PROJEKT Justitia 4.0

API Justitia.Swiss for Developers

Version: - 27

Projektphase: M3

Erstellungs-Datum: 07.12.2023

Autor: Dominik Auf der Maur, Franz Achermann

Klassifizierung: intern

Status: Draft

Table of Contents

PROJEKT
Justitia 4.0

This document is intended for (external) developers that need to access the API justitia.swiss (Justitia 4.0). In case of questions, please contact franz.achermann@kkjpd.ch.

1 Testing Environment

For integration testing of external integrators via API, the TRAIN environment is provided:

`https://j40-train.elca-cloud.com/`

Note: This URL will change in the future

As a precondition, the integrator needs to have whitelisted his source IP range with the infrastructure provider, ELCA. In order to be whitelisted, please contact franz.achermann@kkjpd.ch

The integrator is expected to register a technical API user for his profile via UI, as explained in API and Security.

> **Data persistency**
>
> As of now (December 2023), the test data created via API is deleted with each re-deployment of the platform. Migration towards a data preserving strategy is planned for Q1/2024.
> For this reason, don't plan for test runs that span a long period of time. Also, we advise to use test automation so that integration tests against the API can be re-run quickly, reproducibly and cost effectively.

1.1 Terms of Use

The following terms of use apply to the TRAIN environment:

- ⚠️ Do not use productive data on test environments!
- Do not perform load testing on the TRAIN environment. The TRAIN environment is intended for functional tests, not for performance related testing.
- Do not perform penetration testing on TRAIN. You may be blocked or blacklisted, making the platform unavailable to you.
- Till end of January 2024:
    - TRAIN might be cleared when new software versions are deployed.
    - No strict API Versioning, API changes on short notice

2 Tools

The OpenAPI 3.0 specification is available https://j40-train.elca-cloud.com/swagger-ui/index.html.

For the development of API clients, all the well-known **REST client tooling** can be used, depending on the client development environment. Examples of such tools are:

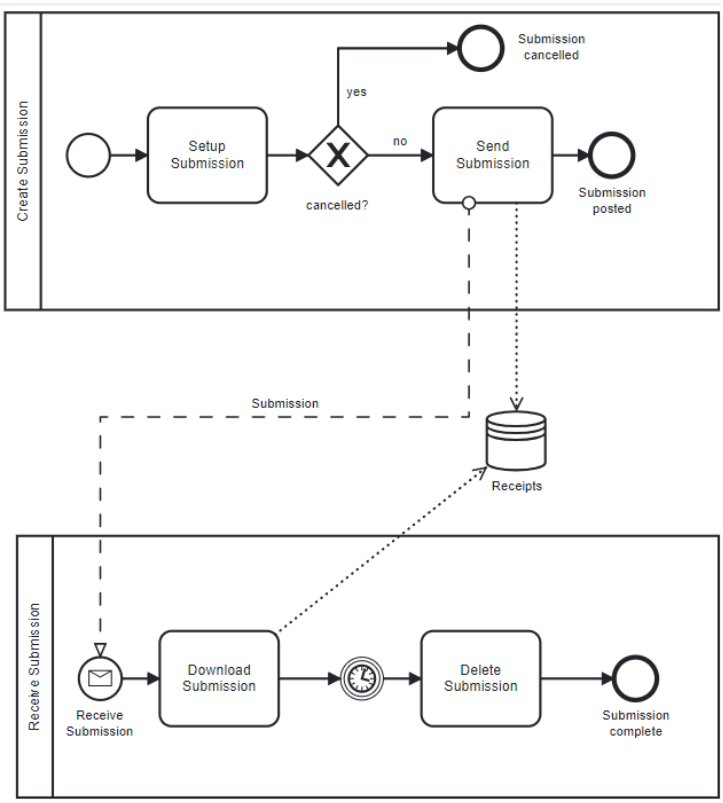
- Java: HTTP Client (built-in with Java 11 and above), Spring WebClient, Apache HttpClient, Square OkHttpClient; in addition, you will require a JSON serializer/deserializer such as Jackson.
- C#: HTTP Client (built-in), RestSharp, Refit
- Python: requests library (built-in), httpx,

If you are relying on client SDK generators such as Swagger Codegen, make sure the generated code complies with the backward compatibility rules defined in API Versioning.

