API as a Service - API - Deutsch Documentation

Release latest

18.08.2020

Einleitung

Willkommen zur Dokumentation von *AaaS-API*.

API as a Service ermöglicht das Konzipieren und automatisierte Erstellen von PHP-APIs über eine entsprechende Benutzeroberfläche. Die Benutzeroberfläche ist eine eigenständige Applikation und vom Rest des Systems separiert. Im Folgenden finden Sie die Dokumentation der API von *API as a Service*.

KAPITEL 1

Einführung

Im Folgenden möchten wir Ihnen die Grundlagen der API von *API as a Service* vermitteln und Sie mit den von uns verwendeten Begrifflichkeiten und deren Definitionen vertraut machen.

1.1 Grundlegendes

Um eine API erstellen zu können sind normalerweise eine Reihe von Vorkehrungen notwendig. So muss beispielsweise ein Datenbankentwickler ein entsprechendes Schema entwerfen und bereitstellen und ein Programmierer Schnittstellen entwickeln und implementieren, damit Klienten, Applikationen oder Drittanbieter lesend und schreibend auf das Datenbanksystem zugreifen können.

Mit *AaaS-API* können Sie programmatisch PHP-APIs generieren und veröffentlichen. Wir stellen Ihnen Schnittstellen zur Verfügung, mit denen Sie sowohl Datenbankschemata entwickeln können als auch entsprechende Programmierschnittstellen für Ihre API-Konsumenten.

1.2 Begrifflichkeiten

AaaS-API bietet die Möglichkeit, ihre *Projekte* in Form von *Repositorien* zu modularisieren. Einem *Repositorium* widerum können eine Reihe von *Services* (Datenmodelle) zugeordnet werden, welche ihrerseits Datenbanktabellen aus dem von Ihnen erstellten Schema in Ihrem Datenbanksystem abbilden. Schlussendlich bestehen Services aus *Feldern*, mit denen Sie Ihren Datenmodellen Eigenschaften (Datenattribute) zuordnen können.

Begriff	**Definition**	**Beispiel**
Projekt	Eines Ihrer API-Projekte	Webanwendung
Repositorium	Eine Reihe von Projektmodulen	Blog, Shop
Service	Einem Projektmodul zugeordnete Datenmodelle	Post, Author, Product, Category
Feld	Einem Service zugeordnete Datenattribute	title, name, color, parent

1.3 Beispiel

Ihr API-Projekt könnte beispielsweise eine Webanwendung sein, die unter anderem verschiedene Module wie einen Blog und einen Onlineshop beinhaltet. Jedes Modul besteht aus entsprechenden Services:

- Das `Blog`-Modul könnte unter anderem einen `Post`-Service umfassen, der datenbanktechnisch Ihre `Blog`-Einträge abbildet.
- Das `Shop`-Modul könnte einen `Product`- und `Category`-Service beinhalten, die Produkte und Kategorien in Ihrer Datenbank repräsentieren.

Um den Services Eigenschaften zuordnen zu können, werden diese mit Attributen versehen:

- Dem `Post`-Service könnte ein `title`-Attribut zugeordnet sein, der den Titel Ihres `Blog`-Eintrages darstellt.
- Der `Product`-Service könnte ein `color`-Attribut enthalten, über den Sie die Farbe eines in Ihrer Datenbank befindlichen Produkte persistieren könnten.
- Der `Category`-Service könnte ein `parent`-Attribut beinhalten, welches Ihren API-Konsumenten die übergeordnete Kategorie anzeigt.

Die Möglichkeiten sind mannigfaltig. So könnten Sie im `Shop`-Modul auch Beziehungen (*Relationen*) unter den Services abbilden um beispielsweise eine Anzahl an Reihen der den Produkten zugeordneten Attribute zu gruppieren. Wenn Sie einen Onlineshop aufbauen möchten um später Bekleidung und Werkzeuge verkaufen zu können, dann benötigen Sie für die Produkte aus der Kategorie *Bekleidung* andere Attribute als für die Produkte aus der Kategorie *Werkzeuge*. Eine mögliche Lösung bestünde dementsprechend in der Separierung und Gruppierung der Attribute der den unterschiedlichen Kategorien zugeordneten Produkte.

Durch die von uns zur Verfügung gestellten Programmierschnittstellen werden Sie in der Lage sein, auch komplexe Anforderungen Ihrer API-Konsumenten zu erfüllen und sauber in Ihrem System abzubilden.

KAPITEL 2

Installation

Wenn Sie *AaaS-API* ausprobieren möchten, verwenden Sie bestenfalls die sich in der README.md befindliche Installationsanleitung aus unserem öffentlich zugänglichen *Github Repositorium*. Der Vollständigkeit halber bieten wir Ihnen aber auch hier eine Anleitung zur Installation unserer Applikation an.

Wir stellen Ihnen Konfigurationsdateien für Docker Compose zur Verfügung, mit denen Sie *AaaS-API* isoliert vom Rest Ihres Systems betreiben können. Wenn Sie zur Orchestrierung der für unsere Anwendung benötigten Dienste keine Containervirtualisierung mittels Docker verwenden wollen, sondern zur Ausführung stattdessen eine virtuelle Maschine wie Vagrant bevorzugen oder wenn Sie unsere Applikation nativ auf einem Ihrer Rechner betreiben möchten, dann muss Ihr System eine Reihe an Anforderungen erfüllen.

2.1 Anforderungen

- Anwendungsserver: PHP in der Version >= 7.2.5.
- Webserver: Wir empfehlen nginx und liefern eine Konfigurationsdatei für PHP-FPM.
- Datenbankserver: Wir empfehlen MariaDB in der Version >= 10.2.7. Sie können auch MySQL in der Version >= 5.7.8 benutzen.
- Cacheserver: Redis.
- Composer zur Verwaltung von Drittanbieterabhängigkeiten.
- OpenSSL zur Erzeugung kryptografischer Schlüssel.
- Git als Versionsverwaltungssystem. (Optional)

Unsere Konfigurationsdateien zur Dienstekomposition mit Docker decken alle Anforderungen an das Endsystem ab. Wir empfehlen dementsprechend sowohl für ihre Entwicklungs- als auch für Ihre Produktivumgebung die Verwendung von Docker. Zur Verwaltung Ihrer Docker-Umgebungen empfehlen wir Portainer oder Rancher.

2.2 Installationsanleitung

Falls Sie *Git* installiert haben, klonen Sie bitte das Projekt in einen Ordner Ihrer Wahl. Als Alternative dazu können Sie auch eine aktuelle Version komprimiert herunterladen und entpacken:

```
$ git clone git@github.com:christiansiewert/aaas-api.git <DIRECTORY>
$ cd <DIRECTORY>
```

Hinweis: Wenn Sie, wie von uns empfohlen, *Docker Compose* benutzen, dann müssen Sie jetzt `docker-compose up` ausführen, um die einzelnen Dienste, wie den Anwendungsserver oder den Webserver, zu starten. Außerdem müssen Sie allen folgenden Kommandos `docker-compose exec php` voranstellen, um das Kommando auf dem entsprechenden *PHP-Container* Ihrer Docker-Umgebung auszuführen. Sie können als Alternative dazu auch `docker-compose exec php /bin/bash` ausführen, um eine *Linux-Shell* auf dem *PHP-Container* zu starten.

