`AssetUID`	`string` Mandatory UID of the DAL asset to retrieve.

Request URL

```
https://api.approval.studio/api/v1/dal/asset?AssetUID=XXXXXXXX
```

Curl:

```
curl -X GET "https://api.approval.studio/api/v1/dal/asset?AssetUID=XXXXXXXX" \
     -H "accept: text/plain" \
     -H "Authorization: Bearer YYYYYYYYYYYYY"
```

Responses

HTTP Code	Response
200	**Success**. The asset is returned in `result.asset`.

```
{
  "version": "1.0",
  "statusCode": 200,
  "message": null,
  "isError": false,
  "result": {
    "asset": {
      "assetUID": "XXXXXXXX",
      "name": "front-label.pdf",
      "description": "string",
      "originProjectUID": "PPPPPPPP",       // Project the asset was copied from,
      "tags": [
        "label", "2025"
      ],
      "addedDate": "2026-01-15T10:24:00.000Z",
      "history": [
        {
          "created": "2026-01-15T10:24:00.000Z",
          "description": "Asset added to the library."
        }
      ]
    }
  }
}
```

HTTP Code	Response
400	**Error**: Parameters' validation failed. See HTTP code 400 description.
404	**Error**: Asset with the given `AssetUID` not found.

GET /api/v1/dal/folder/assets

Returns the assets contained in a given folder.

Parameter	Type & Explanation
`FolderUID`	`string` Mandatory UID of the folder whose assets are requested. Must exist in the client's folder hierarchy.

Request URL

```
https://api.approval.studio/api/v1/dal/folder/assets?FolderUID=XXXXXXXX
```

Curl:

```
curl -X GET "https://api.approval.studio/api/v1/dal/folder/assets?FolderUID=XXXXX
     -H "accept: text/plain" \
     -H "Authorization: Bearer YYYYYYYYYYYYY"
```

Responses

HTTP Code	Response
200	**Success**. The folder's assets are returned in `result.assets`.


```
{
  "version": "1.0",
  "statusCode": 200,
  "message": null,
  "isError": false,
  "result": {
    "assets": [
      {
        "assetUID": "XXXXXXXX",
        "name": "front-label.pdf",
        "description": "string",
        "originProjectUID": "PPPPPPPP",
        "tags": [ "label" ],
        "addedDate": "2026-01-15T10:24:00.000Z",
        "history": [
          {
            "created": "2026-01-15T10:24:00.000Z",
            "description": "Asset added to the library."
```

```
          }
        ]
      }
    ]
  }
}
```


HTTP Code	Response
400	**Error**: Parameters' validation failed (see HTTP code 400 description), or assets could not be retrieved for the folder.


```
{
  "version": "1.0",
  "statusCode": 400,
  "message": "Assets getting failed."
}
```


HTTP Code	Response
404	**Error**: Folder with the given `FolderUID` not found.

PUT /api/v1/dal/asset

Moves a DAL asset to a given folder.

Request

Field	Type & Explanation
`assetUID`	`string` Mandatory UID of the DAL asset to move.
`folderUID`	`string` Mandatory UID of the destination folder. Must exist in the client's folder hierarchy.


```
{
  "assetUID": "XXXXXXXX",
  "folderUID": "YYYYYYYY"
}
```


Curl:

```
curl -X PUT "https://api.approval.studio/api/v1/dal/asset" \
     -H "accept: text/plain" \
     -H "Authorization: Bearer YYYYYYYYYYYYY" \
     -H "Content-Type: application/json" \
     -d "{\"assetUID\":\"XXXXXXXX\",\"folderUID\":\"YYYYYYYY\"}"
```

Responses

HTTP Code	Response
200	**Success**. Asset moved.

```
{
  "version": "1.0",
  "statusCode": 200,
  "message": "Asset moved successfully."
}
```

HTTP Code	Response
400	**Error**: Parameters' validation failed (see HTTP code 400 description), or the move operation failed.

```
{
  "version": "1.0",
  "statusCode": 400,
  "message": "Assets moving failed."
}
```

HTTP Code	Response
404	**Error**: Destination folder (`folderUID`) not found.
	Error: Asset (`assetUID`) not found.

DELETE /api/v1/dal/asset

Deletes a DAL asset.

Request

Field	Type & Explanation
`assetUID`	`string` Mandatory UID of the DAL asset to delete.

```
{
  "assetUID": "XXXXXXXX"
}
```

Curl:

```
curl -X DELETE "https://api.approval.studio/api/v1/dal/asset" \
     -H "accept: text/plain" \
     -H "Authorization: Bearer YYYYYYYYYYYYY" \
     -H "Content-Type: application/json" \
     -d "{\"assetUID\":\"XXXXXXXX\"}"
```

Responses

HTTP Code	Response
200	**Success**. Asset deleted.

```
{
  "version": "1.0",
  "statusCode": 200,
  "message": "Asset deleted successfully."
}
```

HTTP Code	Response
400	**Error**: Parameters' validation failed. See HTTP code 400 description.
404	**Error**: Asset with the given `assetUID` not found.
500	**Error**: Asset deletion failed.

POST /api/v1/dal/asset/project

Copies a DAL asset into a project. A copy of the library asset is placed into the target project; the original DAL asset is left untouched.

Request

Field	Type & Explanation
`assetUID`	`string` Mandatory UID of the DAL asset to copy.
`projectUID`	`string` Mandatory UID of the destination project.
`name`	`string` Optional name to assign to the copied asset in the project.

```
{
  "assetUID": "XXXXXXXX",
  "projectUID": "PPPPPPPP",
  "name": "front-label.pdf"
}
```

Curl:

```
curl -X POST "https://api.approval.studio/api/v1/dal/asset/project" \
     -H "accept: text/plain" \
     -H "Authorization: Bearer YYYYYYYYYYYYY" \
     -H "Content-Type: application/json" \
     -d "{\"assetUID\":\"XXXXXXXX\",\"projectUID\":\"PPPPPPPP\",\"name\":\"front-
```

Responses

HTTP Code	Response
200	**Success**. Asset copied to the project.

```
{
  "version": "1.0",
  "statusCode": 200,
  "message": "Asset copied to project successfully."
}
```

HTTP Code	Response
400	**Error**: Parameters' validation failed (see HTTP code 400 description), or the copy operation failed.

```
{
  "version": "1.0",
  "statusCode": 400,
```

```
  "message": "Copying the asset to the project failed."
}
```

HTTP Code	Response
404	**Error**: DAL asset (`assetUID`) not found.
	Error: Project (`projectUID`) not found.

POST /api/v1/dal/asset/folder

Copies a project's asset into a DAL folder. A copy of the project asset is placed into the target library folder; the original project asset is left untouched.

Request

Field	Type & Explanation
`assetUID`	`string` Mandatory UID of the **project** asset to copy into the library.
`folderUID`	`string` Mandatory UID of the destination DAL folder. Must exist in the client's folder hierarchy.

```
{
  "assetUID": "XXXXXXXX",
  "folderUID": "YYYYYYYY"
}
```

Curl:

```
curl -X POST "https://api.approval.studio/api/v1/dal/asset/folder" \
     -H "accept: text/plain" \
     -H "Authorization: Bearer YYYYYYYYYYYYY" \
     -H "Content-Type: application/json" \
     -d "{\"assetUID\":\"XXXXXXXX\",\"folderUID\":\"YYYYYYYY\"}"
```

Responses

HTTP Code	Response
200	**Success**. Asset copied to the folder.

```
{
  "version": "1.0",
  "statusCode": 200,
  "message": "Asset copied to folder successfully."
}
```

HTTP Code	Response
400	**Error**: Parameters' validation failed (see HTTP code 400 description), or the copy operation failed.
404	**Error**: Project asset (`assetUID`) not found.
	Error: DAL folder (`folderUID`) not found.

Task management

Method/Path	Description
`GET /api/v1/task/all`	Gets a list of tasks assigned to the user.
`GET /api/v1/task`	Gets a task instance.
`DELETE /api/v1/task`	Deletes an task.
`POST /api/v1/task/asset_upload`	Creates a new AssetUpload task.
`POST /api/v1/task/asset_change_upload`	Creates a new AssetChangeUpload task.
`POST /api/v1/task/refdoc_upload`	Creates a new RefDocUpload task.
`POST /api/v1/task/review_asset`	Creates a new CreateReviewAsset task.
`PUT /api/v1/task/complete`	Completes a task.

GET /api/v1/task/all

Gets a list of tasks optionally filtered by task types, assigned to a currently logged-in user. This is what a user sees in Approval Studio web application in the list of tasks in the dashboard.

Parameter	Type & Explanation
`Types`	`string` One or more task types, comma-separated.
	Default value is `UploadAssets, UploadRefDocs, ReviewAssets, ExternalReviewAssets, UploadChangedAsset, UploadVideo, ExternalReviewVideo, ReviewVideo , AssignUserRoles .`
	If no parameter is provided or it is empty, a default list of types is used (see above).

Curl:

```
curl -X GET "http://api.approval.studio/api/v1/task/all?Types=UploadAssets"
     -H "accept: text/plain"
     -H  "Authorization: Bearer YYYYYYYYYY"
```

Responses

HTTP Code	Response
200	**Success**. Task instance returned.

```
{
  "version": "1.0",
  "statusCode": 200,
  "message": "GET Request successful.",
  "result": [
    {
        "taskUID": "XXXXXXXXXXX1",
        "projectUID": "YYYYYYYYYYYY1",
        "projectName": "Project name, string",
        "type": "UploadAssets|UploadRefDocs|ReviewAssets|ExternalReviewAssets|Upl
        "status": "Pending|Closed|Approved|Rejected|ApprovedWithChanges",
        "created": "2020-11-27T15:39:59.864882",
        "closed": "2020-11-27T15:40:51.407384",
        "user": {
          "userUID": "92AFE33153124F0980E43EF80133FE9B",
          "fullName": "John Smith",
          "email": "john.smith@email.com"
        },
        "reviewUrl": string ,                          // URL to launch prooftool f
                                                       // Appears only in ReviewAss
        "requestedAssetName": "butterfly_poster.pdf" // Name of the asset, a task
                                                       // Appears only in UploadCha
```

```
    },
    {
        "taskUID": "XXXXXXXXXXX2",
        "projectUID": "YYYYYYYYYYYY2",
        ...
    },
    ...
  ]
}
```

HTTP Code	Response
400	**Error**: Parameters' validation failed.
	See HTTP code 400 description.

GET /api/v1/task

Gets a task instance for a given task id.

Parameter	Type & Explanation
`TaskUID`	`string` Unique task ID.