3 API Use Cases

This chapter the necessary API calls for the business use cases of the platform.

3.1 Submission



3.1.1 Create Submission (All Participants)

3.1.1.1 Setup Submission

Step

Retrieve all authority profiles as possible recipient of the submission
`curl --location 'https://j40-train.elca-cloud.com//api/participant/v1/authorities' \`
`--header 'accept: application/json' \`
`--header 'Authorization: Bearer *****'`
 	
In the response, you find your desired receiver, more precisely, the UUID of it.

Step

Create the new submission.
`curl --location 'https://j40-train.elca-cloud.com//api/document-exchange/v1/submissions' \`
`--header 'accept: application/json' \`
`--header 'Content-Type: application/json' \`
`--header 'Authorization: Bearer *****' \`
`--data '{`
`  "subject": "myTest",`
`  "recipientProfile": "6cfd0c47-8718-4582-b22c-700f81b9272a"`
`}'`
 	
In the response, you find the UUID of your created submission.

Add a file to the submission. Provide the proper UUID of the submission from the previous step.
`curl --location 'https://j40-train.elca-cloud.com//api/document-exchange/v1/submissions/e9e553f5-6d4e-4d49-b1f3-7783b0017224/documents' \`
`--header 'accept: application/json' \`
`--header 'Authorization: Bearer *****' \`
`--form 'file=@"/tmp/myfile.pdf"'`
 	
In the response, you finde the UUID of the uploaded document. This id is needed to modify the metadata of the document.

Read the current submission:
`curl --location 'https://j40-train.elca-cloud.com//api/document-exchange/v1/submissions/e9e553f5-6d4e-4d49-b1f3-7783b0017224' \`
`--header 'accept: application/json' \`
`--header 'Authorization: Bearer *****'`
 	
The response will contain the details of the submission as well as the details of all documents. Note the version of the submission, which is used for the optimistic locking.

Send the current submission. You need to replace the version with the current version of the submission, as retrieved in the previous step.
`curl --location --request PUT 'https://j40-train.elca-cloud.com//api/document-exchange/v1/submissions/e9e553f5-6d4e-4d49-b1f3-7783b0017224/status' \`
`--header 'accept: application/json' \`
`--header 'Content-Type: application/json' \`
`--header 'Authorization: Bearer *****' \`
`--data '{`
`  "status": "SUBMITTED",`
`  "version": 3`
`}'`