Sie installieren anschließend bitte die Drittanbieterabhängigkeiten:

```
$ composer install
```

Wenn Composer mit einem *Out of memory*-Fehler abbricht, deaktivieren Sie bitte die Speicherbegrenzung von Composer:

```
$ COMPOSER_MEMORY_LIMIT=-1 composer install
```

Sie erzeugen bitte ein *RSA-Schlüsselpaar* zur späteren Verifizierung von *JWT-Access-Tokens*. Bei der Erzeugung werden Sie nach einer *Passphrase* gefragt, welche Sie bitte anschließend als Umgebungsvariable in der Datei .env unterhalb des Wurzelverzeichnisses eintragen. Der Name der Umgebungsvariable lautet `JWT_PASSPHRASE` und ist standardmäßig auf `app!` gesetzt:

```
$ openssl genrsa -out config/jwt/private.pem -aes256 4096
$ openssl rsa -pubout -in config/jwt/private.pem -out config/jwt/public.pem
```

Sie finden innerhalb der `.env`-Datei außerdem noch die Umgebungsvariablen `DATABASE_USER`, `DATABASE_PASSWORD` und `DATABASE_NAME`. Hier tragen Sie bitte die entsprechenden Informationen ein, damit unsere Anwendung auf Ihr Datenbanksystem zugreifen kann. Schlussendlich migrieren Sie bitte Ihre leere Datenbank auf den Ausgangszustand unserer Applikation:

```
$ php bin/console doctrine:migrations:migrate --no-interaction
```

Sie sollten nun einen Administrator-Account erstellen, über den Sie später HTTP-Anfragen an unsere Ressourcen absetzen und damit Ihre API erstellen können:

```
$ php bin/console acl:create-user EMAIL PASSWORD --admin
```

Nachdem Sie das getan haben, können Sie testweise einen JWT-Zugriffstoken vom Server beziehen:

```
curl -X POST -H "Content-Type: application/json" http://localhost/auth/login_check \
-d '{ "email" : "EMAIL", "password" : "PASSWORD" }'
```

Wenn Ihre Zugangsdaten korrekt sind, dann sollte der Server mit einem entsprechenden Token antworten:

```
{
    "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpYXQiOcm9sZXMiOlsiUk9MRV9..."
}
```

Sie haben *AaaS-API* damit erfolgreich installiert und können unsere Anwendung nun benutzen.

KAPITEL 3

Zugriffsrechte

Obschon Sie später für Ihre API entsprechende Zugriffsrechte definieren und implementieren können, integriert auch unsere Anwendung ein angemessenes Rechtesystem. *API as a Service* bietet derzeit zwei Rollen an, die Sie an Ihre Mitarbeiter vergeben können:

Rolle	Definition
ROLE_AAAS_USER	Hat vollen Zugriff, aber nicht auf `/aaas/system`
ROLE_AAAS_ADMIN	Erbt von `ROLE_AAAS_USER` und hat Zugriff auf `/aaas/system`

Die Rolle `ROLE_AAAS_ADMIN` darf demzufolge das System migrieren und den Applikationscache leeren. `ROLE_AAAS_USER` hat hierfür keine Rechte. Wenn Sie die Installationsanleitung in diesem Wiki gelesen haben, dann wissen Sie vielleicht schon, wie Sie entsprechende Benutzer anlegen können, die unsere Applikation später bedienen.

3.1 Benutzer anlegen

Sie setzen dafür ein entsprechendes Kommando von Ihrem Terminal ab und ersetzen `EMAIL` und `PASSWORD` mit validen Werten:

```
$ docker-compose exec php php bin/console acl:create-user EMAIL PASSWORD --admin
```

Wenn Sie einen Benutzer mit der Rolle `ROLE_AAAS_USER` erstellen möchten, dann lassen Sie `--admin` einfach weg.

3.2 JWT-Zugriffstoken

Unsere Anwendung beinhaltet eine Implementierung von RFC 7519 (JSON Web Token). Mithilfe kryptografischer Algorithmen werden dabei Sicherheitsansprüche (Benutzer, Rollen, etc.) verschlüsselt und anschließend kodiert. Beachten Sie bitte, dass Sie sich an allen Endpunkten unterhalb von `/aaas` autorisieren müssen. Sie senden dafür später einen JWT-Zugriffstoken im `Authorization`-HTTP-Header.

Wenn Sie einen Zugriffstoken vom Server beziehen möchten, dann senden Sie eine HTTP-POST-Anfrage an den `/auth/login_check`-Endpunkt:

```
curl -X POST -H "Content-Type: application/json" http://localhost/auth/login_check \
-d '{ "email" : "EMAIL", "password" : "PASSWORD" }'
```

Wenn Ihre Zugangsdaten korrekt sind, dann sollte der Server mit einem entsprechenden Token antworten:

```
{
    "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpYXQiOcm9sZXMiOlsiUk9MRV9..."
}
```

Wenn Sie sich für den Header und Payload des Tokens in dekodierter Form interessieren sollten, dann geben wir Ihnen hier ein Beispiel:

```
{
  "typ": "JWT",
  "alg": "RS256"
}
```

```
{
  "iat": 1597744282,
  "exp": 1597747882,
  "roles": [
    "ROLE_AAAS_ADMIN",
    "ROLE_USER"
  ],
  "username": "christian@sieware.international"
}
```

Unsere JWT-Zugriffstoken besitzen eine Gültigkeitsdauer von 60 Minuten.

3.3 HTTP-Header

Sie senden dann bei allen Anfragen unterhalb unserer `/aaas`-Ressourcen diesen Zugriffstoken im `Authorization`-HTTP-Header mit und setzen den Wert auf `Bearer TOKEN`.

HTTP-Header	Wert
Authorization	Bearer TOKEN

Sollten Sie keinen oder einen invaliden Token senden, antwortet der Server mit einem `401 Unauthorized Error`.

KAPITEL 4

Grundlagen

Wenn Sie *AaaS-API* erfolgreich installiert haben, können Sie anfangen Ihre Projekte, Repositorien und Services anzulegen. Wenn Sie damit fertig sind, erstellt unsere Anwendung aus Ihren Projekten sowohl Datenbankschemata als auch entsprechende Programmierschnittstellen, die anschließend von Ihren API-Konsumenten benutzt werden können. Dabei sind eine Reihe von Besonderheiten zu beachten, die wir Ihnen im Folgenden näher bringen wollen.

Hinweis: Die Leistungsfähigkeit von *AaaS-API* im Entwicklungsmodus, insbesondere wenn der Applikationscache noch nicht generiert wurde oder aufgrund von Änderungen neu generiert werden muss, unterscheidet sich signifikant von der Leistungsfähigkeit im Produktivbetrieb. Während der Produktion sollten Sie `APP_ENV` in der .env-Datei unbedingt auf `prod` setzen. Lesen Sie mehr dazu im Abschnitt *Deployment*.

4.1 Endpunkt-Präfixe

Wir separieren die von Ihnen entworfene und von uns erstellte API von unserem System mittels URI-Präfixen. So können Sie unsere API-Ressourcen vor Ihren API-Konsumenten verbergen oder dereinst die Zugangsberechtigungen per Zugriffslisten (ACL) steuern.

URI-Präfix	Definition
/aaas	Zugriff auf Ressourcen zur Generierung Ihrer API (Unser System)
/api	Zugriff auf Ressourcen der von uns generierten API (Ihr System)
/docs	Swagger, Swagger UI, ReDoc. (Deaktiviert im Produktivbetrieb)
/auth	Endpunkt für Authentifizierung

4.2 HTTP-Clients

Um sich mit *AaaS-API* vertraut zu machen und Anfragen an die API-Endpunkte zu stellen, können Sie einen HTTP-Client Ihrer Wahl benutzen. Im Prinzip reicht dafür schon das Kommandozeilen-Programm cURL. Wir empfehlen Postman oder die Chrome-Erweiterung Restlet-Client. Wenn Sie *AaaS-API* innerhalb Ihrer Entwicklungsumgebung

benutzen, sodass die Umgebungsvariable `APP_ENV` in der .env-Datei auf `dev` gesetzt ist, dann können Sie auch Swagger UI benutzen. Rufen Sie dafür einfach https://localhost/docs in Ihrem Browser auf.