Curl:

```
curl -X GET "http://api.approval.studio/api/v1/task?TaskUID=XXXXXXXXXXXXXXXX"
     -H "accept: text/plain"
     -H  "Authorization: Bearer YYYYYYYYYY"
```

Responses

HTTP Code	Response
200	**Success**. Task instance returned.

```
{
"version": "1.0",
  "statusCode": 200,
  "message": "GET Request successful.",
  "result": {
    "taskUID": "XXXXXXXXXX",
```

```
    "projectUID": "YYYYYYYYYYYY",
    "projectName": "string",
    "clientUID": "ZZZZZZZZZZZZ",
    "type": "UploadAssets|UploadRefDocs|ReviewAssets|ExternalReviewAssets|UploadCl
    "status": "Pending|Closed|Approved|Rejected|ApprovedWithChanges",
    "created": "2020-11-27T15:39:59.864882",
    "closed": "2020-11-27T15:40:51.407384",
    "user": {
      "userUID": "92AFE33153124F0980E43EF80133FE9B",
      "fullName": "John Smith",
      "email": "john.smith@email.com"
    },
    "reviewUrl": string,                // URL to launch prooftool for given tasl
                                        // Appears only for ReviewAsset task.
    "requestedAssetName": "filename.pdf" // Name of the asset, a task references
                                        // Appears only in UploadChangedAsset ta
  }
}
```

HTTP Code	Response
400	**Error**: Parameters' validation failed.
	See HTTP code 400 description.

HTTP Code	Response
404	**Error**: Task with the given ID not found.

```
{
  "version": "1.0",
  "statusCode": 404,
  "message": "Task not found"
}
```

DELETE /api/v1/task

Deletes a task.

Request

Parameter	Type & Explanation
taskUID	`string` Unique task ID.

Curl:

```
curl -X DELETE "https://api.approval.studio/api/v1/task"
-H  "accept: text/plain"
-H  "Authorization: Bearer YYYYYYYYYYYYYYYYYYYYYYYYYY"
-H "Content-Type: application/json-patch+json"
-d "{\"taskUID\":\"XXXXXXXXXXX\"}"
```

Responses

HTTP Code	Response
200	**Success**. Login successful, AUTH token provided.

```
{
  "version": "1.0",
  "statusCode": 200,
  "message": "Task deleted."
}
```

HTTP Code	Response
400	**Error**: Parameters' validation failed.
	See HTTP code 400 description.

HTTP Code	Response
404	**Error**: Task with the given ID not found.

```
{
  "version": "1.0",
  "statusCode": 404,
  "message": "Task not found"
}
```

HTTP Code	Response
410	**Error**: the task is already completed or deleted or approved/rejected. Only pending task can be deleted.

```
{
  "version": "1.0",
  "statusCode": 410,
  "message": "The task is not pending and cannot be deleted."
}
```

POST /api/v1/task/asset_upload

Creates a new **AssetUpload task** for a given user providing optional due date and comment.

Request

```
{
  "projectUID": "XXXXXXXXXXXXXX", // Project this task belongs to, mandatory.
  "userUID": "ZZZZZZZZZZZZZ",     // User this task assigned to, mandatory.
  "dueDate": "2020-12-04",        // Task due date, UTC, optional.
  "comment": "comment text"       // Comment text, optional.
}
```

Curl:

```
curl -X POST "https://api.approval.studio/api/v1/task/asset_upload" \
     -H "accept: text/plain" \
     -H "Authorization: Bearer YYYYYYYYYYYYYYYY" \
     -H "Content-Type: application/json-patch+json" \
     -d "{\"projectUID\":\"XXXXXXXXXXXXXXXX\",
          \"userUID\":\"ZZZZZZZZZZZZZ\",
          \"dueDate\":\"2020-12-04T13:11:11.526Z\",
          \"comment\":\"comment text\"}"
```

Responses

HTTP Code	Response
200	**Success**. Task created.

```
{
  "version": "1.0",
  "statusCode": 200,
  "message": "Task created."
  "result": {
    "task": {
      "taskUID": "XXXXXXXXXXXXXXXXXXXXX",
      "projectId": 981,
      "projectUID": "YYYYYYYYYYYYYYYYYYYYYYYYYYYY",
      "type": "UploadAssets",
      "status": "Pending",
      "comment": "Free-form text comment to the task",
      "dueDate": "2020-12-22",
      "created": "2020-12-22",
      "userUID": "ZZZZZZZZZZZZZ",
      "userName": "John Smith",
      "userEmail": "john.smith@gmail.com"
    }
  }
}
```


HTTP Code	Response
400	**Error**: Parameters' validation failed.
	See HTTP code 400 description.

|

HTTP Code	Response
404	**Error**: Given project not found.


```
{
  "version": "1.0",
  "statusCode": 404,
  "message": "Given project not found."
}
```


HTTP Code	Response
412	**Error**: You cannot create a task for a project in state [Complete]. Only projects that are `Active` or `OnHold` can have tasks.
412	**Error**: User with UID [ZZZZZZZZZZZZZ] not found.

```
{
  "version": "1.0",
  "statusCode": 412,
  "message": "You cannot create a task for a project in state [Complete]."
}
```

POST /api/v1/task/asset_change_upload

Creates a new **AssetChangeUpload task** for a given asset providing optional due date and comment.

This task is providing **a new version of a given asset**, and this asset, when uploaded, would be the next sequential version of this asset, having the same name and incremented version number.

For instance, you have this in a project:

> - picture.pdf (v1)
> - picture.pdf (v2) <-- current version.

Now you create this task, and when it done you will have:

> - picture.pdf (v1)
> - picture.pdf (v2)
> - picture.pdf (v3) <-- new current version.

Request

```
{
  "projectUID": "XXXXXXXXXXXXXX", // Project this task belongs to, mandatory.
  "userUID": "ZZZZZZZZZZZZZ",     // User this task assigned to, mandatory.
  "assetUID": "YYYYYYYYYYYY",     // Asset this task refers to.
  "dueDate": "2020-12-04",        // Task due date, UTC, optional.
  "comment": "comment text"       // Comment text, optional.
}
```

Curl:

```
curl -X POST "https://api.approval.studio/api/v1/task/asset_upload" \
     -H "accept: text/plain" \
     -H "Authorization: Bearer YYYYYYYYYYYYYYYY" \
     -H "Content-Type: application/json-patch+json" \
     -d "{\"projectUID\":\"XXXXXXXXXXXXXXXX\",
```

```
\"userUID\":\"ZZZZZZZZZZZZZ\",
\"assetUID\":\"YYYYYYYYYYYY\",
\"dueDate\":\"2020-12-04T13:11:11.526Z\",
\"comment\":\"comment text\"}"
```

Responses

HTTP Code	Response
200	**Success**. Task created.

```
{
  "version": "1.0",
  "statusCode": 200,
  "message": "Task created."
  "result": {
    "task": {
      "taskUID": "XXXXXXXXXXXXXXXXXXXXXX",
      "projectId": 981,
      "projectUID": "YYYYYYYYYYYYYYYYYYYYYYYYYYYY",
      "type": "UploadAssets",
      "status": "Pending",
      "comment": "Free-form text comment to the task",
      "dueDate": "2020-12-22",
      "created": "2020-12-22",
      "userUID": "ZZZZZZZZZZZZZ",
      "userName": "John Smith",
      "userEmail": "john.smith@gmail.com"
    }
  }
}
```

HTTP Code	Response
400	**Error**: Parameters' validation failed.
	See HTTP code 400 description.

|

HTTP Code	Response
404	**Error**: Given project not found.

```
{
  "version": "1.0",
  "statusCode": 404,
  "message": "Given project not found."
}
```

HTTP Code	Response
412	**Error**: Asset with UID [YYYYYYYYYYY] not found in the given project. You need to provide a valid asset UID that belongs to the given project.
412	**Error**: User with UID [ZZZZZZZZZZZZZ] not found.
412	**Error**: You cannot create a task for a project in state [Complete]. Only projects that are `Active` or `OnHold` can have tasks.

```
{
  "version": "1.0",
  "statusCode": 412,
  "message": "You cannot create a task for a project in state [Complete]."
}
```

POST /api/v1/task/refdoc_upload

Creates a new **RefDocUpload task** for a given user providing optional due date and comment.

Request

```
{
  "projectUID": "XXXXXXXXXXXXXX", // Project this task belongs to, mandatory.
  "userUID": "ZZZZZZZZZZZZZ",     // User this task assign to, mandatory.
  "dueDate": "2020-12-04",        // Task due date, UTC, optional.
  "comment": "comment text"       // Comment text, optional.
}
```

Curl:

```
curl -X POST "https://api.approval.studio/api/v1/task/refdoc_upload" \
     -H "accept: text/plain" \
     -H "Authorization: Bearer YYYYYYYYYYYYYYYY" \
     -H "Content-Type: application/json-patch+json" \
     -d "{\"projectUID\":\"XXXXXXXXXXXXXXXX\",
```

```
\"userUID\":\"ZZZZZZZZZZZZZ\",
\"dueDate\":\"2020-12-04\",
\"comment\":\"comment text\" [...] }"
```

Responses

HTTP Code	Response
200	**Success**. Task created.

```
{
  "version": "1.0",
  "statusCode": 200,
  "message": "Task created."
}
```

HTTP Code	Response
400	**Error**: Parameters' validation failed.
	See HTTP code 400 description.

HTTP Code	Response
404	**Error**: Given project not found.

```
{
  "version": "1.0",
  "statusCode": 404,
  "message": "Given project not found."
  "result": {
    "task": {
      "taskUID": "XXXXXXXXXXXXXXXXXXXXX",
      "projectId": 981,
      "projectUID": "YYYYYYYYYYYYYYYYYYYYYYYYYYYY",
      "type": "UploadRefDocs",
      "status": "Pending",
      "comment": "Free-form text comment to the task",
      "dueDate": "2020-12-22",
      "created": "2020-12-22",
      "userUID": "ZZZZZZZZZZZZZ",
      "userName": "John Smith",
      "userEmail": "john.smith@gmail.com"
    }
  }
}
```

HTTP Code	Response
412	**Error**: You cannot create a task for a project in the state `Complete`. Only projects that are `Active` or `OnHold` can have tasks.
412	**Error**: User with UID [ZZZZZZZZZZZZZ] not found.

```
{
  "version": "1.0",
  "statusCode": 412,
  "message": "You cannot create a task for a project in the state [Complete]."
}
```

POST /api/v1/task/review_asset

Creates a new Review Asset Task for a given user, project, and a list of assets providing optional due date and comment.