3.1.2 Receive Submission (only authorities)

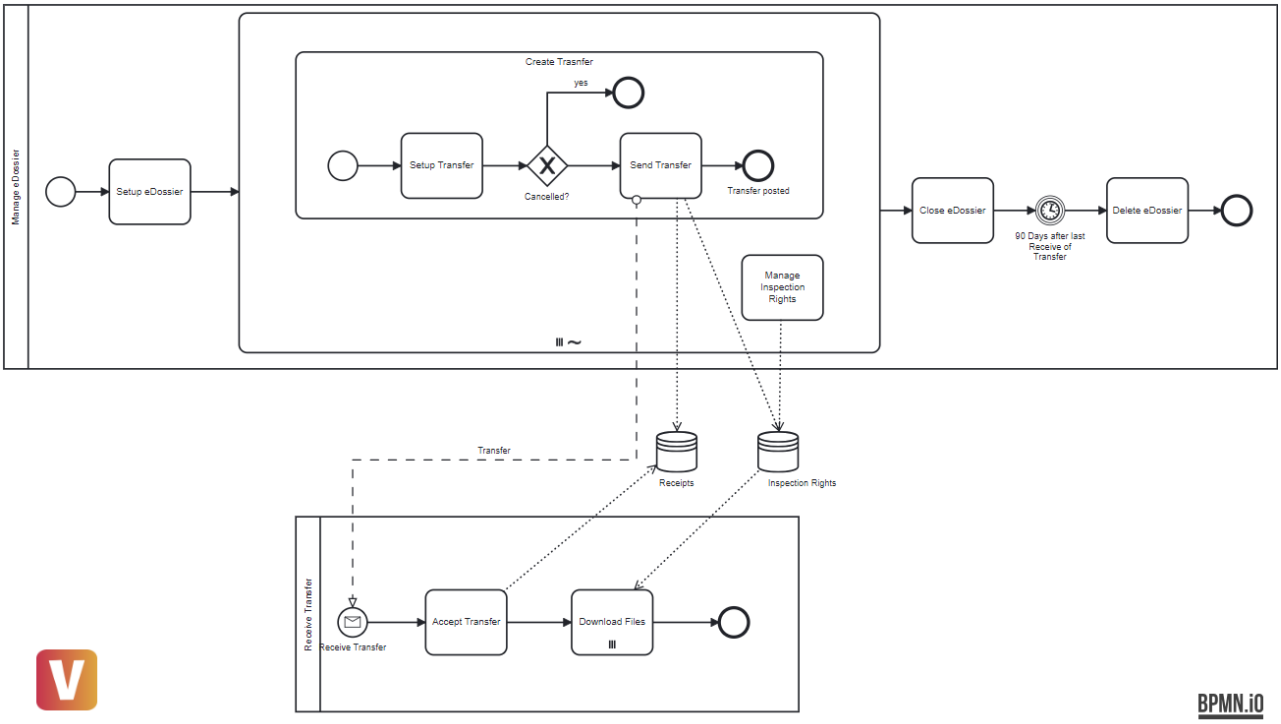
3.1.2.1 Download Submission

Step	API Call
Get all the new transmission (by filtering the inbox on status 'SUBMITTED') Note: the submissions have the same type 'submission'	GET /api/document-exchange/v1/transmissions/inbox
Set the state of the submission to 'RECEIVED' in order to download the contents.	PUT /api/document-exchange/v1/submissions/{submission-id}/status
Get the the details of the submission	GET /api/document-exchange/v1/submissions/{submission-id}
Get a file of the submission	GET /api/document-exchange/v1/submissions/{submission-id}/documents/{document-id}
Get all files of the submission	GET /api/document-exchange/v1/submissions/{submission-id}/all-documents

Note: after 90 days, the transmission will be deleted automatically.

On the web-portal, setting the state to 'RECEIVED' is done behind the scenes by the application.

3.2 Transfer



3.2.1 Manage eDossier (only authorities)

3.2.1.1 Setup eDossier

Step	API Call
Create a new dossier	PUT /api/document-exchange/v1/dossiers
Retrieve (selected) dossiers	GET /api/document-exchange/v1/dossiers

3.2.1.2 Setup Transfer

For a given Dossier (with the dossierId) transfers may be initiated and files uploaded to the dossier.

Manage the dossier itself

Step	API Call
Add a file to the dossier:	
• Upload the file	POST /api/document-exchange/v1/dossiers/{dossier-id}/documents
• Provide Metadata of the file (e.g. `displayName` or your `refKey` to the document) Note: Metadate of files that are already part of a transfer, cannot be changed.	PUT /api/document-exchange/v1/dossiers/{dossier-id}/documents/{document-id}
Get the the details of the dossier (dossier metadata and files metadata)	GET /api/document-exchange/v1/dossiers/{dossier-id}

Manage the transfer

by referring to the files (documentId) of the dossier (dossierId)

Step	API Call
Lookup the `profileId` of the recipient	GET /api/participant/v1/profiles
Create a new transfer by providing the `profileId` ot the recipient.	POST /api/document-exchange/v1/transfers
Link files (of the dossier) to the transfer	PATCH /api/document-exchange/v1/transfers/{transfer-id}/document-grant
Get the the details of the transfer	GET /api/document-exchange/v1/transfers/{transfer-id}
Set the validity period for this recipient	PATCH /api/document-exchange/v1/dossiers/{dossier-id}/permissions/grantees/{grantee-id}
Cancel (delete) the transfer	DELETE /api/document-exchange/v1/transfers/{transfer-id}

