The Qvarn API

Suomen Tilaajavastuu Oy

0.82+git

Contents

Chapter 1

Introduction

This document describes the application programming interface (API) of resource data storage backend provided by Suomen Tilaajavastuu Oy. The backend stores and provides controlled access to data used by various services provided by Tilaajavastuu and its partners. The data may be raw data, or the result of analysis and refinement.

The goal is to document the following:

- Functional and non-functional requirements of the backend.
- The data model for the data stored by the backend.
- The application programming interface to the data.

There are several intended uses for this document:

- Documentation for **partners of Suomen Tilaajavastuu Oy**, who need to use the data via the API to develop their own services.
- Documentation for **developers at Suomen Tilaajavastuu Oy**, who need to develop services or to implement the API backend.
- An **automated acceptance test suite** to specify what functionality is required from the services being produced.
- A **vehicle for communication** between stakeholders about what the services do, and for planning changes.

The intended audiences are:

- **Product management** at Suomen Tilaajavastuu Oy, and partners.
- **Product development** at Suomen Tilaajavastuu Oy, and partners.

All readers are expected to have a basic understanding of computing technology, but are not expected to understand software development.

1.1 Status and scope of this document version

The API documented in this version is implement and available for internal use by Tilaajavastuu. Access to it will be provided later to partners.

1.2 Use as an acceptance test suite

If you're not a developer, you can skip this section.

This document is also an automated test suite for the API, using a testing tool called Yarn. The document source code is given to Yarn, which runs the test code embedded in the document. Yarn is available from http://liw.fi/cmdtest/ and is packaged for Debian. You need at least version 0.16.

The tests in this document are written to be run against an existing, running instance of the API. The test suite itself **does not start the API instance**. The location of the API instance to test is given to the Yarn tool with the **`--env`** option:

EXAMPLE

```
yarn --env API_URL=http://localhost:12765/ *.yarn
```

1.3 History and background

Qvarn started as an internal project in 2015 at Suomen Tilaajavastuu (https://www.tilaajavastuu.fi/en/), a company that provides various reproting services to the Finnish construction industry. Later that year the Swedish Construction Federation (https://www.sverigesbyggindustrier.se/english) adopted Qvarn as a platform for ID06 identity card services in Sweden. From the start, a goal for Qvarn has been to make it easy to comply with the EU General Data Protection Regulation.

In 2016 Qvarn spun off to its own company, QvarnLabs Ab, which provides consulting, development, and support services around Qvarn.

1.4 Legalese

Chapter 2

Requirements

This chapter describes some of the requirements for the API backend, with an emphasis on non-functional requirements. Functional requirements are described primarily as automated tests in later chapters.

2.1 Performance

Performance requirements have not been quantified. Backend performance must be sufficient to handle all the traffic generated by all the users.

2.2 Scalability

Scalability requirements have not been quantified. The architecture design must be such that it can be expected that the system scales to become to serve all of Europe. There should be as few architecture level bottleneck as possible.

2.3 Reliability and availability

Reliability requirements have not been quantified. There are some hard requirements such that if they are not met, there can be real-world consequences up to and including jail time (but typically only reminder letters from authorities).

2.4 Security

Security requirements have not been quantified. The system contains a large amount data that can be sensitive, and security must be high.

Security is audited regularly by external specialists.

Chapter 3

Architecture overview

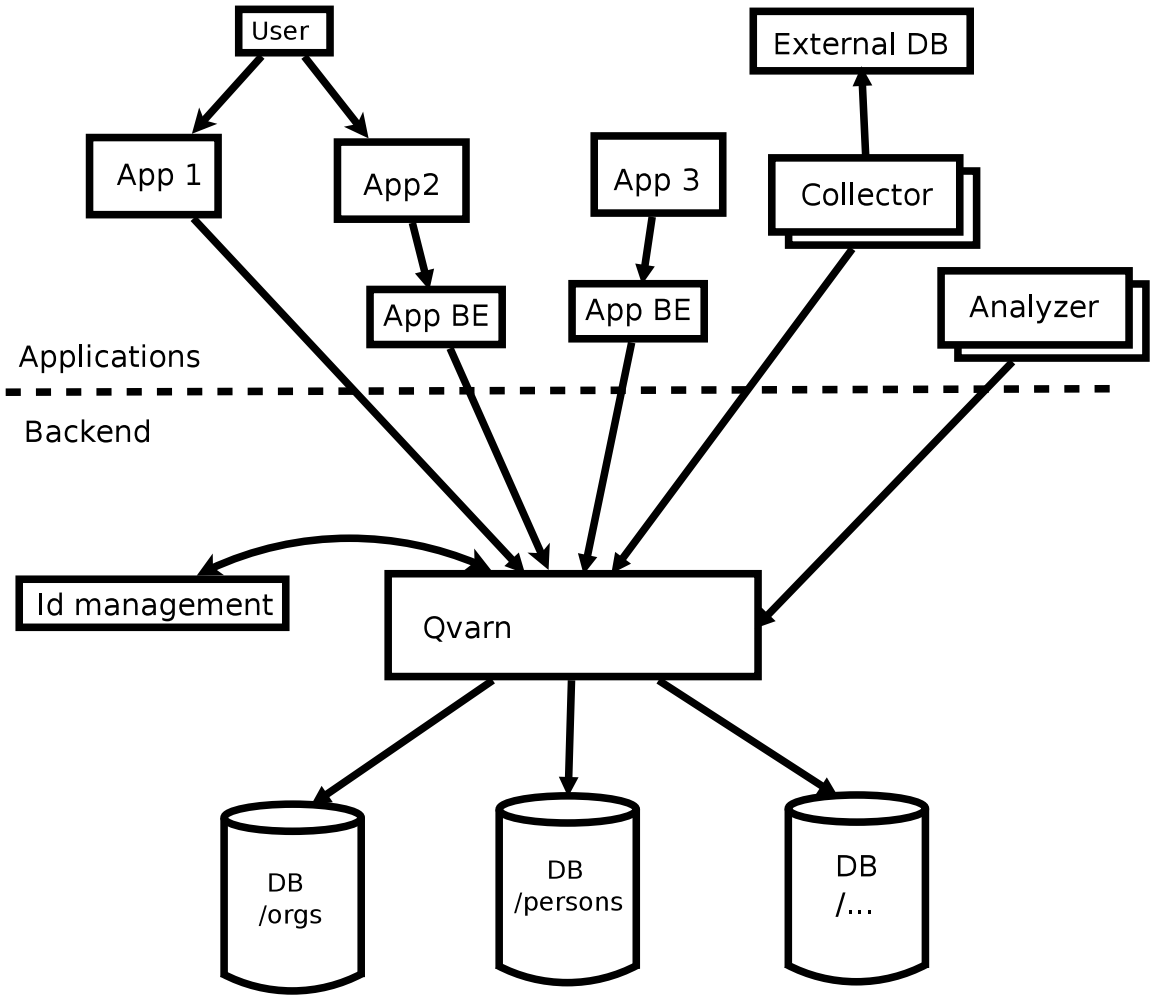

Figure 3.1: Services architecture

This chapter describes the high level architecture of the Tilaajavastuu services and the way various applications fit into the architecture. It is intentionally very high level; components are documented separately, as needed.

The main components are:

- External applications of various kinds. These include:

  - Purely browser-based web applications, typically implemented in Javascript.
  - Applications containing a backend of their own. This will typically contain business logic. The application may have user interfaces (browser-based, mobile apps, desktop apps), but may also be completely non-interactive.
  - Data collector applications, which retrieve data from online services and feed it to the storage backend. These are meant to be simple applications, one per data source. They only retrieve the data, and do not process it.
  - Data analysis applications, which are notified by the storage backend of interesting changes to data, and produce reports or otherwise refine the data to be more useful.

- A data storage backend providing a resource API. This backend only stores data, and notifies applications of changes to the systems. Any processing of the data happens elsewhere.

- An authentication and authorisation system for ensuring only the right users and applications get access and only have access to the right data and operations.

This document describes the resource API for data storage. It does not describe the applications using the API; those are described in the documentation for each application, as needed. It also doesn't describe the implementation of the backend, as that is irrelevant to the API clients.

3.1 Storage backend and its database

The storage backend receives data via its resource API, and stores it some form of persistent storage. The backend does not itself process the data, but notifies interested parties whenever data changes.

The API is an HTTP RESTful API, using JSON to encode data between backend and API client. The backend does not store state related to the user interface session in the backend.

The backend also does not serve any HTML, CSS, Javascript, or any other data that forms part of the user interface implementation.

The backend uses a database, or several databases, or other forms of persistent storage, to store the data. There is no access to this storage except via the API, for any applications. Only the backend accesses the storage directly. Since the API is the only interface, the form of storage is outside the scope of this document and is an internal implementation detail of the backend.

Change notifications are done using a message box. The backend does not call out to applications, instead the applications need to poll for new messages.

3.2 Automatic systems and non-browser applications

The backend API is used by various types applications. For example, a card reader uses an API call to report the log entry to the backend. A mobile application on a smartphone or tablet will use the API to retrieve any information the user is interested in, and renders that in the appropriate way.

For all applications, the backend API provides a de-coupling point. Applications use, and rely on, the API, and all code and data specific to a user interface is only in the application. The API only provides structured data to the user interface. All the logic for how the user interface works is only in the application. This allows applications to be changed, and deployed, separately from the backend, and removes the coupling between user interface logic

and business logic, making it easier to develop both the application and the backend. This applies to applications developed by Suomen Tilaajavastuu Oy as well as those developed by partners.

Some applications may need more backend processing logic than the storage backend provides. Such applications may have their own, application specific backend, which mediates between the application frontend and the storage backend.

3.3 Web applications

Web applications are described separately, as they need to load static files used by the user interface, such as HTML, Javascript, CSS, images, etc. These are served as static files over HTTPS, using a different URL than the API uses. This need for loading resources from a server for the UI itself makes browser based applications somewhat different from other applications. However, when the application has been loaded, it communicates with the backend with the exact same API as every other application.

By loading a different set of files, it becomes easy to vary the user interface by user: a construction company may need a different user interface than a hospital. In addition, it becomes easy to serve different versions of the same user interface, for A/B testing or gradual rollout.

For example, users will normally go to **`https://ui.tilaajavastuu.fi`** to use the web application. This gives them a login screen, and after they log in, the rest of the web application is loaded. At this point, the rest of the application will be loaded from **`https://ui.tilaajavastuu.fi/app/`** normally, but during testing, some users will get the application from **`https://ui.tilaajavastuu.fi/app-beta`** instead, so they test the beta version instead of the latest stable release version.

3.4 Data collection applications

Another special class of application is one that collects data from external sources. There are many such sources, each with their own curiosities and complications as far as how the data is retrieved and in what format it is.

These applications are outside the storage backend, and run separately, injecting the data into the backend via its API. This is so that they can be developed, and deployed, separately from the backend, thanks to the decoupling provided by the API. If a data source changes protocols, for example, it is not necessary to deploy a whole new version of the whole backend.

There is also no technical reason why data collection applications should be developed by Suomen Tilaajavastuu Oy only.

Each data source will have its own application. This will keep each application as simple as possible. If there are similar data sources, it may be possible to write a parameterisable application, to reduce code duplication, as long as the application remains simple. This kind of application otherwise has a tendency to grow in complexity, resulting in code that is fragile and difficult to maintain.

3.5 Data analysis applications

Data gathered into the storage backend can be refined. For example, various kinds of reports may be produced to distill a number of data items into a brief summary. This job is done by data analysis applications. These appliations register their interest in specific items, or types of items, to the storage backend, and the storage backend notifies them whenever anything changes. The analysis application then produces a new report as needed.

Chapter 4

API overview

This chapter gives a brief introduction to the way the API works: its general principles of design and operation. It provides some code examples (which will be tested automatically). The goal of this chapter is to give a programmer a good feeling of how the API is to be used.

The storage backend provides an API for programmatic use. All applications use the same API, including the ones developed by Suomen Tilaajavastuu Oy itself. This chapter gives an overview of the API, and covers its principles of design and operation. It does not go into depth of how each API call functions; that will be covered by the API reference chapter.

The API can be summarised as "RESTful API over HTTPS and using JSON". All calls to the API are done over TLS-encrypted HTTP, and the API design follows the RESTful principles, and structured data is encoded as JSON.

4.1 Overview of data model behind resources

This chapter section gives an overview of the various entities forming the data model behind the API, and their connections with each other. The details of each entity resource are described in the section for each resource.

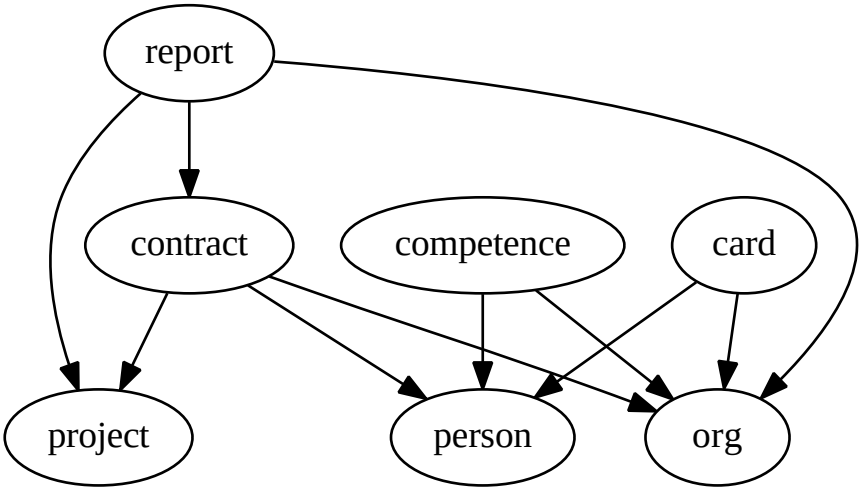

Figure 4.1: API data model

The data model has the following entities:

- **`person`** — a real person. Typically an employee of an organisation, or a user of the API.

- **org** — an organisation of any kind (corporation, association, city, etc).
- **project** — an undertaking for which data needs to be collected as if it were an entity; for example, work done on construction sites may need to be reported together for tax purposes
- **card** — a card for identifying a person, typically a VALTTI card issued by Suomen Tilaajavastuu
- **competence** — describe a formal or informal competence of a person

Each entity is accessible via the API as a separate URL.

4.2 Example: Retrieve the service implementation version

Some API calls are entirely public. One such call is the version of the API and its implementation. Here's an example of how to use that call. This demonstrates how the API is used in simple cases. First of all, you need to make an HTTPS request. On a Unix command line, the **curl** tool works fine:

EXAMPLE

```
curl https://api-demo.tilaajavastuu.fi/version
```

The result is a JSON dictionary:

EXAMPLE

```
{
    "api": {
        "version": "5.2"
    },
    "implementation": {
        "name": "alfred-pennington",
        "version": "2014-12-05"
    }
}
```

The JSON result is typically reasonably self-explanatory, once one has an understanding of the services. However, each API call is documented in detail in the API reference chapter.

4.3 Consistency

The API provides **eventual consistency.** For example, when updating a resource with **PUT /orgs/123** with a new address, it is not guaranteed that an immediately following **GET /orgs/123** returns the new address. However, the new address is returned soon, typically within seconds, though no guarantees are given. This is because there may be multiple independent hosts implementing the **/orgs** resources, and synchronisation of changes to the data between them may take a while. The API does not guarantee that the synchronisation has happened before it returns a success code to the **PUT** call.

Note that the API implementation does guarantee that when the **PUT** call returns, the changed data has been stored persistently.

The API does not guarantee that references between resources are or stay valid. For example, a contract resource between two organisations can be added without either organisation existing. The validity is checked when it is needed, not during updates. For example, a report about contracts requires the organisations to exist, and if they don't, the report will indicate an error.

4.4 Update conflicts and resource revisions

The API provides no locking of resources, or transactions that update multiple resources. It does provide a way to detect conflicting updates to the same resource.

Every resource carries a **`revision`** attribute. This is a unique identifier that identifies a particular version of the resource. When updating a resource, the update must identify the revision it updates. If it is not the current revision, the update fails. A revision identifier is unique, but otherwise arbitrary string, and the API client should not try to use it except to compare for equality.

For example, assume the **`/foos/123`** resource looks like this:

EXAMPLE

```
{
    "id": "123",
    "type": "foo",
    "revision": "cafe",
    "name": "The Blue Sky Foo"
}
```

When an API client requests the resource (**`GET /foos/123`**), it gets the above revision id. When client A updates the resource, it sends the following JSON record (**`PUT /foos/123`**):

EXAMPLE

```
{
    "revision": "cafe",
    "name": "The Red Sea Foo"
}
```

The **`id`** and **`type`** fields are optional in the update, but the **`revision`** field MUST be there. After this is processed, the revision id is changed by the API backend:

EXAMPLE

```
{
    "id": "123",
    "type": "foo",
    "revision": "f00d",
    "name": "The Red SeaFoo"
}
```

If client B now also tries to update the resource, but provides the old revision id (**`"revision": "cafe"`** instead of **`"revision": "f00d"`**), then B's update will fail with an HTTP status code 409 ("Conflict"). Client B can handle such a situation by retrieving the latest revision, and asking the user to change that instead.

4.4.1 Tests

We create a new person and update them with valid and invalid revisions.

```
SCENARIO manage a person with revisions
```

Client has needed access rights for persons resource.

```
GIVEN client has access to scopes
... "uapi_persons_post uapi_persons_id_put"
```

Create a person.

```
WHEN client POSTs /persons with
... {"names": [{"full_name": "James Bond"},
...            {"full_name": "Alfred E. Newman"}]}
THEN HTTP status code is 201
AND result has key "id" containing a string, saved as $ID1
AND result has key "revision" containing a string, saved as $REV1
```

Update a person.

```
WHEN client PUTs /persons/$ID1 with
... {
...     "revision": "$REV1",
...     "names": [{"full_name": "M"}]
... }
THEN HTTP status code is 200
AND result has key "revision" containing a string, saved as $REV3
```

Try to update the record with the old revision.

```
WHEN client PUTs /persons/$ID1 with
... {
...     "revision": "$REV1",
...     "names": [{"full_name": "M"}]
... }
THEN HTTP status code is 409
```

4.5 Conventions

Each resource is described below in its own section. To keep things brief, the following conventions are followed.

- The API is rooted at a **base URL**. This URL may differ between, say, production and beta versions. The base URL might be, for example, `https://api.example.com/1.0`, where `1.0` is a version number. All API resource endpoints are named as paths below the base URL, so that if the endpoint is `/orgs`, the actual URL might be `http://api.example.com/1.0/orgs`.
- For each resource, the **HTTP methods** that work with it are specified: `GET`, `PUT`, `POST`, `DELETE`. Since the API follows RESTful conventions, the actual call is always the same, and to avoid repetition, the descriptions are not repeated for each resource.

- All requests that provide data, provide it as a **JSON document** in the request body. The **`Content-Type`** is **`application/json`** and the character set is UTF-8.
- All responses that provide data, provide it as a JSON document in the body. If there is an error (HTTP status code in the 400 or 500 range), the body may contain free-form text intended for a human instead. The character set is UTF-8.
- When retrieving a list of resources, such as with **`GET /foos`**, the result has the following structure:

```
{
    "resources": [
        {
            "id": "cafef00d",
            ...
        }
    ]
}
```

More top level fields may be added later. The fields for each resource are dependent on the query, but will always include the resource **`id`** field.

- A new resource is added with **`POST /foos`**, with the new resource provided in the body of the request as a JSON document. The response is a copy of the new resource, with the common fields filled in. The request MUST provide the **`type`** field, and MUST NOT provide the **`id`** or **`revision`** fields.
- An individual instance of a **`foo`** is accessed as **`GET /foos/123`**, where **`123`** is the identifier of the resource.
- A resource is modified with **`PUT`** to the resource's own URL. The whole resource is replaced with a new version. It is not possible to update only individual attributes of a resource. Missing attributes are treated as empty. The new version MUST reference the current version with the same **`revision`** identifier field. The request MUST also provide the **`id`** and **`type`** fields with the correct values.
- A resource is deleted with **`DELETE`**. The resource will no longer be accessible.
- Countries are stored using an ISO 3166-1 alpha-2 two-letter country code, in upper case. If a country is not in the standard, contact Tilaajavastuu and we will assign a custom code to use instead.
- Locales are defined in format **`language_COUNTRY`** where **`language`** is an ISO 639 language code and **`COUNTRY`** is an ISO 3166-1 alpha-2 country code in upper case. For example, the locale for British English is **`en_GB`**.
- Dates and timestamps are stored using ISO 8601 formats. Resource attributes may be just a date, or also include a timestamp. A date is always stored as **`YYYY-MM-DD`**, and a timestamp as **`YYYY-MM-DDTHH:MM:SS+HH`** i.e., the time zone is always included as plus or minus some number of hours and minutes. Time zone names are not used, including the **`Z`** suffix, for uniformity.

  When dates or timestamps are paired to indicate a range of time, the end time can be null, to indicate "not ended yet".

4.5.1 Tests

Character set is defined as UTF-8, we try to add data with unicode characters.

```
SCENARIO unicode support
```

Client has needed access rights for persons resource.

```
GIVEN client has access to scopes
... "uapi_persons_post uapi_persons_id_put uapi_orgs_post"

WHEN client POSTs /persons with
... {"names": [
...   {"full_name": "James Bond"},
...   {"full_name": "James Pöntinen"},
...   {"full_name": "□□□·□□"},
...   {"full_name": "□□□□□□□□□□□"},
...   {"full_name": "Джеймс Бонд"},
...   {"full_name": "□□□□□ □□□□"}
... ]}
THEN HTTP status code is 201
AND result matches
... {"names": [
...   {"full_name": "James Bond"},
...   {"full_name": "James Pöntinen"},
...   {"full_name": "□□□·□□"},
...   {"full_name": "□□□□□□□□□□□"},
...   {"full_name": "Джеймс Бонд"},
...   {"full_name": "□□□□□ □□□□"}
... ]}
AND result has key "id" containing a string, saved as $ID1
AND result has key "revision" containing a string, saved as $REV1

WHEN client PUTs /persons/$ID1 with
... {"names": [
...   {"full_name": "James Bond"},
...   {"full_name": "James Pöntinen"},
...   {"full_name": "□□□·□□"},
...   {"full_name": "□□□□□□□□□□□"},
...   {"full_name": "□□□□□ □□□□"}
... ],
... "revision": "$REV1"}
THEN HTTP status code is 200
AND result matches
... {"names": [
...   {"full_name": "James Bond"},
...   {"full_name": "James Pöntinen"},
...   {"full_name": "□□□·□□"},
...   {"full_name": "□□□□□□□□□□□"},
```

```
...   {"full_name": "□□□□□ □□□□"}
... ]}

WHEN client POSTs /orgs with
... {
...     "names": ["Nokia", "□□□"]
... }
THEN HTTP status code is 201
```

We try to create a person with id and/or revision fields set.

```
SCENARIO correctly manage a person
```

Client has needed access rights for persons resource.

```
GIVEN client has access to scopes
... "uapi_persons_post uapi_persons_id_put uapi_persons_id_delete"
```

Fail to create a person with id field set.

```
WHEN client POSTs /persons with
... {"id": "random",
...  "names": [{"full_name": "James Bond"},
...            {"full_name": "Alfred E. Newman"}]}
THEN HTTP status code is 400
```

Fail to create a person with revision field set.

```
WHEN client POSTs /persons with
... {"revision": "random",
...  "names": [{"full_name": "James Bond"},
...            {"full_name": "Alfred E. Newman"}]}
THEN HTTP status code is 400
```

Fail to create a person with id and revision fields set.

```
WHEN client POSTs /persons with
... {"id": "random",
...  "revision": "random",
...  "names": [{"full_name": "James Bond"},
...            {"full_name": "Alfred E. Newman"}]}
THEN HTTP status code is 400
```

Create a person without using id and revision fields.

```
WHEN client POSTs /persons with
... {"names": [{"full_name": "James Bond"},
...            {"full_name": "Alfred E. Newman"}]}
THEN HTTP status code is 201
AND result has key "id" containing a string, saved as $ID1
AND result has key "revision" containing a string, saved as $REV1
```

Try to update a person with wrong id set.

```
WHEN client PUTs /persons/$ID1 with
... {
...     "id": "random",
...     "revision": "$REV1",
...     "names": [{"full_name": "M"}]
... }
THEN HTTP status code is 400
```

Try to send a request with Content-Length of 0. This caused problems in some environments.

```
WHEN client DELETEs /persons/$ID1 with a header "Content-Length: 0"
THEN HTTP status code is 200
```

4.6 Handling names of people

In the Finnish culture, names are almost always very simple: one to three given names, and one family name. For example, a Finnish man might be called “Matti Virtanen”, where “Matti” is his given name and “Virtanen” is his family name. This is so common that a lot of computing systems hardcode the assumption that everyone has at least one given name and one family name, and also that everyone has only one name.

This is cultural assumption that is not valid everywhere, and is not even always valid within Finland, given immigrants. See Falsehoods Programmers Believe About Names, a well-known article on some of the issues.

For the API, we need to be more flexible in how we encode a person’s name. The simplest way to do this is to just store one or more full names, and not try to split them into meaningful parts. However, this is not sufficient: it is sometimes necessary to interface with government and other systems that require providing the given and surnames separately. For this reason, we store the name in a more complicated way:

- **`full_name`** — We **always** store the full name.
- **`given_names`**, **`surnames`** – We **optionally** store given and surnames separately from the full name. These are both lists of individual names.
- **`titles`** — We **optionally** also store the titles of the person, as a list of individual titles. This is any titles they want to show before with their name. In the full name, it should already be included.
- **`sort_key`** — We **optionally** store a sort key, which is a form of the full name to be used for ordering a list of people alphabetically. Commonly, in the Finnish tradition, this is “Lastname, Firstname”, but it can be anything.