4.3 HTTP-Header

Unser System kann API-Ressourcen in unterschiedlichen Formaten serialisieren und deserialisieren. So unterstützen wir neben *JavaScript Object Notation* (JSON) auch *JavaScript Object Notation für verlinkte Daten* (JSON-LD). Eventuell werden wir in Zukunft weitere Formate anbieten. Je nachdem, in welchen Format Sie Ihre Anfragen stellen oder in welchem Format Sie unsere Antworten erwarten, müssen Sie entweder einen HTTP-Header bei der Anfrage mitschicken oder einen Format-Suffix an die entsprechende API-URI anhängen.

HTTP-Header	Wert	Definition
Accept	application/json	Ihr HTTP-Client erwartet JSON. Unsere API antwortet im JSON-Format.
Content-Type	application/json	Unsere API erwartet valides JSON. Sie schicken im HTTP-Request-Body valides JSON.

Als Alternative dazu können Sie auch einen Suffix anhängen, der das Ausgabeformat spezifiziert:

```
HTTP_METHOD SCHEME ":" //HOST/PREFIX/PATH.json # ".json" ist der Suffix.
```

Der API-Server antwortet dann im JSON-Format.

4.4 HTTP-Requests

Wenn Sie *AaaS-API* ohne unsere grafische Benutzeroberfläche verwenden oder wenn Sie eine eigene Benutzeroberfläche entwickeln möchten, dann müssen Sie sich mit unseren API-Endpunkten vertraut machen. Bei unseren API-URIs gilt grundsätzlich, dass Sie mit Ihrem HTTP-Client die entsprechende HTTP-Methode angeben und anschließend eine HTTP-Anfrage an einen unserer Endpunkte stellen:

```
HTTP_METHOD SCHEME ":" //HOST/PREFIX/PATH.FORMAT [ "?" QUERY ]
```

Die von uns generierte und später von Ihren API-Konsumenten zu benutzende API beinhaltet zusätzlich noch den Namen des entsprechenden Repositoriums in der URI. So können Sie unter anderem Ihre Zugriffslisten filigraner gestalten:

```
HTTP_METHOD SCHEME ":" //HOST/PREFIX/REPOSITORY/PATH.FORMAT  [ "?" QUERY ]
```

`FORMAT` und `QUERY` sind optional. Um sich eine Liste aller API-Projekte ausgeben zu lassen, könnten Sie folgende HTTP-GET-Anfrage stellen:

```
curl -X GET "https://localhost/aaas/projects" -H "accept: application/json"
```

Der API-Server gibt Ihnen dann die entsprechende Kollektion zurück:

```
[
  {
    "id": 1,
    "name": "My Online Shop",
    "description": "My fance online shop which uses AaaS.",
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    "repositories": [
      "/aaas/project/repositories/1",
      "/aaas/project/repositories/2"
    ]
  }
]
```

Wenn Sie als `Format-Suffix` `json` angeben, dann müssen Sie den `Accept-Header` nicht mitsenden:

```
curl -X GET "http://localhost/aaas/projects.json"
```

Wenn Sie statt der Kollektion nur auf eine einzelne Ressource zugreifen möchten, dann müssen Sie noch einen eindeutigen Schlüssel angeben, der für jede unserer und später auch Ihrer API-Ressourcen existiert:

```
curl -X GET "https://localhost/aaas/project/repositories/1" -H "accept: application/
↪json"
```

```
{
  "id": 1,
  "name": "Blog",
  "description": "Blog repository holds services for our blog.",
  "services": [
    "/aaas/repository/services/1",
    "/aaas/repository/services/2",
    "/aaas/repository/services/3"
  ]
}
```

```
curl -X GET "https://localhost/aaas/repository/services/1" -H "accept: application/
↪json"
```

```
{
  "id": 1,
  "name": "Article",
  "description": "Articles for our blog repository.",
  "type": "list",
  "repository": "/aaas/project/repositories/1",
  "fields": [
    "/aaas/service/fields/1",
    "/aaas/service/fields/2"
  ]
}
```

```
curl -X GET "https://localhost/aaas/service_fields/1" -H "accept: application/json"
```

```
{
    "id": 1,
    "name": "title",
    "description": "Title for our blog post.",
    "dataType": "string",
    "length": 255,
    "isUnique": false,
    "isNullable": false,
    "dataTypePrecision": null,
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    "dataTypeScale": null,
    "service": "/aaas/repository_services/1",
    "options": [],
    "assertions": [],
    "relation": null
}
```

4.5 HTTP-Methoden

Unsere API stellt Ihnen applikationsübergreifend die HTTP-Methoden `GET`, `PUT`, `POST` und `DELETE` zur Verfügung.

Methode	Definition
GET	Sie beziehen Ressourcen vom Server
POST	Sie legen eine Ressource auf dem Server an
PUT	Sie bearbeiten eine Ressource auf dem Server
DELETE	Sie löschen eine Ressource vom Server

4.6 Eingebettete Ressourcen

Wie Ihnen sicherlich schon aufgefallen ist, serialisiert unser System eingebettete Ressourcen bzw. Relationen über dereferenzierbare IRIs. Sie müssen dementsprechend zusätzliche HTTP-Requests absetzen um Details der entsprechenden Objekte zu erhalten. Aus Gründen der Leistungsfähigkeit ist es allerdings manchmal sinnvoller zusätzliche HTTP-Requests zu vermeiden und stattdessen den API-Server anzuweisen die Details der entsprechenden Relationen ergänzend zu serialisieren. Sie können dafür von uns implementierte Serialisierungsgruppen benutzen. Für Projekte, Repositorien und deren Services geben wir Ihnen hier schon einmal einen kleinen Überblick. Sie erfahren mehr auf den nächsten Seiten.

Gruppe	Erklärung
repository	Serialisiert `repository`-Attribute
service	Serialisiert `service`-Attribute

Mit *AaaS-API* haben Sie später ebenfalls die Möglichkeit Serialisierungsgruppen zu erstellen und Ihren API-Konsumenten zur Verfügung zu stellen. In einigen Fällen ist Ihr System dann leistungsfähiger.

4.6.1 Beispiel

```
curl -X GET "https://localhost/aaas/projects/1?groups[]=repository&groups[]=service" -
↪H "accept: application/json"
```

```
{
    "id": 1,
    "name": "My Online Shop",
    "description": "My fance online shop which uses AaaS.",
    "repositories": [
        {
            "id": 2,
            "name": "Catalog",
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
            "description": "Catalog repository holds services for our catalog.",
            "services": [
                {
                    "id": 4,
                    "name": "Product",
                    "description": "Products service for our catalog repository.",
                    "type": "list",
                    "fields": []
                },
                {
                    "id": 5,
                    "name": "Category",
                    "description": "Categories for our catalog repository.",
                    "type": "tree",
                    "fields": []
                },
                {
                    "id": 6,
                    "name": "ProductDetail",
                    "description": "Details for our products.",
                    "type": "list",
                    "fields": []
        },
        {
            "id": 1,
            "name": "Blog",
            "description": "Blog repository holds services for our blog.",
            "services": [
                {
                    "id": 1,
                    "name": "Article",
                    "description": "Articles for our blog repository.",
                    "type": "list",
                    "fields": []
                },
                {
                    "id": 2,
                    "name": "Label",
                    "description": "Labels for our blog repository.",
                    "type": "list",
                    "fields": []
                },
                {
                    "id": 3,
                    "name": "Comment",
                    "description": "Comments for our blog repository.",
                    "type": "list",
                    "fields": []
                }
            ]
        }
    ]
}
```

KAPITEL 5

Projekte

Wenn Sie den Abschnitt *Einführung* gelesen haben, dann wissen Sie, dass Sie Ihre Datenmodelle in Form von Repositorien modularisieren können. Eine Anzahl an Repositorien widerum bildet Ihr API-Projekt ab. Mit *AaaS-API* können Sie mehrere API-Projekte anlegen. Dafür reicht derzeit ein *Name* und ein optionaler *Beschreibungstext*.