3.2.1.3 Send Transfer

Step	API Call
Send a transfer by updating the status to 'SUBMITTED'	PUT /api/document-exchange/v1/transfers/{transfer-id}/status
Retrieve 'Eingabequittung' after sending the transfer, i.e. by providing status `SUBMITTED`	GET /api/document-exchange/v1/transfers/{transfer-id}/receipts/{status}

3.2.1.4 Manage Inspection Rights

Step	API Call
Retrieve all active grantees (`profileId`) of this dossier	GET /api/document-exchange/v1/dossiers/{dossier-id}/permissions
Set the validity period for this recipient. By setting the `validUntil` Date to the past, the receipient is removed.	PATCH /api/document-exchange/v1/dossiers/{dossier-id}/permissions/grantees/{grantee-id}

3.2.1.5 Close eDossier

Step	API Call
Verify if this dossier may be deleted (i.e. it does not contain any not-yet received transfers)	GET /api/document-exchange/v1/dossiers/{dossier-id}/deletability
Delete this document	DELETE /api/document-exchange/v1/dossiers/{dossier-id}

3.2.2 Receive Transfer (All Participants)

3.2.2.1 Accept Transfer

Step	API Call
Get all the new transmission (by filtering the inbox on status SUBMITTED) Note: the transfer have the same type `TRANSFER`	GET /api/document-exchange/v1/transmissions/inbox
Accept this transfer by passing the status 'RECEIVED'	PUT /api/document-exchange/v1/transfers/{transfer-id}/status

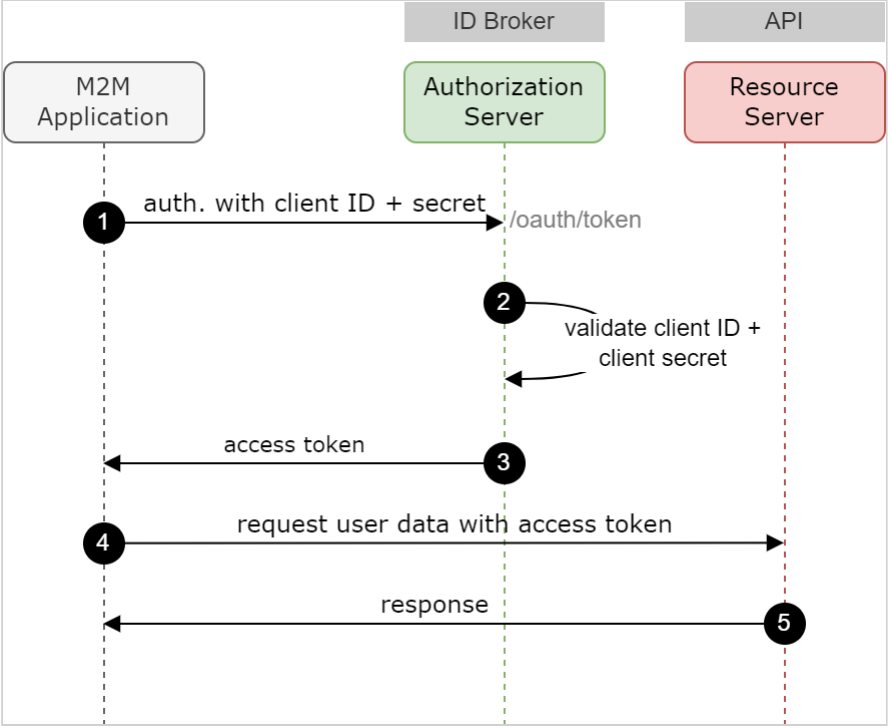
3.2.2.2 Download Files

Step	API Call
Get the the details of the transfer	GET /api/document-exchange/v1/transfers/{transfer-id}
Get receipts of the transfer	GET /api/document-exchange/v1/transfers/{transfer-id}/receipts/{status}
Get a file of the dossier the transfer relates to	GET /api/document-exchange/v1/dossiers/{dossier-id}/documents/{document-id}
Get all files of the transfer	GET /api/document-exchange/v1/transfers/{transfer-id}/all-documents

4 API and Security

4.1 Authentication

Each human and technical user that accesses the APIs provided by the "Justitia.swiss" platform must be authenticated. For regular users, the authentication is conducted by the SPA. For technical API users, we provide functionality via UI (accessible to the organisation admin) to register a specific OIDC client on the platform's Keycloak broker using dynamic client registration. Once set up, the API technical user can then use this client to authenticate with Keycloak using the **OIDC Client Credentials** flow.