The user interface is responsible for determining all the fields: the API implementation **doesn't deduce them**. The user interface can make suitable cultural assumptions to avoid having the user fill out all fields. For example, if the user interface knows that the name is Finnish, it can just ask for given and surnames, and then fill out the full name and the sort key accordingly.

If the API implementation only has a full name stored for a person, and that person needs to be put into a report sent to an external service that requires given and surnames separately, this will trigger an error. The error will happen when the report is being sent: it is **not** a validation error when the person's information is originally entered into the system, since at that time it is not known which external system the person's name is going to be used with.

4.7 Handling contact information

People and organisations, and possibly other resources, have contact information of various kinds. Physical and postal addresses are particularly difficult to encode in a way that is both machine-interpretable and generic. We use the following approach to encode one piece of contact information.

The type of contact information is encoded in its own field:

- **`contact_type`** — the type of the contact; one of
  - **`phone`** — a phone number
  - **`address`** — a physical or postal address
  - **`email`** — an electronic mail address
  - **`einvoice`** — an e-invoicing address

In addition, the contact may also have certain roles that are stored in a list:

- **`contact_roles`** — roles of contact, if any; valid field values are
  - **`billing`** — a contact for billing

All types of contact have the following fields:

- **`contact_source`** — source of the contact information; this should be the string **`self`** to indicate that the person or organisation whose information it is has done the update, or else the name of the other source (free form text)
- **`contact_timestamp`** — timestamp for when the contact was added/updated last

For phone numbers, the number is encoded in the following field:

- **`phone_number`** — a string giving the phone number, in whatever form the user has provided

Email addresses have the following field:

- **`email_address`** — the actual email address.

E-invoicing addresses have the following fields:

- **`einvoice_operator`** — the operator of e-invoicing service
- **`einvoice_address`** — the actual e-invoicing address

Additionally, in the cases where someone is registering on a service that uses Qvarn as a backend, the following fields can be used for email verification:

- **`verification_code`** — a free form string generated by the service application
- **`verification_code_expiration_date`** — a last validity date for the stored verification code
- **`email_verification_timestamp`** — timestamp for when this email address was verified

Addresses are encoded with the following fields, all mandatory:

- **`full_address`** — a free form string, intended for a human to interpret, with the full address
- **`country`** — a two-letter country code for the address

Additionally, an address may be given in a somewhat more structured form:

- **`address_lines`** — **optional** list of address lines
- **`post_code`** — **optional** post code in the syntax appropriate for the country
- **`post_area`** — **optional** name of the area corresponding to the post code

In an ideal world, we would encode a fully parsed address, but given the large amount of variance in addresses across Europe, that is too big a problem to solve at this time.

Example of an address in Finland:

EXAMPLE

```
{
    "full_address": "Tarvonsalmenkatu 17\n02600 Espoo",
    "country": "FI",
    "address_lines": ["Tarvonsalmenkatu 17"],
    "post_code": "02600",
    "post_area": "Espoo"
}
```

Example of an address in the UK:

EXAMPLE

```
{
    "full_address": "102 The Mill\nHigh St\nStalybridge\nSK15 1NS",
    "country": "GB",
    "address_lines": [
        "102 The Mill",
        "High St"
    ],
    "post_code": "SK15 1NS",
    "post_area": "Stalybridge"
}
```

4.8 Error handling

The API uses HTTP status codes for reporting success or failure for an operation. The status code alone is always enough to indicate success or failure, but the response body may contain more information.

The common error cases are:

- **Connection refused** — the API client can't connect to the API provider. In this situation, the client should try again after a reasonable time. If the problem persists, the API provider should be notified out of band.
- **Timeout** — the API provider didn't respond in a reasonable time to a client request. The API provider should be notified out of band.
- **400** (Bad Request) — the client request is malformed, incomplete, or otherwise unacceptable to the server. The client should fix that and try again.
- **401** (Unauthorized) — the client did not authenticate correctly to the API provider. The authentication may have expired, in which case client the client should authenticate again. The credentials may also be wrong, in which case the client should fix them and try again.
- **403** (Forbidden) — the client is not allowed to use the resource. The client shouldn't try again.
- **404** (Not Found) — the client tried to use a resource that doesn't exist. The client shouldn't try again, unless it suspects a timing problem between components of the API implementation getting new data synchronised.
- **405** (Method Not Allowed) — the client tried to use an HTTP method that isn't allowed for the resource, such as **`DELETE /orgs`**. This indicates a client programming error.
- **409** (Conflict) — the client tried to update (**`POST`**) a resource, but gave a **`revision`** field that wasn't the same as that of the latest version of the resource. Client should retrieve the latest version, make changes to that, and try updating again. The latest revision is included as the body of the error response.
- **500** (Internal Server Error) — there is a bug in the API provider. The API provider should be notified out of band. The client shouldn't try again until notified that the problem is fixed.

Any other HTTP status may also be used in the response. The above are the most common ones.

Example: a failure to add a new organisation, when the user lacks the privileges to do so, might result in the following HTTP response:

```
EXAMPLE
401 Unauthorized
Content-type: text/html
<html> <body><p>Don't do that.</p></body> <html>
```

4.8.1 Tests

We try to create a new person but the JSON sent is malformed.

```
SCENARIO sending malformed requests

GIVEN client has access to scopes
... "uapi_persons_post uapi_persons_id_put uapi_persons_id_private_put"

WHEN client POSTs /persons with
... error{"names": [{"full_name": "James Bond",
...              "sort_key": "Bond, James",
...              "titles": [],
...              "given_names": ["James"],
...              "surnames": ["Bond"]}]}
THEN HTTP status code is 400
```

Create a person and try to update the person with malformed JSON.

```
WHEN client POSTs /persons with
... {"names": [{"full_name": "James Bond",
...              "sort_key": "Bond, James",
...              "titles": [],
...              "given_names": ["James"],
...              "surnames": ["Bond"]}]}
THEN result has key "id" containing a string, saved as $ID1
THEN result has key "revision" containing a string, saved as $REV1

WHEN client PUTs /persons/$ID1 with
... {"error"]}]}
THEN HTTP status code is 400

WHEN client PUTs /persons/$ID1/private with
... { "invalid json" ]
THEN HTTP status code is 400
```

4.9 Common attributes

Every resource has some attributes. A resource is represented as a JSON object (key/value map), and attributes are keys in the object. Attribute names are always ASCII strings.

A small number of attributes are common to all resources.

- **`type`** — the type of resource. For example, **`org`** or **`person`**.
- **`id`** — an **internal identifier**, which is unique within an API instance. This identifier is generated by the API provider, and is not used outside of the API, and isn't meaningful to users.. For example, an organisation might have a unique registration number, but this is not used within the API to identify the organisation. Instead, a new identifier is generated by the API. This avoids external identifiers that change (they always change).
- **`revision`** — the revision identifier of the version of the resource

4.10 Searches

Unless otherwise specified, every top level resource (**/orgs**, **/persons**, etc.) can be searched for specific items. The usage pattern is as follows, assuming a resource **/foos**:

- **GET /foos/search/CLAUSE** — do a search for individual resources of type **foo**. The search clause and the result are described below.

The clause contains any number of the following search conditions, concatented in the URL path, which must all match a resource for it to be included in the result:

- **/exact/KEY/VALUE** — the resource matches if it has a field **KEY** at any level, whose exact value is **VALUE**, or if the field value is a list, the list contains a value that is exactly **VALUE**. Strings are compared without case-sensitivity.
- **/ne/KEY/VALUE** — the inverse of exact, value must not be **VALUE**.
- **/startswith/KEY/VALUE** — like **exact**, but the value must be a string that starts with the value.
- **/contains/KEY/VALUE** — like **exact**, but string fields use sub-string matching (**VALUE** may occur anywhere in the resource's value). Other types of fields are matched exactly.
- **/gt/KEY/VALUE** — the resource matches if it has a field **KEY** at any level, whose value is greater than (>) **VALUE**, or if the field value is a list, the list contains a value that is greater than **value**.
- **/ge/KEY/VALUE** — like **gt**, but the value must be greater than or equal (>=) to **VALUE**.
- **/lt/KEY/VALUE** — like **gt**, but the value must be less than (<) **VALUE**.
- **/le/KEY/VALUE** — like **gt**, but the value must be less than or equal(<=) to **VALUE**.

For details on matching, see below.

Note: We'll add more conditions as they're needed, but we'll try to keep the conditions few, simple, and powerful, and rely on some client side logic to refine the results.

Results can be sorted by one or more search keys:

- **/sort/KEY1/sort/KEY2**

Search key may be at any level of the resource. If there are more than one instance of the field (e.g., the **KEY** is a list of names), the first instance is used.

You can you can skip and limit resources returned by Qvarn using **offset** and **limit** operators:

- **/sort/KEY/offset/40/limit/20**

offset and **limit** can only be used together with **sort**.

The clause may also include the following to modify the result:

- **/show/KEY** — include the top level field **KEY** in the result.

- **`/show_all`** — include the whole the resource in the result.

The result is a JSON object that looks like the following:

EXAMPLE

```
{
    "resources": [
        {
            "id": "123"
        }
    ]
}
```

In other words, the object has a single field **`resources`**, whose value is a list of dicts, and those dicts have the single field **`id`**. This is the default. If the **`show`** or **`show_all`** modifiers are used, more fields will be added to the dicts. With **`show_all`**, all the fields of the resource are added.

Note that the result does not have **`id`**, **`type`**, or **`revision`** fields, as it is not a resource. Those fields may, however, be included in each returned match, if specified by **`show`** or **`show_all`**.

4.10.1 Matching rules in conditions

The matching rules are:

- A resource will only match if the API client can read it.
- The key will match any attribute of a resource, at any level of the resource including their sub-records.
- The pattern value is a Boolean or a string, 'true' and 'false' (case-insensitive) are converted to boolean values and can not be used to match string fields.
- The key in the pattern must match the attribute name in the resource exactly.
- Boolean values must match exactly.
- String values match the pattern using case-insensitive matching. The match may need to match the whole string, the beginning, or any part of the string, depending on the search condition. For example, a pattern **`tilaajavastuu`** with a condition **`contains`** would match a string **`Suomen Tilaajavastuu Oy`**, but not with a condition **`exact`**.
- A resource's list value matches the search pattern if any of the list items matches the pattern.
- A key/value map matches the search pattern if any of the values matches the pattern.

4.10.2 Examples

- **GET /persons/search/contains/full_name/bond** would match any person whose full name contains the substring **bond**.
- **GET /persons/search/contains/full_name/bond/exact/given_names/james** would match any person whose full name contains the substring **bond**, but only if their have a given name of James.
- **GET /persons/search/exact/gov_id/SN 00 70 07** would match any person with a key **gov_id** whose value is **SN 00 70 07**, at any level of the data in the person's information.

4.10.3 Notes

- The search URLs are not listed for each resource separately.

4.10.4 Tests

We create a new person and do some searches for that person using different criteria.

```
SCENARIO search a person
```

Client has needed access rights for person resource.

```
GIVEN client has access to scopes
... "uapi_persons_post uapi_persons_id_private_put
...  uapi_persons_search_id_get"
```

Create a test person with sufficient amount of diverse data.

```
WHEN client POSTs /persons with
... {"names": [{"full_name": "James Bond",
...             "sort_key": "Bond, James",
...             "titles": ["Päällikkö", "Seppä"],
...             "given_names": ["James", "???"],
...             "surnames": ["Bond"]}]}
THEN result has key "id" containing a string, saved as $ID1
THEN result has key "revision" containing a string, saved as $REV1
THEN result has a valid Date header

WHEN client PUTs /persons/$ID1/private with
... {
...     "date_of_birth": "1920-11-11",
...     "revision": "$REV1",
...     "gov_ids": [
...         {
...             "country": "GB",
```

```
...             "id_type": "ssn",
...             "gov_id": "SN 00 70 07"
...         }
...     ],
...     "nationalities": ["GB"],
...     "residences": [
...         {
...             "country": "GB",
...             "location": "London"
...         },
...         {
...             "country": "FI",
...             "location": "Ypäjä"
...         },
...         {
...             "country": "NO",
...             "location": "un/known"
...         },
...         {
...             "country": "SE",
...             "location": "search"
...         }
...     ],
...     "contacts": [
...         {
...             "contact_type": "phone",
...             "contact_source": "self",
...             "contact_timestamp": "2038-02-28T01:02:03+0400",
...             "phone_number": "+358 4321"
...         },
...         {
...             "contact_type": "email",
...             "contact_source": "self",
...             "contact_timestamp": "2038-02-28T01:02:03+0400",
...             "email_address": "james.bond@sis.gov.uk"
...         },
...         {
...             "contact_type": "address",
...             "contact_source": "self",
...             "contact_timestamp": "2038-02-28T01:02:03+0400",
...             "country": "GB",
...             "full_address": "61 Horsen Ferry Road\\nLondon S1",
...             "address_lines": ["61 Horsen Ferry Road"],
...             "post_code": "S1",
...             "post_area": "London"
...         }
...     ]
... }
THEN HTTP status code is 200
```

Perform search with a simple parameter inside a list.

```
WHEN client GETs /persons/search/exact/full_name/James%20Bond
THEN HTTP status code is 200
AND result has key "resources", a list containing {"id": "$ID1"}
```

Perform a case insensitive search.

```
WHEN client GETs /persons/search/exact/country/gb
THEN HTTP status code is 200
AND result has key "resources", a list containing {"id": "$ID1"}
```

Perform a search with show_all in the beginning of the condition.

```
WHEN client GETs /persons/search/show_all/exact/given_names/James/exact/date
THEN HTTP status code is 200
AND result has key "resources", a list containing {"id": "$ID1"}
```

Perform a search with **show** to select a specific field to show.

```
WHEN client GETs
... /persons/search/show/names/exact/given_names/James/exact/date_of_birth/1
THEN HTTP status code is 200
AND result matches
... {
...     "resources": [
...         {
...             "id": "$ID1",
...             "names": [
...                 {
...                     "full_name": "James Bond",
...                     "sort_key": "Bond, James",
...                     "titles": ["Päällikkö", "Seppä"],
...                     "given_names": ["James", "???"],
...                     "surnames": ["Bond"]
...                 }
...             ]
...         }
...     ]
... }
```

Perform search with a list parameter inside a list.

```
WHEN client GETs /persons/search/exact/given_names/James
THEN HTTP status code is 200
AND result has key "resources", a list containing {"id": "$ID1"}
```

Perform search with a simple parameter inside a sub-record.

```
WHEN client GETs /persons/search/exact/date_of_birth/1920-11-11
THEN HTTP status code is 200
AND result has key "resources", a list containing {"id": "$ID1"}
```

Perform a search with two conditions.

```
WHEN client GETs
... /persons/search/exact/given_names/James/exact/date_of_birth/1920-11-11
THEN HTTP status code is 200
AND result has key "resources", a list containing {"id": "$ID1"}
```

Perform searches with non-ascii characters. First with location Ypäjä.

```
WHEN client GETs
... /persons/search/exact/location/Yp%C3%A4j%C3%A4
THEN HTTP status code is 200
AND result has key "resources", a list containing {"id": "$ID1"}
```

With given_names □□□.

```
WHEN client GETs /persons/search/exact/given_names/%E8%A9%B9%E5%A7%86%E6%96%
THEN HTTP status code is 200
AND result has key "resources", a list containing {"id": "$ID1"}
```

With title Päällikkö and location Ypäjä.

```
WHEN client GETs
... /persons/search/exact/titles/P%C3%A4%C3%A4llikk%C3%B6/exact/location/Yp%
THEN HTTP status code is 200
AND result has key "resources", a list containing {"id": "$ID1"}
```

Search for a search.

```
WHEN client GETs
... /persons/search/exact/location/search/show_all
THEN HTTP status code is 200
AND result has key "resources", a list containing {"id": "$ID1"}
```

Perform searches with a slash in the condition.

```
WHEN client GETs
... /persons/search/exact/location/un%2fknown
THEN HTTP status code is 200
AND result has key "resources", a list containing {"id": "$ID1"}
```

```
WHEN client GETs
... /persons/search/exact/location/un%2fknown/exact/date_of_birth/1920-11-11
THEN HTTP status code is 200
AND result has key "resources", a list containing {"id": "$ID1"}
```

Perform searches with invalid conditions.

```
WHEN client GETs /persons/search/exact/INVALID_KEY/James
THEN HTTP status code is 400
AND result matches
... {
...   "field": "INVALID_KEY",
...   "message": "Resource does not contain given field",
...   "error_code": "FieldNotInResource"
... }

WHEN client GETs
... /persons/search/exact/location/exact/given_names/James
THEN HTTP status code is 400
AND result matches
... {
...   "message": "Could not parse search condition",
...   "error_code": "BadSearchCondition"
... }

WHEN client GETs
... /persons/search/show_all/location/exact/given_names/James
THEN HTTP status code is 400
AND result matches
... {
...   "message": "Could not parse search condition",
...   "error_code": "BadSearchCondition"
... }

WHEN client GETs
... /persons/search/exact/given_names
THEN HTTP status code is 400
AND result matches
... {
...   "message": "Could not parse search condition",
...   "error_code": "BadSearchCondition"
... }
```

We create multiple contracts and search with comparison and other operators.

```
SCENARIO more search operators
```

Client has needed access rights for contracts resource.

```
GIVEN client has access to scopes
... "uapi_contracts_post uapi_contracts_search_id_get"
```

Create new contracts.

```
WHEN client POSTs /contracts with
... {
... "contract_type": "employment",
... "right_to_work_based_on": "other",
... "start_date": "2013-03-13"
... }
THEN HTTP status code is 201
THEN result has key "id" containing a string, saved as $ID1
```

One for the following day.

```
WHEN client POSTs /contracts with
... {
... "contract_type": "employment",
... "right_to_work_based_on": "other",
... "start_date": "2013-03-14"
... }
THEN HTTP status code is 201
THEN result has key "id" containing a string, saved as $ID2
```

One for the following year.

```
WHEN client POSTs /contracts with
... {
... "contract_type": "employment",
... "right_to_work_based_on": "other",
... "start_date": "2014-03-13"
... }
THEN HTTP status code is 201
THEN result has key "id" containing a string, saved as $ID3
```

Use different comparison operators to find correct contracts by **`start_date`**.

Greater than.

```
WHEN client GETs
... /contracts/search/gt/start_date/2013-03-13
THEN HTTP status code is 200
AND result has key "resources", a list that does not contain {"id": "$ID1"}
AND result has key "resources", a list containing {"id": "$ID2"}
AND result has key "resources", a list containing {"id": "$ID3"}

WHEN client GETs
... /contracts/search/gt/start_date/2014-03-12
THEN HTTP status code is 200
AND result has key "resources", a list that does not contain {"id": "$ID1"}
AND result has key "resources", a list that does not contain {"id": "$ID2"}
AND result has key "resources", a list containing {"id": "$ID3"}
```

Greater than or equal.

```
WHEN client GETs
... /contracts/search/ge/start_date/2013-03-14
THEN HTTP status code is 200
AND result has key "resources", a list that does not contain {"id": "$ID1"}
AND result has key "resources", a list containing {"id": "$ID2"}
AND result has key "resources", a list containing {"id": "$ID3"}

WHEN client GETs
... /contracts/search/ge/start_date/2014-03-13
THEN HTTP status code is 200
AND result has key "resources", a list that does not contain {"id": "$ID1"}
AND result has key "resources", a list that does not contain {"id": "$ID2"}
AND result has key "resources", a list containing {"id": "$ID3"}
```

Less than.

```
WHEN client GETs
... /contracts/search/lt/start_date/2013-03-14
THEN HTTP status code is 200
AND result has key "resources", a list containing {"id": "$ID1"}
AND result has key "resources", a list that does not contain {"id": "$ID2"}
AND result has key "resources", a list that does not contain {"id": "$ID3"}

WHEN client GETs
... /contracts/search/lt/start_date/2014-03-12
THEN HTTP status code is 200
AND result has key "resources", a list containing {"id": "$ID1"}
AND result has key "resources", a list containing {"id": "$ID2"}
AND result has key "resources", a list that does not contain {"id": "$ID3"}
```

Less than or equal.

```
WHEN client GETs
... /contracts/search/le/start_date/2013-03-13
THEN HTTP status code is 200
AND result has key "resources", a list containing {"id": "$ID1"}
AND result has key "resources", a list that does not contain {"id": "$ID2"}
AND result has key "resources", a list that does not contain {"id": "$ID3"}

WHEN client GETs
... /contracts/search/le/start_date/2014-03-12
THEN HTTP status code is 200
AND result has key "resources", a list containing {"id": "$ID1"}
AND result has key "resources", a list containing {"id": "$ID2"}
AND result has key "resources", a list that does not contain {"id": "$ID3"}
```

Not equal.

```
WHEN client GETs
... /contracts/search/ne/start_date/2013-03-13
THEN HTTP status code is 200
AND result has key "resources", a list that does not contain {"id": "$ID1"}
AND result has key "resources", a list containing {"id": "$ID2"}
AND result has key "resources", a list containing {"id": "$ID3"}

WHEN client GETs
... /contracts/search/ne/start_date/2013-03-14
THEN HTTP status code is 200
AND result has key "resources", a list containing {"id": "$ID1"}
AND result has key "resources", a list that does not contain {"id": "$ID2"}
AND result has key "resources", a list containing {"id": "$ID3"}
```

We create a new organisation and do a search for that organisation.

```
SCENARIO search an organisation
```

Client has needed access rights for orgs resource.

```
GIVEN client has access to scopes
... "uapi_orgs_post uapi_orgs_search_id_get"

WHEN client POSTs /orgs with
... {"names": ["Suomen Tilaajavastuu Oy"],
... "country": "FI"
... }
THEN HTTP status code is 201
THEN result has key "id" containing a string, saved as $ID2
```

Only get fields specifically required:

```
WHEN client GETs /orgs/search/contains/names/Tilaajavastuu/show/country
THEN HTTP status code is 200
AND search result contains match
... {
...     "id": "$ID2",
...     "country": "FI"
... }
AND result doesn't match
... {
...     "resources": [
...         {
...             "id": "$ID2",
...             "contacts": [],
...             "gov_org_ids": []
...         }
...     ]
... }
```

We create different resource instances and search for them using multiple search conditions.

```
SCENARIO search with multiple conditions

GIVEN client has access to scopes
... "uapi_persons_post uapi_persons_id_private_put
...  uapi_persons_search_id_get
...  uapi_orgs_post uapi_orgs_search_id_get
...  uapi_contracts_post uapi_contracts_search_id_get"
```

Create an organisation, a person and an employment contract between them.

```
WHEN client POSTs /orgs with
... {
...     "names": ["Test Company, Inc.", "Test Company"],
...     "country": "FI",
...     "gov_org_ids": [
...         {
...             "country": "FI",
...             "org_id_type": "registration_number",
...             "gov_org_id": "1234567-8"
...         }
...     ],
...     "contacts": [
...         {
...             "contact_type": "address",
...             "country": "FI",
...             "full_address": "Katu 1\\n00000 Helsinki"
...         },
...         {
...             "contact_type": "email",
...             "email_address": "info@testcompany.fi"
...         }
...     ]
... }
THEN HTTP status code is 201
THEN result has key "id" containing a string, saved as $ORG_ID1

WHEN client POSTs /persons with
... {
...     "names": [
...         {
...             "full_name": "Test Person",
...             "sort_key": "Person, Test",
...             "given_names": ["Test"],
...             "surnames": ["Person"]
...         }
...     ]
... }
```

```
THEN HTTP status code is 201
THEN result has key "id" containing a string, saved as $PERSON_ID1

WHEN client POSTs /contracts with
... {"contract_type": "employment",
... "start_date": "2016-01-01",
... "contract_parties": [
...     {
...         "type": "org",
...         "resource_id": "$ORG_ID1",
...         "role": "employer"
...     },
...     {
...         "type": "person",
...         "resource_id": "$PERSON_ID1",
...         "role": "employee"
...     }
... ],
... "right_to_work_based_on": "eea_citizen",
... "contract_state": "active"
... }
THEN HTTP status code is 201
AND result has key "id" containing a string, saved as $CONTRACT_ID1
```

Search for names beginning with a string:

```
WHEN client GETs /orgs/search/startswith/names/Test
THEN HTTP status code is 200
AND result has key "resources", a list containing {"id": "$ORG_ID1"}

WHEN client GETs /orgs/search/startswith/names/Nessie
THEN HTTP status code is 200
AND result has key "resources",
... a list that does not contain {"id": "$ORG_ID1"}
```

Search for names containing a string:

```
WHEN client GETs /orgs/search/contains/names/Company
THEN HTTP status code is 200
AND result has key "resources", a list containing {"id": "$ORG_ID1"}

WHEN client GETs /orgs/search/contains/names/Nessie
THEN HTTP status code is 200
AND result has key "resources",
... a list that does not contain {"id": "$ORG_ID1"}
```

Perform searches with multiple conditions on resource's main level.

```
WHEN client GETs /contracts/search/exact/start_date/2016-01-01/exact/contrac
THEN HTTP status code is 200
AND result has key "resources", a list containing {"id": "$CONTRACT_ID1"}

WHEN client GETs /contracts/search/exact/start_date/2016-01-01/exact/contrac
THEN HTTP status code is 200
AND result matches
... {
...     "resources": []
... }
```

Perform searches targeting several levels of nested elements.

```
WHEN client GETs /orgs/search/exact/names/Test%20Company/exact/country/FI
THEN HTTP status code is 200
AND result has key "resources", a list containing {"id": "$ORG_ID1"}

WHEN client GETs /orgs/search/exact/names/Test%20Company/exact/country/SE
THEN HTTP status code is 200
AND result matches
... {
...     "resources": []
... }
```

Perform searches matching different instances of the same nested element.

```
WHEN client GETs /contracts/search/exact/resource_id/$ORG_ID1/exact/resource
THEN HTTP status code is 200
AND result has key "resources", a list containing {"id": "$CONTRACT_ID1"}

WHEN client GETs /contracts/search/exact/resource_id/$ORG_ID1/exact/resource
THEN HTTP status code is 200
AND result matches
... {
...     "resources": []
... }
```