5.1 Projekte anlegen

Sie senden einen HTTP-POST-Request an den entsprechenden Endpoint und schicken valides JSON im HTTP-Request-Body:

```
curl -X POST "https://localhost/aaas/projects" -H CONTENT_TYPE_HEADER -H ACCEPT_
↪HEADER -d HTTP_REQUEST_BODY
```

Für obiges Beispiel könnte Ihr HTTP-Request-Body wie folgt aussehen:

```
{
  "name": "My API",
  "description": "This is my API project."
}
```

Wenn Sie keinen Beschreibungstext angeben möchten, können Sie ihn entweder auf `null` setzen oder Sie lassen das entsprechende Attribut weg:

```
{
  "name": "My API",
  "description": null
}
```

```
{
  "name": "My API"
}
```

Der API-Server antwortet dann mit dem erstellten Datensatz:

```
{
    "id": 1,
    "name": "My API",
    "description": "This is my API project.",
    "repositories": [
    ]
}
```

Hinweis: In allen folgenden Beispielen lassen wir den Header- und Request-Body-Parameter weg. In den Beispielen setzen wir voraus, dass Sie in allen HTTP-Requests den HTTP-Header `Content-Type` und `Accept` auf `application/json` setzen. Zur besseren Lesbarkeit fügen wir JSON-Code-Blöcke ein.

5.2 Projekte bearbeiten

Wenn Sie ein vorhandes Projekt bearbeiten möchten, dann senden Sie einen HTTP-PUT-Request an die entsprechende Resource und geben im HTTP-Request-Body an, welche Attribute Sie bearbeiten möchten:

```
curl -X PUT "https://localhost/aaas/projects/1" ...
```

```
{
  "name": "My edited API"
}
```

```
{
    "id": 1,
    "name": "My edited API",
    "description": "This is my API project.",
    "repositories": [
    ]
}
```

Es ist ausreichend, nur jene Objekt-Schlüssel zu senden, die Sie auch bearbeiten möchten:

```
{
  "description": "This is my edited API project."
}
```

```
{
    "id": 1,
    "name": "My edited API",
    "description": "This is my edited API project.",
    "repositories": [
    ]
}
```

5.3 Projekte löschen

Wenn Sie eines Ihrer API-Projekte löschen, dann senden Sie einen HTTP-DELETE-Request an die entsprechende Ressource:

```
curl -X DELETE "https://localhost/aaas/projects/1" ...
```

Der API-Server antwortet dann mit dem HTTP-Statuscode 204, welcher anzeigt, dass die Operation erfolgreich ausgeführt wurde und ansonsten kein weiterer Inhalt gesendet wird.

Hinweis: Löschoperationen wirken generell kaskadierend. Wenn Sie also ein Projekt löschen, dann werden zugehörige Repositorien und Services ebenfalls gelöscht.

5.4 Serialisierungsgruppen

Wenn Sie den Abschnitt *Eingebettete Ressourcen* im Kapitel *Grundlagen* gelesen haben, dann wissen Sie, dass Sie Serialisierungsgruppen benutzen können um eingebettete Ressourcen zusätzlich vom API-Server serialisieren zu lassen:

Gruppe	Definition
repository	Serialisiert/Deserialisiert `repository`-Attribute
service	Serialisiert/Deserialisiert `service`-Attribute

5.4.1 Deseralisierung von Repositorien

Sie können Serialisierungsgruppen auch zur Deserialisierung benutzen. So sind Sie nicht nur in der Lage, eingebettete Ressourcen ohne zusätzliche HTTP-Requests abzufragen, sondern ebenso anzulegen und auch zu bearbeiten. Im folgenden Beispiel legen wir ein Projekt und zusätzlich zwei Repositorien an:

```
curl -X POST "https://localhost/aaas/projects?groups[]=repository" ...
```

```
{
  "name": "Web Application",
  "description": "This is my Web Application.",
  "repositories" : [
    {
      "name": "Blog",
      "description": "Blog repository."
    },
    {
      "name": "Shop",
      "description": "Shop repository"
    }
  ]
}
```

Der API-Server antwortet dann mit dem HTTP-Statuscode 201 (Ressource angelegt) und gibt Ihnen die entsprechende Ressource zurück:

```
{
    "id": 1,
    "name": "Web Application",
    "description": "This is my Web Application.",
    "repositories": [
        {
            "id": 1,
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
            "name": "Blog",
            "description": "Blog repository.",
            "project": "/aaas/projects/1",
            "services": [
            ]
        },
        {
            "id": 2,
            "name": "Shop",
            "description": "Shop repository",
            "project": "/aaas/projects/1",
            "services": [
            ]
        }
    ]
}
```


5.4.2 Deseralisierung von Repositorien und Services

Analog dazu können Sie auch gleich Services mit anlegen:

```
curl -X POST "https://localhost/aaas/projects?groups[]=repository&groups[]=service" ..
↪.
```


```
{
  "name": "Web Application",
  "description": "This is my Web Application.",
  "repositories" : [
    {
      "name": "Blog",
      "description": "Blog repository.",
      "services": [
        {
          "name": "Post",
          "description": "Service holds blog articles."
        }
      ]
    },
    {
      "name": "Shop",
      "description": "Shop repository"
    }
  ]
}
```



```
{
    "id": 3,
    "name": "Web Application",
    "description": "This is my Web Application.",
    "repositories": [
        {
            "id": 9,
            "name": "Blog",
            "description": "Blog repository.",
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
            "project": "/aaas/projects/3",
            "services": [
                {
                    "id": 2,
                    "name": "Post",
                    "description": "Service holds blog articles.",
                    "type": "list",
                    "repository": "/aaas/project/repositories/9",
                    "fields": [
                    ]
                }
            ]
        },
        {
            "id": 10,
            "name": "Shop",
            "description": "Shop repository",
            "project": "/aaas/projects/3",
            "services": [
            ]
        }
    ]
}
```


Sie erfahren mehr zu Repositorien und Services auf den nächsten Seiten.

KAPITEL 6

Repositorien

Wenn Sie das Kapitel *Projekte* gelesen haben, dann haben Sie eventuell schon eine gewisse Vorstellung davon, wie Sie Repositorien anlegen können. So, wie bei Projekten auch, reicht dafür derzeit ebenfalls lediglich ein *Name* und ein optionaler *Beschreibungstext*.

6.1 Repositorien anlegen

Um ein Repositorium anlegen zu können, senden Sie, wie oben bereits erwähnt, einen Namen und einen optionalen Beschreibungstext. Sie müssen bei einem Repositorium zusätzlich das Projekt angeben, welches Sie dem Repositorium zuordnen möchten. Sie können dafür entweder eine dereferenzierbare IRI angeben oder Sie benutzen Serialisierungsgruppen und fügen das Projekt als eingebette Ressource hinzu.