Request

```
{
  "projectUID": "XXXXXXXXXXXXXX", // Project this task belongs to, mandatory.
  "userUID": "ZZZZZZZZZZZZZ",     // User this task assign to, mandatory.
  "assetUIDs": [
    "AAAAAAAAAAAAAAA1",           // One or more assets to review, namdatory.
    "AAAAAAAAAAAAAAA2"
  ],
  "dueDate": "2020-12-04",        // Task due date, UTC, optional.
  "comment": "comment text"       // Comment text, optional.
}
```

Curl:

```
curl -X POST "https://api.approval.studio/api/v1/task/review_asset" \
     -H "accept: text/plain" \
     -H "Authorization: Bearer YYYYYYYYYYYYYYYY" \
     -H "Content-Type: application/json-patch+json" \
     -d "{\"projectUID\":\"XXXXXXXXXXXXXXX\",
          \"userUID\":\"ZZZZZZZZZZZZZ\",
          \"assetUIDs\":[\"AAAAAAAAA1\", \"AAAAAAAAA2\"],
          \"dueDate\":\"2020-12-04\",
          \"comment\":\"comment text\" [...]}"
```

Responses

HTTP Code	Response
200	**Success**. Task created.


```
{
  "version": "1.0",
  "statusCode": 200,
  "message": "Task created.",
  "result": {
    "task": {
      "taskUID": "XXXXXXXXXXXXXXXXXXXXX",
      "projectId": 981,
      "projectUID": "YYYYYYYYYYYYYYYYYYYYYYYYYYYY",
      "type": "ReviewAssets",
      "status": "Pending",
      "comment": "Free-form text comment to the task",
      "dueDate": "2020-12-22",
      "created": "2020-12-22",
      "userUID": "ZZZZZZZZZZZZZ",
      "userName": "John Smith",
      "userEmail": "john.smith@gmail.com"
    }
  }
}
```


HTTP Code	Response
400	**Error**: Parameters' validation failed.
	Error: An asset [XXXXXXXXX] must be a valid UID that points to a successfully processed asset.
	See HTTP code 400 description.

HTTP Code	Response
404	**Error**: Given project not found.
404	**Error**: Asset [AAAAAAAAAAAAAAA1] not found...
404	**Error**: Asset [AAAAAAAAAAAAAAA1] does not belong to project [XXXXXXXXXXXXXXX].

```
{
  "version": "1.0",
  "statusCode": 404,
  "message": "Given project not found."
}
```

HTTP Code	Response
412	**Error**: You cannot create a task for a project in the state [Complete]. Only projects that are `Active` or `OnHold` can have tasks.
412	**Error**: User with UID [ZZZZZZZZZZZZZ] not found.

```
{
  "version": "1.0",
  "statusCode": 412,
  "message": "You cannot create a task for a project in the state [Complete]."
}
```

POST /api/v1/task/review_asset_ext

Creates a new External Review Asset Task for someone who doesn't have an account in Approval Studio.
The flow is the following:

1. When a task is created, an email is sent to an email address from the request.
2. Email contains a link (URL) to a proof report review session.
3. Approve or reject terminates the review task and makes the URL expire.

Request

```
{
      "projectUID": "XXXXXXXXXXXXXX", // Project this task belongs to, mandatory.
      "email": "string",              // User's email, mandatory.
      "assetUIDs": [
        "AAAAAAAAAAAAAAA1",           // One or more assets to review, namdatory.
        "AAAAAAAAAAAAAAA2"
      ],
      "dueDate": "2020-12-04",        // Task due date, UTC, optional.
      "password": "string",           // Optional password. When provided, the re
                                      // to enter it before the review session.
      "emailSubject": "string",       // Custom email subject line; override temp
```

```
      "emailLanguage": "English",     // Email language, optional. English is defa
                                      // English, German, French, Polish, Spanish
      "comment": "comment text"       // Comment text, optional.
      "isAllowDownloadAssets": true,  // When true, allows user to download origi
      "isReadOnly": true              // When true, makes a review session read-o
                                      // no comments, no approve/reject. Optional
}
```

Curl:

```
curl -X POST "https://api.approval.studio/api/v1/task/review_asset_ext" \
     -H "accept: text/plain" \
     -H "Authorization: Bearer YYYYYYYYYYYYYYYY" \
     -H "Content-Type: application/json-patch+json" \
     -d "{\"projectUID\":\"XXXXXXXXXXXXXXXX\",
          \"email\":\"ZZZZZZZZZZZZZ\",
          \"assetUIDs\":[\"AAAAAAAAAA1\", \"AAAAAAAAAA2\"],
          \"dueDate\":\"2020-12-04\",
          \"comment\":\"comment text\" [...]}"
```

Responses

HTTP Code	Response
200	**Success**. Task created.

```
{
  "version": "1.0",
  "statusCode": 200,
  "message": "Task created.",
  "result": {
    "task": {
      "taskUID": "XXXXXXXXXXXXXXXXXXXXXX",
      "projectId": 981,
      "projectUID": "YYYYYYYYYYYYYYYYYYYYYYYYYYYY",
      "type": "ExternalReviewAssets",
      "status": "Pending",
      "comment": "Free-form text comment to the task",
      "dueDate": "2020-12-22",
      "created": "2020-12-22",
      "userUID": "ZZZZZZZZZZZZZ",
      "userName": "John Smith",
      "userEmail": "john.smith@gmail.com"
    }
  }
}
```

HTTP Code	Response
400	**Error**: Parameters' validation failed.
	Error: An asset [XXXXXXXXX] must be a valid UID that points to a successfully processed asset.
	See HTTP code 400 description.

HTTP Code	Response
404	**Error**: Given project not found.
404	**Error**: Asset [AAAAAAAAAAAAAAA1] not found...
404	**Error**: Asset [AAAAAAAAAAAAAAA1] does not belong to project [XXXXXXXXXXXXXXX].

```
{
  "version": "1.0",
  "statusCode": 404,
  "message": "Given project not found."
}
```

HTTP Code	Response
412	**Error**: You cannot create a task for a project in the state [Complete]. Only projects that are `Active` or `OnHold` can have tasks.

```
{
  "version": "1.0",
  "statusCode": 412,
  "message": "You cannot create a task for a project in the state [Complete]."
}
```

PUT /api/v1/task/complete

Completes a given **task**.

Note: **AssignUserRoles** task, related to workflow initialing, is processing in the following way:

- If required data (internal and/or external users) **is already provided**, then task would be completed.
- If required data **is not provided**, then `data->assignUsers` group (see below) must be provided to complete the task.

> For example, a workflow is configured to a have two internal users, uploader and reviewer, and
> two external ones. Then you need to have two users in the group `data->assignUser->internalUsers`:
> `[ "UploaderUserUID1" ], <-- Uploader.`
> `[ "ReviewerUserUID2" ], <-- Reviewer.`
> You may also provide multiple UIDs for every user in the group:
> `[“UploaderUserUID1”, “UploaderUserUID2”], <-- Uploaders

- This method validates if all the required data provided, and if not, then HTTP code **412** would be returned with an appropriate comment.

Request

```
{
  "taskUID": "XXXXXXXXXXXXXX" // Task id to complete.
  "data": {
        "assignUsers": { // AssignUser-task specific data.
          "internalUsers": [
            [ "UserUID1", "UserUID2"... ], <-- first internal users(s)
            [ "UserUID3", "UserUID4"... ], <-- second internal user(s)
            [...]
          ],
          "externallUsers": [
            [ "email1@email.com", "email2@email.com"... ],
            [ "email3@email.com", "email4@email.com"... ],
          ]
        }
      }
}
```

Curl:

```
curl -X POST "https://api.approval.studio/api/v1/task/complete" \
     -H "accept: text/plain" \
     -H "Authorization: Bearer YYYYYYYYYYYYYYYY" \
     -H "Content-Type: application/json-patch+json" \
     -d "{\"taskUID\":\"XXXXXXXXXXXXXXX\", \"data\": {...}}"
```

Responses

HTTP Code	Response
200	**Success**. Task marked as completed.

```
{
  "version": "1.0",
  "statusCode": 200,
  "message": "Task completed."
}
```

HTTP Code	Response
400	**Error**: Parameters' validation failed.

HTTP Code	Response
404	**Error**: Given task not found.

See HTTP code 400 description.

```
{7
  "version": "1.0",
  "statusCode": 404,
  "message": "Task not found."
}
```

HTTP Code	Response
410	**Error**: the task is already completed or deleted or approved/rejected. Only pending tasks can be completed.

```
{
  "version": "1.0",
  "statusCode": 410,
  "message": "The task is not pending and cannot be completed."
}
```

HTTP Code	Response
412	**Error**: Only upload-related tasks might be marked as completed:

HTTP Code	Response
	Allowed task types: `UploadAssets` , `UploadChangedAsset` , `UploadRefDocs` , `UploadVideo` , `AssignUserRoles`

```
{
  "version": "1.0",
  "statusCode": 412,
  "message": "This type of task cannot be manually completed" |
             "This task can not be completed as provided list of users does not ma
}
```

Annotations management

Method/Path	Description
`GET /api/v1/annotation/all`	Returns a list of annotations for a given asset .
`GET /api/v1/annotation`	Returns an annotation.
`POST /api/v1/annotation`	Creates a new annotation.
`DELETE /api/v1/annotation`	Deletes an annotation.
`PUT /api/v1/annotation/hide`	Hides an annotation.
`PUT /api/v1/annotation/complete`	Completes an annotation.
`PUT /api/v1/annotation/uncomplete`	Un-completes an annotation.

GET /api/v1/annotation/all

Returns a list of annotations for a given asset and page.

Request

Parameter	Type & Explanation
AssetUID	`string` Unique project ID, **mandatory**.
PageNum	`int` Page number, zero-based, **mandatory**.

```
curl -X GET "http://api.approval.studio/api/v1/annotation/all?AssetUID=XXXXXXXXXX
     -H  "accept: text/plain" \
     -H "Authorization: Bearer YYYYYYYYYYYYYYYY"
```

Responses

HTTP Code	Response
200	**Success**. Annotations returned and a hierarchy built.

```
{
  "version": "1.0",
  "statusCode": 200,
  "message": "GET Request successful.",
  "result": {
        {
          "0": [ // Key is the page number
            {
              "commentId": 0,
              "commentUID": "CCCCCCCCCCCCCC",
              "body": "Annotation body",
              "drawingCode": "XXXXXXXXXXXXX",
              "created": "2020-12-04T16:35:11.083Z",
              "pageNum": 0,
              "sequenceId": 0,
              "isCompleted": true,
              "replies": [
                  { annotation instance 1},         // The same annotation instan
                  { annotation instance 2} [...]    // tree-like annotation hiera
              ],
              "user": {
                "userUID": "ZZZZZZZZZZ",
                "fullName": "string",
                "email": "string"
              }
            }
          ],
          "1" : [...]
    [...]
```

```
  }
}
```

HTTP Code	Response
400	**Error**: Parameters' validation failed.
	See HTTP code 400 description.

HTTP Code	Response
404	**Error**: Asset with the given ID not found.

```
{
  "version": "1.0",
  "statusCode": 404,
  "message": "Asset not found."
}
```

GET /api/v1/annotation

Returns an annotation for a given annotation id.