1. *The application authenticates with the Authorization Server using its Client ID and Client Secret (`/oauth/token` endpoint).*
2. *The Authorization Server validates the Client ID and Client Secret.*
3. *The Authorization Server responds with an access token.*
4. *The application can use the access token to call an API on behalf of itself.*
5. *The API responds with requested data.*

Note: The Client Credentials flow assumes that the integrator is able to securely store the credentials obtained when registering the technical user via UI. Since the Justitia.Swiss platform does not store the credentials, a new API technical user has to be generated in case the credentials are compromised or lost.

The following call gets the access token server, client_id and client_secret.

```
curl --location 'https://j40-train.elca-cloud.com/auth/realms/JustitiaPlatform/
protocol/openid-connect/token' \
--header 'Content-Type: application/x-www-form-urlencoded' \
--data-urlencode 'grant_type=client_credentials' \
--data-urlencode 'scope=openid profile roles' \
```

```
--data-urlencode 'client_id=****' \
--data-urlencode 'client_secret=****'
```

After authentication, the API clients must send the **JWT** obtained before as a **bearer token** in each request (stateless).

The access token returned as authentication result has an expiry. When expired, a new access token has to be obtained by another Client Credentials authentication request to the Keycloak broker.

4.2 Authorization

Authorization of technical API users is **identical to regular UI users**:

- Each technical API user is linked to exactly one profile.
- A technical API user has access to all ordinary business functionality, but not to organisation management related features. These are available to the organisation administrator via UI only.

4.3 Rate Limiting

The following rate-limiting for specific API endpoints will be employed:

- Number of requests (based on leaky bucket algorithm)
- Number of requests to APIs by a given count per time (fixed window algorithm)
- Number of concurrent requests to your services

The specific rates are determined based on load tests on the platform as this gives a good indication on how many requests the platform can handle. This can be further adapted during the operation of the platform.

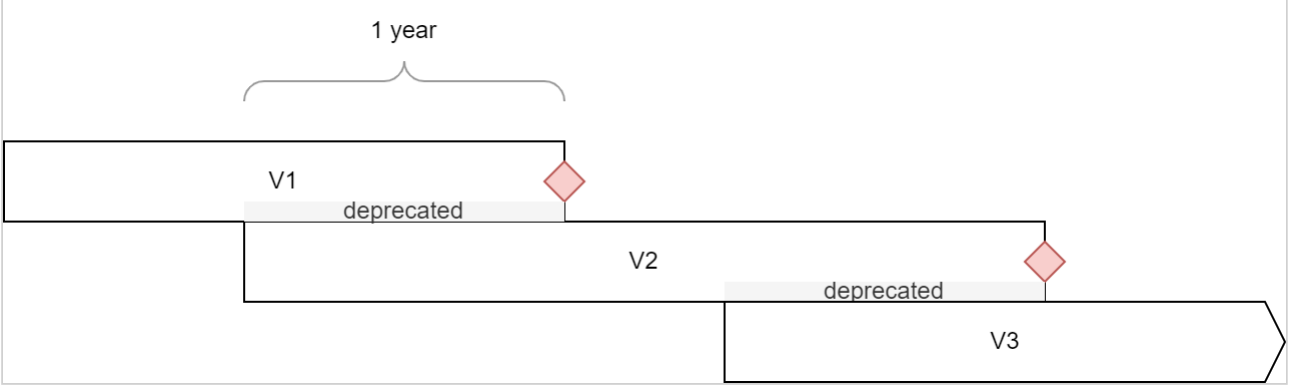
4.4 Blocking of Requests

In addition to not exposing internal APIs, we will also explicitly block them in the API gateway.