We create multiple person resources and search with sorting.

```
SCENARIO search with sort

GIVEN client has access to scopes
... "uapi_persons_post uapi_persons_search_id_get"
```

Create several person resources.

```
GIVEN unique identifier $UID

WHEN client POSTs /persons with
... {
...     "gluu_user_id": "$UID",
...     "names": [
...         {
...             "full_name": "Person A",
...             "sort_key": "3, Person",
...             "given_names": ["Test"],
...             "surnames": ["Person", "A"]
...         }
...     ]
... }
THEN HTTP status code is 201
THEN result has key "id" containing a string, saved as $ID3

WHEN client POSTs /persons with
... {
...     "gluu_user_id": "$UID",
...     "names": [
...         {
...             "full_name": "Person B",
...             "sort_key": "1, Person",
...             "given_names": ["Test"],
...             "surnames": ["Person", "B"]
...         }
...     ]
... }
THEN HTTP status code is 201
THEN result has key "id" containing a string, saved as $ID1

WHEN client POSTs /persons with
... {
...     "gluu_user_id": "$UID",
...     "names": [
...         {
...             "full_name": "Person C",
...             "sort_key": "2, Person",
...             "given_names": ["Test"],
...             "surnames": ["Person", "C"]
...         }
...     ]
... }
THEN HTTP status code is 201
THEN result has key "id" containing a string, saved as $ID2
```

Search person resources and sort results by **sort_key**.

```
WHEN client GETs /persons/search/exact/gluu_user_id/$UID/sort/sort_key
```

```
THEN HTTP status code is 200
AND result matches
... {
...     "resources": [
...         {"id": "$ID1"},
...         {"id": "$ID2"},
...         {"id": "$ID3"}
...     ]
... }
```

Sort person resources using different sort key.

```
WHEN client GETs /persons/search/exact/gluu_user_id/$UID/sort/full_name
THEN HTTP status code is 200
AND result matches
... {
...     "resources": [
...         {"id": "$ID3"},
...         {"id": "$ID1"},
...         {"id": "$ID2"}
...     ]
... }
```

Search person resources and sort results by two search keys, where first search key is a list containing more than one value. First key is **surnames**, where each resource has same first surname, and second key is **sort_key**. Since each first surname is the same, results should fall back to the second sort key.

```
WHEN client GETs
... /persons/search/exact/gluu_user_id/$UID/sort/surnames/sort/sort_key
THEN HTTP status code is 200
AND result matches
... {
...     "resources": [
...         {"id": "$ID1"},
...         {"id": "$ID2"},
...         {"id": "$ID3"}
...     ]
... }
```

Search with only search operator should also work, returning all available resource ids.

```
WHEN client GETs /persons/search/sort/sort_key
THEN HTTP status code is 200
AND result has key "resources", a list containing {"id": "$ID1"}
AND result has key "resources", a list containing {"id": "$ID2"}
AND result has key "resources", a list containing {"id": "$ID3"}
```

We create several resources and search for resources but results are limited using LIMIT and OFFSET operators.

```
SCENARIO limit search results with LIMIT and OFFSET

GIVEN client has access to scopes
... "uapi_persons_post uapi_persons_search_id_get"
```

Create several person resources.

```
GIVEN unique identifier $UID

WHEN client POSTs /persons with
... {
...     "gluu_user_id": "$UID",
...     "names": [
...         {
...             "full_name": "Person A",
...             "sort_key": "3, Person",
...             "given_names": ["Test"],
...             "surnames": ["Person", "A"]
...         }
...     ]
... }
THEN HTTP status code is 201
THEN result has key "id" containing a string, saved as $ID3

WHEN client POSTs /persons with
... {
...     "gluu_user_id": "$UID",
...     "names": [
...         {
...             "full_name": "Person B",
...             "sort_key": "1, Person",
...             "given_names": ["Test"],
...             "surnames": ["Person", "B"]
...         }
...     ]
... }
THEN HTTP status code is 201
THEN result has key "id" containing a string, saved as $ID1

WHEN client POSTs /persons with
... {
...     "gluu_user_id": "$UID",
...     "names": [
...         {
...             "full_name": "Person C",
...             "sort_key": "2, Person",
...             "given_names": ["Test"],
...             "surnames": ["Person", "C"]
...         }
```

```
...     ]
... }
THEN HTTP status code is 201
THEN result has key "id" containing a string, saved as $ID2
```

Search person resources and sort results by **sort_key** and limit number of returned resources.

```
WHEN client GETs /persons/search/exact/gluu_user_id/$UID/sort/sort_key/limit
THEN HTTP status code is 200
AND result matches
... {
...     "resources": [
...         {"id": "$ID1"},
...         {"id": "$ID2"}
...     ]
... }
```

Skip one resource and then limit to 1.

```
WHEN client GETs /persons/search/exact/gluu_user_id/$UID/sort/sort_key/offse
THEN HTTP status code is 200
AND result matches
... {
...     "resources": [
...         {"id": "$ID2"}
...     ]
... }
```

It is an error to use limit without sort.

```
WHEN client GETs /persons/search/exact/gluu_user_id/$UID/limit/2
THEN HTTP status code is 400
AND result matches
... {
...   "message": "LIMIT and OFFSET can only be used with together SORT.",
...   "error_code": "LimitWithoutSortError"
... }
```

4.11 File resources

Some resources, e.g. contracts, allow files to be associated with them. The file content is transfered in the HTTP request body instead of JSON so some special rules apply:

- Header **Content-Type** specifies the media type of the uploaded file.
- Header **Revision** specifies the current item revision.

The response to file attachment PUT is in JSON format and contains only the **id** and **revision** fields.

A resource that has no associated document responds to HTTP GET with status code 404. Otherwise HTTP GET will return the file contents in response body with headers **Content-Type** and **Revision** set.

4.11.1 Examples

- A successful **`PUT /contracts/123/document`** with headers **`Content-Type: application/pdf`** and **`Revision: an-excellent-revision`** and PDF contents in the request body would return a JSON dictionary **`{"id": "123", "revision": "a-new-revision"}`**

4.12 Change notifications

The API allows a client to register to be notified of changes to specific resources or resource types. This avoids the need to have the client check every resource repeatedly to check for updates.

The notifications use a message box design, rather than callbacks to the API client. There are **no callbacks** to the API client. The API client needs to poll its messages at a frequency that suits it. This is to avoid having to arrange and maintain access through firewalls, NAT, and other complications. The API client needs to be able to access the API anyway, and the same access can be used to query for messages.

The message boxes are per top-level resource. The API client does need to poll multiple notification message boxes if it needs notifications from multiple top-level resources.

4.12.1 Listeners

Each top-level resource that is a collection of resources (e.g., **`/orgs`**, **`/persons`**, but not **`/version`**) has a listener interface:

- **`GET /orgs/listeners`** — list all listener objects the caller has access to
- **`POST /orgs/listeners`** — create a new listener. The body is a **`listener`** object, as described below.
- **`PUT /orgs/listeners/123`** — update a listener, e.g., to add new resources to listen to.
- **`DELETE /orgs/listeners/123`** — delete a listener.

Note: Only the API client itself should have access to its listener, but it should be possible to add multiple listeners. This is not yet specified, as it ties into the authentication and authorisation framework that is still a work in progress.

A **listener object** looks like this:

`EXAMPLE`

```
{
    "type": "listener",
    "id": "abcd",
    "revision": "kuhg",
    "notify_of_new": true,
    "listen_on_all": false,
    "listen_on": [
        "678"
    ]
}
```

The fields in a listener object have the following meanings:

- **`type`**, **`id`**, **`revision`** — as usual.
- **`notify_of_new`** — set to **`true`** if the listener should be notified of new resources (e.g., new organisations). It will only be notified of new resources it can access.
- **`listen_on_all`** — set to **`true`** if the listener should be notified of the changes in all of the resources (e.g., new organisations). It will only be notified of the resources it can access.
- **`listen_on`** — a list of resources the listener is interested in. Note that new resources are **not added** automatically to the list: the client needs to add them itself. An API client may only listen on resources it can access.

4.12.2 Message box for notifications

Each listener has a message box for notifications:

- **`GET /orgs/listeners/123/notifications`** — list of all notification messages for the listener.
- **`GET /orgs/listeners/123/notifications/567`** — a specific notification message.
- **`DELETE /orgs/listeners/123/notifications/567`** — delete a message.

Note that the API client can't create or update the notification messages: it can only see them and delete them. Messages are not deleted automatically: the client is responsible for deleting messages it no longer cares about.

The API implementation **may delete messages** to keep resource usage in control or for other reasons. The API client must not assume messages persist. There is no notification of deleted notification messages. The API client should consider doing an occasional full scan of the resources it's interested in to handle missed messages.

4.12.3 Notification messages

Notification messages have the following structure:

EXAMPLE

```
{
    "type": "notification",
    "id": "qwer",
    "revision": "456",
    "resource_id": "890",
    "resource_revision": "cafe",
    "resource_change": "created",
    "timestamp": "created"
}
```

The fields are:

- **`type`**, **`id`**, **`revision`** — as usual. The revision won't change, but it's included for consistency with other objects.
- **`resource_id`** — the id of the resource that has changed in some way: it was created, deleted, or updated (including updates to its sub-resources, such as **`/persons/007/photos`**).
- **`resource_revision`** — the revision after the change; if the resource was deleted, the revision is **`null`**.

- **`resource_change`** — an indication of the kind of change that happened: the value is one of **`created`**, **`updated`**, or **`deleted`**.

The notification messages intentionally do not carry deltas of the changes. The API client will have to retrieve the new revision of the resource itself.

4.12.4 Example

- The company Universal Exports wants to be notified whenever any new contracts are added that it can read.
- UE makes a **`POST /contracts/listeners`** request to add a listener:

```
EXAMPLE
{
    "type": "listener",
    "notify_of_new": true,
    "resources": []
}
```

- The new listener has an **`id`** of "M".
- UE then gets a list of all contracts it can see, and adds them to the listener: **`PUT /contracts/listeners/M`** with an updated **`listener`** object as above, but with a really long **`resources`** list.
- UE then starts polling the **`/contracts/listeners/M/notifications`** resource. When it returns a non-empty list of notification message ids, UE retrieves the message, and either examines the existing record, or adds a new contract to the **`resources`** list.
- After that, UE deletes the notification message: **`DELETE /contracts/listeners/notifications/`** for example.

4.12.5 Tests

We set up a listener for new items and another that listens for no items, and create two organisations, and update the organisations, checking for the correct notifications to appear.

```
SCENARIO listen on change notifications
```

Client has needed access rights for orgs resource.

```
GIVEN client has access to scopes
... "uapi_orgs_listeners_post uapi_orgs_listeners_id_get
...  uapi_orgs_listeners_get uapi_orgs_listeners_id_notifications_get
...  uapi_orgs_post uapi_orgs_listeners_id_notifications_id_get
...  uapi_orgs_listeners_id_put uapi_orgs_id_put uapi_orgs_id_delete
...  uapi_orgs_listeners_id_delete
...  uapi_orgs_listeners_id_notifications_id_delete"
```

```
WHEN client POSTs /orgs/listeners with
... {
...     "notify_of_new": true
... }
THEN HTTP status code is 201
AND result matches
... {
...     "type": "listener",
...     "notify_of_new": true,
...     "listen_on": []
... }
AND result has key "id" containing a string, saved as $LISTENID1
AND HTTP Location header is API_URL/orgs/listeners/$LISTENID1

WHEN client POSTs /orgs/listeners with
... {
...     "notify_of_new": false
... }
THEN HTTP status code is 201
AND result matches
... {
...     "type": "listener",
...     "notify_of_new": false,
...     "listen_on": []
... }
AND result has key "id" containing a string, saved as $LISTENID2
AND HTTP Location header is API_URL/orgs/listeners/$LISTENID2
AND result has key "revision" containing a string, saved as $REV1

WHEN client POSTs /orgs/listeners with
... {
...     "notify_of_new": false,
...     "listen_on_all": true
... }
THEN HTTP status code is 201
AND result matches
... {
...     "type": "listener",
...     "notify_of_new": false,
...     "listen_on_all": true,
...     "listen_on": []
... }
AND result has key "id" containing a string, saved as $LISTENID3
AND HTTP Location header is API_URL/orgs/listeners/$LISTENID3

WHEN client GETs /orgs/listeners/$LISTENID1
THEN HTTP status code is 200
AND result matches
... {
```

```
...     "id": "$LISTENID1",
...     "type": "listener",
...     "notify_of_new": true,
...     "listen_on": []
... }

WHEN client GETs /orgs/listeners
THEN HTTP status code is 200
THEN result has key "resources", a list containing
... {"id": "$LISTENID1"}
THEN result has key "resources", a list containing
... {"id": "$LISTENID2"}
```

A listener has no notifications initially.

```
WHEN client GETs /orgs/listeners/$LISTENID1/notifications
THEN HTTP status code is 200
AND result matches
... {
...     "resources": []
... }

WHEN client POSTs /orgs with
... {
...     "names": ["Universal Exports"]
... }
THEN result has key "id" containing a string, saved as $ORGID1
AND result has key "revision" containing a string, saved as $REV2

WHEN client POSTs /orgs with
... {
...     "names": ["Telebulvania Ltd"]
... }
THEN result has key "id" containing a string, saved as $ORGID2
```

After adding the new organizations the first listener should be notified while the second and third should have no notifications.

```
WHEN client GETs /orgs/listeners/$LISTENID1/notifications
THEN HTTP status code is 200
AND result lists resources, id of index 0 saved as $MSGID1
AND result lists resources, id of index 1 saved as $MSGID2

WHEN client GETs /orgs/listeners/$LISTENID1/notifications/$MSGID1
THEN result matches
... {
...     "id": "$MSGID1",
...     "type": "notification",
```

```
...     "resource_id": "$ORGID1",
...     "resource_change": "created"
... }

WHEN client GETs /orgs/listeners/$LISTENID1/notifications/$MSGID2
THEN result matches
... {
...     "id": "$MSGID2",
...     "type": "notification",
...     "resource_id": "$ORGID2",
...     "resource_change": "created"
... }

WHEN client GETs /orgs/listeners/$LISTENID2/notifications
THEN HTTP status code is 200
AND result matches
... {
...     "resources": []
... }

WHEN client GETs /orgs/listeners/$LISTENID3/notifications
THEN HTTP status code is 200
AND result matches
... {
...     "resources": []
... }
```

We update the empty listener to listen on organization changes and update the organization checking for the correct notification to appear. The third listener listening to all the changes should get the notification, too.

```
WHEN client PUTs /orgs/listeners/$LISTENID2 with
... {
...     "notify_of_new": false,
...     "listen_on": ["$ORGID1"],
...     "revision": "$REV1"
... }
THEN HTTP status code is 200
AND result matches
... {
...     "type": "listener",
...     "notify_of_new": false,
...     "listen_on": ["$ORGID1"]
... }

WHEN client PUTs /orgs/$ORGID1 with
... {
...     "names": ["Universal Experts"],
...     "revision": "$REV2"
... }
```

```
THEN HTTP status code is 200

WHEN client GETs /orgs/listeners/$LISTENID2/notifications
THEN HTTP status code is 200
AND result lists resources, id of index 0 saved as $MSGID3

WHEN client GETs /orgs/listeners/$LISTENID2/notifications/$MSGID3
THEN result matches
... {
...     "id": "$MSGID3",
...     "type": "notification",
...     "resource_id": "$ORGID1",
...     "resource_change": "updated"
... }

WHEN client GETs /orgs/listeners/$LISTENID3/notifications
THEN HTTP status code is 200
AND result lists resources, id of index 0 saved as $MSGID4

WHEN client GETs /orgs/listeners/$LISTENID3/notifications/$MSGID4
THEN result matches
... {
...     "id": "$MSGID4",
...     "type": "notification",
...     "resource_id": "$ORGID1",
...     "resource_change": "updated"
... }
```

The first listener gets no additional notifications.

```
WHEN client GETs /orgs/listeners/$LISTENID1/notifications
THEN HTTP status code is 200
AND result matches
... {
...     "resources": [
...         {"id": "$MSGID1"}, {"id": "$MSGID2"}
...     ]
... }
```

We delete the organization and check for the correct notification to appear.

```
WHEN client DELETEs /orgs/$ORGID1
THEN HTTP status code is 200

WHEN client GETs /orgs/listeners/$LISTENID2/notifications
THEN HTTP status code is 200
AND result lists resources, id of index 1 saved as $MSGID5
```

```
WHEN client GETs /orgs/listeners/$LISTENID2/notifications/$MSGID5
THEN result matches
... {
...     "id": "$MSGID5",
...     "type": "notification",
...     "resource_id": "$ORGID1",
...     "resource_revision": null,
...     "resource_change": "deleted"
... }

WHEN client GETs /orgs/listeners/$LISTENID3/notifications
THEN HTTP status code is 200
AND result lists resources, id of index 1 saved as $MSGID6

WHEN client GETs /orgs/listeners/$LISTENID3/notifications/$MSGID6
THEN result matches
... {
...     "id": "$MSGID6",
...     "type": "notification",
...     "resource_id": "$ORGID1",
...     "resource_revision": null,
...     "resource_change": "deleted"
... }
```

The first listener gets no additional notifications.

```
WHEN client GETs /orgs/listeners/$LISTENID1/notifications
THEN HTTP status code is 200
AND result matches
... {
...     "resources": [
...         {"id": "$MSGID1"}, {"id": "$MSGID2"}
...     ]
... }
```

Deletion of a listener deletes also the notifications.

```
WHEN client DELETEs /orgs/listeners/$LISTENID1
THEN HTTP status code is 200

WHEN client GETs /orgs/listeners/$LISTENID1/notifications/$MSGID1
THEN HTTP status code is 404

WHEN client GETs /orgs/listeners/$LISTENID1/notifications/$MSGID2
THEN HTTP status code is 404

WHEN client GETs /orgs/listeners/$LISTENID1
THEN HTTP status code is 404
```

Notification can be deleted.

```
WHEN client DELETEs /orgs/listeners/$LISTENID2/notifications/$MSGID3
THEN HTTP status code is 200

WHEN client GETs /orgs/listeners/$LISTENID2/notifications/$MSGID3
THEN HTTP status code is 404

WHEN client DELETEs /orgs/listeners/$LISTENID3/notifications/$MSGID4
THEN HTTP status code is 200

WHEN client GETs /orgs/listeners/$LISTENID3/notifications/$MSGID4
THEN HTTP status code is 404

WHEN client DELETEs /orgs/listeners/$LISTENID2/notifications/$MSGID5
THEN HTTP status code is 200

WHEN client DELETEs /orgs/listeners/$LISTENID3/notifications/$MSGID6
THEN HTTP status code is 200
```

Chapter 5

Authentication

The API uses OAuth 2.0 protocol for authentication and OpenID Connect 1.0 as an identity layer.

5.1 API endpoints

The API provides the following end-points for authentication. (FIXME: These are not yet implemented.)

- **`/auth/authorize`** — start the authentication flow
- **`/auth/token`** — get access tokens
- **`/auth/introspection`** — validate access tokens

5.2 Authentication flows

In the following sections we describe the basic steps of the most common authentication flows:

- Authorization code flow (OpenID Connect 1.0 + OAuth 2.0)
- Implicit flow (OpenID Connect 1.0 + OAuth 2.0)
- Client credentials flow (OAuth 2.0)

More detailed descriptions can be found in the standardisation documents:

- https://tools.ietf.org/html/rfc6749
- http://openid.net/specs/openid-connect-core-1_0.html

5.3 Authorization code flow

The authorization code flow is mostly used by server-based clients that are used by end-users through a browser (or some other additional client). The access token is not exposed to the end-user. The authorization code flow is suitable for services that can securely maintain a client secret between themselves and the server (this excludes browser-only clients).

5.3.1 Client application redirects end-user's browser to authorization server's authorize endpoint

- **`GET /auth/authorize`**

Parameters

- **`scope`** — **Optional.** Contain scope values that the client application is requesting access to, e.g. **`uapi_orgs_post`**. Values are separated with spaces.
- **`response_type`** — **Required.** Value must be **`code`** when using authorization code flow.
- **`client_id`** — **Required.** Client application's unique client identifier.
- **`redirect_uri`** — **Required.** Response will be sent to this URI. This URI must match one of the redirect URI values registered for the client.
- **`state`** — **Required.** Unique random string that is used to maintain session between the request and the callback. CSRF mitigation is done by binding the value to browser cookie and validating the state in callback.

Example (line breaks are for display purposes only)

```
EXAMPLE
GET /auth/authorize
?scope=uapi_orgs_get uapi_orgs_id_get
&response_type=code
&client_id=@!1E2D.4C48.2272.F616!0001!CC3B.680A!0008!974A.61B3
&redirect_uri=https://yourservice.com/callback
&state=eyJjbGllbnRJZCI6I
```

Authorization server now checks if the end-user has valid session. If not, the end-user must login to authenticate.

After end-user has successfully authenticated authorization server checks if the client application is trusted. If not, the end-user is requested to provide consent. This means the end-user must explicitly allow the client application to execute operations on the end-user's behalf. **The operations the client application can execute are defined by the scopes requested.**

After end-user has provided consent, end-user's browser is redirected back to the the URI defined in the request's **`redirect_uri`**.

5.4 Authorization server redirects back to the client application

- **`GET https://yourservice.com/callback`**

5.4.0.1 Successful response

Parameters

- **`code`** — The authorization code client application can now use to request an access token. This code has limited lifetime and is single use only.

- **`state`** — State parameter that was sent to the authorization server. Client application must check that this value matches the value stored in the session.

and additionally

- **`session_id`** — Authorization server session identifier.
- **`scope`** — Scopes that were granted. The client application will only be granted access to the scopes that have been registered for the client.

Example (line breaks are for display purposes only)

```
EXAMPLE
GET https://yourservice.com/callback
?code=025d464b-edd7-4546-a478-f14862710084
&state=eyJjbGllbnRJZCI6I
&session_id=f3e8a7f2-b341-45f7-9d50-88c92cba5a8b
&scope=uapi_orgs_get uapi_orgs_id_get
```

Client application parses the query parameters and verifies **`state`**.

5.4.0.2 Error response

Parameters

- **`error`** — A machine readable error code.
- **`error_description`** — A human readable text description of the error.
- **`state`** — State parameter that was sent to the authorization server.

Example (line breaks are for display purposes only)

```
EXAMPLE
GET https://yourservice.com/callback
?error=access_denied
&error_description=The+resource+owner+or+authorization+server+denied+the+req
&state=eyJjbGllbnRJZCI6I
```

Possible error codes are listed in https://tools.ietf.org/html/rfc6749#section-4.1.2.1 and http://openid.net/specs/openid-connect-core-1_0.html#AuthError.

5.4.1 Client application exchanges the authorization code for an access token

- **`POST /auth/token`**

Headers

- **`Authorization`** — **Required.** Contains client application's credentials (**`client_id`** and **`client_secret`** using the HTTP-Basic scheme. Most HTTP request libraries do this automatically when providing the credentials. In summary, the header value is constructed as follows:

    1. Client id and client secret are combined into a string **`client_id:client_secret`**.
    2. The resulting string is then encoded using the base64 encoding.
    3. The authorization method and a space **`"Basic "`** is then put before the encoded string.

- **`Content-Type`** — Value must be **`application/x-www-form-urlencoded`** as request parameters are sent in the HTTP request body.

Parameters

- **`grant_type`** — **Required.** Value must be **`authorization_code`** when using authorization code flow.
- **`redirect_uri`** — **Required.** This URI must match the redirect URI where end-user's browser was redirected after successful authentication and authorization.
- **`code`** — **Required.** Authorization code received from the authorization server.

Example

```
EXAMPLE
POST /auth/token
```

with request body (line breaks between values are for display purposes only)

```
EXAMPLE
grant_type=authorization_code
&redirect_uri=https://yourservice.com/callback
&code=025d464b-edd7-4546-a478-f14862710084
```

If the token request fails, the authorization server will return HTTP status code 400 or 401 as defined in OAuth 2.0 specification http://tools.ietf.org/html/rfc6749#section-5.2.

If the token request is successful, the authorization server will respond with HTTP status code 200 and a JSON response with the following key-value pairs:

- **`access_token`** — The access token
- **`token_type`** — Type of the access token. The value is always **`bearer`**.
- **`expires_in`** — Lifetime of the access token in seconds.
- **`refresh_token`** — Single use token that can be exchanged for an access token.

Access tokens have limited lifetime and a new token must be acquired during longer sessions.

5.4.2 Client application can now make requests with the access token.

Using the access token in API requests is described in a later section.

5.5 Implicit flow

The implicit flow is mainly used by JavaScript browser clients and is suitable for clients that can not securely hold the client secret. The access token is returned to the client which may expose it to the end-user and other applications that have access to the end-user's browser. Implicit flow uses only the authorize endpoint, which also returns the access token.

5.6 Client credentials flow

Client credentials flow is mostly limited to internal services that communicate directly with the API without an end-user. Client credentials flow uses only the token endpoint to directly acquire an access token. This flow is not defined in OpenID Connect 1.0 as it does not verify the end-user identity, only the client identity (which in this case is also the end-user).

5.6.1 Client application gets the access token

- **`POST /auth/token`**

Headers

- **`Authorization`** — **Required.** Contains client application's credentials (**`client_id`** and **`client_secret`** using the HTTP-Basic scheme (Value is **`Basic`** + base64 encoded **`client_id`** + **`:`** + **`client_secret`**).
- **`Content-Type: application/x-www-form-urlencoded`** — Parameters are sent in the HTTP request body.

Parameters

- **`grant_type`** — **Required.** Value must be **`client_credentials`**.
- **`scope`** — **Required.** Contain scope values that the client application is requesting access to, e.g. **`uapi_orgs_post`**. Values are separated with spaces.