Request

Parameter	Type & Explanation
AnnotationUID	`string` Unique annotation ID, **mandatory**.

```
curl -X GET "http://api.approval.studio/api/v1/annotation?AnnotationUID=XXXXXXXXX
     -H  "accept: text/plain" \
     -H "Authorization: Bearer YYYYYYYYYYYYYYYY"
```

Responses

HTTP Code	Response
200	**Success**. Annotations returned and a hierarchy built.

```
{
  "version": "1.0",
```

```
  "statusCode": 200,
  "message": "GET Request successful.",
  "result": {
      "commentUID": "XXXXXXXXXXXXXXX",
      "body": "Annotation body",
      "drawingCode": "DDDDDDDDDDD",          // Simple json-based markup code.
      "created": "2020-12-04T16:35:11.083Z",
      "pageNum": 0,
      "sequenceId": 0,
      "isCompleted": true,
      "user": {
        "userUID": "ZZZZZZZZZZ",
        "fullName": "string",
        "email": "string"
      }
  }
}
```

HTTP Code	Response
400	**Error**: Parameters' validation failed.
	See HTTP code 400 description.

HTTP Code	Response
404	**Error**: Annotation with the given ID not found.

```
{
  "version": "1.0",
  "statusCode": 404,
  "message": "Annotation not found."
}
```

POST /api/v1/annotation

Creates a new high-level comment for a given asset.

Request

Parameter	Type & Explanation
`assetUID`	`string` Unique asset ID.

Parameter	Type & Explanation
pageNum	`int` Zero-based page number.
body	`string` Text of the annotation, a free-form text or safe html.

Curl:

```
curl -X 'POST' \
  'https://api.dev.lam.hitech.dev/api/v1/annotation' \
  -H 'accept: text/plain' \
  -H 'Content-Type: application/json-patch+json' \
  -d '{
  "assetUID": "XXXXXXXXXXXXXXXXXXXXXXXX",
  "pageNum": 0,
  "body": "Some text."
}'
```

Responses

HTTP Code	Response
200	**Success**. Annotation created.

```
{
  "version": "1.0",
  "statusCode": 200,
  "message": "Annotation created."
}
```

HTTP Code	Response
400	**Error**: Parameters' validation failed.
	See HTTP code 400 description.

HTTP Code	Response
404	**Error**: Asset with the given ID not found.

```
{
  "version": "1.0",
  "statusCode": 404,
  "message": "Asset not found."
}
```

DELETE /api/v1/annotation

Deletes an annotation for a given annotation id.

Request

Parameter	Type & Explanation
`annotationUID`	`string` Unique annotation ID.

Curl:

```
curl -X DELETE "https://api.approval.studio/api/v1/annotation"
-H  "accept: text/plain"
-H  "Authorization: Bearer YYYYYYYYYYYYYYYYYYYYYYYYYY"
-H "Content-Type: application/json-patch+json"
-d "{\"annotationUID\":\"XXXXXXXXXXX\"}"
```

Responses

HTTP Code	Response
200	**Success**. Annotation deleted.

```
{
  "version": "1.0",
  "statusCode": 200,
  "message": "Annotation deleted."
}
```

HTTP Code	Response
400	**Error**: Parameters' validation failed.
	See HTTP code 400 description.

HTTP Code	Response
404	**Error**: Annotation with the given ID not found.

```
{
  "version": "1.0",
  "statusCode": 404,
```

```
  "message": "Annotation not found."
}
```

HTTP Code	Response
412	**Error**: Only the user that created the annotation or comment can delete it. You cannot delete other's users' annotations and this annotation does not belong to you. Deletion failed.

```
{
  "version": "1.0",
  "statusCode": 404,
  "message": "You can delete only your annotation."
}
```

PUT /api/v1/annotation/hide

Hides an annotation for a given annotation id.

Request

Parameter	Type & Explanation
`taskUID`	`string` Unique annotation ID.

Curl:

```
curl -X PUT "https://api.approval.studio/api/v1/annotation/hide"
-H  "accept: text/plain"
-H  "Authorization: Bearer YYYYYYYYYYYYYYYYYYYYYYYYY"
-H "Content-Type: application/json-patch+json"
-d "{\"annotationUID\":\"XXXXXXXXXX\"}"
```

Responses

HTTP Code	Response
200	**Success**. Annotation was hidden.

```
{
  "version": "1.0",
  "statusCode": 200,
  "message": "Annotation was hidden."
}
```

HTTP Code	Response
400	**Error**: Parameters' validation failed.
	See HTTP code 400 description.
406	**Error**: A user can hide only their own annotation.
404	**Error**: Annotation with the given ID not found.

```
{
  "version": "1.0",
  "statusCode": 404,
  "message": "Annotation not found."
}
```

PUT /api/v1/annotation/complete

Completes an annotation for a given annotation id.

Request

Parameter	Type & Explanation
`taskUID`	`string` Unique annotation ID.

Curl:

```
curl -X PUT "https://api.approval.studio/api/v1/annotation/complete"
-H  "accept: text/plain"
-H  "Authorization: Bearer YYYYYYYYYYYYYYYYYYYYYYYYY"
-H "Content-Type: application/json-patch+json"
-d "{\"annotationUID\":\"XXXXXXXXXX\"}"
```

Responses

HTTP Code	Response
200	**Success**. Annotation was marked as completed.

```
{
  "version": "1.0",
  "statusCode": 200,
  "message": "Annotation was completed."
}
```

HTTP Code	Response
400	**Error**: Parameters' validation failed.
	See HTTP code 400 description.
404	**Error**: Annotation with the given ID not found.
406	**Error**: Annotation is already completed.

```
{
  "version": "1.0",
  "statusCode": 404,
  "message": "Annotation not found."
}
```

PUT /api/v1/annotation/uncomplete

Un-completes an annotation for a given annotation id.

Request

Parameter	Type & Explanation
`taskUID`	`string` Unique annotation ID.

Curl:

```
curl -X PUT "https://api.approval.studio/api/v1/annotation/uncomplete"
-H  "accept: text/plain"
-H  "Authorization: Bearer YYYYYYYYYYYYYYYYYYYYYYYYY"
-H "Content-Type: application/json-patch+json"
-d "{\"annotationUID\":\"XXXXXXXXXX\"}"
```

Responses

HTTP Code	Response
200	**Success**. Annotation was marked as uncompleted.

```
{
  "version": "1.0",
  "statusCode": 200,
  "message": "Annotation was uncompleted."
}
```

HTTP Code	Response
400	**Error**: Parameters' validation failed.
	See HTTP code 400 description.
404	**Error**: Annotation with the given ID not found.
406	**Error**: Annotation is already completed.

```
{
  "version": "1.0",
  "statusCode": 404,
  "message": "Annotation not found."
}
```

Users management

Method/Path	Description
`GET /api/v1/users`	Returns all clients(tenants) with users.
`GET /api/v1/user/loggedin`	Returns currently logged in user user.

GET /api/v1/users

This method returns a list of available active users per tenant (client) and a template list for every client.
Please read more about the concept of project templates in the project methods' group.
Curl:

```
curl -X GET "https://api.approval.studio/api/v1/users"
     -H "accept: text/plain"
     -H "Authorization: Bearer YYYYYYYYYYYYYYYYYYYYYYYYYY"
```

Responses

HTTP Code	Response
200	**Success**. List of users with companies they belong to returned.

```
{
  "version": "1.0",
  "statusCode": 200,
  "message": "GET Request successful.",
  "result": {
    "clients": [
     {
        "clientUID": "XXXXXXXXXXXXXXXXXXX",
        "name": "HiTech Service",
        "users": [
          {
            "userUID": "UUUUUUUUUUUUUUUUUU1",
            "fullName": "John Smith",
            "email": "john.smith@gmail.com",
            "role": "Administrator|RegularUser"
          },
          [...]
        ],
        "templates": [
          {
            "templateUID": "TTTTTTTTTTTTTTTTTT",
            "name": "First template",
            "data": {
              "key": "value",
              "key1": "value1"
            }
          },
          [...]
        ],
        "kanbanColumns": [ // List of kanban columns configured for the given cli
          {
```

```
            "name": "Active Projects",
            "kanbanColumnUID": "p8i41"
          },
          {
            "name": "In Production",
            "kanbanColumnUID": "d4df5a"
          }

        ],
        "folders": [
          {
            "folderUID": "#shared#",
            "name": ""
          },
          {
            "folderUID": "ib0t",
            "name": "10_04_2024_01"
          }
        ]
    },
    [...]
  }
}
```

HTTP Code	Response
400	**Error**: Parameters' validation failed.
	See HTTP code 400 description.

GET /api/v1/user/loggedin

This method returns a currently logged in user; this is the used that was used when creating an authorization token in method `POST /api/v1/token/login`.

Curl:

```
curl -X GET "https://api.approval.studio/api/v1/users"
     -H "accept: text/plain"
     -H "Authorization: Bearer YYYYYYYYYYYYYYYYYYYYYYYYYY"
```

Responses

HTTP Code	Response
200	**Success**. List of users with companies they belong to returned and a complete list of clients including inactive ones.

```
{
  "version": "1.0",
  "statusCode": 200,
  "message": "GET Request successful.",
  "result": {
    "user": {
      "userUID": "XXXXXXXXXXXXXXXXXXX",
      "fullName": "John Smith",
      "email": "john.smith@company.com",
      "role": "Administrator"
    },
    "clients": [
      {
        "clientUID": "YYYYYYYYYYYYYYYYYYY",
        "name": "Client Name",
        "status": "Active|Inactive",
        "clientType": "Pro|Lite",
        "kanbanColumns": [
          {
            "kanbanColumnUID": "XYZ",
            "name": "Column One"
          },
          {
            "kanbanColumnUID": "ZYX",
            "name": "Column Two"
          }
        ]
      },
     ...
     ]
  }
}
```

HTTP Code	Response
400	**Error**: Parameters' validation failed.
	See HTTP code 400 description.

Webhooks

Approval Studio API uses webhooks to notify your application when an event happens. Webhooks are particularly useful for asynchronous events like when an asset is accepted or rejected or when a new task is created or someone made a comment or a new asset version is uploaded etc.