Wenn Sie das `project`-Attribut weglassen, antwortet der API-Server mit einer entsprechenden Fehlermeldung:

```
{
    "type": "https://tools.ietf.org/html/rfc2616#section-10",
    "title": "An error occurred",
    "detail": "project: This value should not be blank.",
    "violations": [
        {
            "propertyPath": "project",
            "message": "This value should not be blank."
        }
    ]
}
```

Wir legen ein Repositorium über eine dereferenzierbare IRI an:

```
curl -X POST "https://localhost/aaas/project/repositories" ...
```

```
{
  "name": "Blog",
  "description": "Blog repository.",
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
  "project" : "/aaas/projects/1"
}
```

Der API-Server antwortet mit dem angelegten Repositorium:

```
{
    "id": 1,
    "name": "Blog",
    "description": "Blog repository.",
    "project": "/aaas/projects/1",
    "services": [
    ]
}
```

6.2 Repositorien bearbeiten

Beim Bearbeiten eines Repositoriums gehen Sie gleichermaßen vor, wie beim Bearbeiten eines API-Projekte. Es ändert sich lediglich der API-Endpunkt:

```
curl -X PUT "https://localhost/aaas/project/repositories/1" ...
```

```
{
    "name": "Edited Blog",
    "description": "My edited Blog repository"
}
```

```
{
    "id": 1,
    "name": "Edited Blog",
    "description": "My edited Blog repository",
    "project": "/aaas/projects/1",
    "services": [
    ]
}
```

6.3 Repositorien anzeigen

Um eine ungefilterte Ergebnisliste zu erhalten, senden Sie einen HTTP-GET-Request an den entsprechenden Endpoint:

```
curl -X GET "https://localhost/aaas/project/repositories" ...
```

```
[
    {
        "id": 1,
        "name": "Edited Blog",
        "description": "My edited Blog repository",
        "project": "/aaas/projects/1",
        "services": [
            "/aaas/repository/services/1"
        ]
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    },
    {
        "id": 2,
        "name": "Blog",
        "description": "Blog repository.",
        "project": "/aaas/projects/2",
        "services": [
            "/aaas/repository/services/2"
        ]
    }
]
```

Um sich alle Repositorien anzeigen zu lassen, die einem bestimmten API-Projekt zugeordnet sind, verwenden Sie einen entsprechenden Query-Parameter und übergeben den eineindeutigen Schlüssel des entsprechenden Projekts:

```
curl -X GET "https://localhost/aaas/project/repositories?project=1" ...
```

```
[
    {
        "id": 1,
        "name": "Blog",
        "description": "Blog repository."
        "project": "/aaas/projects/1",
        "services": [
        ]
    },
    {
        "id": 2,
        "name": "Shop",
        "description": "Shop repository.",
        "project": "/aaas/projects/1",
        "services": [
        ]
    }
]
```

6.4 Repositorien löschen

Sie senden einen HTTP-DELETE-Request an den entsprechenden API-Endpoint:

```
curl -X DELETE "https://localhost/aaas/project/repositories/1" ...
```

Das Repositorium wird gelöscht, wenn es sich in Ihrer Datenbank befindet.

6.5 Serialisierungsgruppen

Sie finden im Folgenden eine Übersicht der verwendeten Serialisierungsgruppen von Repositorien:

Gruppe	Definition
project	Serialisiert/Deserialisiert `project`-Attribute
service	Serialisiert/Deserialisiert `service`-Attribute

Hinweis: Wie Sie anhand der Tabelle erkennen können, können Sie Projekte über Repositorien deserialisieren. So ist es beispielsweise möglich, ein Repositorium anzulegen und gleichzeitig auch ein zugehöriges Projekt. Objekthierarchisch gesehen, befinden sich Repositorien unterhalb von Projekten. Aus Gründen der Designphilosophie empfehlen wir daher immer den umgekehrten Weg: Legen Sie Projekte samt Repositorien an, aber legen Sie keine Repositorien samt Projekte an.

6.5.1 Deseralisierung von Projekten

Folgendes würde funktionieren, wird von uns aber nicht empfohlen:

```
curl -X POST https://localhost/aaas/project/repositories?groups[]=project
```

```
{
  "name": "Blog",
  "description": "Blog repository.",
  "project" : {
    "name" : "My awesome API project"
  }
}
```

```
{
    "id": 3,
    "name": "Blog",
    "description": "Blog repository.",
    "project": {
        "id": 3,
        "name": "My awesome API project",
        "description": null,
        "repositories": [
        ]
    },
    "services": [
    ]
}
```

Sie sollten Serialisierungsgruppen benutzen, um über- und untergeordnete Objekte zu serialisieren. Deserialisieren sollten Sie immer nur untergeordnete Objekte.

6.5.2 Deseralisierung von Services

So können Sie die Gruppen benutzen, um beispielsweise Repositorien samt Services anzulegen:

```
curl -X POST https://localhost/aaas/project/repositories?groups[]=service
```

```
{
  "name": "Blog",
  "description": "Blog repository.",
  "project" : "/aaas/projects/1",
  "services": [
    {
      "name": "Post",
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
      "description": "Service holds blog articles."
    },
    {
      "name": "Author",
      "description": "Service holds blog authors."
    }
  ]
}
```

```
{
    "id": 4,
    "name": "Blog",
    "description": "Blog repository.",
    "project": "/aaas/projects/1",
    "services": [
        {
            "id": 3,
            "name": "Post",
            "description": "Service holds blog articles.",
            "type": "list",
            "repository": "/aaas/project/repositories/4",
            "fields": [
            ]
        },
        {
            "id": 4,
            "name": "Author",
            "description": "Service holds blog authors.",
            "type": "list",
            "repository": "/aaas/project/repositories/4",
            "fields": [
            ]
        }
    ]
}
```

6.5.3 Serialisierung von Projekten

Um `project`- oder `service`-Attribute zusätzlich serialisieren zu lassen, geben Sie einfach die entsprechenden Serialisierungsgruppen an:

```
curl -X GET "https://localhost/aaas/project/repositories/4?groups[]=project" ...
```

```
{
    "id": 4,
    "name": "Blog",
    "description": "Blog repository.",
    "project": {
        "id": 1,
        "name": "Web Application",
        "description": "This is my Web Application.",
        "repositories": [
            "/aaas/project/repositories/4"
        ]
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    },
    "services": [
        "/aaas/repository/services/3",
        "/aaas/repository/services/4"
    ]
}
```

6.5.4 Serialisierung von Projekten und Services

```
curl -X GET "https://localhost/aaas/project/repositories/4?groups[]=project&
↪groups[]=service" ...
```

```
{
    "id": 4,
    "name": "Blog",
    "description": "Blog repository.",
    "project": {
        "id": 1,
        "name": "Web Application",
        "description": "This is my Web Application.",
        "repositories": [
            "/aaas/project/repositories/4"
        ]
    },
    "services": [
        {
            "id": 3,
            "name": "Post",
            "description": "Service holds blog articles.",
            "type": "list",
            "repository": "/aaas/project/repositories/4",
            "fields": [
            ]
        },
        {
            "id": 4,
            "name": "Author",
            "description": "Service holds blog authors.",
            "type": "list",
            "repository": "/aaas/project/repositories/4",
            "fields": [
            ]
        }
    ]
}
```

KAPITEL 7

Services

Services formen Ihre Datenmodelle und bilden datenbankteschnisch Ihre Tabellen ab. Sie ordnen später Ihren Services Datenattribute (Felder) zu, mit denen Sie Ihren Datenmodellen Eigenschaften verleihen können. Sie können verschiedene Arten von Services erzeugen.

7.1 Servicetypen

Servicetyp	Beschreibung
list	Ihr Service repräsentiert eine Liste. (Default)
tree	Ihr Service repräsentiert eine verschachtelte Menge.

Hinweis: Wenn Sie bei der Erzeugung eines Services keinen Typ angeben, wird standardmäßig ein Datenmodell vom Typ `list` erzeugt.

In den meisten Fällen werden Sie Services vom Typ `list` erzeugen wollen um beispielsweise Produkt-, Kunden- oder Artikellisten zu erzeugen. In einigen Fällen, wie beispielsweise bei einem Kategoriebaum oder anderen baumartigen Strukturen, wollen Sie allerdings verschachtelte Mengen speichern. Für diese Fälle legen Sie einen Service vom Typ `tree` an.