5 API Versioning

5.1 Rationale and Goals

External integrators cannot be forced to do coordinated "big bang" migrations when the platform API changes in a non-backward-compatible way. For this reason, we version the API and support up to two versions of the platform API simultaneously. The following picture shows the typical scenario.



Once a new API version is introduced, the existing one is deprecated but provided for 1 more year. During this period, the integrators are expected to migrate to the new API version.

Note that during the lifetime of an API version, there may be backward compatible changes to the API. For this reason, it is important to implement API clients according to the compatibility rules described below.

5.2 Platform API Policies

Deprecation

Interfaces or parts thereof (operations, data elements, ...) that are deprecated are documented. In OpenAPI, the keyword deprecated is used for this purpose. The documentation explains the reasons and points to the "new" (replacement) mechanism or data element.

Decommissioning

A deprecated interface is still supported for 1 year before it is decommissioned, i.e., taken out of operation.

5.3 Compatibility Rules

The **platform client** must be built in a **forward compatible** way since certain changes on the platform side may be activated without special notice. In detail, the following changes must be accepted and ignored by the consumer:

- Added REST services must be ignored
- Additional properties in JSON response objects must be ignored ("tolerant reader" pattern)
- Depending on case: unknown enum values in response should be ignored ("tolerant reader" pattern)
  Remark: Whether or not this type of change can be interpreted as backward compatible depends on the concrete case and will be mentioned in the documentation.
- Additional enum values used as input parameters must be acceptable
  Remark: However, re-interpretation of existing enum values is not backward compatible.
- Added optional query parameters must be ignored
- Removed input parameters or properties on input objects must not break the client

Code Generators

Note that client SDK generators such as Swagger Codegen in general do a bad job at generating forward compatible libraries. E.g., Swagger Codegen for Java does not even ignore additional response properties.

In contrast, the client is not expected to accept changes such as:

- Removed mandatory property in response
- Additional mandatory input parameter
- Property renaming

Such types of changes will be introduced in a new major API version only.

Semantics and Compatibility

The above rules are simple to understand and guarantee compatibility on a syntactic level. In reality, the more difficult questions about compatibility have to be asked on higher semantic levels. We try to convey as many of those business rules as possible in the description. When deciding whether or not a change platform API change is breaking and should lead to a major version change, we do not focus on syntax alone. Instead, semantic changes (e.g. using a field in a different way) may also lead to a major API version update.

5.4 Technical Implementation of Versioning

We implement **URI Path Versioning**: REST resources are versioned by introducing a major version number in the URI path, directly after the domain name:

`https://hostname/api/domain/Vn/...`

A concrete example would be:

`https://justita-swiss.ch/api/participant/V1/profiles/2b0937d2-f165-450c-b947-f151f111c953`

The version is updated if and only if a change is breaking.

The **scope** of one version is a domain, so there is a separate version for the "participant" and "document-exchange" parts of the API.

5.5 References

The approach presented here is one of the best practices, followed by large players such as SalesForce. There are many resources on the Web, some of them are:

- https://blog.hubspot.com/website/api-versioning
- https://restfulapi.net/versioning/
- https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design#uri-versioning

5.6 Implementation Hints

Ignore added REST services

No special measures necessary. With or without generated client SDKs, the additional operations will just remain unused.

Igore additional properties in response

Usually, the object mapper can be configured to ignore unknown properties on de-serialization.

Ignore unknown enum values in response or input parameter objects

Just ignoring unknown enum values is the wrong choice in many situations. If the consumer expects to handle all enum cases exhaustively, ignoring new cases would lead to undetected failures.

Solution A: Map the field to a string instead of an enum on the client side, even if the field is declared as an enum type in OpenAPI. In the client logic, do the appropriate validation and throw an error if unknown values are not acceptable.

Solution B: Map the field to an Enum type, but parse the JSON string in such a way that all unknown values are mapped to an Unknown value defined in the Enum. In the client logic, handle Unknown appropriately.

Note: Standard code generators such as Swagger Codegen generate Enums without further measures and are therefore not tolerant!