Example

```
EXAMPLE
POST /auth/token
```

with request body (line breaks between values are for display purposes only)

```
EXAMPLE
grant_type=client_credentials
&scope=uapi_orgs_post uapi_orgs_get
```

If the token request fails, the authorization server will return HTTP status code 400 or 401 as defined in OAuth 2.0 specification http://tools.ietf.org/html/rfc6749#section-5.2.

If the token request is successful, the authorization server will respond with HTTP status code 200 and a JSON response with the following key-value pairs:

- **`access_token`** — The access token
- **`scope`** — Scopes that were granted. The client application will only be granted access to the scopes that have been registered for the client.
- **`token_type`** — Type of the access token. The value is always **`bearer`**.
- **`expires_in`** — Lifetime of the access token in seconds.
- **`refresh_token`** — Single use token that can be exchanged for an access token.

Access tokens have limited lifetime and a new token has to be acquired during longer sessions.

5.6.2 Client application can now make requests with the access token.

Using the access token in API requests is described in a later section.

Chapter 6

Authorization

The access to API resources is controlled by access scopes. The scopes are defined per resource (and resource listing) and per HTTP method. Client needs to define, and to have been given access to, the scopes when authenticating to the server.

6.1 Access scopes

The API scopes are prefixed with **`uapi`** and suffixed with the lowercase HTTP method name. Otherwise the scope naming follows the resource path with resource ids replaced with the **`id`** and slashes replaced with underscores.

For search scopes the whole search condition part is replaced with **`id`**. Otherwise the naming rules are the same.

If a client makes a request with a token that has insufficient access scopes, the API returns **`WWW-Authenticate`** header with value **`Bearer error="insufficient_scope"`** as specified in RFC 6750.

6.1.1 Examples

To add a new organisation, POST to /orgs, requires the scope **`uapi_orgs_post`**.

To get a list of organisations, GET to /orgs, requires the scope **`uapi_orgs_get`**.

To get a single organisation, GET to /orgs/123, requires the scope **`uapi_orgs_id_get`**.

To change a contract's document, PUT to /contracts/123/document, requires the scope **`uapi_contracts_id_docu`**

To search for persons, GET to /persons/search/SEARCH_CONDITION, requires the scope **`uapi_persons_search`**

6.2 Using the access token with the API

All API resource requests must contain the access token in **`Authorization`** header. Header value must be **`Bearer`** and the access token concatenated and separated with a space.

Example

`EXAMPLE`
`GET /orgs`

Headers

- **`Authorization`** — **Required.** Value is **`Bearer`** + the access token.

If an expired or otherwise invalid token is provided, the response contains **`WWW-Authenticate`** header with value **`Bearer error="invalid_token"`**.

6.2.1 Tests

We use persons and orgs resources with limited access.

```
SCENARIO client access authorization
```

Client has access to get a list of persons.

```
GIVEN client has access to scopes
... "uapi_persons_get"
```

Request for the list of persons is successful.

```
WHEN client GETs /persons
THEN HTTP status code is 200
```

The client can not create new persons.

```
WHEN client POSTs /persons with {}
THEN HTTP status code is 403
```

The client can not access other resources.

```
WHEN client GETs /orgs
THEN HTTP status code is 403
AND HTTP header WWW-Authenticate is Bearer error="insufficient_scope"
```

A request with an invalid token.

```
GIVEN an invalid access token
WHEN client GETs /persons
THEN HTTP status code is 401
AND HTTP header WWW-Authenticate is Bearer error="invalid_token"
```

Chapter 7

API resources

This chapter is the detailed reference manual for the API resources and simultaneously its automated acceptance test suite. The API is divided into a set of resources, most of which correspond to entities in the data model described in the previous chapter. For each API resources, the following are documented:

- The purpose of the resource.
- URL path to the resource.
- What HTTP operations are supported for the resource.
- What input is to be provided.
- What output is provided in the successful case.
- What error situations are and what the API client needs to do about them.
- Any other information that seems useful to the API client.
- Example code.
- Automated tests for verifying the API call works correctly; these function as additional examples.

7.1 `/version`

This read-only resource reports version information of the API instance. It requires no authentication and is available to everyone.

Synopsis:

- **`GET /version`** with an empty body.

Note that **`/version`** does not support searching.

Errors:

- There are no errors that are specific to this endpoint.

Example result:

```
EXAMPLE
{
    "api": {
        "version": "1.0"
    },
    "implementation": {
        "version": "2014-12-05",
        "name": "Alfred Pennington"
    }
}
```

Fields in the result (main and subkey):

- **api**, **version** — API version
- **implementation**, **version** — implementation version
- **implementation**, **name** — name of implementation

All values are meant to be human-readable and are not meant to be interpreted programmatically.

7.1.1 Tests

We do the GET call to get **/version**, and check that the result is a JSON dict with the fields as described above. There may be other fields, but we don't care about that.

```
SCENARIO retrieve version information
WHEN client GETs /version
THEN HTTP status code is 200
AND result has key "api" with subkey "version"
AND result has key "implementation" with subkey "name"
AND result has key "implementation" with subkey "version"
```

y

Chapter 8

`/resource_types` – resource type definitions

NOTE THAT this section is a glimpse of the future, left here by a confused time traveller.

API clients may define and manage resource types, if properly authorized.

Synopsis:

- **`GET /resource_types`** — get a list of ids of all resource types.
- **`POST /resource_types`** — add a new resource type.
- **`GET /resource_types/<id>`** — get information about a specific resource type.
- **`PUT /resource_types/<id>`** — update a resource type
- **`DELETE /resource_types/<id>`** — remove a resource type.

Errors:

- There are no errors that are specific to this resource.

Example result:

```
EXAMPLE information about a resource type
{
    "type": "resource_type",
    "id": "12313",
    "name": "foo",
    "versions": [
        {
            "prototype": {
                "blerp": "",
                "glurp": ""
            },
            "version": "v0"
        }
    ]
}
```

Chapter 9

Resource type definition using prototypes

NOTE: This chapter documents a temporary phase in the development of Qvarn: one where resource types are defined by adding YAML files into **`/etc/qvarn`**. In the future, resource types will be defined via the Qvarn HTTP API, but that time is not yet here. Even after the API works, it will use the same "languaged" based on prototypes as described here.

9.1 Defining resource types for Qvarn

When Qvarn is run in "prepare mode", it reads resource type specifications from **`/etc/qvarn`** and inserts them into a database table (**`resource_types`**). When it's run in normal mode, Qvarn will only look at the database table.

The resource type specifications are YAML files with the following structure:

```
EXAMPLE a resource type specification
type: person
path: /persons
versions:
- version: v0
  prototype:
    type: ""
    id: ""
    revision: ""
- version: v1
  prototype:
    type: ""
    id: ""
    revision: ""
    names:
    - full_name: ""
      sort_key: ""
      titles: [""]
      given_names: [""]
      surnames: [""]
  subpaths:
```

```
    photo:
      prototype:
        body: blob
        content_type: ""
    private:
      prototype:
        date_of_birth: ""
  files:
  - photo
```

Some notes:

- **`type`** is the name of the resource type. Any resources of that type need to specify this name.
- **`path`** is the path under which the resources are available on the API. Note that Qvarn does **not** create the path automatically (by adding an s to the end of the type).
- **`versions`** is a list of resource type versions. Every version must have a unique version string, which need not be in any particular format. Version strings are only compared for equality and do not define an order. The order in which versions occur in the **`versions`** list defines the order.

Note that if there is any change at all to a resource type, a new version **must** be added. Qvarn does **not** support fields being added or otherwise changed. Qvarn keeps track of which version (and thus which database schema) it is using by the version string only. Also, a new version **must** be done by adding a new version to the list, not by just changing the latest entry's version string. Otherwise Qvarn's automatic transition of the database schema won't work.

Each version of a resource type consists of a "prototype", a list of sub-resources, and list of which sub-resources are actually files. The prototype defines what fields the resource has, and what types of value each field has. Sub-resources are defined similarly with prototypes.

EXAMPLE simple version

```
- version: v0
  prototype:
    type: ""
    id: ""
    revision: ""
```

In the v0 example, the resource type has three fields: the standard three fields we put in every resource type. A field is specified by giving its name, and the type of the value. The value type is specified using one of the following:

- **`""`** – value is a Unicode string. String may be empty or null.
- **`blob`** – value is an arbitrary binary string (e.g., a JPEG). May be empty or null. Cannot be part of a list.
- **`[""]`** – value is a list of Unicode strings. List may be empty.
- A list with a dict with fields – value is a list of dicts. List may be empty. See version v1, field **`names`**.

To define a sub-resource, add it to the **`subpaths`** part of a resource version. To make the sub-resource be a "file attachment" instead of a JSON object, add the name of the sub-resource to the **`files`** part of a resource version; see **`photo`** in the example above. A file attachment should have the fields **`body`** and **`content_type`** as in the example.

9.2 /persons

This resource manages all the people known to the service.

Synopsis:

- **GET /persons** — get a list of ids of all persons.
- **POST /persons** — add a new person.
- **GET /persons/<id>** — get information about a specific person.
- **GET /persons/<id>/private** — get a person's sensitive, private information.
- **PUT /persons/<id>** — replace the information for a person.
- **PUT /persons/<id>/private** — replace the sensitive, private information for a person.
- **GET /persons/<id>/photo** — get the attached photo for a specific person
- **PUT /persons/<id>/photo** — replace the attached photo for a person
- **DELETE /persons/<id>** — remove a person (including private information).

Errors:

- There are no errors that are specific to this resource.

Information about a person is divided into two parts: semi-public and private. Access to the two parts is controlled separately.

Example result:

```
EXAMPLE information about a person
{
    "type": "person",
    "id": "12313",
    "names": [
        {
            "full_name": "James Bond",
            "sort_key": "Bond, James",
            "titles": [],
            "given_names": ["James"],
            "surnames": ["Bond"]
        },
        {
            "full_name": "Alfred E. Newman",
            "sort_key": "Newman, Alfred E.",
            "titles": [],
            "given_names": ["Alfred", "E."],
            "surnames": ["Newman"]
        }
    ],
    "gluu_user_id": "@!2027.861B.4505.5885!0001!200B.B5FE!0000!5DBD.0EA8"
}
```

Example of a person's private information:

```
EXAMPLE private information about a person
{
    "id": "12313",
    "date_of_birth": null,
    "gov_ids": [
        {
            "country": "GB",
            "id_type": "ssn",
            "gov_id": "SN 00 70 07"
        }
    ],
    "nationalities": [
        "GB"
    ],
    "residences": [
        {
            "country": "GB",
            "location": "London"
        }
    ],
    "contacts": [
        {
            "contact_type": "phone",
    "contact_roles": [],
            "contact_source": "self",
            "contact_timestamp": "2038-02-28T01:02:03+0400",
            "phone_number": "+358 4321"
        }
    ]
}
```

Fields in the result:

- **`names`** — all names known about the person, each name specified as described in Handling names of people.
- **`gluu_user_id`** — the Gluu user identifier (user inum) corresponding to this person (not filled in by Qvarn at this time)

Fields in the private information:

- **`date_of_birth`** — ***optional*** date of birth information
- **`gov_ids`** — all known government-issued identifiers, such as social security numbers; see below for details
- **`nationalities`** — current nationalities of the person, list of two-letter country codes
- **`residences`** — known residences; see below
- **`contacts`** — known contact information; see Handling contact information

Government-issued identifiers are encoded as a JSON object with the following fields:

- **`country`** — two-letter country code

- **id_type** — the type of the id, one of:
  - **ssn** — social security number or similar, including Finnish “henkilötunnus”)
  - **tax_number** — a tax office identifier (including Finnish “veronumero”)
- **gov_id** — the actual identifier

Residences are stored as a JSON object with the following fields:

- **country** — two-letter country code
- **location** — free form string for location within the country

9.2.1 Tests

We create a new person, update them, and delete them.

```
SCENARIO manage a person
```

Client has needed access rights for persons resource.

```
GIVEN client has access to scopes
... "uapi_persons_post uapi_persons_get uapi_persons_id_get
...  uapi_persons_id_private_get uapi_persons_id_put
...  uapi_persons_id_private_put uapi_persons_id_delete
...  uapi_persons_id_photo_get uapi_persons_id_photo_put"
```

Try to create a new person, but with invalid data. These must all fail, of course.

Only specific fields are allowed. The API does not accept arbitrary data from the client.

```
WHEN client POSTs /persons with {"invalid_field": true}
THEN HTTP status code is 400
```

Extra fields for a name are not allowed.

```
WHEN client POSTs /persons with
... {"names": [{"full_name": "James Bond", "nickname": "Jimbo"}]}
THEN HTTP status code is 400
```

Create a new person. Note that even if we only submit the mandatory fields, we get a result with all optional fields filled in with empty values.

```
WHEN client POSTs /persons with
... {
...     "names": [
...         {
...             "full_name": "James Bond",
...             "sort_key": "Bond, James",
```

```
...             "titles": ["Mr"],
...             "given_names": ["James"],
...             "surnames": ["Bond"]
...         },
...         {
...             "full_name": "Alfred E. Newman"
...         }
...     ]
... }
THEN HTTP status code is 201
AND result matches
... {
...     "names": [
...         {
...             "full_name": "James Bond",
...             "sort_key": "Bond, James",
...             "titles": ["Mr"],
...             "given_names": ["James"],
...             "surnames": ["Bond"]
...         },
...         {
...             "full_name": "Alfred E. Newman"
...         }
...     ]
... }
AND result has key "id" containing a string, saved as $ID1
AND HTTP Location header is API_URL/persons/$ID1
AND result has key "revision" containing a string, saved as $REV1
```

Check that the record is there.

```
WHEN client GETs /persons
THEN HTTP status code is 200
AND result has key "resources", a list containing { "id": "$ID1" }

WHEN client GETs /persons/$ID1
THEN HTTP status code is 200
AND result matches
... {
...     "names": [
...         {
...             "full_name": "James Bond",
...             "sort_key": "Bond, James",
...             "titles": ["Mr"],
...             "given_names": ["James"],
...             "surnames": ["Bond"]
...         },
...         {
...             "full_name": "Alfred E. Newman"
```

```
...         }
...     ]
... }
AND result has key "id" containing a string, saved as $ID2
AND values "$ID1" and "$ID2" are identical
AND result has key "revision" containing a string, saved as $REV2
AND values "$REV1" and "$REV2" are identical
```

The private sub-record should also be there, without being added explicitly, even if it has no data.

```
WHEN client GETs /persons/$ID1/private
THEN HTTP status code is 200
```

Update the record.

```
WHEN client PUTs /persons/$ID1 with
... {
...     "revision": "$REV1",
...     "names": [{"full_name": "M"}]
... }
THEN HTTP status code is 200
AND result matches { "names": [{"full_name": "M"}]}
AND result has key "id" containing a string, saved as $ID3
AND values "$ID1" and "$ID3" are identical
AND result has key "revision" containing a string, saved as $REV3
```

Subrecord should still be there.

```
WHEN client GETs /persons/$ID1/private
THEN HTTP status code is 200
```

Updating with the wrong revision should fail.

```
WHEN client PUTs /persons/$ID1 with
... {
...     "revision": "$REV1",
...     "names": [{"full_name": "Blofeld"}]
... }
THEN HTTP status code is 409
WHEN client GETs /persons/$ID1
THEN result matches
... {
...     "revision": "$REV3",
...     "names": [{"full_name": "M"}]
... }
```

Update the private sub-record. Note that we give the revision of the main record.

```
WHEN client PUTs /persons/$ID1/private with
... {
...     "revision": "$REV3",
...     "date_of_birth": "1920-11-11"
... }
THEN HTTP status code is 200
AND result matches { "date_of_birth": "1920-11-11" }
AND result has key "revision" containing a string, saved as $REV4
```

(Not bothering to check that all the fields are set. Let's assume they are, to keep the test a bit shorter.)

Updating the sub-record without the correct revision should fail.

```
WHEN client PUTs /persons/$ID1/private with
... {
...     "revision": "this-is-the-wrong-revision",
...     "date_of_birth": "1920-11-11"
... }
THEN HTTP status code is 409
```

Can not get unsent photo.

```
WHEN client GETs /persons/$ID1/photo
THEN HTTP status code is 404
```

Send photo.

```
GIVEN file photo.png containing "\x89"

WHEN client PUTs file photo.png with content type image/png
...  and revision $REV4 to /persons/$ID1/photo
THEN HTTP status code is 200

WHEN client GETs /persons/$ID1/photo
THEN HTTP status code is 200
AND HTTP header Content-Type is image/png
AND HTTP header Revision exists
AND result body matches file photo.png
```

Without request body (no Content-Length header) server responds with status Length Required 411.

```
WHEN client POSTs /persons with
... {"names": [{"full_name": "James Bond"},
...            {"full_name": "Alfred E. Newman"}]}
THEN result has key "id" containing a string, saved as $ID4
AND result has key "revision" containing a string, saved as $REV5

WHEN client PUTs no file with content type image/png
...  and revision $REV5 to /persons/$ID4/photo
THEN HTTP status code is 411
```

Update the main record, this should not affect the subrecord(s).

```
WHEN client GETs /persons/$ID1
THEN result has key "revision" containing a string, saved as $REV6

WHEN client PUTs /persons/$ID1 with
... {
...     "revision": "$REV6",
...     "names": [{"full_name": "X"}]
... }
THEN HTTP status code is 200
AND result matches { "names": [{"full_name": "X"}]}

WHEN client GETs /persons/$ID1/private
THEN HTTP status code is 200
AND result matches { "date_of_birth": "1920-11-11" }

WHEN client GETs /persons/$ID1/photo
THEN HTTP status code is 200
AND HTTP header Content-Type is image/png
AND HTTP header Revision exists
AND result body matches file photo.png
```

Delete the record.

```
WHEN client DELETEs /persons/$ID1
THEN HTTP status code is 200

WHEN client GETs /persons/$ID1
THEN HTTP status code is 404
```

The private sub-record should be gone too.

```
WHEN client GETs /persons/$ID1/private
THEN HTTP status code is 404
```

9.3 `/orgs`

This resource manages all the organisations known to the service. An organisation can be a company, but also a non-profit association, city, or similar, which is why the more general term is used.

Synopsis:

- **`GET /orgs`** — get a list of ids of all organisations.
- **`POST /orgs`** — add a new organisation.
- **`GET /orgs/<id>`** — get the information about a specific organisation.
- **`PUT /orgs/<id>`** — replace the information for an organisation.

- **`DELETE /orgs/<id>`** — remove an organisation.

Errors:

- There are no errors that are specific to this resource.

Example result for an organisation:

EXAMPLE

```
{
    "names": [
        "Full Company Name Ltd",
        "Da Company"
    ],
    "country": "FI",
    "gov_org_ids": [
        {
            "country": "FI",
            "org_id_type": "registration_number",
            "gov_org_id": "1234567-1"
        }
    ],
    "contacts": [
        {
            "contact_type": "address",
            "contact_source": "self",
            "contact_timestamp": "2015-03-04T16:47:12+0200",
            "country": "FI",
            "full_address": "Street 1\n00000 Nowhere"
        }
    ]
}
```

Fields in the result:

- **`names`** — list of all names known about the organisation
- **`country`** — two letter country code for the reported home country of the organisation
- **`gov_org_ids`** — all known government-issued identifiers; see below
- **`contacts`** — known contact information; see Handling contact information

Government-issued identifiers (**`gov_org_ids`**) have the following fields:

- **`country`** — two letter country code
- **`gov_id_type`** — the type of the id, one of:
  - **`registration_number`** — organisation registration number, which would be “Y-tunnus” in Finland
  - **`vat_number`** — a VAT identification number, used within the EU and elsewhere
- **`gov_org_id`** — the actual identifier

9.3.1 Tests

We create a new organisation, update its info, and delete it.

```
SCENARIO manage an organisation
```

Client has needed access rights for orgs resource.

```
GIVEN client has access to scopes
... "uapi_orgs_post uapi_orgs_id_get uapi_orgs_id_put
...  uapi_orgs_id_delete"
```

Create a new organisation.

```
WHEN client POSTs /orgs with
... {"names": ["Suomen Tilaajavastuu Oy"],
... "country": "FI",
... "gov_org_ids": [{"country": "FI",
...                  "org_id_type": "registration_number",
...           "gov_org_id": "1234567-1"}],
... "contacts": [{"contact_type": "address",
...               "contact_roles": ["billing"],
...               "contact_source": "self",
...            "contact_timestamp": "2015-03-04T16:47:12+0200",
...            "country": "FI",
...            "full_address": "Street 1\\n00000 Nowhere"},
...           {"contact_type": "phone",
...               "contact_source": "self",
...            "contact_timestamp": "2015-03-04T16:47:12+0200",
...            "phone_number": "+15551234567890"},
...           {"contact_type": "einvoice",
...               "contact_roles": ["billing"],
...               "contact_source": "self",
...            "contact_timestamp": "2015-03-04T16:47:12+0200",
...            "einvoice_operator": "FOOBANK",
...            "einvoice_address": "003712345671"}]
... }
THEN HTTP status code is 201
AND result matches
... { "names": ["Suomen Tilaajavastuu Oy"],
...   "country": "FI",
...   "gov_org_ids": [{"country": "FI",
...              "org_id_type": "registration_number",
...              "gov_org_id": "1234567-1"}],
... "contacts": [{"contact_type": "address",
...               "contact_roles": ["billing"],
...               "contact_source": "self",
...            "contact_timestamp": "2015-03-04T16:47:12+0200",
```

```
...           "country": "FI",
...           "full_address": "Street 1\\n00000 Nowhere"},
...          {"contact_type": "phone",
...               "contact_source": "self",
...           "contact_timestamp": "2015-03-04T16:47:12+0200",
...           "phone_number": "+15551234567890"},
...          {"contact_type": "einvoice",
...               "contact_roles": ["billing"],
...               "contact_source": "self",
...           "contact_timestamp": "2015-03-04T16:47:12+0200",
...           "einvoice_operator": "FOOBANK",
...           "einvoice_address": "003712345671"}]
... }
AND result has key "id" containing a string, saved as $ID1
AND HTTP Location header is API_URL/orgs/$ID1
AND result has key "revision" containing a string, saved as $REV1
```

Check that the record is there.

```
WHEN client GETs /orgs/$ID1
THEN HTTP status code is 200
AND result matches
... { "names": ["Suomen Tilaajavastuu Oy"],
...   "country": "FI",
...   "gov_org_ids": [{"country": "FI",
...             "org_id_type": "registration_number",
...             "gov_org_id": "1234567-1"}],
... "contacts": [{"contact_type": "address",
...               "contact_roles": ["billing"],
...               "contact_source": "self",
...           "contact_timestamp": "2015-03-04T16:47:12+0200",
...           "country": "FI",
...           "full_address": "Street 1\\n00000 Nowhere"},
...          {"contact_type": "phone",
...               "contact_source": "self",
...           "contact_timestamp": "2015-03-04T16:47:12+0200",
...           "phone_number": "+15551234567890"},
...          {"contact_type": "einvoice",
...               "contact_roles": ["billing"],
...               "contact_source": "self",
...           "contact_timestamp": "2015-03-04T16:47:12+0200",
...           "einvoice_operator": "FOOBANK",
...           "einvoice_address": "003712345671"}]
... }
AND result has key "id" containing a string, saved as $ID2
AND values "$ID1" and "$ID2" are identical
AND result has key "revision" containing a string, saved as $REV2
AND values "$REV1" and "$REV2" are identical
```

Update the record.

```
WHEN client PUTs /orgs/$ID1 with
... {
...     "revision": "$REV1",
...     "names": ["Tilaajavastuu"]
... }
THEN HTTP status code is 200
AND result matches
... {
...     "names": ["Tilaajavastuu"]
... }
AND result has key "id" containing a string, saved as $ID3
AND values "$ID1" and "$ID3" are identical
AND result has key "revision" containing a string, saved as $REV3
```

Delete the record.

```
WHEN client DELETEs /orgs/$ID1
THEN HTTP status code is 200

WHEN client GETs /orgs/$ID1
THEN HTTP status code is 404
```

Attempt to send invalid data.

```
WHEN client POSTs /orgs with {"nonexistent_field": "data"}
THEN HTTP status code is 400
```

9.4 `/contracts`

This resource manages various types of contracts between organisations and persons. A contract can be of any type: a contract to supply services on a construction site; employment; or similar. All the types are collected into one resource type to avoid an explosion in the types of resources.

Synopsis:

- **`GET /contracts`** — get a list of ids of all contracts
- **`POST /contracts`** — add a new contract
- **`GET /contracts/<id>`** — get the information about a specific contract
- **`GET /contracts/<id>/document`** — get the attached document for a specific contract
- **`PUT /contracts/<id>`** — replace the information for a contract
- **`PUT /contracts/<id>/document`** — replace the attached document for a contract
- **`DELETE /contracts/<id>`** — remove a contract

Errors:

- There are no errors that are specific to this resource.

Possible contract documentation can be stored as attachments to contracts. The attached documentation is accessed separately.