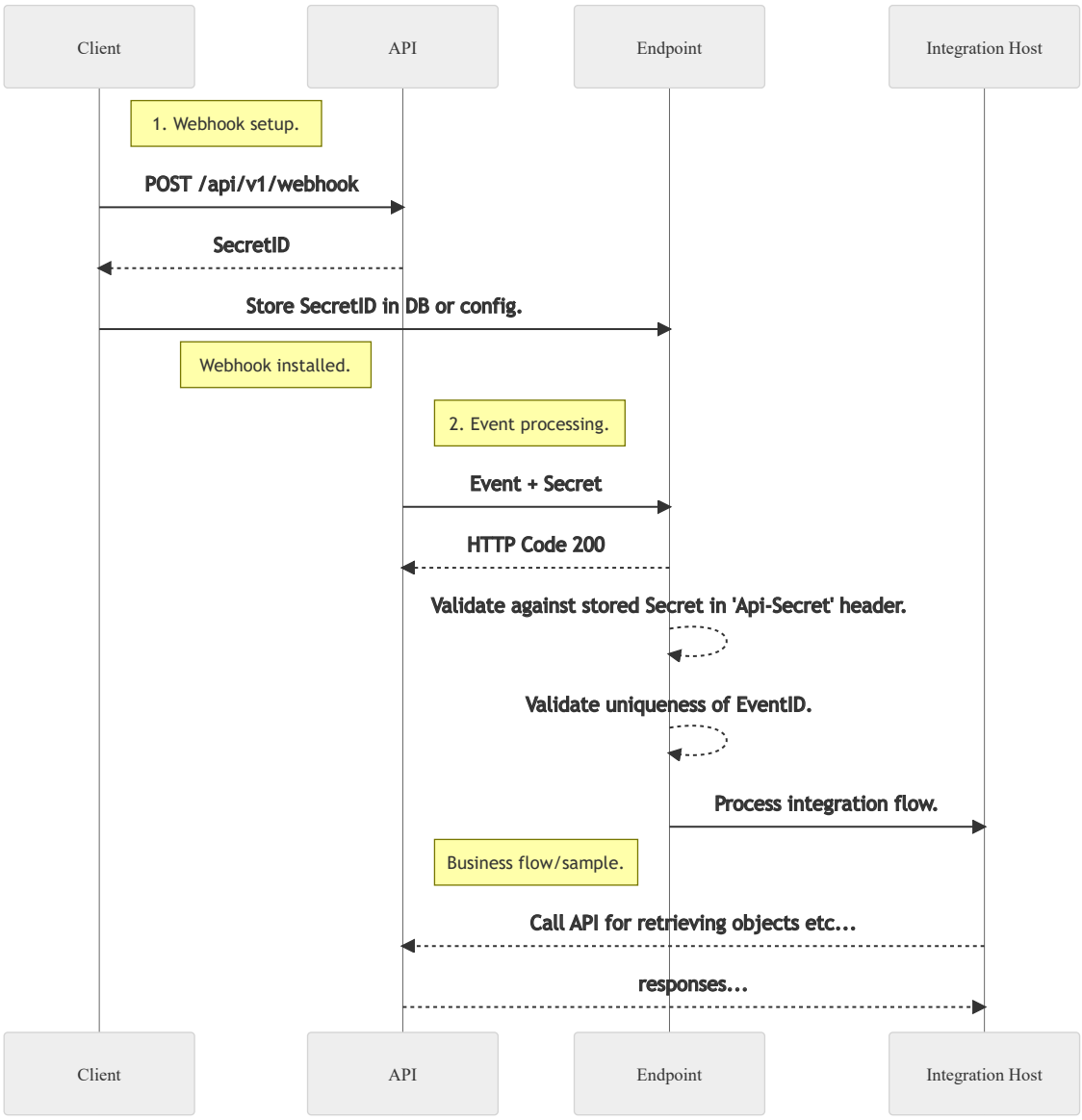
Begin using the webhooks in three steps:

1. Create a webhook endpoint on your server.
2. Register the endpoint with Webhook management method **POST /api/v1/webhook**.
3. Test it to ensure that you can receive events using method **PUT /api/v1/webhook/test**.

Webhooks generally refer to a combination of elements that collectively create a notification and reaction system within a larger integration.

The webhook endpoint is code on your server, which could be written in C#, Java, Ruby, PHP, Node.js, or any other language/technology you prefer. The webhook endpoint has an associated URL (e.g., **https://example.com/webhook**).
Note: Explanations on how to build an endpoint are out of the scope of this document.

The Approval Studio notifications are Event objects. Those event objects contain all the relevant information about what just happened, including the type of event and the data associated with that event. The webhook endpoint uses the event details to take any required actions.



Security

The API host provides a **Secret ID** with every event object posted on an endpoint. The endpoint needs to check against this ID for every incoming post to avoid spamming. Keep secret id safe.

Retry logic

Webhook host implements a simple retry logic in order to try to deliver events in case of possible troubles on the endpoint side
If the endpoint does not return HTTP code 200, the webhooks host starts retrying posting the same event object up to ~30 minutes. After that time the event delivery silently fails.

Event object

An event object is a JSON document that API posts on a selected endpoint. It generally has the following structure:

```
{
    "EventId":"DY3h1Pl040uIRvL74oKFlQ",      // Unique event id.
    "EventType":"annotation.added",          // Codename of the event, see below.
    "Created":"2021-02-18T14:08:08.615088Z", // UTC datetime when the event was g
    "Data": {                                // Data block, event-type dependent.
        "SomeData": "XXXXXXXX"
    }
}
```

EventID is unique for every Event object. If for whatever reason, the API host makes more than one post on your endproint with the same event object, you can identify it by reading EventID.
Event types are self-explanatory:

Event code	Description
`project.created`	Fires when a new project is created.
`project.edited`	Fires when a project's attribute changes, like name, description, due date, etc.
`project.state`	Fires when project state changes.
`asset.uploaded`	Fires when an asset or a new version of an existing asset is uploaded.
`asset.deleted`	Fires when asset deleted.
`refdoc.uploaded`	A new reference document uploaded.
`refdoc.deleted`	An existing reference document is deleted.

Event code	Description
`annotation.added`	An annotation to asset is created.
`annotation.edited`	An annotation is edited.
`annotation.deleted`	An annotation is deleted.
`task.created`	A new task of any type is created.
`task.completed`	A task is marked as completed.
`task.deleted`	A task is deleted.
`task.approved`	An asset review task marked as approved.
`task.rejected`	An asset review task marked as rejected.
`webhook.test`	Dummy event object for the testing endpoint.

project.created

Fires when a new project is created.

```
{
    "EventId": "XppRxqUyNEqYrZtY9lRBdA",
    "EventType": "project.created",
    "Created": "2021-02-18T23:22:33.2944308Z",
    "Data": {
        "ProjectUID": "B0CADA6D72824F85A38BE8471503D204", // Unique ID of a newly
        "Name": "ProjectName plain text", // Project name.
        "Project": { // Here is a Project entity as it is
                     // in the GET /api/v1/project method.
            [...]
            "ProjectUID": "XXXXXXXXXXXX",
            "ProjectState": "OnHold",
            "Name": "string",
            "Customer": "string",
            "Project": "string",
            "Design": "string",
            "Revision": "string",
            "Description": "string",
            "Tags": [
                "string",
                "string"
            ],
            "ReviewStatus": {
```

```
                "PendingCount": 1,
                "ApprovedCount": 2,
                "RejectedCount": 3
            },
            "Created": "2021-04-29T11:56:44.690266Z",
            "Owners": [
                "YYYYYYYYYYYY0",
                "YYYYYYYYYYYY1"
            ],
            "Client": {
                "ClientUID": "ZZZZZZZZZZZZ",
                "Name": "string"
            },
            "FolderUID": "string",
            "Workflow": {
                "WorkflowUID": "string",
                "Name": "string"
            }
        }
    }
}
```

project.edited

Fires when one of the project's attributes is changed: `Name`, `Customer`, `Project`, `Design`, `Revision`, `Description`, `Tags`, `DueDate`, `ProjectOwners`.

```
 {
        "EventId":"xDPuRZ1F_Ea03fO-WPZqnQ",
        "EventType":"project.edited",
        "Created":"2021-02-18T23:47:17.7572002Z",
        "Data": {
            "ProjectUID":"XXXXXXXXXXXXXXXXXX",          // Unique ID of a proje
            "Attribute":"Name",                         // Attribute name, see
            "Value":"New project name",                 // New attribute value.
            "Project" : {                               // Here is a Project en
                                                        // in the GET /api/v1/p
              [...]
          }
         }
}
```

project.state

Fires when the project's state is changed, interactively or automatically by Approval Studio itself.

```
{
    "EventId": "xDPuRZ1F_Ea03fO-WPZqnQ",
    "EventType": "project.edited",
    "Created": "2021-02-18T23:47:17.7572002Z",
    "Data": {
        "ProjectUID": "XXXXXXXXXXXXXXXXXX", // Unique ID of a project where the cl
        "Attribute": "Name", // Attribute name, see the list above.
        "Value": "New project name", // New attribute value.
        "Project": { // Here is a Project entity as it is
                     // in the GET /api/v1/project method.
            [...]
        }
    }
}
```

asset.uploaded

Fires when an asset (or an asset version) is uploaded and successfully processed. If asset processing fails due to any possible reason, an event is not sent.

```
{
    "EventId": "inuBFhnHn0CUZ4u4OO-8Ww",
    "EventType": "asset.uploaded",
    "Created": "2021-02-19T12:57:53.9413529Z",
    "Data": {
        "ProjectUID": "XXXXXXXXXXXXXXXXXX",
        "AssetUID": "YYYYYYYYYYYYYYYYYY",
        "Name": "filename.png",
        "Project": { // Here is a Project entity as it is
                     // in the GET /api/v1/project method.
            [...]
        }
    }
}
```

asset.deleted

Fires when an asset (or an asset version) is deleted.
Note: when an asset is deleted, all the related objects – tasks, comments, attachments – are deleted as well, but you will not get separate webhook calls for them.

```
{
    "EventId": "__Z6LClFFkyxeEPeVx-ytQ",
    "EventType": "asset.deleted",
    "Created": "2021-02-19T13:12:39.2669051Z",
    "Data": {
        "ProjectUID": "XXXXXXXXXXXXXXXXXX",
        "AssetUID": "YYYYYYYYYYYYYYYYYY",
        "Project": { // Here is a Project entity as it is
                     // in the GET /api/v1/project method.
            [...]
        }
    }
}
```

refdoc.uploaded

Fires when a reference document is uploaded and processed successfully.

```
{
    "EventId": "q2bwNSmAoUO0wXMVTyn7Cg",
    "EventType": "refdoc.uploaded",
    "Created": "2021-02-19T13:33:57.6370653Z",
    "Data": {
        "ProjectUID": "XXXXXXXXXXXXXXXXXX",
        "RefDocUID": "YYYYYYYYYYYYYYYYYY",
        "Name": "refdocument.docx",
        "Project": { // Here is a Project entity as it is
                     // in the GET /api/v1/project method.
            [...]
        }
    }
}
```

refdoc.deleted

Fires when a reference document is deleted.

```
{
    "EventId": "AT4ShAVLN0Ck7kVhMHYzUw",
    "EventType": "refdoc.deleted",
    "Created": "2021-02-19T13:37:59.3381091Z",
    "Data": {
        "ProjectUID": "XXXXXXXXXXXXXXXXXX",
        "RefDocUID": "YYYYYYYYYYYYYYYYYY",
        "Project": { // Here is a Project entity as it is
```

```
                    // in the GET /api/v1/project method.
            [...]
        }
    }
}
```

annotation.added

Fires when an annotation is added to an asset.
This can be a high-level annotation associated with the selected region on the image, or a comment (to an annotation).

```
{
    "EventId": "Vx5S8r4os0K5WlwYFsYRhg",
    "EventType": "annotation.added",
    "Created": "2021-02-19T13:45:24.7158494Z",
    "Data": {
        "ProjectUID": "XXXXXXXXXXXXXXXXXX",
        "AnnotationUID": "YYYYYYYYYYYYYYYYYY",
        "AssetUID": "AAAAAAAAAAAAAAAAAA",
        "Text": "Annotation text",
        "Project": { // Here is a Project entity as it is
                    // in the GET /api/v1/project method.
            [...]
        }
    }
}
```