Hinweis: Wir implementieren derzeit das Modell Nested Sets zur Abbildung Ihrer Baumstrukturen. In Zukunft werden eventuell noch weitere Modelle implementiert.

7.2 Services anlegen

Um einen Service anzulegen, senden Sie einen Namen und einen optionalen Beschreibungstext an den entsprechenden Endpunkt. Das dazugehörige Repositorium spezifizieren Sie über eine dereferenzierbare IRI. Außerdem können Sie

noch den Servicetyp angeben. Wenn Sie das `type`-Attribut weglassen, wird ein Service vom Typ `list` erzeugt:

```
curl -X POST "https://localhost/aaas/repository/services" ...
```

```
{
  "name": "Product",
  "description": "This holds our product list.",
  "type": "list",
  "repository": "/aaas/project/repositories/1"
}
```

Der API-Server antwortet gibt Ihnen den angelegten Service zurück.

```
{
    "id": 2,
    "name": "Product",
    "description": "This holds our product list.",
    "type": "list",
    "repository": "/aaas/project/repositories/1",
    "fields": [
    ]
}
```

7.3 Services bearbeiten

Beim Bearbeiten eines Services übergeben Sie in der URI den eineindeutigen Schlüssel und senden die entsprechenden Attribute, die Sie bearbeiten möchten:

```
curl -X PUT "https://localhost/aaas/repository/services/2" ...
```

```
{
  "description": "Updated description."
}
```

```
{
    "id": 2,
    "name": "Product",
    "description": "Updated description.",
    "type": "list",
    "repository": "/aaas/project/repositories/1",
    "fields": [
    ]
}
```

7.4 Services anzeigen

Beim Anzeigen aller Services, die einem bestimmten Repositorium zugeordnet sind, senden Sie einen HTTP-GET-Request an den betreffenden Endpunkt:

```
curl -X GET "https://localhost/aaas/repository/services?repository=1" ...
```

```
[
    {
        "id": 1,
        "name": "Pharaoh",
        "description": null,
        "type": "tree",
        "repository": "/aaas/project/repositories/1",
        "fields": [
            "/aaas/service/fields/1"
        ]
    },
    {
        "id": 2,
        "name": "Product",
        "description": "Updated description.",
        "type": "list",
        "repository": "/aaas/project/repositories/1",
        "fields": [
        ]
    }
]
```


7.5 Services löschen

Sie senden einen HTTP-DELETE-Request:

```
curl -X DELETE "https://localhost/aaas/repository/services/2"
```

Der API-Server antwortet mit dem HTTP-Statuscode 204.

7.6 Serialisierungsgruppen

Sie finden im Folgenden eine Übersicht der verwendeten Serialisierungsgruppen von Services:

Gruppe	Definition
project	Serialisiert/Deserialisiert `project`-Attribute
repository	Serialisiert/Deserialisiert `repository`-Attribute

7.6.1 Serialisierung von Repositorien

```
curl -X GET "https://localhost/aaas/repository/services/1?groups[]=repository" ...
```


```
{
    "id": 2,
    "name": "Product",
    "description": "This holds our product list.",
    "type": "list",
    "repository": {
        "id": 1,
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        "name": "Blog",
        "description": "Blog repository.",
        "project": "/aaas/projects/1",
        "services": [
            "/aaas/repository/services/2"
        ]
    },
    "fields": [
    ]
}
```

7.6.2 Serialisierung von Projekten und Repositorien

```
curl -X GET "https://localhost/aaas/repository/services/1?groups[]=repository&
↪groups[]=project" ...
```

```
{
    "id": 2,
    "name": "Product",
    "description": "This holds our product list.",
    "type": "list",
    "repository": {
        "id": 1,
        "name": "Blog",
        "description": "Blog repository.",
        "project": {
            "id": 1,
            "name": "Web Application",
            "description": "This is my Web Application.",
            "repositories": [
                "/aaas/project/repositories/1"
            ]
        },
        "services": [
            "/aaas/repository/services/2"
        ]
    },
    "fields": [
    ]
}
```

KAPITEL 8

Felder

Felder sind Datenattribute, mit denen Sie Ihren Services Eigenschaften zuordnen können. Wenn Sie eine Liste von Produkten persistieren möchten und jedes Produkt hat beispielsweise einen Namen und einen Beschreibungstext, dann legen Sie für diese beiden Datenattribute entsprechende Felder an. Beim Anlegen eines Feldes müssen Sie den entsprechenden Datentyp des Attributes angeben. Bei der Generierung Ihrer Datenbanktabellen wird diese Information weitergereicht, sodass auf Datenbankebene Ihre Tabellenspalten den richtigen Datentypen zugeordnet werden. Wir implementieren derzeit folgende Datentypen:

8.1 Datentypen

Datentyp	Beschreibung	Beispiel
string	Variabler Text.	Produktname
text	Sehr lange Zeichenkette.	Produktbeschreibung
integer	Ganzzahl.	Produktanzahl im Sortiment
decimal	Gleitkommazahl.	Produktpreis
float	Gleitkommazahl für Berechnungen.	
boolean	Boolesche Variable mit zwei möglichen Werten.	Ist Produkt auf Lager?
date	Datum.	Produkthaltbarkeit
time	Uhrzeit.	
datetime	Datum und Uhrzeit.	
relation	Beziehungen zu anderen Feldern.	Siehe Kapitel *Feldrelationen*

8.2 HTTP-Request-Body

Es reicht aus den Namen und Typ des entsprechenden Datenattributes zu spezifizieren, wenn Sie ein Feld anlegen wollen. Wenn Sie keinen Datentyp spezifizieren, dann wird standardmäßig eine variable Zeichenkette mit einer Länge von 255 Zeichen angelegt. Je nachdem welchen Datentyp Sie benutzen möchten, müssen Sie noch weitere Daten im HTTP-Request-Body mitsenden.

Schlüssel	Beschreibung	Optional
name	Der Name des Datenattributes.	Nein
dataType	Der Datentyp Ihres Feldes.	Ja
description	Beschreibungstext.	Ja
dataTypePrecision	Genauigkeit. Nur für `decimal`-Typ.	Ja
dataTypeScale	Anzahl Nachkommastellen. Nur für `decimal`-Typ.	Ja
length	Maximale Länge. Nur für `string`-Typ.	Ja
isUnique	Wert darf in der Spalte nur einmal vorkommen.	Ja
isNullable	Wert darf auf `null` gesetzt werden.	Ja

8.3 Felder anlegen

Sie schicken valides JSON an den entsprechenden Endpunkt und spezifizieren den dazugehörigen Service:

```
curl -X POST "https://localhost/aaas/service/fields" ...
```

```
{
  "name" : "color",
  "dataType" : "string",
  "length" : 30,
  "service" : "/aaas/repository/services/1"
}
```