Example result for a contract:

EXAMPLE An example of an employment contract

```
{
    "type": "contract",
    "id": "1234",
    "contract_type": "employment",
    "start_date": "2015-03-01",
    "end_date": "2015-08-31",
    "contract_parties": [
        {
            "type": "org",
            "resource_id": "123",
            "role": "employer",
        },
        {
            "type": "person",
            "resource_id": "7",
            "role": "employee"
        }
    ],
    "right_to_work_based_on": "eea_citizen",
    "contract_state": "active"
}
```

Fields in the result:

- **contract_type** — type of the contract, one of:
  - **employment** — an employment contract
  - **induction** — an induction document between an organisation and a person
  - **ID06_cards** — contract between an ID06 card issuer and supplier (see the cards resource for details)
  - **tilaajavastuu_account** — a user account for Tilaajavastuu service
  - **SBCC_account** — a user account for SBCC service
  - **service_application** — an application for a service by an organisation, applied by a person
- **start_date** — the date when the contract takes effect
- **end_date** — ending date of the contract
- **date_created** — date when the contract was created
- **date_updated** — date when the contract was last updated
- **contract_parties** — all parties in the contract, where each party is defined by the following fields:
  - **type** — type of the party, one of **org**, **person**
  - **resource_id** — identifier of the party
  - **role** — role in the contract, see below
  - **username** — username for the service, if this contract is a service account contract
  - **user_role** — name of the user role, if this contract is a service account contract

  - **permissions** — a list of permissions, if this contract is a service account contract, see below
  - **global_permissions** — a list of global permissions, if this contract is a service account contract, see below

- **terms_of_service** — information on the terms of a service contract
- **right_to_work_based_on** — basis for a foreign employee's right to work, one of:

  - **eea_citizen** — citizenship of an EU or EEA country
  - **residence_permit** — an unspecified residence permit
  - **other** — an unspecified other basis

- **id06_issuer_requires_bankid** — does the issuer require BankID authentication for enabling each virtual ID06 card?
- **id06_contact_name** — name of ID06 contact person
- **id06_contact_email** — email address of the ID06 contact person
- **contract_state** — state of the contract, if applicable
- **contract_state_history** — state history for the contract; the last entry in the list is the current state; see below for list items
- **signers** — list of signers for the contract; see below for list items
- **preferred_language** — preferred language of communication in cases where this contract is a user account contract

Contract parties have a role in which they appear in the contract. The allowed roles are dependent on the type of contract, as follows:

- **employment** contracts:

  - **employer** — must be an **org**
  - **employee** — must be a **person**

- **induction** contracts:

  - **inductor** — must be an **org**
  - **inductee** — must be a **person**

- **ID06_cards** contracts:

  - **issuer** — must be an **org**
  - **supplier** — must be an **org**
  - **issuer_contact** — person at issuer responsible for ordering ID06 card; must be a **person**

- **tilaajavastuu_account** contracts:

  - **user** — must be a **person**
  - **target** — must be an **org** or a **person**

- **SBCC_account** contracts:

  - **user** — must be a **person**
  - **target** — must be an **org** or a **person**

- **service_application** contracts:

  - **applicant** — must be an **org** or a **person**

Contract state history is a list of structures of the following form:

- **state** — the new state of the contract
- **modified_by** — must be a **person**
- **modification_timestamp** — time of the state change

The **signers** list consists of structures of the following form:

- **signer** — must be a **person**
- **signing_request_timestamp** — time when the request was sent
- **signing_request_message** — a custom message for the signer
- **signing_timestamp** — time when the signer signed the contract

The **permissions** and **global_permissions** lists consist of structures of the following form:

- **permission_name** — name of the granted permission

The **terms_of_service** list consists of structures of the following form:

- **terms_of_service_version** — version of these terms of service
- **terms_of_service_language** — language of the accepted terms
- **acceptance_time** — timestamp when these terms where accepted
- **accepter_person_id** — id of the **person** who accepted these terms

9.4.1 Tests

We create a new contract, update its info, and delete it.

```
SCENARIO manage a contract
```

Client has needed access rights for contracts resource.

```
GIVEN client has access to scopes
... "uapi_contracts_post uapi_contracts_id_get uapi_contracts_id_put
...  uapi_contracts_id_document_put uapi_contracts_id_document_get
...  uapi_contracts_id_delete"
```

Create a new contract.

```
WHEN client POSTs /contracts with
... {
... "contract_type": "employment",
... "start_date": "2013-03-13",
... "contract_parties": [
...     {
```

```
...         "type": "org",
...         "resource_id": "123",
...         "role": "employer"
...     },
...     {
...         "type": "person",
...         "resource_id": "7",
...         "role": "employee"
...     }
... ],
... "right_to_work_based_on": "eea_citizen",
... "contract_state": "active"
... }
THEN HTTP status code is 201
AND result matches
... {
... "type": "contract",
... "contract_type": "employment",
... "start_date": "2013-03-13",
... "contract_parties": [
...     {
...         "type": "org",
...         "resource_id": "123",
...         "role": "employer",
...         "contacts": []
...     },
...     {
...         "type": "person",
...         "resource_id": "7",
...         "role": "employee",
...         "contacts": []
...     }
... ],
... "right_to_work_based_on": "eea_citizen",
... "contract_state": "active"
... }
AND result has key "id" containing a string, saved as $ID1
AND HTTP Location header is API_URL/contracts/$ID1
AND result has key "revision" containing a string, saved as $REV1
```

Check that the record is there.

```
WHEN client GETs /contracts/$ID1
THEN HTTP status code is 200
AND result matches
... {
... "type": "contract",
... "contract_type": "employment",
... "start_date": "2013-03-13",
```

```
... "contract_parties": [
...     {
...         "type": "org",
...         "resource_id": "123",
...         "role": "employer",
...         "contacts": []
...     },
...     {
...         "type": "person",
...         "resource_id": "7",
...         "role": "employee",
...         "contacts": []
...     }
... ],
... "right_to_work_based_on": "eea_citizen",
... "contract_state": "active"
... }
AND result has key "id" containing a string, saved as $ID2
AND values "$ID1" and "$ID2" are identical
AND result has key "revision" containing a string, saved as $REV2
AND values "$REV1" and "$REV2" are identical
```

Update the record.

```
WHEN client PUTs /contracts/$ID1 with
... {
... "revision": "$REV1",
... "type": "contract",
... "contract_type": "employment",
... "start_date": "2013-03-13",
... "end_date": "2015-10-21",
... "contract_parties": [
...     {
...         "type": "org",
...         "resource_id": "123",
...         "role": "employer",
...         "contacts": [
...             {
...                 "contact_type": "address",
...                 "contact_source": "hat",
...                 "contact_timestamp": "20160630T141516",
...                 "phone_number": "",
...                 "email_address": "",
...                 "full_address": "10 Downing St, London, UK",
...                 "country": "UK",
...                 "address_lines": ["10 Downing St", "London", "UK"],
...                 "post_code": "1 BP",
...                 "post_area": "Buckingham Palace"
...             }
```

```
...         ]
...     },
...     {
...         "type": "person",
...         "resource_id": "7",
...         "role": "employee",
...         "contacts": []
...     }
... ],
... "right_to_work_based_on": "residence_permit"
... }
THEN HTTP status code is 200
AND result matches
... {
... "type": "contract",
... "contract_type": "employment",
... "start_date": "2013-03-13",
... "end_date": "2015-10-21",
... "contract_parties": [
...     {
...         "type": "org",
...         "resource_id": "123",
...         "role": "employer",
...         "contacts": [
...             {
...                 "contact_type": "address",
...                 "contact_source": "hat",
...                 "contact_timestamp": "20160630T141516",
...                 "phone_number": "",
...                 "email_address": "",
...                 "full_address": "10 Downing St, London, UK",
...                 "country": "UK",
...                 "address_lines": ["10 Downing St", "London", "UK"],
...                 "post_code": "1 BP",
...                 "post_area": "Buckingham Palace"
...             }
...         ]
...     },
...     {
...         "type": "person",
...         "resource_id": "7",
...         "role": "employee",
...         "contacts": []
...     }
... ],
... "right_to_work_based_on": "residence_permit"
... }
AND result has key "id" containing a string, saved as $ID3
AND values "$ID1" and "$ID3" are identical
```

```
AND result has key "revision" containing a string, saved as $REV3
```

Can not get unsent documentation.

```
WHEN client GETs /contracts/$ID1/document
THEN HTTP status code is 404
```

Send documentation.

```
GIVEN file test.file containing "\x89\xff This pretends to be a pdf"

WHEN client PUTs file test.file with content type application/pdf
...  and revision $REV3 to /contracts/$ID1/document
THEN HTTP status code is 200

WHEN client GETs /contracts/$ID1/document
THEN HTTP status code is 200
AND HTTP header Content-Type is application/pdf
AND HTTP header Revision exists
AND result body matches file test.file
```

Can not send documentation with no Content-Length header set.

```
WHEN client POSTs /contracts with
... {
... "contract_type": "employment",
... "contract_parties": [
...     {
...         "type": "org",
...         "resource_id": "123",
...         "role": "employer",
...         "contacts": []
...     },
...     {
...         "type": "person",
...         "resource_id": "7",
...         "role": "employee",
...         "contacts": []
...     }
... ],
... "right_to_work_based_on": "eea_citizen"
... }
THEN result has key "id" containing a string, saved as $ID4
AND result has key "revision" containing a string, saved as $REV4
```

Without request body (no Content-Length header) server responds with status Length Required 411.

```
WHEN client PUTs no file with content type application/pdf
...  and revision $REV4 to /contracts/$ID4/document
THEN HTTP status code is 411
```

Update the main record, this should not affect the subrecord(s).

```
WHEN client GETs /contracts/$ID1
THEN result has key "revision" containing a string, saved as $REV5

WHEN client PUTs /contracts/$ID1 with
... {
... "revision": "$REV5",
... "contract_type": "employment",
... "contract_parties": [
...     {
...         "type": "org",
...         "resource_id": "123",
...         "role": "employer",
...         "contacts": []
...     },
...     {
...         "type": "person",
...         "resource_id": "7",
...         "role": "employee",
...         "contacts": []
...     }
... ],
... "right_to_work_based_on": "eea_citizen"
... }
THEN HTTP status code is 200
AND result matches
... {
... "contract_type": "employment",
... "contract_parties": [
...     {
...         "type": "org",
...         "resource_id": "123",
...         "role": "employer",
...         "contacts": []
...     },
...     {
...         "type": "person",
...         "resource_id": "7",
...         "role": "employee",
...         "contacts": []
...     }
... ],
... "right_to_work_based_on": "eea_citizen"
... }
```

```
WHEN client GETs /contracts/$ID1/document
THEN HTTP status code is 200
AND HTTP header Content-Type is application/pdf
AND HTTP header Revision exists
AND result body matches file test.file
```

Delete the record.

```
WHEN client DELETEs /contracts/$ID1
THEN HTTP status code is 200

WHEN client GETs /contracts/$ID1
THEN HTTP status code is 404
```

Create a new service account contract.

```
WHEN client POSTs /contracts with
... {
... "contract_type": "SBCC_account",
... "start_date": "2013-03-13",
... "contract_parties": [
...     {
...         "type": "person",
...         "resource_id": "123",
...         "role": "user",
...         "username": "mrbond",
...         "user_role": "secret_agent",
...         "contacts": []
...     },
...     {
...         "type": "org",
...         "resource_id": "MI6",
...         "role": "target",
...         "contacts": []
...     }
... ],
... "preferred_language": "en"
... }
THEN HTTP status code is 201
AND result matches
... {
... "type": "contract",
... "contract_type": "SBCC_account",
... "start_date": "2013-03-13",
... "contract_parties": [
...     {
...         "type": "person",
```

```
...         "resource_id": "123",
...         "role": "user",
...         "username": "mrbond",
...         "user_role": "secret_agent",
...         "contacts": []
...     },
...     {
...         "type": "org",
...         "resource_id": "MI6",
...         "role": "target",
...         "contacts": []
...     }
... ],
... "preferred_language": "en"
... }
AND result has key "id" containing a string, saved as $ID5
AND HTTP Location header is API_URL/contracts/$ID5
AND result has key "revision" containing a string, saved as $REV6
```

Check that the record is there.

```
WHEN client GETs /contracts/$ID5
THEN HTTP status code is 200
AND result matches
... {
... "type": "contract",
... "contract_type": "SBCC_account",
... "start_date": "2013-03-13",
... "contract_parties": [
...     {
...         "type": "person",
...         "resource_id": "123",
...         "role": "user",
...         "username": "mrbond",
...         "user_role": "secret_agent",
...         "contacts": []
...     },
...     {
...         "type": "org",
...         "resource_id": "MI6",
...         "role": "target",
...         "contacts": []
...     }
... ],
... "preferred_language": "en"
... }
AND result has key "id" containing a string, saved as $ID6
AND values "$ID5" and "$ID6" are identical
AND result has key "revision" containing a string, saved as $REV7
```

```
AND values "$REV6" and "$REV7" are identical
```

Create a new service application contract.

```
WHEN client POSTs /contracts with
... {
... "contract_type": "service_application",
... "date_created": "2015-11-30",
... "contract_parties": [
...     {
...         "type": "person",
...         "resource_id": "123",
...         "role": "applicant",
...         "contacts": []
...     },
...     {
...         "type": "org",
...         "resource_id": "536457",
...         "role": "applicant",
...         "contacts": []
...     }
... ],
... "contract_state_history": [
...     {
...         "state": "incomplete",
...         "modified_by": "123",
...         "modification_timestamp": "2015-11-30T14:30:45+0200"
...     }
... ],
... "signers": [
...     {
...         "signer": "1234",
...         "signing_request_timestamp": "2015-11-30T14:30:45+0200",
...         "signing_request_message": "Please, sign this."
...     }
... ]
... }
THEN HTTP status code is 201
AND result matches
... {
... "contract_type": "service_application",
... "date_created": "2015-11-30",
... "contract_parties": [
...     {
...         "type": "person",
...         "resource_id": "123",
...         "role": "applicant",
...         "contacts": []
...     },
```

```
...     {
...         "type": "org",
...         "resource_id": "536457",
...         "role": "applicant",
...         "contacts": []
...     }
... ],
... "contract_state_history": [
...     {
...         "state": "incomplete",
...         "modified_by": "123",
...         "modification_timestamp": "2015-11-30T14:30:45+0200"
...     }
... ],
... "signers": [
...     {
...         "signer": "1234",
...         "signing_request_timestamp": "2015-11-30T14:30:45+0200",
...         "signing_request_message": "Please, sign this."
...     }
... ]
... }
AND result has key "id" containing a string, saved as $ID7
AND HTTP Location header is API_URL/contracts/$ID7
AND result has key "revision" containing a string, saved as $REV8
```

Check that the record is there.

```
WHEN client GETs /contracts/$ID7
THEN HTTP status code is 200
AND result matches
... {
... "contract_type": "service_application",
... "date_created": "2015-11-30",
... "contract_parties": [
...     {
...         "type": "person",
...         "resource_id": "123",
...         "role": "applicant",
...         "contacts": []
...     },
...     {
...         "type": "org",
...         "resource_id": "536457",
...         "role": "applicant",
...         "contacts": []
...     }
... ],
... "contract_state_history": [
```

```
...     {
...         "state": "incomplete",
...         "modified_by": "123",
...         "modification_timestamp": "2015-11-30T14:30:45+0200"
...     }
... ],
... "signers": [
...     {
...         "signer": "1234",
...         "signing_request_timestamp": "2015-11-30T14:30:45+0200",
...         "signing_request_message": "Please, sign this."
...     }
... ]
... }
AND result has key "id" containing a string, saved as $ID8
AND values "$ID7" and "$ID8" are identical
AND result has key "revision" containing a string, saved as $REV9
AND values "$REV8" and "$REV9" are identical
```

Create a new ID06 contract.

```
WHEN client POSTs /contracts with
... {
... "contract_type": "ID06_cards",
... "start_date": "2016-01-01",
... "end_date": "2038-01-19",
... "contract_parties": [
...     {
...         "type": "org",
...         "resource_id": "123",
...         "role": "issuer",
...         "contacts": []
...     },
...     {
...         "type": "org",
...         "resource_id": "536457",
...         "role": "supplier",
...         "contacts": []
...     },
...     {
...         "type": "person",
...         "resource_id": "543534",
...         "role": "issuer_contact",
...         "contacts": []
...     }
... ],
... "id06_issuer_requires_bankid": false,
... "id06_contact_name": "Mikael Blomkvist",
... "id06_contact_email": "mikael.blomkvist@millennium.se"
```

```
... }
THEN HTTP status code is 201
AND result matches
... {
... "contract_type": "ID06_cards",
... "start_date": "2016-01-01",
... "end_date": "2038-01-19",
... "contract_parties": [
...     {
...         "type": "org",
...         "resource_id": "123",
...         "role": "issuer",
...         "contacts": []
...     },
...     {
...         "type": "org",
...         "resource_id": "536457",
...         "role": "supplier",
...         "contacts": []
...     },
...     {
...         "type": "person",
...         "resource_id": "543534",
...         "role": "issuer_contact",
...         "contacts": []
...     }
... ],
... "id06_issuer_requires_bankid": false,
... "id06_contact_name": "Mikael Blomkvist",
... "id06_contact_email": "mikael.blomkvist@millennium.se"
... }
AND result has key "id" containing a string, saved as $ID9
AND HTTP Location header is API_URL/contracts/$ID9
AND result has key "revision" containing a string, saved as $REV10
```

Check that the record is there.

```
WHEN client GETs /contracts/$ID9
THEN HTTP status code is 200
AND result matches
... {
... "contract_type": "ID06_cards",
... "start_date": "2016-01-01",
... "end_date": "2038-01-19",
... "contract_parties": [
...     {
...         "type": "org",
...         "resource_id": "123",
...         "role": "issuer",
```

```
...             "contacts": []
...         },
...         {
...             "type": "org",
...             "resource_id": "536457",
...             "role": "supplier",
...             "contacts": []
...         },
...         {
...             "type": "person",
...             "resource_id": "543534",
...             "role": "issuer_contact",
...             "contacts": []
...         }
... ],
... "id06_issuer_requires_bankid": false,
... "id06_contact_name": "Mikael Blomkvist",
... "id06_contact_email": "mikael.blomkvist@millennium.se"
... }
AND result has key "id" containing a string, saved as $ID10
AND values "$ID9" and "$ID10" are identical
AND result has key "revision" containing a string, saved as $REV11
AND values "$REV10" and "$REV11" are identical
```

Create a new Tilaajavastuu account contract.

```
WHEN client POSTs /contracts with
... {
... "contract_type": "tilaajavastuu_account",
... "start_date": "2013-03-13",
... "contract_parties": [
...     {
...         "type": "person",
...         "resource_id": "125",
...         "role": "user",
...         "username": "sherlockholmes"
...     },
...     {
...         "type": "org",
...         "resource_id": "secret",
...         "role": "target",
...         "user_role": "detective",
...         "permissions": [
...             {
...                 "permission_name": "taito_user"
...             }
...         ],
...         "global_permissions": [
...             {
```

```
...                 "permission_name": "korttivarasto_user"
...             }
...         ]
...     }
... ],
... "preferred_language": "en"
... }
THEN HTTP status code is 201
AND result matches
... {
... "contract_type": "tilaajavastuu_account",
... "start_date": "2013-03-13",
... "contract_parties": [
...     {
...         "type": "person",
...         "resource_id": "125",
...         "role": "user",
...         "username": "sherlockholmes"
...     },
...     {
...         "type": "org",
...         "resource_id": "secret",
...         "role": "target",
...         "user_role": "detective",
...         "permissions": [
...             {
...                 "permission_name": "taito_user"
...             }
...         ],
...         "global_permissions": [
...             {
...                 "permission_name": "korttivarasto_user"
...             }
...         ]
...     }
... ],
... "preferred_language": "en"
... }
AND result has key "id" containing a string, saved as $ID11
AND HTTP Location header is API_URL/contracts/$ID11
AND result has key "revision" containing a string, saved as $REV12
```

Check that the record is there.

```
WHEN client GETs /contracts/$ID11
THEN HTTP status code is 200
AND result matches
... {
... "contract_type": "tilaajavastuu_account",
```

```
... "start_date": "2013-03-13",
... "contract_parties": [
...     {
...         "type": "person",
...         "resource_id": "125",
...         "role": "user",
...     "username": "sherlockholmes"
...     },
...     {
...         "type": "org",
...         "resource_id": "secret",
...         "role": "target",
...     "user_role": "detective",
...     "permissions": [
...             {
...                 "permission_name": "taito_user"
...             }
...         ],
...     "global_permissions": [
...             {
...                 "permission_name": "korttivarasto_user"
...             }
...         ]
...     }
... ],
... "preferred_language": "en"
... }
AND result has key "id" containing a string, saved as $ID12
AND values "$ID11" and "$ID12" are identical
AND result has key "revision" containing a string, saved as $REV13
AND values "$REV12" and "$REV13" are identical
```

Create a new Taito contract.

```
WHEN client POSTs /contracts with
... {
... "contract_type": "taito_subscription",
... "start_date": "2013-03-13",
... "contract_parties": [
...     {
...         "type": "org",
...         "resource_id": "secret",
...         "role": "customer"
...     }
... ]
... }
THEN HTTP status code is 201
AND result matches
... {
```

```
... "contract_type": "taito_subscription",
... "start_date": "2013-03-13",
... "contract_parties": [
...     {
...         "type": "org",
...         "resource_id": "secret",
...         "role": "customer"
...     }
... ]
... }
AND result has key "id" containing a string, saved as $ID13
AND HTTP Location header is API_URL/contracts/$ID13
AND result has key "revision" containing a string, saved as $REV14
```

Check that the record is there.

```
WHEN client GETs /contracts/$ID13
THEN HTTP status code is 200
AND result matches
... {
... "contract_type": "taito_subscription",
... "start_date": "2013-03-13",
... "contract_parties": [
...     {
...         "type": "org",
...         "resource_id": "secret",
...         "role": "customer"
...     }
... ]
... }
AND result has key "id" containing a string, saved as $ID14
AND values "$ID13" and "$ID14" are identical
AND result has key "revision" containing a string, saved as $REV15
AND values "$REV14" and "$REV15" are identical
```

Create a new Tilaajavastuu subscription contract.

```
WHEN client POSTs /contracts with
... {
... "contract_type": "tilaajavastuu_subscription",
... "start_date": "2016-09-01",
... "contract_parties": [
...     {
...         "type": "org",
...         "resource_id": "secret",
...         "role": "customer"
...     }
... ],
```

```
... "terms_of_service": [
...     {
...         "terms_of_service_version": "1.0",
...         "terms_of_service_language": "en",
...         "acceptance_time": "2016-09-01T11:32:59",
...         "accepter_person_id": "a good employee"
...     }
... ]
... }
THEN HTTP status code is 201
AND result matches
... {
... "contract_type": "tilaajavastuu_subscription",
... "start_date": "2016-09-01",
... "contract_parties": [
...     {
...         "type": "org",
...         "resource_id": "secret",
...         "role": "customer"
...     }
... ],
... "terms_of_service": [
...     {
...         "terms_of_service_version": "1.0",
...         "terms_of_service_language": "en",
...         "acceptance_time": "2016-09-01T11:32:59",
...         "accepter_person_id": "a good employee"
...     }
... ]
... }
AND result has key "id" containing a string, saved as $ID15
AND HTTP Location header is API_URL/contracts/$ID15
AND result has key "revision" containing a string, saved as $REV16
```

Check that the record is there.

```
WHEN client GETs /contracts/$ID15
THEN HTTP status code is 200
AND result matches
... {
... "contract_type": "tilaajavastuu_subscription",
... "start_date": "2016-09-01",
... "contract_parties": [
...     {
...         "type": "org",
...         "resource_id": "secret",
...         "role": "customer"
...     }
... ],
```

```
... "terms_of_service": [
...     {
...         "terms_of_service_version": "1.0",
...         "terms_of_service_language": "en",
...         "acceptance_time": "2016-09-01T11:32:59",
...         "accepter_person_id": "a good employee"
...     }
... ]
... }
AND result has key "id" containing a string, saved as $ID16
AND values "$ID15" and "$ID16" are identical
AND result has key "revision" containing a string, saved as $REV17
AND values "$REV16" and "$REV17" are identical
```

Attempt to send invalid data.

```
WHEN client POSTs /contracts with {"nonexistent_field": "data"}
THEN HTTP status code is 400
```

9.5 /projects

This resource manages various types of projects. A project is an abstraction to allow linking of contracts, supply chains, invoices, and other pieces of data that relates to a single undertaking. Such linking is often required for tax reporting, or similar reasons. An example of a project is a construction site.

Synopsis:

- **GET /projects** — get a list of ids of all projects
- **POST /projects** — add a new project
- **GET /projects/<id>** — get the information about a specific project
- **PUT /projects/<id>** — replace the information for a project
- **DELETE /projects/<id>** — remove a project

Errors:

- There are no errors that are specific to this resource.

Example result for a project

```
EXAMPLE
{
    "type": "project",
    "id": "1234",
    "names": [
        "Construction of the Empire State Building"
    ],
    "project_responsible_org": "123",
```

```
    "project_responsible_person": "456",
    "project_ids": [
        {
            "project_id_type": "finnish_construction_site_key",
            "project_id": "blahblahblah"
        }
    ]
}
```

Fields in the result:

- **`names`** — all the known names of the project
- **`project_responsible_org`** — **optional** id of the organisation who owns, or is otherwise responsible for the project, or is its main contact
- **`project_responsible_person`** — **optional** id of the person who owns, or is otherwise responsible for the project, or is its main contact
- **`project_ids`** — "real world" identifiers for the project: identifiers known to people or other computing systems
  - **`project_id_type`** — must be **`finnish_construction_site_key`** (Finnish "työmaa-avain"), but new types may be added later
  - **`project_id`** — the key itself

9.5.1 Tests

We create a new project, update them, and delete them.

```
SCENARIO manage a project
```

Client has needed access rights for projects resource.

```
GIVEN client has access to scopes
... "uapi_projects_post uapi_projects_get uapi_projects_id_get
...  uapi_projects_id_put uapi_projects_id_delete"
```

Try to create a new project. Test with an invalid field.

```
WHEN client POSTs /projects with {"invalid_field": true}
THEN HTTP status code is 400
```

Create a new project.

```
WHEN client POSTs /projects with
... {
...     "names": [
...         "Construction of the Empire State Building"
...       ],
```

```
...     "project_responsible_org": "123",
...     "project_responsible_person": "456",
...     "project_ids": [
...       {
...           "project_id_type": "finnish_construction_site_key",
...           "project_id": "blahblahblah"
...       }
...     ]
... }
THEN HTTP status code is 201
AND result matches
... {
...     "names": [
...         "Construction of the Empire State Building"
...       ],
...     "project_responsible_org": "123",
...     "project_responsible_person": "456",
...     "project_ids": [
...       {
...           "project_id_type": "finnish_construction_site_key",
...           "project_id": "blahblahblah"
...       }
...     ]
... }
AND result has key "id" containing a string, saved as $ID1
AND HTTP Location header is API_URL/projects/$ID1
AND result has key "revision" containing a string, saved as $REV1
```

Check that the record is there.

```
WHEN client GETs /projects
THEN HTTP status code is 200
AND result has key "resources", a list containing {"id": "$ID1"}

WHEN client GETs /projects/$ID1
THEN HTTP status code is 200
AND result matches
... {
...     "names": [
...         "Construction of the Empire State Building"
...       ],
...     "project_responsible_org": "123",
...     "project_responsible_person": "456",
...     "project_ids": [
...       {
...           "project_id_type": "finnish_construction_site_key",
...           "project_id": "blahblahblah"
...       }
...     ]
```

```
... }
AND result has key "id" containing a string, saved as $ID2
AND values "$ID1" and "$ID2" are identical
AND result has key "revision" containing a string, saved as $REV2
AND values "$REV1" and "$REV2" are identical
```

Update the record.

```
WHEN client PUTs /projects/$ID1 with
... {
...     "revision": "$REV1",
...     "names": ["Construction of Saint Isaac's Cathedral"]
... }
THEN HTTP status code is 200
AND result matches {"names": ["Construction of Saint Isaac's Cathedral"]}
AND result has key "id" containing a string, saved as $ID3
AND values "$ID1" and "$ID3" are identical
AND result has key "revision" containing a string, saved as $REV3
```

Delete the record.

```
WHEN client DELETEs /projects/$ID1
THEN HTTP status code is 200

WHEN client GETs /projects/$ID1
THEN HTTP status code is 404
```

9.6 `/cards`

This resource manages identification cards of various kinds.