annotation.deleted

Fires when an annotation or comment to an annotation is deleted.

```
{
    "EventId": "HdBv5m13CEaZPBAr4E0AIw",
    "EventType": "annotation.deleted",
    "Created": "2021-02-19T14:51:23.9392566Z",
    "Data": {
        "ProjectUID": "XXXXXXXXXXXXXXXXXX",
        "AnnotationUID": "YYYYYYYYYYYYYYYYYY",
        "AssetUID": "AAAAAAAAAAAAAAAAAA",
        "Project": { // Here is a Project entity as it is
                    // in the GET /api/v1/project method.
            [...]
        }
```

```
    }
}
```

task.created

Fires when a task is created. Only a task ID is passed to an Event object.

```
{
    "EventId": "QbLNa0r3UkuLPJ3KZpoDNg",
    "EventType": "task.created",
    "Created": "2021-02-19T15:10:19.7757392Z",
    "Data": {
        "TaskUID": "XXXXXXXXXXXXXXXXXX",
        "Project": { // Here is a Project entity as it is
                     // in the GET /api/v1/project method.
            [...]
        },
        "Task": { // Here is a Task entity as it is
            // in the GET /api/v1/task.
            "taskUID": "XXXXXXXXXXX",
            "projectUID": "YYYYYYYYYYYY",
            "projectName": "string",
            "clientUID": "ZZZZZZZZZZZZ",
            "type": "UploadAssets|UploadRefDocs|ReviewAssets|ExternalReviewAssets
            "status": "Pending|Closed|Approved|Rejected|ApprovedWithChanges",
            "created": "2020-11-27T15:39:59.864882",
            "closed": "2020-11-27T15:40:51.407384",
            "user": {
                "userUID": "92AFE33153124F0980E43EF80133FE9B",
                "fullName": "John Smith",
                "email": "john.smith@email.com"
            },
            "reviewUrl": string, // URL to launch prooftool for given task.
            // Appears only for ReviewAsset task.
            "requestedAssetName": "filename.pdf" // Name of the asset, a task ref
            // Appears only in UploadChangedAsset task.
        }
    }
}
```

task.deleted

Fires when a task is deleted, interactively, or through API.

```
{
    "EventId": "0Jd-zSoOKU2_PoG6zSb5uQ",
    "EventType": "task.deleted",
    "Created": "2021-02-19T15:20:58.5833087Z",
    "Data": {
        "TaskUID": "XXXXXXXXXXXXXXXXXX",
        "Project": { // Here is a Project entity as it is
                     // in the GET /api/v1/project method.
            [...]
        },
        "Task": {    // Here is a Task entity as it is
                     // in the GET /api/v1/task.
            [...]
        }
    }
}
```

task.approved

Fires when an asset is approved in a proof tool or using API.
TaskUID is an ID of an asset review task associated with a proof tool session.

```
{
    "EventId": "0Jd-zSoOKU2_PoG6zSb5uQ",
    "EventType": "task.deleted",
    "Created": "2021-02-19T15:20:58.5833087Z",
    "Data": {
        "TaskUID": "XXXXXXXXXXXXXXXXXX",
        "Project": { // Here is a Project entity as it is
                     // in the GET /api/v1/project method.
            [...]
        },
        "Task": {    // Here is a Task entity as it is
                     // in the GET /api/v1/task.
            [...]
        }
    }
}
```

task.rejected

Fires when an asset is rejected in a proof tool or using API.
TaskUID is an ID of an asset review task associated with a proof tool session.

```
{
    "EventId": "0Jd-zSoOKU2_PoG6zSb5uQ",
    "EventType": "task.deleted",
    "Created": "2021-02-19T15:20:58.5833087Z",
    "Data": {
        "TaskUID": "XXXXXXXXXXXXXXXXXX",
        "Project": { // Here is a Project entity as it is
                     // in the GET /api/v1/project method.
            [...]
        },
        "Task": { // Here is a Task entity as it is
                  // in the GET /api/v1/task.
            [...]
        }
    }
}
```

webhook.test

Fires when API method **PUT /api/v1/webhook/test** is invoked. It works like any other real event including a secret UID in an HTTP header `Api-Secret`.

```
{
    "EventId": "0Jd-zSoOKU2_PoG6zSb5uQ",
    "EventType": "webhook.test",
    "Created": "2021-02-19T15:20:58.5833087Z",
    "Data": {
        "DummyData": "random string"
    }
}
```

Webhooks management

Method/Path	Description
`GET /api/v1/webhooks`	Returns all webhooks for the current user's tenant.
`POST /api/v1/webhook`	Sets us a new webhook.
`DEL /api/v1/webhook`	Deletes a webhook.
`PUT /api/v1/webhook/test`	Test a webhook endpoint.

GET /api/v1/webhooks

Returns all the installed webhooks.

Curl:

```
curl -X GET "https://api.approval.studio/api/v1/webhooks"
     -H "accept: text/plain"
     -H "Authorization: Bearer YYYYYYYYYYYYYYYYYYYYYYYYYY"
```

Responses

HTTP Code	Response
200	**Success**. List of webhooks.

```
{
  "version": "1.0",
  "statusCode": 200,
  "message": "GET Request successful.",
  "result": [
    {
      "webhookUID": "XXXXXXXXXXXXXXXXXXXX01",
      "url": "http://xxxxx.com/part/of/url01",
      "secret": "YYYYYYYYYYYYYYYYYYYY01",       // This secret will be sent with
      "created": "2021-02-09T09:17:27.904911"    // to ensure that this is API ho
    },
    {
      "webhookUID": "XXXXXXXXXXXXXXXXXXXX02",
      "url": "http://xxxxx.com/part/of/url02",
      "secret": "YYYYYYYYYYYYYYYYYYYY02",
      "created": "2021-02-09T09:17:27.904911"
    }
  ]
}
```

POST /api/v1/webhook

Sets up a new webhook.

Request

```
{
  "url": "string",       // Valid URL string, http or https.
```

```
  "clientUID": "string", // Client UID, optional.
  "eventType": "string"  // One of the event type names, optional.
}
```

clientUID is an optional client identificator. Events are client-wide, so in the case of multitenancy, when a user belongs to multiple clients, the events through webhooks are produced based on the client you provided when creating a webhook. You need to install multiple webhooks to get events from all tenants you belong to.
If no clientUID is provided, the first client in the list is used.
eventType is an optional event type, please see Event object When provided, it limits the webhook to be invoked only when the given event type occurs. If it's not provided, all events will be sent to the webhook and in this case an API consumer is responsible for filtering events (when necessary).

Note: The API host might need up to ~1 minute to start processing a newly installed webhook.
Curl:

```
curl -X POST "https://api.approval.studio/api/v1/webhook"
     -H "accept: text/plain"
     -H "Authorization: Bearer YYYYYYYYYYYYYYYYYYYYYYYYYY"
     -H  "Content-Type: application/json-patch+json"
     -d "{\"url\":\"http://xxxx.com/part/of/url\", \"clientUID\": \"XXXX\", \"ever
```

Responses

HTTP Code	Response
200	**Success**. A newly created webhook instance returned.

```
{
  "version": "1.0",
  "statusCode": 200,
  "message": "POST Request successful.",
  "result": {
      "webhookUID": "XXXXXXXXXXXXXXXXXXXX01", // Newly assigned webhook ID.
      "url": "http://xxxx.com/part/of/url",   // Url of the webhook, as in the re
      "secret": "YYYYYYYYYYYYYYYYYYYY01",     // This secret will be sent with ev
      "created": "2021-02-09T09:17:27.904911" // to ensure that this is API host
                                              // see HTTP response header "Api-Se
  }
}
```

HTTP Code	Response
400	**Error**: Parameters' validation failed.
	See HTTP code 400 description.
412	**Error**: Webhook with the given URL is already registered for this tenant. API does not allow to setup duplicates.

DELETE /api/v1/webhook

Deletes a webhook with the given ID.
Note that deletion of a webhook might need up to ~1 minute for API host to update.
Curl:

```
curl -X DELETE "https://api.approval.studio/api/v1/webhook"
     -H "accept: text/plain"
     -H "Authorization: Bearer YYYYYYYYYYYYYYYYYYYYYYYYYY"
     -H  "Content-Type: application/json-patch+json"
     -d "{\"webHookUID\":\"XXXXXXXXXXXXXXXXXXXXXX\"}"
```

Responses

HTTP Code	Response
200	**Success**. Webhook with the given ID deleted.

```
{
  "version": "1.0",
  "statusCode": 200,
  "message": "Webhook is successfully deleted."
}
```

HTTP Code	Response
400	**Error**: Parameters' validation failed.
	See HTTP code 400 description.
404	**Error**: Webhook with the given ID not found or was already deleted.

PUT /api/v1/webhook/test

Test a webhook's endpoint with dummy data in form of JSON:

```
{
        "EventId":"0Jd-zSoOKU2_PoG6zSb5uQ",
        "EventType":"webhook.test",
        "Created":"2021-02-19T15:20:58.5833087Z",
        "Data": {
            "DummyData":"random string"
        }
    }
```

When posting test data, the API host uses the same retry logic as it is for real calls, i.e. tries to re-deliver it in case of failure, so multiple calls are expected if the endpoint fails to accept the call instantly.
Curl:

```
curl -X PUT "https://api.approval.studio/api/v1/webhook/test"
     -H "accept: text/plain"
     -H "Authorization: Bearer YYYYYYYYYYYYYYYYYYYYYYYYYY"
     -H  "Content-Type: application/json-patch+json"
     -d "{\"webHookUID\":\"XXXXXXXXXXXXXXXXXXXXXXX\"}"
```

Responses

HTTP Code	Response
200	**Success**. Webhook is tested, see your endpoint logs.

```
{
  "version": "1.0",
  "statusCode": 200,
  "message": "Webhook is tested, see your endpoint logs."
}
```

HTTP Code	Response
400	**Error**: Parameters' validation failed.
	See HTTP code 400 description.
404	**Error**: Webhook with the given ID not found.

Workflow Management

Introduction

Workflow in Approval Studio is a event-based subsystem that allows to automate business flow based on users' activities.

There are a predefined list of events that trigger one or other actions.

It is like this: you may configure the workflow to create task(s) when a new project created. Or send assets to review when an asset is uploaded, and so on.

So in other words, workflow is a set of trigger <-> action pairs. Those actions are completely independent on each other.