Der Server gibt Ihnen das angelegte Feld zurück:

```
{
    "id": 1,
    "name": "color",
    "description": null,
    "dataType": "string",
    "dataTypePrecision": null,
    "dataTypeScale": null,
    "length": 30,
    "isUnique": false,
    "isNullable": false,
    "service": "/aaas/repository/services/1",
    "options": [
    ],
    "constraints": [
    ],
    "relation": null
}
```

```
curl -X POST "https://localhost/aaas/service/fields" ...
```

```
{
  "name" : "price",
  "dataType" : "decimal",
  "dataTypePrecision" : 10,
  "dataTypeScale" : 2,
  "service" : "/aaas/repository/services/1"
}
```

```
{
    "id": 2,
    "name": "price",
    "description": null,
    "dataType": "decimal",
    "dataTypePrecision": 10,
    "dataTypeScale": 2,
    "length": null,
    "isUnique": false,
    "isNullable": false,
    "service": "/aaas/repository/services/1",
    "options": [
    ],
    "constraints": [
    ],
    "relation": null
}
```


```
curl -X POST "https://localhost/aaas/service/fields" ...
```


```
{
  "name" : "email",
  "dataType" : "string",
  "length" : 255,
  "isUnique" : true,
  "service" : "/aaas/repository/services/2"
}
```



```
{
    "id": 3,
    "name": "email",
    "description": null,
    "dataType": "string",
    "dataTypePrecision": null,
    "dataTypeScale": null,
    "length": 255,
    "isUnique": true,
    "isNullable": false,
    "service": "/aaas/repository/services/2",
    "options": [
    ],
    "constraints": [
    ],
    "relation": null
}
```


8.4 Felder bearbeiten

Sie senden die Schlüssel, die Sie bearbeiten möchten an den entsprechenden Endpunkt:

```
curl -X PUT "https://localhost/aaas/service/fields/3" ...
```

```
{
  "length" : 100
}
```


```
{
    "id": 3,
    "name": "email",
    "description": null,
    "dataType": "string",
    "dataTypePrecision": null,
    "dataTypeScale": null,
    "length": 100,
    "isUnique": true,
    "isNullable": false,
    "service": "/aaas/repository/services/1",
    "options": [
    ],
    "constraints": [
    ],
    "relation": null
}
```


8.5 Felder anzeigen

Sie können sich alle Felder anzeigen lassen, die einem bestimmten Service zugeordnet sind, indem Sie einen entsprechenden HTTP-Query-Parameter verwenden:

```
curl -X GET "https://localhost/aaas/service/fields?service=1" ...
```


```
[
    {
        "id": 1,
        "name": "color",
        "description": null,
        "dataType": "string",
        "length": 30,
        "dataTypePrecision": null,
        "dataTypeScale": null,
        "isUnique": false,
        "isNullable": false,
        "service": "/aaas/repository/services/1",
        "fieldOptions": [
        ],
        "constraints": [
        ],
        "relation": null
    },
    {
        "id": 2,
        "name": "price",
        "description": null,
        "dataType": "decimal",
        "length": null,
        "dataTypePrecision": 10,
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        "dataTypeScale": 2,
        "isUnique": false,
        "isNullable": false,
        "service": "/aaas/repository/services/1",
        "fieldOptions": [
        ],
        "constraints": [
        ],
        "relation": null
    }
]
```

8.6 Felder löschen

Sie senden einen HTTP-DELETE-Request an den entsprechenden Endpunkt:

```
curl -X DELETE "https://localhost/aaas/service/fields/18"
```

Wenn das Feld gelöscht wurde, antwortet der API-Server mit dem HTTP-Statuscode 204.

8.7 Serialisierungsgruppen

Sie finden im Folgenden eine Übersicht der verwendeten Serialisierungsgruppen von Feldern:

Gruppe	Definition
relation	Serialisiert/Deserialisiert `relation`-Attribute
constraint	Serialisiert/Deserialisiert `constraint`-Attribute
constraintOption	Serialisiert/Deserialisiert `constraintOption`-Attribute
fieldOption	Serialisiert/Deserialisiert `fieldOption`-Attribute

8.7.1 Deserialiserung von Feldoptionen

So können Sie Felder und Feldoptionen in einem Schritt anlegen. Beim Generieren Ihrer Datenbankstruktur, werden Feldoptionen an die sich unter Ihrer Applikation befindliche Datenbank weitergereicht:

```
curl -X POST "https://localhost/aaas/service/fields?groups[]=fieldOption" ...
```

```
{
  "name" : "sku",
  "dataType" : "string",
  "length" : "14"
  "service" : "/aaas/repository/services/1",
  "fieldOptions": [
    {
      "name" : "default",
      "value" : "PRD-123456789"
    }
  ]
}
```

```
{
    "id": 7,
    "name": "sku",
    "description": null,
    "dataType": "integer",
    "length": null,
    "dataTypePrecision": null,
    "dataTypeScale": null,
    "isUnique": false,
    "isNullable": false,
    "service": "/aaas/repository/services/1",
    "fieldOptions": [
        {
            "id": 2,
            "name": "default",
            "value": "PRD-123456789",
            "field": "/aaas/service/fields/7"
        }
    ],
    "constraints": [
    ],
    "relation": null
}
```


Sie erfahren mehr zu Feldoptionen auf den nächsten Seiten.

8.7.2 Deseralisierung von Feldvalidierungen

```
curl -X POST "https://localhost/aaas/service/fields?groups[]=constraint&
↪groups[]=constraintOption" ...
```


```
{
  "name" : "title",
  "dataType" : "string",
  "service" : "/aaas/repository/services/2",
  "constraints" : [
    {
      "name": "NotBlank",
      "constraintOptions": [
        {
          "name": "message",
          "value": "This value should be not blank."
        }
      ]
    }
  ]
}
```



```
{
    "id": 4,
    "name": "title",
    "description": null,
    "dataType": "string",
    "length": null,
    "dataTypePrecision": null,
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
    "dataTypeScale": null,
    "isUnique": false,
    "isNullable": false,
    "service": "/aaas/repository/services/2",
    "fieldOptions": [
    ],
    "constraints": [
        {
            "id": 2,
            "name": "NotBlank",
            "field": "/aaas/service/fields/19",
            "constraintOptions": [
                {
                    "id": 1,
                    "name": "message",
                    "value": "This value should be not blank.",
                    "constraint": "/aaas/field/constraints/2"
                }
            ]
        }
    ],
    "relation": null
}
```


Sie erfahren mehr zu Validierungsbeschränkungen auf den nächsten Seiten.

8.7.3 Deseralisierung von Feldrelationen

Mithilfe der `relation`-Serialisierungsgruppe können Sie auch Felder anlegen, die ihrerseits als Relation für andere Felder dienen:

```
curl -X POST "https://localhost/aaas/service/fields?groups[]=relation" ...
```


```
{
  "name": "author",
  "dataType": "relation",
  "service": "/aaas/repository/services/1",
  "relation": {
    "service": "/aaas/repository/services/2",
    "type": "ManyToOne"
  }
}
```



```
{
    "id": 4,
    "name": "author",
    "description": null,
    "dataType": "relation",
    "length": null,
    "dataTypePrecision": null,
    "dataTypeScale": null,
    "isUnique": false,
    "isNullable": false,
    "service": "/aaas/repository/services/1",
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
    "fieldOptions": [
    ],
    "constraints": [
    ],
    "relation": {
        "id": 1,
        "type": "ManyToOne",
        "mappedBy": null,
        "inversedBy": null,
        "orphanRemoval": false,
        "joinColumnName": null,
        "joinColumnReferencedColumnName": "id",
        "joinColumnIsUnique": false,
        "joinColumnIsNullable": true,
        "field": "/aaas/service/fields/4",
        "service": "/aaas/repository/services/2"
    }
}
```


Sie erfahren mehr zu Relationen auf den nächsten Seiten.

KAPITEL 9

Feldoptionen

Sie können Ihren Feldern eine bestimmte Anzahl an Optionen zuordnen. Diese Optionen werden beim Generieren Ihrer Datenbanktabellen den entsprechenden Spalten zugewiesen. Wir implementieren derzeit folgende Optionen:

Option	Beschreibung
unsigned	Vorzeichenbehafteter Datentyp. Nur für Ganzzahlen.
default	Standardwert.
comment	Spaltenkommentar.