Synopsis:

- **`GET /cards`** — get a list of ids of all cards
- **`POST /cards`** — add a new card
- **`GET /cards/<id>`** — get the information about a specific card
- **`PUT /cards/<id>`** — replace the information for a card
- **`GET /cards/<id>/holder_photo`** — get the photo of the card holder
- **`PUT /cards/<id>/holder_photo`** — set the photo of the card holder
- **`GET /cards/<id>/issuer_logo`** — get the logo of the card issuer
- **`PUT /cards/<id>/issuer_logo`** — set the logo of the card issuer
- **`DELETE /cards/<id>`** — remove a card

Errors:

- There are no errors that are specific to this resource.

9.6.1 Description

A **card** resource describes an identification card. There are various types of cards supported, and the **card_type** field will specify the type for a specific resource.

Each type of card has different attributes, but many of of them are identical from type to type. The field descriptions below describe which types of cards they apply to. In addition, each type of card has a short section below describing the card.

There are two sub-resource of cards:

- **holder_photo** is a photograph of the card holder.
- **issuer_logo** is the logotype of the card issuer.

In both cases the image shows what was recorded or printed on the card, not whatever is current.

9.6.2 Glossary

Some terminology for the **card** resource.

- **card issuer** — the company who issued the card, i.e., decided it to be created and paid for that to happen. Usually, but not necessarily, the employer of the card holder.
- **card holder** — the named person to whom the card was issued. Usually, but not necessarily, an employee of the card issuer.
- **card supplier** — the company who made the card, based on an order from the card issuer.

9.6.3 Finnish VALTTI cards

The VALTTI cards are operated by the company Suomen Tilaajavastuu Oy. These cards contain an RFID chip with minimal digital information, only the digital serial number, called **uid** below. On the card is printed a one-dimensional bar code, called **barcode_id** below. In addition, some information about the card holder, and the company who issued the card. That information is not readable from the RFID chip, and a lookup via this API must be made to retrieve it in a machine readable form.

FIXME: Add a URL here to a public description of the cards.

EXAMPLE An example resource for a VALTTI card

```
{
    "type": "card",
    "id": "1234",
    "card_type": "VALTTI",
    "card_ids": [
        {
            "card_id_type": "uid",
            "card_id": "1",
        },
        {
            "card_id_type": "barcode_id",
            "card_id": "1",
```

```
        }
    ],
    "issued_date": "2015-04-01",
    "created_date": "2015-04-01",
    "valid_until_date": null,
    "expiration_date": null,
    "card_status_history": [
        {
            "card_status": "NotConfirmed",
            "modified_by": "123",
            "modified_timestamp": "2015-02-25T12:13:14+0200",
            "modification_reason": "NewCard"
            "modification_description": "M. issued card"
        }
    ],
    "current_status": "NotConfirmed",
    "org": "4444",
    "person": "7777"
}
```

9.6.4 Swedish ID06 cards

The Swedish construction business uses a card standard known as ID06. It is managed by the Swedish Construction Federation, and has some information readable over RFID as well printed on the face of the card.

There are three types of ID06 card: physical card, virtual card, and visitor card. A physical card is issued to a specific person and has their identifying information on them. The virtual card is effectively an access token installed on a mobile device (phone, tablet) and tied to a physical card. A visitor card is not specific to a person, and has no personal information on it. It can be given to someone temporarily visiting a construction site.

FIXME: Add a URL here to the card specification.

EXAMPLE An example resource for an ID06 card

```
{
    "type": "card",
    "id": "1234",
    "card_type": "ID06",
    "card_ids": [
        {
            "card_id_type": "ID06_id",
            "card_id": "12456",
        }
    ],
    "issued_date": "2015-01-01",
    "created_date": "2015-02-25",
    "valid_from_date": "2015-04-01",
    "valid_until_date": "2018-04-01",
    "expiration_date": "2038-01-19",
    "issuer_gov_org_ids": [
        {
```

```
        "country": "SE",
        "org_id_type": "registration_number",
        "gov_org_id": "1111111"
    }
],
"holder_gov_ids": [
    {
        "country": "SE",
        "id_type": "ssn",
        "gov_id": "1212313123"
    }
],
"holder_nationalities": [
    "SE"
],
"holder_names": [
    {
        "full_name": "Lisbeth Salander",
        "sort_key": "Salander, Lisbeth",
        "title": "Miss",
        "given_names": ["Lisbeth"],
        "surnames": ["Salander"]
    }
],
"id06_virtual_enabled": true,
"id06_virtual_history": [
    {
        "full_name": "Mikael Blomqvist",
        "date": "2001-01-01",
        "enabled": true
    }
],
"id06_bankid_required": true,
"id06_virtual_devices": [
    "+44 007 007 007 007"
],
"id06_supplier_employee_full_name": "Mikael Blomqvist",
"card_status_history": [
    {
        "card_status": "active",
        "modified_by": "123",
        "modified_timestamp": "2015-02-25T12:13:14+0200",
        "modification_reason": ""
        "modification_description": "Card created"
    }
],
"current_status": "active",
"org": "4444",
"person": "7777"
```

```
}
```

9.6.5 Resource fields

The following table describes all the possible fields for a **`card`** resource. The "card types" column indicates which card types each field is valid for.

field	card types	description
`card_type`	all	type of card; one of **`VALTTI`**, **`ID06`**, **`ID06_visitor`**
`card_ids`	all	list of unique identifiers for the card; see below for list items
`issued_date`	all	date when issued, i.e., when the issuer ordered it
`created_date`	all	date when the card was created
`valid_from_date`	all	first date when card is valid
`valid_until_date`	all	last date when card is valid
`expiration_date`	all	last date when card can be valid; it should be terminated
`card_status_history`	all	history status of the card; the last entry in the list is the current status; see below for list items
`current_status`	all	current status of the card; the valid values are the same as in **`card_status`** field of the **`card_status_history`** record; the value should be the same as in the last entry of the **`card_status_history`**
`org`	all	resource identifier of the organisation who issued the card
`issuer_name`	**`ID06`**, **`ID06_visitor`**	name of the issuer, as printed on the card
`issuer_gov_org_ids`	**`ID06`**, **`ID06_visitor`**	issuer's government organisation ids, as recorded on the card (not necessarily current info); same structure as in the **`org.gov_org_ids`** list
`person`	all	resource identifier of the person for whom the card was issued
`holder_gov_ids`	**`ID06`**	card holder's government identification numbers as recorded on the card (not necessarily current); same structure as **`person.gov_ids`**
`holder_nationalities`	**`ID06`**	card holder's nationalities (list of country codes)
`holder_names`	all	card holder's names as recorded on the card (not necessarily current); same structure as **`person.names`**
`id06_taxation_country`	**`ID06`**	country code of country to which card holder pays taxes
`id06_virtual_devices`	**`ID06`**	phones or other devices which may get an ID06 virtual card

field	card types	description
id06_virtual_enabled	**ID06**	are virtual cards allowed?
id06_virtual_valid_from	**ID06**	date from which virtual cards are valid
id06_virtual_valid_until	**ID06**	date until which virtual cards are valid
id06_virtual_history	**ID06**	list of virtual card enable/disable changes; see below for details
id06_bankid_required	**ID06**	is card holder required to identify themselves using a Swedish BankID to put a virtual card on a mobile device?
id06_supplier_employee_full_name	**ID06**	full name of employer at the card supplier who created the card

The **card_ids** list has items with the following fields:

- **card_id_type** — type of card identifier
  - for **VALTTI** cards, one of **uid**, **barcode_id**
  - for **ID06** and **ID06_visitor** cards: **ID06_id**
- **card_id** — the actual card id

The **card_status_history** list has items with the following fields:

- **card_status** — status of the card; the allowed values depend on type of card; the last entry in the list is the current status
  - for **VALTTI** cards, the following states are defined:
    * **NotConfirmed** — card has not been confirmed yet, and should not be used
    * **Active** — card is active, and may be used
    * **Closed** — card has been closed, and should not be used
    * **Expired** — card has expired, and should not be used
  - for **ID06** and **ID06_visitor** cards, the following states are defined:
    * **active** — card is active and may be used
    * **inactive** — card is inactive, and may not be used; it may be re-activated later
    * **terminated** — card is inactive, and may not be re-activated again
- **modified_by** — the **/persons** resource id of the person who added the card (new cards) or changed the card status
- **modified_timestamp** — timestamp of when the modification was made (set by the backend, not the API client)
- **modification_reason** — reason for the modification; the allowed values are dependent on the card type
  - for **VALTTI** cards:
    * **NewCard** — a new card was created

        * **CardClosed** — card was closed
        * **CardLost** — card was lost
        * **CardError** — other error
    - for **ID06** cards:
        * This, at least for now, a free-form text field. An ID06 external application should define a list of codes for the allowed values and validate against that list. The API backend doesn't validate this.

The **id06_virtual_history** field for **ID06** cards is a (possibly empty) list of records like this:

- **full_name** – full name of employee at card issuer who changed the status of virtual cards
- **date** — when the status was changed
- **enabled** — are virtual cards enabled after the change? note that the current status must always be in the **id06_virtual_enabled** field

9.6.6 Tests

We create a new card, update them, and delete them.

```
SCENARIO manage a card
```

Client has needed access rights for cards resource.

```
GIVEN client has access to scopes
... "uapi_cards_post uapi_cards_get uapi_cards_id_get uapi_cards_id_put
...  uapi_cards_id_delete uapi_cards_id_holder_photo_get
...  uapi_cards_id_holder_photo_put uapi_cards_id_issuer_logo_get
...  uapi_cards_id_issuer_logo_put"
```

Try to create a new card, but with invalid data. These must all fail.

Only specific fields are allowed. Test with an invalid field.

```
WHEN client POSTs /cards with {"invalid_field": true}
THEN HTTP status code is 400
```

Create a new card.

```
WHEN client POSTs /cards with
... {"card_type": "VALTTI",
...  "card_ids": [
...    {
...        "card_id_type": "uid",
...        "card_id": "1"
...    },
...    {
...        "card_id_type": "barcode_id",
```

```
...         "card_id": "1"
...       }
...     ],
...   "issued_date": "2015-04-01",
...   "created_date": "2015-04-08",
...   "valid_until_date": null,
...   "expiration_date": null,
...   "card_status_history": [
...     {
...         "card_status": "NotConfirmed",
...         "modified_by": "123",
...         "modified_timestamp": "2015-02-25T12:13:14+0200",
...         "modification_reason": "NewCard",
...         "modification_description": "M. didn't want to activate card yet"
...     }
...   ],
...   "current_status": "NotConfirmed",
...   "org": "4444",
...   "person": "7777"
... }
THEN HTTP status code is 201
AND result matches
... {"card_type": "VALTTI",
...   "card_ids": [
...     {
...         "card_id_type": "uid",
...         "card_id": "1"
...     },
...     {
...         "card_id_type": "barcode_id",
...         "card_id": "1"
...     }
...   ],
...   "issued_date": "2015-04-01",
...   "created_date": "2015-04-08",
...   "valid_until_date": null,
...   "expiration_date": null,
...   "card_status_history": [
...     {
...         "card_status": "NotConfirmed",
...         "modified_by": "123",
...         "modified_timestamp": "2015-02-25T12:13:14+0200",
...         "modification_reason": "NewCard",
...         "modification_description": "M. didn't want to activate card yet"
...     }
...   ],
...   "current_status": "NotConfirmed",
...   "org": "4444",
...   "person": "7777"
```

```
... }
AND result has key "id" containing a string, saved as $ID1
AND HTTP Location header is API_URL/cards/$ID1
AND result has key "revision" containing a string, saved as $REV1
```

Check that the record is there.

```
WHEN client GETs /cards
THEN HTTP status code is 200
AND result has key "resources", a list containing {"id": "$ID1"}

WHEN client GETs /cards/$ID1
THEN HTTP status code is 200
AND result matches
... {"card_type": "VALTTI",
...  "card_ids": [
...    {
...        "card_id_type": "uid",
...        "card_id": "1"
...    },
...    {
...        "card_id_type": "barcode_id",
...        "card_id": "1"
...    }
...   ],
...  "issued_date": "2015-04-01",
...  "created_date": "2015-04-08",
...  "valid_until_date": null,
...  "expiration_date": null,
...  "card_status_history": [
...    {
...        "card_status": "NotConfirmed",
...        "modified_by": "123",
...        "modified_timestamp": "2015-02-25T12:13:14+0200",
...        "modification_reason": "NewCard",
...        "modification_description": "M. didn't want to activate card yet"
...    }
...   ],
...  "current_status": "NotConfirmed",
...  "org": "4444",
...  "person": "7777"
... }
AND result has key "id" containing a string, saved as $ID1
```

Update the record.

```
WHEN client PUTs /cards/$ID1 with
... {"revision": "$REV1",
```

```
...  "card_type": "VALTTI",
...  "card_ids": [
...    {
...        "card_id_type": "uid",
...        "card_id": "1"
...    },
...    {
...        "card_id_type": "barcode_id",
...        "card_id": "1"
...    }
...   ],
...  "issued_date": "2015-04-01",
...  "created_date": "2015-04-08",
...  "valid_until_date": "2016-04-01",
...  "expiration_date": "2038-01-19",
...  "card_status_history": [
...    {
...        "card_status": "NotConfirmed",
...        "modified_by": "123",
...        "modified_timestamp": "2015-02-25T12:13:14+0200",
...        "modification_reason": "NewCard",
...        "modification_description": "M. didn't want to activate card yet"
...    }
...   ],
...  "current_status": "NotConfirmed",
...  "org": "4444",
...  "person": "7777"
... }
THEN HTTP status code is 200
AND result matches {"valid_until_date": "2016-04-01"}
AND result matches {"expiration_date": "2038-01-19"}
AND result has key "id" containing a string, saved as $ID2
AND values "$ID1" and "$ID2" are identical
AND result has key "revision" containing a string, saved as $REV2
```

Cannot get an unsent card holder photo.

```
WHEN client GETs /cards/$ID1/holder_photo
THEN HTTP status code is 404
```

Send a card holder photo.

```
GIVEN file photo.png containing "\x89 this is a binary file"

WHEN client PUTs file photo.png with content type image/png
...  and revision $REV2 to /cards/$ID1/holder_photo
THEN HTTP status code is 200
```

```
WHEN client GETs /cards/$ID1/holder_photo
THEN HTTP status code is 200
AND HTTP header Content-Type is image/png
AND HTTP header Revision exists
AND result body matches file photo.png

WHEN client GETs /cards/$ID1
THEN HTTP status code is 200
AND result has key "revision" containing a string, saved as $REV3
```

Cannot send a photo with no Content-Length header set. Without request body (no Content-Length header) server responds with status Length Required 411.

```
WHEN client PUTs no file with content type image/png
...  and revision $REV3 to /cards/$ID1/holder_photo
THEN HTTP status code is 411
```

Cannot get an unsent issuer logo file.

```
WHEN client GETs /cards/$ID1/issuer_logo
THEN HTTP status code is 404
```

Send an issuer logo file.

```
GIVEN file logo.png containing "\x89 this a binary file"

WHEN client PUTs file logo.png with content type image/png
...  and revision $REV3 to /cards/$ID1/issuer_logo
THEN HTTP status code is 200

WHEN client GETs /cards/$ID1/issuer_logo
THEN HTTP status code is 200
AND HTTP header Content-Type is image/png
AND HTTP header Revision exists
AND result body matches file logo.png

WHEN client GETs /cards/$ID1
THEN HTTP status code is 200
AND result has key "revision" containing a string, saved as $REV4
```

Cannot send a file with no Content-Length header set. Without request body (no Content-Length header) server responds with status Length Required 411.

```
WHEN client PUTs no file with content type image/png
...  and revision $REV4 to /cards/$ID1/issuer_logo
THEN HTTP status code is 411
```

Update the main record, this should not affect the subrecord(s).

```
WHEN client GETs /cards/$ID1
THEN result has key "revision" containing a string, saved as $REV5

WHEN client PUTs /cards/$ID1 with
... {"revision": "$REV5",
...  "card_type": "VALTTI",
...  "card_ids": [
...    {
...        "card_id_type": "uid",
...        "card_id": "1"
...    },
...    {
...        "card_id_type": "barcode_id",
...        "card_id": "1"
...    }
...   ],
...  "issued_date": "2015-04-01",
...  "valid_until_date": "2016-04-01",
...  "card_status_history": [
...    {
...        "card_status": "NotConfirmed",
...        "modified_by": "123",
...        "modified_timestamp": "2015-02-25T12:13:14+0200",
...        "modification_reason": "NewCard",
...        "modification_description": "M. didn't want to activate card yet"
...    }
...   ],
...  "org": "4444",
...  "person": "7777"
... }
THEN HTTP status code is 200
AND result matches {"valid_until_date": "2016-04-01"}

WHEN client GETs /cards/$ID1/holder_photo
THEN HTTP status code is 200
AND HTTP header Content-Type is image/png
AND HTTP header Revision exists
AND result body matches file photo.png

WHEN client GETs /cards/$ID1/issuer_logo
THEN HTTP status code is 200
AND HTTP header Content-Type is image/png
AND HTTP header Revision exists
AND result body matches file logo.png
```

Delete the record.

```
WHEN client DELETEs /cards/$ID1
THEN HTTP status code is 200
```

```
WHEN client GETs /cards/$ID1
THEN HTTP status code is 404
```

9.7 /reports

This resource manages all the reports known to the service.

Synopsis:

- **GET /reports** — get a list of ids of all reports
- **POST /reports** — create a new report
- **GET /reports/<id>** — get information about a specific report
- **GET /reports/<id>/pdf** — get the generated report as PDF
- **PUT /reports/<id>/pdf** — save a PDF report
- **DELETE /reports/<id>** — remove a report

Errors:

- There are no errors that are specific to this resource.

The **GET /report/<id>** request gives metadata about the report:

EXAMPLE

```
{
"type": "report",
"id": "123",
"org": "1234",
"report_type": "tilaajavastuu.fi_ci",
"generated_timestamp": "2015-05-31T14:00:00+0200",
"tilaajavastuu_status": "OK"
}
```

Fields in the result:

- **org** — the id of the organisation the report is about
- **report_type** — the type of the report; one of:
  - **tilaajavastuu.fi_ci** — the Finnish “tilaajavastuu” report, company information
- **generated_timestamp** — timestamp when the generation of the report started

If **report_type** is **tilaajavastuu.fi_ci** then following fields are present:

- **tilaajavastuu_status** — status of the report according to Finnish act on the contractor’s obligations and liability, one of:

- **`Seis`** — the company of the report has not fulfilled its contractor's obligations and is not allowed to do business
- **`Selvitä`** — the company of the report has not fulfilled its contractor's obligations and should fulfill them before any agreements are made with them
- **`Tietoja odotetaan`** — the report is missing some information from the company it is related to
- **`Huomioi`** — the company of the report has some kind of exceptional rule that should be verified by other means before making any agreements with them
- **`OK`** — the company of the report has fulfilled all of its contractor's obligations

9.7.1 Tests

We create a new report, search for it, and delete it.

```
SCENARIO manage a report
```

Client has needed access rights for reports resource.

```
GIVEN client has access to scopes
... "uapi_reports_post uapi_reports_get uapi_reports_id_get
...  uapi_reports_id_pdf_put uapi_reports_id_pdf_get
...  uapi_reports_id_delete uapi_reports_id_put"
```

Create a new report.

```
WHEN client POSTs /reports with
... {
... "org": "1234",
... "report_type": "tilaajavastuu.fi_ci",
... "generated_timestamp": "2015-05-31T14:00:00+0200",
... "tilaajavastuu_status": "OK"
... }
THEN HTTP status code is 201
AND result matches
... {
... "org": "1234",
... "report_type": "tilaajavastuu.fi_ci",
... "generated_timestamp": "2015-05-31T14:00:00+0200",
... "tilaajavastuu_status": "OK"
... }
AND result has key "id" containing a string, saved as $ID1
AND HTTP Location header is API_URL/reports/$ID1
AND result has key "revision" containing a string, saved as $REV1
```

Check that the record is there.

```
WHEN client GETs /reports
THEN HTTP status code is 200
AND result has key "resources", a list containing { "id": "$ID1" }

WHEN client GETs /reports/$ID1
THEN HTTP status code is 200
AND result matches
... {
... "org": "1234",
... "report_type": "tilaajavastuu.fi_ci",
... "generated_timestamp": "2015-05-31T14:00:00+0200",
... "tilaajavastuu_status": "OK"
... }
AND result has key "id" containing a string, saved as $ID1
AND result has key "revision" containing a string, saved as $REV1
```

Save PDF report.

```
GIVEN file report_test.pdf containing "This pretends to be a pdf"

WHEN client PUTs file report_test.pdf with content type application/pdf
...  and revision $REV1 to /reports/$ID1/pdf
THEN HTTP status code is 200

WHEN client GETs /reports/$ID1/pdf
THEN HTTP status code is 200
AND HTTP header Content-Type is application/pdf
AND HTTP header Revision exists
AND result body matches file report_test.pdf
```

Update the main record, this should not affect the subrecord(s).

```
WHEN client GETs /reports/$ID1
THEN result has key "revision" containing a string, saved as $REV5

WHEN client PUTs /reports/$ID1 with
... {
... "revision": "$REV5",
... "org": "1234",
... "report_type": "tilaajavastuu.fi_ci",
... "generated_timestamp": "2015-05-31T14:00:00+0200",
... "tilaajavastuu_status": "OK"
... }
THEN HTTP status code is 200

WHEN client GETs /reports/$ID1/pdf
THEN HTTP status code is 200
AND HTTP header Content-Type is application/pdf
AND HTTP header Revision exists
AND result body matches file report_test.pdf
```

Delete the record.

```
WHEN client DELETEs /reports/$ID1
THEN HTTP status code is 200

WHEN client GETs /reports/$ID1
THEN HTTP status code is 404
```

9.8 /events

This resource manages various kinds of timestamped events.

Synopsis:

- **GET /events** — get a list of ids of all events
- **GET /events/<id>** — get the information about a specific event
- **POST /events** — add a new event
- **DELETE /events/<id>** — remove an event

Errors:

- There are no errors that are specific to this resource.

Example result:

```
EXAMPLE
{
    "type": "event",
"id": "123456",
"event_type": "card_event",
"generated_timestamp": "2015-06-24T23:00:00+0200",
"person": "2345",
"org": "1234",
"project": "12121212",
"card": "56565656",
"card_event_type": "out"
}
```

Fields in the result:

- **event_type** — the type of the event, one of:
  - **card_event** — clocking of an identification card with a card reading device
  - **presence_event** — indicator of a person's presence on a site without exact timestamp information
- **generated_timestamp** — timestamp of the event
- **person** — identifier of the person attached to this event, if any

- **org** — identifier of the organisation attached to this event, if any
- **project** — identifier of the project attached to this event, if any

If **event_type** is **card_event** following fields are also present in the result:

- **card** — identifier of the card used in logging this event
- **card_event_type** — type of the clocking event, one of:
  - **in** — check-in to a project
  - **out** — check-out from a project

9.8.1 Tests

We create a new event, and delete it.

```
SCENARIO manage an event
```

Client has needed access rights for events resource.

```
GIVEN client has access to scopes
... "uapi_events_post uapi_events_get uapi_events_id_get
...  uapi_events_id_delete"
```

Try to create a new event, but with invalid data. These must all fail.

Then, attempt with non-existent fields:

```
WHEN client POSTs /events with {"non-existent": []}
THEN HTTP status code is 400
```

Create a new event with all the mandatory fields filled.

```
WHEN client POSTs /events with
... {
...     "event_type": "presence_event",
...     "generated_timestamp": "2015-06-24",
...     "person": "2345",
...     "org": "1234",
...     "project": "12121212",
...     "card": "56565656"
... }
THEN HTTP status code is 201
AND result matches
... {
...     "event_type": "presence_event",
...     "generated_timestamp": "2015-06-24",
...     "person": "2345",
```

```
...     "org": "1234",
...     "project": "12121212",
...     "card": "56565656"
... }
AND result has key "id" containing a string, saved as $ID1
AND HTTP Location header is API_URL/events/$ID1
```

Check that the record is there.

```
WHEN client GETs /events
THEN HTTP status code is 200
AND result has key "resources", a list containing { "id": "$ID1" }

WHEN client GETs /events/$ID1
THEN HTTP status code is 200
AND result matches
... {
...     "event_type": "presence_event",
...     "generated_timestamp": "2015-06-24",
...     "person": "2345",
...     "org": "1234",
...     "project": "12121212",
...     "card": "56565656"
... }
AND result has key "id" containing a string, saved as $ID2
AND values "$ID1" and "$ID2" are identical
```

Delete the record.

```
WHEN client DELETEs /events/$ID1
THEN HTTP status code is 200

WHEN client GETs /events/$ID1
THEN HTTP status code is 404
```

9.9 /competences

This resource stores information about competences: formal or informal certifications that a person is able and allowed to do a specific type of work.