The whole workflow from the point of view of API is an array of actions and optional list of roles:

Actions:

```
[
    {
      "uid": "XXXX",
      "friendlyName": "string",
      "triggerType": "ProjectCreated|InternalUploadAssetsTaskCompleted|...",
      "reactionType": "CreateInternalUploadAssetsTask|CreateInternalUploadRefDocs
      [...]
      // Trigger-action - specific fields.
    },
    {
      "uid": "YYYY",
      "friendlyName": "string",
      "triggerType": "ProjectCreated|InternalUploadAssetsTaskCompleted|...",
      "reactionType": "CreateInternalUploadAssetsTask|CreateInternalUploadRefDocs
      [...]
      // Trigger-action - specific fields.
    },
    [...]
]
```

The complete action is the following:

```
{
  "uid": "string",
  "friendlyName": "string",
  "triggerType": TriggerType,
  "reactionType": ReactionType,
  "userUIDs": [
    "string"
  ],
  "roleUids": [
    "string"
  ],
  "triggerExecutorUid": "string",
  "triggerRoleUid": "string",
  "triggerEmails": "string",
    "triggerRoleIds": [
    "string"
  ],
  "emails": "string",
  "emailSubject": "string",
  "emailLanguage": "string",
  "password": "string",
  "comment": "string",
  "checklistName": "string",
  "checklistItems": [
    "string"
  ],
  "daysToDueDate": [1...N],
  "allowDownload": true|false,
  "isSimpleMode": true|false,
  "isCustomPayload": true|false,
  "webhookURL": "string",
  "webhookRequestType": "string",
  "webhookContentType": "string",
  "webhookHeaders": [
    {
      "additionalProp1": "string",
      "additionalProp2": "string",
      "additionalProp3": "string"
    }
  ],
  "webhookFields": [
    "string"
  ],
  "webhookCustomPayload": "string",
  "contacts": "string",
  "targetKanbanColumnUid": "string"
}
```

Roles:

```
[
      {
        "uid": "XXXX",
        "name": "Roole Name",
        "isInternal": true|false
      },
      {
         [...]
      }
]
```

Triggers

Triggers initiate actions.

Trigger	Explanation
ProjectCreated	It triggers when a new project created and is fully initialized. It is occurred only once during the project's lifecycle.
InternalUploadAssetsTaskCompleted	Triggers when Uploading Asset task is completed, new asset or asset versions are uploaded and processed.
InternalUploadRefDocsTaskCompleted	Triggers when Uploading Reference Document task is completed.
InternalReviewTaskRejected	Triggers when asset review task is completed and the review result is reject.
InternalReviewTaskApproved	Triggers when asset review task is completed and the review result is approve.
ExternalUploadAssetsTaskCompleted	Triggers when external (addressed to a user outside your company) Uploading Asset task is completed, new asset or asset versions are uploaded and processed.
ExternalUploadRefDocsTaskCompleted	Triggers when external Uploading Reference Document task is completed.

Trigger	Explanation
ExternalReviewTaskRejected	Triggers when external (addressed to a user outside your company, usually using email) asset review task is completed and the review result is reject.
ExternalReviewTaskApproved	Triggers when external (addressed to a user outside your company, usually using email) asset review task is completed and the review result is approve.
UserRolesAssigned	The concept of this trigger is the following:
ManualTrigger	

Actions

Action is what is the reaction to a trigger. Actions contain different number of fields to provide, depends on the nature of the action.
Please see Workflow Fields to have a detailed set set fields to provides for every action.

Reaction	Explanation
CreateInternalUploadAssetsTask	Create a new asset uploading task.
CreateInternalUploadRefDocsTask	Create a new reference document uploading task.
CreateExternalUploadAssetsTask	Create a new external (the task assignee is outside the company) asset uploading task.
CreateExternalUploadRefDocsTask	Create a new external (the task assignee is outside the company) reference document uploading task.
CreateInternalReviewTask	Create a new asset review task. **Note**: It can be used in conjunction with asset upload task, making a upload-review cycle until reach an asset approve event.
CreateExternalReviewTask	Create a new asset review task (the task assignee is outside the company).

Reaction	Explanation
SendAssetsByEmail	Sends links to the project's asset(s) to a recipient's email(s) to download.
SendRefDocsByEmail	Sends links to the project's reference documents to a recipient's email(s) to download.
CompleteProject	Marks the project as **Complete** (changes the project status'). This ultimately ends the workflow processing. To restart workflow you need manually, or using API, move the project back to **Active** state.
SendDataWithWebhook	Makes an HTTP call to an external webhook. You may use this action to provide some kind of automation outside the scope of the Approval Studio. Think of it as an custom workflow action implemented separately. **Note**: This is not related to the API's webhooks described earlier in the documentation.
AssignUserRoles	This action is important to create an adaptive workflow. Approval Studio does nto have a concept of roles. The assigning role action allows to assign users to roles at runtime instead of hardcoding them in the workflow itself. At design time, you may define one or more roles, like "Uploader", "Reviewer", "External Reviewer" etc and ask some user, usually project owner, to assign concrete user to those roles. This makes the workflows adaptable when you need to assign different people to different projects.
SendCustomEmail	This action just send a email to one or more recipients. Use it when you need to notify someone.
MoveInKanban	This action moves the project to a predefined Canban column. This allows you to track visually the workflow progress.
SwitchToWorkflow	This allows to to switch currently running project to another workflow.

Trigger-Reaction dependencies

Some actions make sense to particular triggers only.

For example, it makes no sense to create asset review task without any assets uploaded, or sending reference documents not having any of them attached to a project.

Below is the list of dependencies between triggers and allowed actions:

Trigger	Available reactions
ProjectCreated	`AssignUserRoles`, `CreateInternalUploadAssetsTask`, `CreateInternalUploadRefDocsTask`, `CreateExternalUploadAssetsTask`, `CreateExternalUploadRefDocsTask`, `SendDataWithWebhook`, `SendCustomEmail`, `SwitchToWorkflow`
UserRolesAssigned, **ManualTrigger**, **InternalUploadAssetsTaskCompleted**, **InternalUploadRefDocsTaskCompleted**, **ExternalUploadAssetsTaskCompleted**, **ExternalUploadRefDocsTaskCompleted**	`AssignUserRoles`, `CreateInternalUploadAssetsTask`, `CreateInternalUploadRefDocsTask`, `CreateExternalUploadAssetsTask`, `CreateExternalUploadRefDocsTask`, `CreateInternalReviewTask`, `CreateExternalReviewTask`, `SendAssetsByEmail`, `SendRefDocsByEmail`, `CompleteProject`, `SendDataWithWebhook`, `SendCustomEmail`, `SwitchToWorkflow`
InternalReviewTaskApproved, **InternalReviewTaskRejected**, **ExternalReviewTaskRejected**, **ExternalReviewTaskApproved**	`AssignUserRoles`, `CreateInternalUploadAssetsTask`, `CreateInternalUploadRefDocsTask`, `CreateExternalUploadAssetsTask`, `CreateExternalUploadRefDocsTask`, `CreateInternalReviewTask`, `CreateExternalReviewTask`, `SendAssetsByEmail`, `SendRefDocsByEmail`, `CompleteProject`, `SendDataWithWebhook`, `SendCustomEmail`, `SwitchToWorkflow`

Roles

The concept of roles is the following: instead of hardcoding users in the actions, you may define roles like “Designer”, “Reviewer”, “External Reviewer” etc and ask someone provide the concrete persons before workflow starts.

That is done by – a). by assigning a “AssignUserRoles” task or, b). interactively by a project owner who provides users for the roles in the project popup.

Note: there is always available a pre-defined role `"projectOwners"`. You may use it as a role UID wherever you need it:

```
"roleUids": [
    "projectOwners", <-- predefined role UID.
    "xxxx",
    "yyyy"
]
```

Workflow fields

Common fields

Field	Description
UID	[Mandatory] A unique action identifier, string.
TriggerType	[Mandatory] Trigger type, see Triggers.
ReactionType	[Mandatory] reaction type, see Actions.
Comment	[Optional] A free-form text that associated with the action. An end-user will see this comment on the UI and in the notifications like emails, WhatsUp message or Slack message.
FriendlyName	[Optional] Name of the workflow step (action).

Action-specific Fileds

Field	Type	Description
userUIDs	`string[]`	Along with RoleUIDs presented with all task-based actions. Provide here UID(s) of user(s) to have task assigned to.

Field	Type	Description
roleUids	`string[ ]`	The same as roleUids except this is role's UIDs, not user's.
triggerExecutorUid	`string`	
triggerRoleUid	`string`	
emailSubject	`string`	A custom subject in the email to be sent for email-related actions.
password	`string`	An optional password in the asset review-related actions.
comment	`string`	An optional comment in the task-related actions.
checklistName	`string`	An optional checklist name in asset-review based actions.
checklistItems	`string[]`	A list of the checklist items that is assigned to asset review task. Along with `checklistName` this filed makes up the checklist.
daysToDueDate	`int`	An optional parameter of any task-based activities, that defines how many days an assignee has to do the assigned task. The due date is calculated at the moment of task created plus this number of days.
allowDownload	`boolean`	When true allows review task assignee to download assets original files.
isSimpleMode	`boolean`	When true, the review task's UI will be simplified to contains only minimal set of controls like Zoom, Pan, Accept/Reject.
isCustomPayload	`boolean`	
webhookURL	`string`	
webhookRequestType	`string`	
webhookContentType	`string`	

Field	Type	Description
webhookHeaders	`key-value}[]`	
webhookFields	`string[]`	
webhookCustomPayload	`string`	
contacts	`string`	
targetKanbanColumnUid	`string`	

Workflow fields per reactions

Reaction	Fields
AssignUserRoles	RoleUids, UserUids, daysToDueDate
CreateInternalUploadAssetsTask	RoleUids, UserUids, DaysToDueDate


```
{
    "uid": "uqag",
    "comment": "comment",
    "roleUids": [
      "projectOwners"
    ],
    "userUids": [
      "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
    ],
    "triggerType": "ProjectCreated",
    "friendlyName": "Upload Ref Docs",
    "reactionType": "CreateInternalUploadAssetsTask",
    "daysToDueDate": 3
  }
```


CreateInternalUploadRefDocsTask	RoleUids, UserUids, DaysToDueDate


```
{
    "uid": "uqag",
    "comment": "comment",
    "roleUids": [
      "projectOwners"
    ],
    "userUids": [
```

```
    "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
  ],
  "triggerType": "ProjectCreated",
  "friendlyName": "Upload Ref Docs",
  "reactionType": "CreateInternalUploadRefDocsTask",
  "daysToDueDate": 3
}
```

CreateInternalUploadAssetsTask	RoleUids, UserUids, DaysToDueDate

```
{
  "uid": "uqag",
  "comment": "comment",
  "roleUids": [
    "projectOwners"
  ],
  "userUids": [
    "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
  ],
  "triggerType": "ProjectCreated",
  "friendlyName": "Upload Ref Docs",
  "reactionType": "CreateInternalUploadAssetsTask",
  "daysToDueDate": 3
}
```

CreateExternalUploadAssetsTask	RoleUids, UserUids, DaysToDueDate, subject, contacts, password,

```
{
  "uid": "uqag",
  "comment": "Comment",
  "subject": "Optional subject",
  "contacts": "email:email@email.com, email:email2@email.com",
  "password": "Optional password",
  "roleUids": [
    "projectOwners"
  ],
  "triggerType": "ProjectCreated",
  "friendlyName": "aaaq",
  "reactionType": "CreateExternalUploadAssetsTask",
  "daysToDueDate": 3
}
```