Don't fail due to additional enum values in input

No special measures necessary. With or without generated client SDKs, the additional enum values will just remain unused.

Don't fail due to additional (optional) input parameters

No special measures necessary. With or without generated client SDKs, the additional parameters will just remain unused.

Don't fail due to removed input parameters or input properties

We assume that removing the input parameter does not affect functionality. E.g., we will not remove a filter parameter such as limit and offset used for paging and then just return all data.

The client may emit excess parameters, but since the platform also respects the "tolerant reader" pattern, this data is just ignored.

6 Frequently asked API-Questions

6.1 Update Status

Q: Why don’t you use patch method to update status ?

- You create a specific status resource for managing it.
- The status is one property of the resource submission or Transfer
- Why don’t you use the patch verb to update the status like other patch method on others ressources

A: It’s a tradeoff of readability and understandability on the swagger.

6.2 Delete Dossier

Q: Why don’t you use the OPTION verb to know if the resource can be deleted ?

- On the dossier resource you define a delectability sub resource to know if the dossier can be deleted
- The OPTION verb can be used to get info about action that can be allowed on a resource

A: It’s a tradeoff of readability and understandability on the swagger.

Q: Could we understand that the deletion of one dossier is only scheduled for a latter batch deletion ?

A: We need to keep the document (according to E-BEKJ) for 90 days, hence 'closing' a dossier makes it immutable and the physical deletion happens in a batch..

6.3 Delete Document from Submission

Q: Why is there no 'delete' endpoint to delete a single file from a dossier:

A: we expect the client system to send the right files. More precisely, the we assume the business use case is only started by the client, when all the files are known to the client system.

6.4 Get all documents

Q: the method used to retrieve all document on a transfer or submission should use an additional resource:

- Why don’t you use the standard url (/api/document-exchange/v1/submissions/{submission-id}/documents) with a specific accept header
- This allow client to request resource in his preferred format. It can put "accept : application/json" http header to get the json representation of the list of documents or "accexp: application/zip" http header to get the zip representation of all documents"

A: it might be the case that we put other information (like structure infos) into a zip, hence to encode ‘what the clients’ expects seems more clear.

6.5 Handling and Generation of UUIDs

Q: All resource identifier are uuids. Why do you use the POST method to create ressources (like submission or document in submission …). The post method is not idempotent and in case of error during creation, if we have to send another request we potentially will create a duplicated ressources. Why don’t you use directly the PUT verb to create resources and let the client set the unique identifier. You cans almost respond with the 201 status for

resource creation or 200 for resource modification. In case of network pb, if we have to resend the request we don't create a new ressources.

A: for security reasons we want to 'invent' the uuid on the platform side to make it harder for adversaries to guess uuids. The api allos the 'refKey' on documents to keep your own technical identifier. All the other uuids that the model uses, does not need to be persisted for a long time on the client side. The only key introduced by the platform is the delivery address.

6.6 Get Authorities

Q: Why is there no pagination on authorities services? (/api/participant/v1/authorities

A: Contrary to the inbox, the list of authorities is (for the time being) small enough.

6.7 Versioning

Q: Some resources have a version property. How this version is managed ?

A: The versioning is used for optimistic locking wrt. the Web-interface. Whenever you update a (patch) a resource, you need to pass the proper version, in order to *show* that you know the current state of the ressource. For instance, when you send a submission, it must be of the proper 'version.

```
Get_Submission (Id = e9e553f5-6d4e-4d49-b1f3-7783b0017224)  returns:

{
    "id": "7b239523-fe70-47bb-965a-5a02eddd7fbd",
    "version": 1,
    "subject": "Test mit Thomas",
    "creatorProfileId": "479c9954-bf08-410c-aae8-7cf6988a7d1a",
    "recipientProfileId": "6cfd0c47-8718-4582-b22c-700f81b9272a",
    "status": "DRAFT",
    "createdAt": "2023-11-30T16:27:35.576Z",
    "documents": [ ... ]}
```

Note that the version is 1. When we try to send this submission (using PUT submissions/{{submission_id}/status), we need to pass this version number. Otherwise we would get an 409 Conflict error.