Synopsis:

- **GET /competences** — get a list of ids of all competence certifications.
- **POST /competences** — add a new competence certification.
- **GET /competences/<id>** — get the information about a specific competence certification.
- **PUT /competences/<id>** — update the information about a specific competence certification.
- **DELETE /competences/<id>** — remove a competence certification.

Errors:

- There are no errors that are specific to this resource.

Example result for a competence certification:

EXAMPLE

```
{
    "type": "competence",
    "id": "123",
    "employee_person_id": "007",
    "employer_org_id": "9",
    "valid_from_date": "2015-05-11",
    "valid_until_date": "2015-05-11",
    "competence_type_id": "abc",
    "granted_by_org_id": "654",
    "competence_registry_id": "987",
    "competence_card_id": "876",
    "competence_card_holder_names": [
      {
          "full_name": "James Bond",
          "sort_key": "Bond, James",
          "given_names": ["James"],
          "surnames": ["Bond"]
      }
    ],
"competence_trainer": "Bill Tanner",
    "validation_status": "unknown"
}
```

Fields in the result:

- **`employee_person_id`** — id of person who has been certified as competent
- **`employer_org_id`** — id of organisation this competence is registered under
- **`valid_from_date`** — **optional** date from which the certification is valid
- **`valid_until_date`** — **optional** date until which the certification is valid
- **`competence_type_id`** — id of the type of the competence (see competence types)
- **`granted_by_org_id`** — id of organisation certifying the competence
- **`competence_registry_id`** — id of the organisation that keeps the registry of this competence
- **`competence_card_id`** — **optional** id of the card proving the certification, as recorded on the card.
- **`competence_card_holder_names`** — **optional** names as recorded on the card. Same structure as **`person.names`**.
- **`competence_trainer`** — **optional** person listed as giving the training for this competence
- **`validation_status`** — status of the validation of this competence against the external registry specified in the **`competence_registry_id`** field; the value must be one of the following:
  - **`unknown`** — it is not known if the competence is valid
  - **`invalid`** — the competence is not valid
  - **`valid`** — the competence is valid

9.9.1 Tests

We create a new competence, update them, and delete them.

```
SCENARIO manage a competence
```

Client has needed access rights for competence resource.

```
GIVEN client has access to scopes
... "uapi_competences_post uapi_competences_get uapi_competences_id_get
...  uapi_competences_id_put uapi_competences_id_delete"
```

Try to create a new competence. Test with an invalid field.

```
WHEN client POSTs /competences with {"invalid_field": true}
THEN HTTP status code is 400
```

Create a new competence.

```
WHEN client POSTs /competences with
... {
...     "employee_person_id": "007",
...     "employer_org_id": "9",
...     "valid_from_date": "2015-05-11",
...     "valid_until_date": "2015-05-11",
...     "competence_type_id": "abc",
...     "granted_by_org_id": "123",
...     "competence_registry_id": "456",
...     "competence_card_id": "876",
...     "competence_card_holder_names": [
...       {
...           "full_name": "James Bond",
...           "sort_key": "Bond, James",
...           "given_names": ["James"],
...           "surnames": ["Bond"]
...       }
...     ],
...     "competence_trainer": "Bill Tanner",
...     "validation_status": "unknown"
... }
THEN HTTP status code is 201
AND result matches
... {
...     "type": "competence",
...     "employee_person_id": "007",
...     "employer_org_id": "9",
...     "valid_from_date": "2015-05-11",
...     "valid_until_date": "2015-05-11",
```

```
...     "competence_type_id": "abc",
...     "granted_by_org_id": "123",
...     "competence_registry_id": "456",
...     "competence_card_id": "876",
...     "competence_card_holder_names": [
...       {
...           "full_name": "James Bond",
...           "sort_key": "Bond, James",
...           "given_names": ["James"],
...           "surnames": ["Bond"]
...       }
...     ],
...     "competence_trainer": "Bill Tanner",
...     "validation_status": "unknown"
... }
AND result has key "id" containing a string, saved as $ID1
AND HTTP Location header is API_URL/competences/$ID1
AND result has key "revision" containing a string, saved as $REV1
```

Check that the record is there.

```
WHEN client GETs /competences
THEN HTTP status code is 200
AND result has key "resources", a list containing {"id": "$ID1"}

WHEN client GETs /competences/$ID1
THEN HTTP status code is 200
AND result matches
... {
...     "type": "competence",
...     "employee_person_id": "007",
...     "employer_org_id": "9",
...     "valid_from_date": "2015-05-11",
...     "valid_until_date": "2015-05-11",
...     "competence_type_id": "abc",
...     "granted_by_org_id": "123",
...     "competence_registry_id": "456",
...     "competence_card_id": "876",
...     "competence_card_holder_names": [
...       {
...           "full_name": "James Bond",
...           "sort_key": "Bond, James",
...           "given_names": ["James"],
...           "surnames": ["Bond"]
...       }
...     ],
...     "competence_trainer": "Bill Tanner",
...     "validation_status": "unknown"
... }
```

```
AND result has key "id" containing a string, saved as $ID2
AND values "$ID1" and "$ID2" are identical
AND result has key "revision" containing a string, saved as $REV2
AND values "$REV1" and "$REV2" are identical
```

Update the record.

```
WHEN client PUTs /competences/$ID1 with
... {
...     "revision": "$REV1",
...     "valid_from_date": "2015-05-09"
... }
THEN HTTP status code is 200
AND result matches {"valid_from_date": "2015-05-09"}
AND result has key "id" containing a string, saved as $ID3
AND values "$ID1" and "$ID3" are identical
AND result has key "revision" containing a string, saved as $REV3
```

Delete the record.

```
WHEN client DELETEs /competences/$ID1
THEN HTTP status code is 200

WHEN client GETs /competences/$ID1
THEN HTTP status code is 404
```

9.10 /competence_types

This resource stores information about different types of competences. See competences.

Synopsis:

- **GET /competence_types** — get a list of ids of all competence types.
- **POST /competence_types** — add a new competence type.
- **GET /competence_types/<id>** — get the information about a specific competence type.
- **PUT /competence_types/<id>** — update the information about a specific competence type.
- **DELETE /competence_types/<id>** — remove a competence certification.
- **GET /competence_types/<id>/card_front** — get the photo of the example competence card front
- **PUT /competence_types/<id>/card_front** — set the photo of the example competence card front
- **GET /competence_types/<id>/card_back** — get the logo of the example competence card back
- **PUT /competence_types/<id>/card_back** — set the logo of the example competence card back
- **GET /competence_types/<id>/registry_logo** — get the logo of the competence registry organisation

- **PUT /competence_types/<id>/registry_logo** — set the logo of the competence registry organisation

There are three sub-resource of competence_types:

- **card_front** is an example photo of the competence card front.
- **card_back** is an example photo of the competence card back.
- **registry_logo** is the logotype of the competence registry organisation.

Errors:

- There are no errors that are specific to this resource.

Example result for a competence type:

EXAMPLE


```
{
    "type": "competence_type",
    "id": "123",
    "competence_type_id": "NAP",
    "names": [
        {
            "locale": "fi_FI",
            "name": "torkkuminen"
        },
        {
            "locale": "en_GB",
            "name": "napping"
        }
    ],
"customer_org_ids": [
    "1a2b3c",
    "4d5e6f"
],
    "is_verified": true,
"attribute_validation": [
    {
        "attribute": "valid_from_date",
    "required": true,
    "regex": "2016-*"
    }
],
    "descriptions": [
        {
            "locale": "fi_FI",
            "description": "Saa torkkua tyoaikana"
        },
        {
```

```
            "locale": "en_GB",
            "description": "Allowed to nap during work hours"
        }
    ]
}
```

Fields in the result:

- **`competence_type_id`** — a user defined identification for this competence type
- **`names`** — a list of names for the competence type, perhaps in different locales; there can be multiple names (aliases) in one locale; the fields for one entry are:
  - **`locale`** — the locale code
  - **`name`** — a name for the competence in the given locale
- **`customer_org_ids`** — a list of organisation ids this competence type is intended for
- **`is_verified`** — indicates if the competences of this type need to be verified
- **`attribute_validation`** — a list of validation rules for competences of this type
  - **`attribute`** — attribute of the competence
  - **`required`** — a boolean value to indicate if this attribute is required
  - **`regex`** — a regex to describe a valid form for this attribute
- **`descriptions`** — a list of descriptions for the competence type; the description can be given in any locale, but only once per locale
  - **`locale`** — the locale code
  - **`description`** — the description for the competence in the given locale

9.10.1 Tests

We create a new competence type, update them, and delete them.

```
SCENARIO manage a competence type
```

Client has needed access rights for competence type resource.

```
GIVEN client has access to scopes
... "uapi_competence_types_post uapi_competence_types_get
...  uapi_competence_types_id_get uapi_competence_types_id_put
...  uapi_competence_types_id_delete
...  uapi_competence_types_id_card_front_get
...  uapi_competence_types_id_card_front_put
...  uapi_competence_types_id_card_back_get
...  uapi_competence_types_id_card_back_put
...  uapi_competence_types_id_registry_logo_get
...  uapi_competence_types_id_registry_logo_put"
```

Try to create a new competence type. Test with an invalid field.

```
WHEN client POSTs /competence_types with {"invalid_field": true}
THEN HTTP status code is 400
```

Create a new competence type.

```
WHEN client POSTs /competence_types with
... {
...     "competence_type_id": "NAP",
...     "names": [
...         {
...             "locale": "fi_FI",
...             "name": "torkkuminen"
...         },
...         {
...             "locale": "en_GB",
...             "name": "napping"
...         }
...     ],
...     "customer_org_ids": [
...             "mattress_testing_org"
...     ],
...     "is_verified": true,
...     "attribute_validation": [
...         {
...             "attribute": "valid_from_date",
...             "required": true,
...             "regex": "2016-*"
...         }
...     ],
...     "descriptions": [
...         {
...             "locale": "fi_FI",
...             "description": "Saa torkkua tyoaikana"
...         },
...         {
...             "locale": "en_GB",
...             "description": "Allowed to nap during work hours"
...         }
...     ]
... }
THEN HTTP status code is 201
AND result matches
... {
...     "type": "competence_type",
...     "competence_type_id": "NAP",
...     "names": [
...         {
...             "locale": "fi_FI",
...             "name": "torkkuminen"
```

```
...         },
...         {
...             "locale": "en_GB",
...             "name": "napping"
...         }
...     ],
...     "customer_org_ids": [
...             "mattress_testing_org"
...     ],
...     "is_verified": true,
...     "attribute_validation": [
...         {
...             "attribute": "valid_from_date",
...             "required": true,
...             "regex": "2016-*"
...         }
...     ],
...     "descriptions": [
...         {
...             "locale": "fi_FI",
...             "description": "Saa torkkua tyoaikana"
...         },
...         {
...             "locale": "en_GB",
...             "description": "Allowed to nap during work hours"
...         }
...     ]
... }
AND result has key "id" containing a string, saved as $ID1
AND HTTP Location header is API_URL/competence_types/$ID1
AND result has key "revision" containing a string, saved as $REV1
```

Check that the record is there.

```
WHEN client GETs /competence_types
THEN HTTP status code is 200
AND result has key "resources", a list containing {"id": "$ID1"}

WHEN client GETs /competence_types/$ID1
THEN HTTP status code is 200
AND result matches
... {
...     "type": "competence_type",
...     "competence_type_id": "NAP",
...     "names": [
...         {
...             "locale": "fi_FI",
...             "name": "torkkuminen"
...         },
```

```
...         {
...             "locale": "en_GB",
...             "name": "napping"
...         }
...     ],
...     "customer_org_ids": [
...             "mattress_testing_org"
...     ],
...     "is_verified": true,
...     "attribute_validation": [
...         {
...             "attribute": "valid_from_date",
...             "required": true,
...             "regex": "2016-*"
...         }
...     ],
...     "descriptions": [
...         {
...             "locale": "fi_FI",
...             "description": "Saa torkkua tyoaikana"
...         },
...         {
...             "locale": "en_GB",
...             "description": "Allowed to nap during work hours"
...         }
...     ]
... }
AND result has key "id" containing a string, saved as $ID2
AND values "$ID1" and "$ID2" are identical
AND result has key "revision" containing a string, saved as $REV2
AND values "$REV1" and "$REV2" are identical
```

Update the record.

```
WHEN client PUTs /competence_types/$ID1 with
... {
...     "revision": "$REV1",
...     "competence_type_id": "NAP",
...     "names": [
...         {
...             "locale": "fi_FI",
...             "name": "torkkuminen"
...         }
...     ],
...     "descriptions": [
...         {
...             "locale": "fi_FI",
...             "description": "Saa torkkua tyoaikana"
...         }
```

```
...     ]
... }
THEN HTTP status code is 200
AND result matches
... {
...     "competence_type_id": "NAP",
...     "names": [
...         {
...             "locale": "fi_FI",
...             "name": "torkkuminen"
...         }
...     ],
...     "descriptions": [
...         {
...             "locale": "fi_FI",
...             "description": "Saa torkkua tyoaikana"
...         }
...     ]
... }
AND result has key "id" containing a string, saved as $ID3
AND values "$ID1" and "$ID3" are identical
AND result has key "revision" containing a string, saved as $REV3
```

Cannot get an unsent competence card front image.

```
WHEN client GETs /competence_types/$ID1/card_front
THEN HTTP status code is 404
```

Send competence card front image.

```
GIVEN file test.png containing "This pretends to be a png"

WHEN client PUTs file test.png with content type image/png
...  and revision $REV3 to /competence_types/$ID1/card_front
THEN HTTP status code is 200

WHEN client GETs /competence_types/$ID1/card_front
THEN HTTP status code is 200
AND HTTP header Content-Type is image/png
AND HTTP header Revision exists
AND result body matches file test.png

WHEN client GETs /competence_types/$ID1
THEN HTTP status code is 200
AND result has key "revision" containing a string, saved as $REV4
```

Cannot send a photo with no Content-Length header set. Without request body (no Content-Length header) server responds with status Length Required 411.

```
WHEN client PUTs no file with content type image/png
...  and revision $REV4 to /competence_types/$ID1/card_front
THEN HTTP status code is 411
```

Cannot get an unsent competence card back image.

```
WHEN client GETs /competence_types/$ID1/card_back
THEN HTTP status code is 404
```

Send competence card front image.

```
GIVEN file test.png containing "This pretends to be a png"

WHEN client PUTs file test.png with content type image/png
...  and revision $REV4 to /competence_types/$ID1/card_back
THEN HTTP status code is 200

WHEN client GETs /competence_types/$ID1/card_back
THEN HTTP status code is 200
AND HTTP header Content-Type is image/png
AND HTTP header Revision exists
AND result body matches file test.png

WHEN client GETs /competence_types/$ID1
THEN HTTP status code is 200
AND result has key "revision" containing a string, saved as $REV5
```

Cannot send a photo with no Content-Length header set. Without request body (no Content-Length header) server responds with status Length Required 411.

```
WHEN client PUTs no file with content type image/png
...  and revision $REV5 to /competence_types/$ID1/card_back
THEN HTTP status code is 411
```

Cannot get an unsent competence registry logo.

```
WHEN client GETs /competence_types/$ID1/registry_logo
THEN HTTP status code is 404
```

Send competence card front image.

```
GIVEN file test.png containing "This pretends to be a png"

WHEN client PUTs file test.png with content type image/png
...  and revision $REV5 to /competence_types/$ID1/registry_logo
THEN HTTP status code is 200
```

```
WHEN client GETs /competence_types/$ID1/registry_logo
THEN HTTP status code is 200
AND HTTP header Content-Type is image/png
AND HTTP header Revision exists
AND result body matches file test.png

WHEN client GETs /competence_types/$ID1
THEN HTTP status code is 200
AND result has key "revision" containing a string, saved as $REV6
```

Cannot send a photo with no Content-Length header set. Without request body (no Content-Length header) server responds with status Length Required 411.

```
WHEN client PUTs no file with content type image/png
...  and revision $REV6 to /competence_types/$ID1/registry_logo
THEN HTTP status code is 411
```

Delete the record.

```
WHEN client DELETEs /competence_types/$ID1
THEN HTTP status code is 200

WHEN client GETs /competence_types/$ID1
THEN HTTP status code is 404
```

9.11 /bolagsfakta_suppliers

This resource stores information about project supplier relationships for Bolagsfakta project.

Synopsis:

- **GET /bolagsfakta_suppliers** — get a list of ids of suppliers
- **POST /bolagsfakta_suppliers** — create a new supplier relationship
- **GET /bolagsfakta_suppliers/<id>** — get the information about a specific supplier relationship
- **PUT /bolagsfakta_suppliers/<id>** — update supplier relationship
- **DELETE /bolagsfakta_suppliers/<id>** — update supplier relationship
- **GET /bolagsfakta_suppliers/search/exact/materialized_path/<org_id>** — find all supplier that are descendands of org_id organisation

Errors:

- There are no errors that are specific to this resource.

Example result for a supplier:

EXAMPLE

```
{
    "type": "bolagsfakta_supplier",
    "id": "12345",
    "bolagsfakta_status": "ok",
    "contract_start_date": "2016-06-01",
    "contract_end_date": "2016-12-31",
    "materialized_path": [
        "23456",
        "34567",
        "45678"
    ],
    "parent_org_id": "34567",
    "parent_supplier_id": "34567",
    "project_resource_id": "123456",
    "supplier_org_id": "45678",
    "supplier_type": "linked"
}
```

Fields in the result:

- **bolagsfakta_status** — cached status of the company in question
- **contract_start_date** — start of the contract for this supplier relationship
- **contract_end_date** — end of the contract for this supplier relationship
- **materialized_path** — a path of organisation within the tree.
- **parent_org_id** — suppliers direct parent organisation in the tree
- **parent_supplier_id** — ID of parent supplier organisation. Used when searching for direct descendants.
- **supplier_type** — supplier type, can be one of:
  - **linked** — Supplier is fully linked to other suppliers in the tree
  - **unlinked** — Supplier links to other suppliers not known

9.11.1 Tests

Create a new bolagsfakta_supplier entry, update it and delete it

SCENARIO manage bolagsfakta supplier

CLient has access rights to bolagsfakta_suppliers resource

```
GIVEN client has access to scopes
... "uapi_bolagsfakta_suppliers_get uapi_bolagsfakta_suppliers_id_delete
...  uapi_bolagsfakta_suppliers_id_get uapi_bolagsfakta_suppliers_id_put
...  uapi_bolagsfakta_suppliers_post uapi_bolagsfakta_suppliers_search_id_ge
```

Attempt to create a new bolagsfakta supplier with invlaid fields.

```
WHEN client POSTs /bolagsfakta_suppliers with {"some_field": 100}
THEN HTTP status code is 400
```

Create a new bolagsfakta supplier.

```
WHEN client POSTs /bolagsfakta_suppliers with
... {
...     "type": "bolagsfakta_supplier",
...     "bolagsfakta_status": "ok",
...     "contract_start_date": "2016-06-01",
...     "contract_end_date": "2016-12-31",
...     "materialized_path": [
...         "23456",
...         "34567",
...         "45678"
...     ],
...     "parent_org_id": "34567",
...     "parent_supplier_id": "34567",
...     "project_resource_id": "123456",
...     "supplier_org_id": "45678",
...     "supplier_type": "linked"
... }
THEN HTTP status code is 201
AND result matches
... {
...     "type": "bolagsfakta_supplier",
...     "bolagsfakta_status": "ok",
...     "contract_start_date": "2016-06-01",
...     "contract_end_date": "2016-12-31",
...     "materialized_path": [
...         "23456",
...         "34567",
...         "45678"
...     ],
...     "parent_org_id": "34567",
...     "parent_supplier_id": "34567",
...     "project_resource_id": "123456",
...     "supplier_org_id": "45678",
...     "supplier_type": "linked"
... }
AND result has key "id" containing a string, saved as $ID1
AND result has key "revision" containing a string, saved as $REV1
```

Check that the bolagsfakta supplier exists.

```
WHEN client GETs /bolagsfakta_suppliers
THEN HTTP status code is 200
AND result has key "resources", a list containing {"id": "$ID1"}
```

```
WHEN client GETs /bolagsfakta_suppliers/$ID1
THEN HTTP status code is 200
AND result matches
... {
...     "type": "bolagsfakta_supplier",
...     "bolagsfakta_status": "ok",
...     "contract_start_date": "2016-06-01",
...     "contract_end_date": "2016-12-31",
...     "materialized_path": [
...         "23456",
...         "34567",
...         "45678"
...     ],
...     "parent_org_id": "34567",
...     "parent_supplier_id": "34567",
...     "project_resource_id": "123456",
...     "supplier_org_id": "45678",
...     "supplier_type": "linked"
... }
AND result has key "id" containing a string, saved as $ID2
AND values "$ID1" and "$ID2" are identical
AND result has key "revision" containing a string, saved as $REV2
AND values "$REV1" and "$REV2" are identical
```

Update the supplier record.

```
WHEN client PUTs /bolagsfakta_suppliers/$ID1 with
... {
...     "revision": "$REV1",
...     "type": "bolagsfakta_supplier",
...     "bolagsfakta_status": "stop",
...     "contract_start_date": "2016-06-01",
...     "contract_end_date": "2016-12-31",
...     "materialized_path": [
...         "23456",
...         "34567",
...         "45678"
...     ],
...     "parent_org_id": "34567",
...     "parent_supplier_id": "34567",
...     "project_resource_id": "123456",
...     "supplier_org_id": "45678",
...     "supplier_type": "linked"
... }
THEN HTTP status code is 200
AND result matches
... {
...     "type": "bolagsfakta_supplier",
```

```
...     "bolagsfakta_status": "stop",
...     "contract_start_date": "2016-06-01",
...     "contract_end_date": "2016-12-31",
...     "materialized_path": [
...         "23456",
...         "34567",
...         "45678"
...     ],
...     "parent_org_id": "34567",
...     "parent_supplier_id": "34567",
...     "project_resource_id": "123456",
...     "supplier_org_id": "45678",
...     "supplier_type": "linked"
... }
AND result has key "id" containing a string, saved as $ID3
AND values "$ID1" and "$ID3" are identical
AND result has key "revision" containing a string, saved as $REV3
```

Search for descendant suppliers via materialized_path.

```
WHEN client GETs /bolagsfakta_suppliers/search/exact/materialized_path/34567
THEN HTTP status code is 200
AND result has key "resources", a list containing {"id": "$ID1"}
```

Finally delete the supplier.

```
WHEN client DELETEs /bolagsfakta_suppliers/$ID1
THEN HTTP status code is 200
WHEN client GETs /bolagsfakta_suppliers/$ID1
THEN HTTP status code is 404
```

9.12 /data_cache

This resource acts as a temporary cache for applications. It is a fairly generic key-value store.

Synopsis:

- **GET /data_cache** — get a list of all the stored cache entries
- **POST /data_cache** — add a new cache entry
- **GET /data_cache/<id>** — get the specific cache entry
- **DELETE /data_cache/<id>** — remove a cache entry

Errors:

- There are no errors that are specific to this resource.

Example of a cache entry:

```
EXAMPLE
{
    "type": "data_cache",
    "id": "54353",
    "cache_name": "My shopping list",
    "key": "price",
    "value": "$5.98"
    "expires_at": "2016-12-24T08:00:00+0200"
}
```

Fields in the result:

- **`cache_name`** — name given to the cache this entry is part of
- **`key`** — name of this cache entry
- **`value`** — value of this cache entry
- **`expires_at`** — timestamp when this cache entry is last valid

9.12.1 Tests

We create a new data cache entry, search for it and delete it.

```
SCENARIO manage a data cache entry
```

Client has needed access rights for data cache.

```
GIVEN client has access to scopes
... "uapi_data_cache_post uapi_data_cache_get
...  uapi_data_cache_search_id_get
...  uapi_data_cache_id_get
...  uapi_data_cache_id_delete"
```

Create a data cache entry.

```
WHEN client POSTs /data_cache with
... {
...     "cache_name": "My list",
...     "key": "Name",
...     "value": "Sherlock",
...     "expires_at": "2020-01-03T07:56:30+0200"
... }
THEN HTTP status code is 201
AND result matches
... {
...     "cache_name": "My list",
...     "key": "Name",
...     "value": "Sherlock",
...     "expires_at": "2020-01-03T07:56:30+0200"
```

```
... }
AND result has key "id" containing a string, saved as $ID1
AND HTTP Location header is API_URL/data_cache/$ID1
AND result has key "revision" containing a string, saved as $REV1
```

Check that the record is there.

```
WHEN client GETs /data_cache/$ID1
THEN HTTP status code is 200
AND result matches
... {
...     "cache_name": "My list",
...     "key": "Name",
...     "value": "Sherlock",
...     "expires_at": "2020-01-03T07:56:30+0200"
... }
AND result has key "id" containing a string, saved as $ID2
AND values "$ID1" and "$ID2" are identical
AND result has key "revision" containing a string, saved as $REV2
AND values "$REV1" and "$REV2" are identical
```

Make a search by key for the created entry.

```
WHEN client GETs /data_cache/search/exact/key/Name
THEN HTTP status code is 200
AND result has key "resources", a list containing {"id": "$ID1"}
```

We delete the created data cache entry.

```
WHEN client DELETEs /data_cache/$ID1
THEN HTTP status code is 200

WHEN client GETs /data_cache/$ID1
THEN HTTP status code is 404
```

9.13 `/files`

This resource manages various kind of binary files.