CreateExternalUploadRefDocsTask	RoleUids, UserUids, webhookHeaders, Subject, Contacts, Password


```
{
    "uid": "uqag",
    "comment": "Comment",
    "subject": "Optional subject",
    "contacts": "email:email@email.com, email:email2@email.com",
    "password": "Optional password",
    "roleUids": [
      "projectOwners"
    ],
    "triggerType": "ProjectCreated",
    "friendlyName": "aaaq",
    "reactionType": "CreateExternalUploadRefDocsTask",
    "daysToDueDate": 3
  }
```


SendDataWithWebhook	WebhookUrl, WebhookFields, IsCustomPayload, WebhookContentType, WebhookRequestType, WebhookCustomPayload


```
{
  "uid": "uqag",
  "webhookUrl": "https://url.com",
  "triggerType": "ProjectCreated",
  "friendlyName": "Send Data With Webhook",
  "reactionType": "SendDataWithWebhook",
  "webhookFields": [
    "projectUID",
    "projectState",
    "projectCreateDate",
    "projectDescription",
    "projectDueDate",
    "projectCustomer",
    "projectRevision",
    "projectTags",
    "projectOwnersUIDs",
    "projectAssetsUIDs",
    "projectAssetsThumbnails",
    "projectAssetsReviewStatus",
    "projectTemplateName",
    "projectWorkflowName",
    "projectName",
    "projectDesign",
```

```
    "projectProject",
    "projectOwnersNames",
    "projectAssetsNames",
    "projectAssetsURLs",
    "projectAssetsStatus",
    "projectTemplateUID",
    "projectWorkflowUID"
  ],
  "webhookHeaders": [
    {
      "name": "Header1",
      "value": "Value1"
    }
  ],
  "isCustomPayload": false,
  "webhookContentType": "application/json",
  "webhookRequestType": "POST|GET|PUT"
}
```

or

```
{
  "uid": "uqag",
  "webhookUrl": "url",
  "triggerType": "ProjectCreated",
  "friendlyName": "aaaq",
  "reactionType": "SendDataWithWebhook",
  "isCustomPayload": true,
  "webhookContentType": "application/json",
  "webhookRequestType": "POST",
  "webhookCustomPayload": "Any payload"
}
```

SendCustomEmail	Contacts, RoleUids

```
{
  "uid": "uqag",
  "contacts": "email:address@mail.com",
  "roleUids": [
    "projectOwners"
  ],
  "triggerType": "ProjectCreated",
  "friendlyName": "Send Custom Email",
  "reactionType": "SendCustomEmail"
}
```

SwitchWorkflow	TargetWorkflowUid

```
{
  "uid": "uqag",
  "triggerType": "ProjectCreated",
  "friendlyName": "Switch To Workflow",
  "reactionType": "SwitchToWorkflow",
  "targetWorkflowUid": "YYYYYYYYYYY"
}
```

CompleteProject	-

```
{
  "uid": "uqag",
  "triggerType": "InternalReviewTaskRejected",
  "friendlyName": "aaaq",
  "reactionType": "CompleteProject"
}
```

SendRefDocsByEmail	Contacts, DaysToDueDate

```
  {
    "uid": "uqag",
    "comment": "Comment xxxx",
    "contacts": "email:mailgmail.com",
    "roleUids": [
      "projectOwners"
    ],
    "triggerType": "InternalReviewTaskRejected",
    "friendlyName": "Send Reference Docs By Email",
    "reactionType": "SendRefDocsByEmail",
    "daysToDueDate": 3
  }
```

CreateInternalReviewTask	RoleUids, UserUids, ChecklistName, DaysToDueDate, ChecklistItems, TriggerRoleUid

```
{
  "uid": "uqag",
  "comment": "Comment",
  "roleUids": [
    "projectOwners"
```

```
  ],
  "userUids": [
    "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
  ],
  "triggerType": "InternalReviewTaskRejected",
  "friendlyName": "aaaqa",
  "reactionType": "CreateInternalReviewTask",
  "checklistName": "CH",
  "daysToDueDate": 3,
  "checklistItems": [
    "A1",
    "A2"
  ],
  "triggerRoleUid": "projectOwners"
}
```

CreateExternalReviewTask	Comment, Subject, Contacts, Language, Password, IsSimpleMode, AllowDownload, DaysToDueDate

```
{
  "uid": "yi59",
  "comment": "Custom task comment",
  "subject": "Custom subject",
  "contacts": "email:oleg.gladun@hitech-ua.com",
  "language": "en",
  "password": "SecurePassword",
  "triggerType": "InternalReviewTaskRejected",
  "isSimpleMode": true,
  "reactionType": "CreateExternalReviewTask",
  "allowDownload": true,
  "daysToDueDate": 3,
  "triggerExecutorUid": "F84D43FF0BAE469595F695C1D3CAA9FC" or  "triggerRoleUid":
}
```

Workflow management

Method/Path	Description
`GET /api/v1/workflows`	Returns all webhooks for the current user's tenant.
`GET /api/v1/workflows`	Returns all webhooks for the current user's tenant.
`POST /api/v1/workflow`	Sets us a new webhook.

Method/Path	Description
`DEL /api/v1/workflow`	Deletes a webhook.
`PUT /api/v1/webhook/test`	Test a webhook endpoint.

GET /api/v1/workflows

Returns list of the all workflows available.

Request URL

```
https://api.approval.studio/api/v1/workflows
```

Curl:

```
curl -X 'GET' 'https://api.approval.studio/api/v1/workflows'
```

Response:

```
{
    "workflows": [
      {
        "workflowUID": "XXXX",
        "name": "string",
        "created": "2025-12-20T18:39:10.913Z"
      },
      {...}
    ]
}
```

GET /api/v1/workflow

Returns a complete workflow object instance by a given workflow UID.

Request URL

```
https://api.approval.studio/api/v1/workflow?WorkflowUID=XXXXXXX
```

Curl:

```
curl -X 'GET' 'https://api.approval.studio/api/v1/workflows'
```

Result:

```
{
    "actions": [ // List of actions.
      {
        "uid": "string",
        "friendlyName": "string",
        "triggerType": TriggerType,
        "reactionType": ReactionType,
        "userUIDs": [
          "string"
        ]
        "roleUids": [
          "string"
        ],
        "triggerExecutorUid": "string",
        "triggerRoleUid": "string",
        "triggerEmails": "string",
        "triggerRoleIds": [
          "string"
        ],
        "emailSubject": "string",
        "emailLanguage": "string",
        "password": "string",
        "comment": "string",
        "checklistName": "string",
        "checklistItems": [
          "string"
        ],
        "daysToDueDate": 0,
        "allowDownload": true|false,
        "isSimpleMode": true|false,
        "isCustomPayload": true|false,
        "webhookURL": "string",
        "webhookRequestType": "string",
        "webhookContentType": "string",
        "webhookHeaders": [
          {
            "Header": "Value",
            [...]
          }
        ],
        "webhookFields": [
          "string"
        ]
        "webhookCustomPayload": "string",
```

```
        "contacts": "string",
        "targetKanbanColumnUid": "string"
      }
    ],
    "roles": [ // List of roles.
      {
        "uid": "XXXX",
        "name": "string",
        "isInternal": true|false
      }
    ]
  }
}
```

Responses

HTTP Code	Response
200	**Success**. List of workflows.
400	**Error**: Parameters' validation failed.
	See HTTP code 400 description.
404	**Error**: A requested workflow not found, `workflowUID` parameter is invalid.

POST /api/v1/workflow

Creates a new active workflow.

Curl:

```
```bash
curl -X 'POST' \
  'https://api.approval.studio/api/v1/workflow' \
  -H 'accept: text/plain' \
  -H 'Content-Type: application/json-patch+json' \
  -d '{
  "clientUID": "XXXXXXXXX", // Client UID that this workflow will be belong to.
  "name": "Name",           // Optioanl workflow name.
  "actions": [              // List of actions.
    {
        ...
    },
    [...]
```
```

```
  ],
  "roles": [                // Optional list of roles.
    {
      ...
    }
  ]
}'
```

Result:

```
{
    "workflowUID": "XXXXXX",
    "name": "Name",
    "created": "2025-12-20T20:06:21.703Z",
    "actions": [ // List of actions.
      {
        ...
      }
    ],
    "roles": [ // List of roles.
      {
        ...
      }
    ]
  }
}
```

Responses

HTTP Code	Response
200	**Success**. List of workflows.
400	**Error**: Parameters' validation failed.
	See HTTP code 400 description.
412	**Error**: `ClientUID` parameter is invalid, no active client with the given ID found for this user.

PUT /api/v1/workflow

Updates (edits) a given workflow.

Curl:

```
curl -X 'PUT' \
  'https://api.approval.studio/api/v1/workflow' \
  -H 'accept: text/plain' \
  -H 'Content-Type: application/json-patch+json' \
  -d '{
  "name": "Name",           // Optioanl workflow name.
  "actions": [              // List of actions.
    {
        ...
    },
    [...]
  ],
  "roles": [                // Optional list of roles.
    {
      ...
    }
  ]
}'
```

Result:

```
{
    "workflowUID": "XXXXXX",
    "name": "Name",
    "created": "2025-12-20T20:06:21.703Z",
    "actions": [ // List of actions.
      {
        ...
      }
    ],
    "roles": [ // List of roles.
      {
        ...
      }
    ]
  }
}
```

Responses

HTTP Code	Response
200	**Success**. List of workflows.

HTTP Code	Response
400	**Error**: Parameters' validation failed.
	See HTTP code 400 description.
404	**Error**: A requested workflow not found, `workflowUID` parameter is invalid.

DELETE /api/v1/workflow

Deletes (marks as deleted) a given workflow.

Curl:

```
curl -X 'DELETE' \
  'https://api.approval.studio/api/v1/workflow' \
  -H 'accept: text/plain' \
  -H 'Content-Type: application/json-patch+json' \
  -d '{
  "workflowUID": "string"
}'
```

Responses

HTTP Code	Response
200	**Success**. List of workflows.
400	**Error**: Parameters' validation failed.
	See HTTP code 400 description.
404	**Error**: A requested workflow not found, `workflowUID` parameter is invalid.

POST /api/v1/project/workflow/run

Runs a manual action of the workflow, assigned to a given project.

Curl:

```
curl -X 'POST' \
  'https://api.approval.studio/api/v1/project/workflow/run' \
  -H 'accept: text/plain' \
```

```
  -H 'Content-Type: application/json-patch+json' \
  -d '{
  "projectUID": "string"
}'
```

Responses

HTTP Code	Response
200	**Success**. Workflow run triggered.
400	**Error**: Parameters' validation failed.
	See HTTP code 400 description.
404	**Error**: A requested project not found, `workflowUID` parameter is invalid.
412	**Error**: No workflow assigned on this project.
412	**Error**: No any manual running action is in the list of the workflow's actions for this project.