Synopsis:

- **`GET /files`** — get a list of ids of all files
- **`GET /files/<id>`** — get the information about a specific file
- **`POST /files`** — add a new file
- **`PUT /files/<id>`** — replace the information about a specific file
- **`GET /files/<id>/file`** — get the contents of the file
- **`PUT /files/<id>/file`** — set the contents of the file

- **DELETE /files/<id>** — remove a file

Errors:

- There are no errors that are specific to this resource.

Example result:

```
EXAMPLE information about a file
{
    "type": "file",
    "id": "34rf",
    "filename": "my_secret_diary.odt"
}
```

Fields in the result:

- **filename** — name of the stored file

9.13.1 Tests

We create a new file, update them, and delete them.

```
SCENARIO manage a file
```

Client has needed access rights for files resource.

```
GIVEN client has access to scopes
... "uapi_files_post uapi_files_get uapi_files_id_get
... uapi_files_id_put uapi_files_id_delete
... uapi_files_id_file_put uapi_files_id_file_get"
```

Create a new file.

```
WHEN client POSTs /files with
... {
...     "filename": "my_very_own_file.txt"
... }
THEN HTTP status code is 201
AND result matches
... {
...     "filename": "my_very_own_file.txt"
... }
AND result has key "id" containing a string, saved as $ID1
AND HTTP Location header is API_URL/files/$ID1
AND result has key "revision" containing a string, saved as $REV1
```

Check that the record is there.

```
WHEN client GETs /files/$ID1
THEN HTTP status code is 200
AND result matches
... {
...     "filename": "my_very_own_file.txt"
... }
AND result has key "id" containing a string, saved as $ID2
AND values "$ID1" and "$ID2" are identical
AND result has key "revision" containing a string, saved as $REV2
AND values "$REV1" and "$REV2" are identical
```

Send the contents of the created file.

```
GIVEN file my_very_own_file.txt containing "This is my text."

WHEN client PUTs file my_very_own_file.txt with content type
...  text/plain and revision $REV2 to /files/$ID1/file
THEN HTTP status code is 200

WHEN client GETs /files/$ID1/file
THEN HTTP status code is 200
AND HTTP header Content-Type is text/plain
AND HTTP header Revision exists
AND result body matches file my_very_own_file.txt
```

Delete the file.

```
WHEN client DELETEs /files/$ID1
THEN HTTP status code is 200

WHEN client GETs /files/$ID1
THEN HTTP status code is 404
```

9.14 `/jobs`

This resource manages job queues from external sources.

Synopsis:

- **`GET /jobs`** — get a list of ids of all jobs
- **`GET /jobs/<id>`** — get the information about a specific job
- **`POST /jobs`** — add a new job
- **`PUT /jobs/<id>`** — updates a specific job
- **`DELETE /jobs/<id>`** — removes a job

Errors:

- There are no errors that are specific to this resource.

Example result:

```
EXAMPLE information about a job
{
    "type": "job",
    "id": "j0b8",
    "job_type": "endless_wheel",
    "person_id": "4trytr6",
    "org_id": "123",
    "submitted_at": "1323-08-12T16:04:23+0200",
    "started_at": "1323-08-12T16:05:00+0200",
    "done_at": "2016-11-02T13:40:38+0200",
    "status": "done",
    "reserved_until": "2016-11-01T00:00:00+0200",
    "parameters": [
        {
            "key": "is_this_fun?",
            "value": "yes"
        }
    ]
}
```

Fields in the result:

- **job_type** — name of the job type, can be an application or a source or such
- **person_id** — resource id of the originator or sender of the job
- **org_id** — resource id of the organisation that this job belongs to
- **submitted_at** — timestamp when this job was submitted
- **started_at** — timestamp when the processing of this job started
- **done_at** — timestamp when the processing of this job completed
- **status** — current status of this job
- **reserved_until** — timestamp of the time this job is reserved for its primary handler only
- **parameters** — a list of parameters to be processed by this job, containing pairs of the following:
- **key** — name of this parameter
- **value** — value of this parameter

9.14.1 Tests

We create a new job, update them, and delete them.

```
SCENARIO manage a job
```

Client has needed access rights for files resource.

```
GIVEN client has access to scopes
... "uapi_jobs_post uapi_jobs_get uapi_jobs_id_get
... uapi_jobs_id_put uapi_jobs_id_delete"
```

Create a new job.

```
WHEN client POSTs /jobs with
... {
...     "job_type": "gotta_do_this",
...     "person_id": "34567",
...     "org_id": "098123",
...     "submitted_at": "2016-09-01T02:36:45+0300",
...     "started_at": "2016-09-01T03:00:00+0300",
...     "done_at": "2016-11-02T14:00:00+0200",
...     "status": "almost_done",
...     "reserved_until": "2016-11-02T13:00:00+0200",
...     "parameters": [
...         {
...             "key": "my_key",
...             "value": "54676yq"
...         },
...         {
...             "key": "your_key",
...             "value": "347564t"
...         }
...     ]
... }
THEN HTTP status code is 201
AND result matches
... {
...     "job_type": "gotta_do_this",
...     "person_id": "34567",
...     "org_id": "098123",
...     "submitted_at": "2016-09-01T02:36:45+0300",
...     "started_at": "2016-09-01T03:00:00+0300",
...     "done_at": "2016-11-02T14:00:00+0200",
...     "status": "almost_done",
...     "reserved_until": "2016-11-02T13:00:00+0200",
...     "parameters": [
...         {
...             "key": "my_key",
...             "value": "54676yq"
...         },
...         {
...             "key": "your_key",
...             "value": "347564t"
...         }
...     ]
... }
```

```
AND result has key "id" containing a string, saved as $ID1
AND HTTP Location header is API_URL/jobs/$ID1
AND result has key "revision" containing a string, saved as $REV1
```

Check that the record is there.

```
WHEN client GETs /jobs/$ID1
THEN HTTP status code is 200
AND result matches
... {
...     "job_type": "gotta_do_this",
...     "person_id": "34567",
...     "org_id": "098123",
...     "submitted_at": "2016-09-01T02:36:45+0300",
...     "started_at": "2016-09-01T03:00:00+0300",
...     "done_at": "2016-11-02T14:00:00+0200",
...     "status": "almost_done",
...     "reserved_until": "2016-11-02T13:00:00+0200",
...     "parameters": [
...         {
...             "key": "my_key",
...             "value": "54676yq"
...         },
...         {
...             "key": "your_key",
...             "value": "347564t"
...         }
...     ]
... }
AND result has key "id" containing a string, saved as $ID2
AND values "$ID1" and "$ID2" are identical
AND result has key "revision" containing a string, saved as $REV2
AND values "$REV1" and "$REV2" are identical
```

Delete the job.

```
WHEN client DELETEs /jobs/$ID1
THEN HTTP status code is 200

WHEN client GETs /jobs/$ID1
THEN HTTP status code is 404
```

9.15 /report_accesses

This resource acts as a log of report access events. This is done because earlier allowed access to a certain report allows later accesses to the same report even after the access privileges have otherwise been revoked. This resource takes into account that some of the requests will come through partner APIs that do not pass enough information

for Qvarn maintainer to actually identify the actual accessor. However, the partner (identified by **client_id**) must be able to identify the accessor using the stored **customer_id**.

Synopsis:

- **GET /report_accesses** — get a list of ids of all the report accesses
- **GET /report_accesses/<id>** — get the information about a specific report access
- **POST /report_accesses** — add a new report access

Notes:

As this resource is basically a log that is accessed through the API updating and removing items should never be allowed.

Example result:

```
EXAMPLE report access information
{
    "type": "report_access",
    "id": "378iu",
    "client_id": "qwerty-23456-zxcvb",
    "customer_id": "that_strange_customer",
    "org_id": "345",
    "report_id": "234eg4",
    "arkisto_id": "201104231235345",
    "access_time": "2011-04-23T12:35:45+0200"
}
```

Fields in the result:

- **client_id** — id of the client making the access request
- **customer_id** — id of the accessing customer in the client's maintainer's system
- **org_id** — id of the organisation in the report
- **report_id** — id of the report
- **arkisto_id** — id of the report in Suomen Tilaajavastuu's pre-Qvarn Arkisto system
- **access_time** — timestamp when this report access was done

9.15.1 Tests

We create a new report access item, update, and remove it.

```
SCENARIO manage report access log
```

Client has needed access rights for report access resource.

```
GIVEN client has access to scopes
... "uapi_report_accesses_post uapi_report_accesses_get
... uapi_report_accesses_id_get uapi_report_accesses_id_delete"
```

Create a new report access event.

```
WHEN client POSTs /report_accesses with
... {
...     "client_id": "2345678",
...     "customer_id": "good_customer",
...     "org_id": "qwerty1",
...     "report_id": "6868686868",
...     "arkisto_id": "20142345246678",
...     "access_time": "2016-05-03T12:45:38+0300"
... }
THEN HTTP status code is 201
AND result matches
... {
...     "client_id": "2345678",
...     "customer_id": "good_customer",
...     "org_id": "qwerty1",
...     "report_id": "6868686868",
...     "arkisto_id": "20142345246678",
...     "access_time": "2016-05-03T12:45:38+0300"
... }
AND result has key "id" containing a string, saved as $ID1
AND HTTP Location header is API_URL/report_accesses/$ID1
AND result has key "revision" containing a string, saved as $REV1
```

Check that the record is there.

```
WHEN client GETs /report_accesses/$ID1
THEN HTTP status code is 200
AND result matches
... {
...     "client_id": "2345678",
...     "customer_id": "good_customer",
...     "org_id": "qwerty1",
...     "report_id": "6868686868",
...     "arkisto_id": "20142345246678",
...     "access_time": "2016-05-03T12:45:38+0300"
... }
AND result has key "id" containing a string, saved as $ID2
AND values "$ID1" and "$ID2" are identical
AND result has key "revision" containing a string, saved as $REV2
AND values "$REV1" and "$REV2" are identical
```

Delete the report access event.

```
WHEN client DELETEs /report_accesses/$ID1
THEN HTTP status code is 200

WHEN client GETs /report_accesses/$ID1
THEN HTTP status code is 404
```

Chapter 10

APPENDIX: Plans for the future

This chapter contains plans for the future that will probably be implemented, hopefully soon, and may change from the description here when they do get implemented. Feedback is welcome.

10.1 Modification timestamps

When individual resources are queried with GET, the HTTP response has a **`Last-Modified`** header set to the modification time of the resource. The **`If-Modified-Since`** request header can be used in the normal way.

These headers do not apply for requests and responses that concern multiple resources. For example, **`GET /foos`** won't have **`Last-Modified`**, nor obey **`If-Modified-Since`**. However, they both work for **`GET /foos/123`**.

10.2 `/competence_registries`

This resource stores information about external registries with data about competences: which persons have which competences. This is used to validate that a person has the competence they are claimed to have.

Synopsis:

- **`GET /competence_registries`** — get a list of ids of all competence registries
- **`POST /competence_registries`** — add a new competence registry
- **`GET /competence_registries/<id>`** — get the information about a specific competence registry
- **`PUT /competence_registries/<id>`** — update the information about a specific competence registry
- **`DELETE /competence_registries/<id>`** — remove a competence registry

Errors:

- There are no errors that are specific to this resource.

Example result for a competence certification:

EXAMPLE

```
{
    "type": "competence_registry",
    "id": "123",
    "names": ["S.P.E.C.T.R.E. enemies list"],
    "validation_interval": 86400,
    "competence_type_id": "456"
}
```

Fields in the result:

- **`names`** — list of names for the registry
- **`validation_interval`** — interval between validations of competencies against this registry, in seconds
- **`competence_type_id`** — id of the type of competencies in this registry (see competences)

10.2.1 Tests

Add tests.

Chapter 11

APPENDIX: Implementations for scenario steps

This chapter shows the actual implementations for each scenario step. If you're not a developer (of this test suite), you can skip this chapter entirely.

11.1 Saved values

For some scenarios, we need to capture some part of a response, such as the identifier for an entity, and then use that in, say, a future request. We do this by the following mechanism.

- A value can be saved to a named value. The scenario step specifies the name.
- The relevant steps expand references to saved values using the **`$FOO`** syntax.

Each step that supports this, mentions it explicitly.

Saved values can also be compared explicitly.

```
IMPLEMENTS THEN values "\$([^"]+)" and "\$([^"]+)" are identical
test "$(expand_values "\$$MATCH_1")" = "$(expand_values "\$$MATCH_2")"
```

11.2 Access tokens

Access token is requested with given scopes and saved to a token file.

```
IMPLEMENTS GIVEN client has access to scopes? "(.+)"
echo "$MATCH_1" | "$SRCDIR/createtoken" "$API_URL" > "$DATADIR/token"
```

Saves a string to a token file.

```
IMPLEMENTS GIVEN an invalid access token
echo "OHNO" > "$DATADIR/token"
```

11.3 Files

Creates a file with text content.

```
IMPLEMENTS GIVEN file (\S+) containing "(.+)"
/usr/bin/printf "$MATCH_2" > "$DATADIR/$MATCH_1"
```

11.4 Unique identifier

Creates a unique identifier used to uniquely identify several resources.

```
IMPLEMENTS GIVEN unique identifier \$(\S+)
save_value "$MATCH_1" "$(python -c 'import uuid; print(uuid.uuid4())')"
```

11.5 Requests

We do various kinds of requests. After each request, the headers (including the HTTP status line), and body are stored in **`$DATADIR/curl.headers`** and **`$DATADIR/curl.out`**, respectively. These can be examined by other scenario steps later on.

Note that a request step will not fail even if the HTTP status code is an error code. A step will only fail if curl itself returns a non-zero exit code.

For each requested, saved values are expanded in the path of the request and its body.

GET something. No body.

```
IMPLEMENTS WHEN client GETs (.+)
# FIXME: There is a propagation delay between updates (POST, PUT
# DELETE), which an mean that a modification plus an immediate
# GET results in an error (404), or pre-modified data. To avoid
# that, we artificially wait. This is not a good solution, but
# until we fix this properly, the kludge will have to do.
sleep 3
curl -k -D "$DATADIR/curl.headers" \
    -H "Authorization: Bearer $(cat "$DATADIR/token")" \
    "$API_URL$(expand_values "$MATCH_1")" \
    > "$DATADIR/curl.out" 2> "$DATADIR/curl.err"
```

POST or PUT with a JSON body.

```
IMPLEMENTS WHEN client (POST|PUT)s (\S+) with (.*)
expand_values "$MATCH_3" | tee "$DATADIR/curl.request.body"
curl -k -D "$DATADIR/curl.headers" \
    -X "$MATCH_1" \
    -H "Authorization: Bearer $(cat "$DATADIR/token")" \
    -H 'Content-Type: application/json' \
```

```
    -d "$(expand_values "$MATCH_3")" \
    "$API_URL$(expand_values "$MATCH_2")" \
    > "$DATADIR/curl.out" 2> "$DATADIR/curl.err"
```

DELETE with a specific header.

```
IMPLEMENTS WHEN client DELETEs (.+) with a header "(.+)"
curl -k -X DELETE -D "$DATADIR/curl.headers" \
    -H "Authorization: Bearer $(cat "$DATADIR/token")" \
    -H "$MATCH_2" \
    "$API_URL$(expand_values "$MATCH_1")" \
    > "$DATADIR/curl.out" 2> "$DATADIR/curl.err"
```

DELETE something. No body.

```
IMPLEMENTS WHEN client DELETEs ([^\s]+)
curl -k -X DELETE -D "$DATADIR/curl.headers" \
    -H "Authorization: Bearer $(cat "$DATADIR/token")" \
    "$API_URL$(expand_values "$MATCH_1")" \
    > "$DATADIR/curl.out" 2> "$DATADIR/curl.err"
```

PUT a file.

```
IMPLEMENTS WHEN client PUTs file (\S+) with content type (\S+) and revision
curl -k -D "$DATADIR/curl.headers" \
    -X PUT \
    -H "Authorization: Bearer $(cat "$DATADIR/token")" \
    -H "Content-Type: $(expand_values "$MATCH_2")" \
    -H "Revision: $(expand_values "$MATCH_3")" \
    --data-binary "@$DATADIR/$(expand_values "$MATCH_1")" \
    "$API_URL$(expand_values "$MATCH_4")" \
    > "$DATADIR/curl.out" 2> "$DATADIR/curl.err"
```

Puts a file without content.

```
IMPLEMENTS WHEN client PUTs no file with content type (\S+) and revision (\S
curl -k -D "$DATADIR/curl.headers" \
    -X PUT \
    -H "Authorization: Bearer $(cat "$DATADIR/token")" \
    -H "Content-Type: $(expand_values "$MATCH_1")" \
    -H "Revision: $(expand_values "$MATCH_2")" \
    "$API_URL$(expand_values "$MATCH_3")" \
    > "$DATADIR/curl.out" 2> "$DATADIR/curl.err"
```

11.6 Response examination

These steps examine the response to the latest HTTP request.

Check for a specific HTTP status code.

```
IMPLEMENTS THEN HTTP status code is (\d+)
status="$(awk 'NR == 1 { print $2 }' "$DATADIR/curl.headers")"
if [ "$status" != "$MATCH_1" ]
then
    echo "Expected status $MATCH_1, but got $status" 1>&2
    cat 1>&2 "$DATADIR/curl.headers" "$DATADIR/curl.out"
    exit 1
fi
```

Check for a specific header and its value.

```
IMPLEMENTS THEN HTTP header (\S+) is (.*)
value="$(cat "$DATADIR/curl.headers" | grep -i "$MATCH_1" | cut -d' ' -f2- |
if [ "$value" != "$MATCH_2" ]
then
    echo "Expected header $MATCH_1 to be $MATCH_2, but got $value" 1>&2
    cat 1>&2 "$DATADIR/curl.headers"
    exit 1
fi
```

Check for a location header value.

```
IMPLEMENTS THEN HTTP Location header is API_URL(\S+)
value="$(cat "$DATADIR/curl.headers" | grep "Location" | cut -d' ' -f2 | tr
if [ "$value" != "$API_URL$(expand_values "$MATCH_1")" ]
then
    echo "Expected location header to be $API_URL$(expand_values "$MATCH_1")
    cat 1>&2 "$DATADIR/curl.headers"
    exit 1
fi
```

Check if a specific header exists in the response headers.

```
IMPLEMENTS THEN HTTP header (\S+) exists
cat "$DATADIR/curl.headers" | grep "^$MATCH_1: "
```

Check result Date header.

```
IMPLEMENTS THEN result has a valid Date header
if ! grep Date: "$DATADIR/curl.headers"
then
    echo No Date: header  in response
```

```
    exit 1
fi
now="$(date +%s)"
date="$(sed -n '/^Date: /s///p' "$DATADIR/curl.headers" | tr -d '\n\r')"
hdr="$(date --date="$date" +%s)"
delta="$(echo "$now" - "$hdr" | bc -l)"
if [ "$delta" -lt -5 ] || [ "$delta" -gt 5 ]
then
    echo "Date: header is too old or too new"
    exit 1
fi
```

If the response had a body, we can check if the body matches file contents.

```
IMPLEMENTS THEN result body matches file (\S+)
cmp "$DATADIR/curl.out" "$DATADIR/$MATCH_1"
```

In most cases we interpret the response body as JSON hashmap (Python dict), and examine specific keys and their values.

Look for a specific a key, check that it's value is a dictionary, and check that the value has another key. Ignore the value of the subkey.

```
IMPLEMENTS THEN result has key "([^"]+)" with subkey "([^"]+)"
# We interpret curl.out as a JSON dict. This is easiest done in
# Python. Don't want to parse JSON in shell.

python -c '
import json, sys, os
obj = json.load(sys.stdin)
mainkey = os.environ["MATCH_1"]
subkey = os.environ["MATCH_2"]
assert type(obj) is dict, "expected a dictionary"
assert mainkey in obj, "expected dictionary to have key %s" % mainkey
assert type(obj[mainkey]) is dict, "expected value to be a dictionary"
assert subkey in obj[mainkey], "expected value to have key %s" % subkey
' < "$DATADIR/curl.out"
```

Check for a specific key in a dict, and save the value.

```
IMPLEMENTS THEN result has key "([^"]+)" containing a string, saved as \$(\S
# Run jsonextract separately so that if it fails, the whole thing
# fails.
"$SRCDIR/jsonextract" "$MATCH_1" < "$DATADIR/curl.out" \
    > "$DATADIR/value"
save_value "$MATCH_2" "$(cat "$DATADIR/value")"
```

Extract the resource id of a list of resources returned by a query.

```
IMPLEMENTS THEN result lists resources, id of index ([0-9]+) saved as \$(\S+
echo ----
cat "$DATADIR/curl.out"
echo ----
python -c '
import json, sys, os
obj = json.load(sys.stdin)
index = int(os.environ["MATCH_1"])
assert type(obj) is dict, "expected a dictionary"
assert "resources" in obj, "expected dictionary to have key resources"
sys.stdout.write(obj["resources"][index]["id"])
' < "$DATADIR/curl.out" > "$DATADIR/saved.tmp"

save_value "$MATCH_2" "$(cat "$DATADIR/saved.tmp")"
```

Look for a specific resource in a search result. This is so that we can have steps that look for known matches, but don't get confused by additional matches.

```
IMPLEMENTS THEN search result contains match (.*)
python -c '

import json, sys, os
obj = json.load(sys.stdin)
wanted = json.loads(sys.argv[1])
print "wanted:", repr(wanted)
for match in obj["resources"]:
    print "match:", repr(match)
    for key in wanted:
        if match.get(key) == wanted[key]:
            sys.exit(0)
print "no match found"
sys.exit(1)

' "$(expand_values "$MATCH_1")" < "$DATADIR/curl.out"
```

Match the whole JSON response against a JSON value. A value matches according to the following rules:

- if the pattern is a dict, the value must be a dict, and all the keys in the pattern must exist in the value and have matching values
- if the pattern is a list, the value must be a list and have the same values, in the same order
- if the pattern is a string, the value must be an identical string

Thus `{ "names": ["foo", "bar"] }` as a pattern will match the value `{ "id": "123", "names": ["foo", "bar"] }`.

Any references to saved values are expanded in the pattern.

```
IMPLEMENTS THEN result matches (.*)
"$SRCDIR/jsonmatch" whole \
    "$(expand_values "$MATCH_1")" < "$DATADIR/curl.out"
```

Test for not matching a pattern.

```
IMPLEMENTS THEN result doesn't match (.*)
! "$SRCDIR/jsonmatch" whole \
    "$(expand_values "$MATCH_1")" < "$DATADIR/curl.out"
```

Sometimes we want to check if a result has a list which contains a specific value, rather than a specific set of values. This is true, for example, for requests of the type of **`GET /foos`**, when run against a system in production.

```
IMPLEMENTS THEN result has key "([^"]*)", a list containing (.*)
"$SRCDIR/jsonmatch" list-item "$(expand_values "$MATCH_2")" \
    "$MATCH_1" < "$DATADIR/curl.out"

IMPLEMENTS THEN result has key "([^"]*)", a list that does not contain (.*)
! "$SRCDIR/jsonmatch" list-item "$(expand_values "$MATCH_2")" \
    "$MATCH_1" < "$DATADIR/curl.out"
```

Chapter 12

APPENDIX: Release Notes

This appendix contains a summary of relevant changes since the previous release. Minor changes, such as corrections to spelling, are not listed here.

12.1 Version 0.3, released 2015-06-04

- Things that are planned, but not implemented, have been moved to an appendix. Wild ideas that are not currently needed have been dropped. Both will be added back as they get implemented. This should clarify what parts of the API can and can't be used for each release.
- Competence related resources have been tweaked to better match applications being written.
- API tests have been improved.
- JSON field names now use underscores instead of dashes. For example, `gov-org-ids` is now `gov_org_ids`. This makes it easier to handle the resouces in Javascript.
- `org` resources now have a `country` field.

12.2 Version 0.4, released 2015-07-01

- The `contract` resource has been added, supporting employment contracts and inductions to worksites (Finnish *perehdytys*), which is modelled as a contract.
- An `event` resource type has been added. It is meant to be used for various time-stamped events, such as gate access (Finnish *leimatieto*).
- A `report` resource types has been added, for storing reports of various kinds.
- We've added the ability to have **binary file attachments** for resources. For example, a scanned PDF of a signed contract may be attached to a `contract` resource.
- **Lists of resources** returned by various requests, such as `GET /foos` or a search, all now use the same response format. This is an **incompatible** change, sorry, but there's no external users of the API yet, so it's still OK to make such changes.
- **Searches** have been added. You can search for one type of resource at a time, for any field in the resource.

- **Change notifications** for resources have been added and are ready to be used.
- **Resource revisions** and update conflicts have been implemented. When updating a resource you now have to specify what you think is the current revision, and if the API backend disagrees, the update fails. The HTTP status code for update conflicts has been corrected to be 409, instead of 408, as the document previously suggested.
- The separate **mockup implementation** of the API (`mockup.py`) has been dropped, in favour of automated provisioning of API instances of the production implementation.

12.3 Version 0.5, released 2015-07-23

- The `card` resource type has moved from ideas to being real. As part of that, we dropped some complications, which are not needed at this time. (Tell us when you need them, and they'll be added back.)
- The `project` resource type has moved form ideas to being real.
- Preliminary specification for authentication and authorisation has been added to the API. These are based on OAuth 2.0 and OpenID Connect, using JSON Web Tokens.

12.4 Version 0.6, released UNRELEASED

- Creation of resources now returns HTTP status 201. This is the customary and expected status code for that.
- Authentication and authorization have documented in much more detail, with tests added.
- The "Services" chapter has been removed. It was irrelevant for the scope of a storage-only API.
- The whole text has been improved and clarified.
- The term "message box" is now used consistently for change notifications. Previously, "mailbox" was sometimes used instead, for no good reason.
- New resourced have been added: `competence`, `competence_type`.
- Support for Swedish `ID06` identity cards has been added. This affects `card` and `contract` resources.
- Persons may now have titles associated with their names.