9.1 Feldoptionen anlegen

Beim Anlegen einer Option übergeben Sie den Namen und den Wert der entsprechenden Option:

```
curl -X POST "https://localhost/aaas/field/options" ...
```

```
{
  "name": "default",
  "value" : "Fancy product",
  "field" : "/aaas/service/fields/6"
}
```

Der Server antwortet mit der angelegten Option:

```
{
    "id": 4,
    "name": "default",
    "value": "Fancy product",
    "field": "/aaas/service/fields/6"
}
```

9.2 Feldoptionen bearbeiten

Sie senden einen HTTP-PUT-Request an den entsprechenden Endpunkt:

```
curl -X PUT "https://localhost/aaas/field/options/4" ...
```

```
{
  "value" : "Fancy category"
}
```

```
{
    "id": 4,
    "name": "default",
    "value": "Fancy category",
    "field": "/aaas/service/fields/6"
}
```

9.3 Feldoptionen anzeigen

Sie können sich alle Feldoptionen anzeigen lassen, die einem bestimmten Feld zugeordnet sind, indem Sie einen entsprechenden HTTP-Query-Parameter verwenden:

```
curl -X GET "https://localhost/aaas/field/options?field=1" ...
```

```
[
    {
        "id": 3,
        "name": "unsigned",
        "value": "false",
        "field": "/aaas/service/fields/1"
    }
]
```

9.4 Feldoptionen löschen

Sie senden einen HTTP-DELETE-Request an den entsprechenden Endpunkt:

```
curl -X DELETE "https://localhost/aaas/field/options/4"
```

Der API-Server antwortet mit dem HTTP-Statuscode 204.

9.5 Serialisierungsgruppen

Sie finden im Folgenden eine Übersicht der verwendeten Serialisierungsgruppen von Feldoptionen:

Gruppe	Definition
field	Serialisiert/Deserialisiert `field`-Attribute

9.5.1 Serialiserung von Feldern

Um zusätzlich zu den Feldoptionen die zugehörigen Felder serialisieren zu lassen, senden Sie einen entsprechenden HTTP-Query-Paramater und setzen ihn auf `field`:

```
curl -X DELETE https://localhost/aaas/field/options?field=1&groups[]=field ...
```


```
[
    {
        "id": 3,
        "name": "unsigned",
        "value": "false",
        "field": {
            "id": 1,
            "name": "email",
            "description": null,
            "dataType": "string",
            "length": 200,
            "dataTypePrecision": null,
            "dataTypeScale": null,
            "isUnique": true,
            "isNullable": false,
            "service": "/aaas/repository/services/2",
            "fieldOptions": [
                "/aaas/field/options/3"
            ],
            "constraints": [
            ],
            "relation": null
        }
    }
]
```

KAPITEL 10

Feldvalidierungen

Wenn Sie Ihre konstruierten Services Ihren API-Konusumenten zur Verfügung stellen und dabei Eingabewerte restriktieren möchten, dann können Sie die von *AaaS-API* zur Verfügung gestellten Validierungsoptionen benutzen.

Wir bieten Ihnen eine ganze Reihe an möglichen Validierungsbeschränkungen an und benutzen dabei intern *Symfony Validation Constraints*. Wenn Sie sich für Datenvalidierung und deren Konfiguration interessieren und erfahren möchten, welche Möglichkeiten wir Ihnen zur Verfügung stellen, dann konsultieren Sie bitte die entsprechende Seite in der Symfony-Dokumentation oder schauen Sie sich die `VALID_CONSTRAINTS`-Konstante in Constraint.php in unserem Github-Repositorium an.

Auf dieser Seite geben wir Ihnen einen Überblick, wie Sie Feldeingaben beschränken können.

10.1 Feldvalidierungen anlegen

Eine Feldvalidierung in *AaaS-API* besteht immer aus dem Namen des Validators und einer Reihe an möglichen Konfigurationsoptionen. Die Optionen unterscheiden sich je nach verwendetem Validator. Im folgenden belegen wir ein `String`-Feld mit einem `NotBlank`-Validator, welcher dafür sorgt, dass Ihre API-Konsumenten später einen Wert für das entsprechende Feld mitsenden müssen:

```
curl -X POST "https://localhost/aaas/field/constraints" ...
```

```
{
  "name": "NotBlank",
  "field": "/aaas/service/fields/1"
}
```

Der Server antwortet mit dem angelegten Validator:

```
{
    "id": 1,
    "name": "NotBlank",
    "field": "/aaas/service/fields/1",
    "constraintOptions": [
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    ]
}
```

Wenn Sie später Ihre API generieren lassen und Ihre Konsumenten senden für das entsprechende Feld keinen Wert mit, dann antwortet der API-Server mit dem HTTP-Status-Code 400 (Bad Request):

```
curl -X POST "https://localhost/aaas/api/blog/authors" -H "accept: application/json" -
↪H "Content-Type: application/ld+json" -d "{ \"name\": \"\"}"
```

Beachten Sie bitte im Curl-Aufruf, dass der Wert für `name` leer ist. Der API-Server antwortet dann mit einem entsprechenden clientseitige Fehler:

```
{
  "type": "https://tools.ietf.org/html/rfc2616#section-10",
  "title": "An error occurred",
  "detail": "name: This value should not be blank.",
  "violations": [
    {
      "propertyPath": "name",
      "message": "This value should not be blank."
    }
  ]
}
```

10.2 Validierungsoptionen anlegen

Sie senden einen HTTP-POST-Request an den entsprechenden Endpunkt:

```
curl -X POST "https://localhost/aaas/field/constraint/options" ...
```

```
{
  "name": "message",
  "value": "This value should be not blank.",
  "constraint": "/aaas/field/constraints/1"
}
```

Der API-Server antwortet mit der angelegten Ressource:

```
{
    "id": 1,
    "name": "message",
    "value": "This value should be not blank.",
    "constraint": "/aaas/field/constraints/1"
}
```

Als Alternative dazu können Sie auch Serialisierungsgruppen benutzen, um Feldvalidierungen und deren Konfigurationsoptionen in einem Schritt anzulegen:

```
curl -X POST "https://localhost/aaas/field/constraints?groups[]=constraintOption" ...
```

```
{
  "name": "NotBlank",
  "field": "/aaas/service/fields/1",
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
  "constraintOptions": [
    {
      "name": "message",
      "value": "This value should be not blank."
    }
  ]
}
```

```
{
    "id": 2,
    "name": "NotBlank",
    "field": "/aaas/service/fields/1",
    "constraintOptions": [
        {
            "id": 2,
            "name": "message",
            "value": "This value should be not blank.",
            "constraint": "/aaas/field/constraints/2"
        }
    ]
}
```

10.3 Feldvalidierungen bearbeiten

Sie senden einen HTTP-PUT-Request an den entsprechenden Endpunkt:

```
curl -X PUT "https://localhost/aaas/field/constraints/1" ...
```

```
{
  "name": "Email"
}
```

```
{
    "id": 1,
    "name": "Email",
    "field": "/aaas/service/fields/1",
    "constraintOptions": [
        "/aaas/field/constraint/options/1"
    ]
}
```

Im obigen Beispiel wird der Eingabewert für das entsprechende Feld auf den `Email`-Validator geändert und damit auf valide E-Mail-Adressen beschränkt.

10.4 Feldvalidierungen löschen

Sie senden einen HTTP-DELETE-Request an den entsprechenden Endpunkt:

```
curl -X DELETE "https://localhost/aaas/field/constraints/1"
```

Der API-Server antwortet mit dem HTTP-Statuscode 204.

10.5 Feldvalidierungen anzeigen

Um sich alle Validatoren anzeigen zu lassen, die einem bestimmten Servicefeld zugeordnet sind, können Sie einen entsprechenden HTTP-Query-Parameter benutzen. Beachten Sie bitte, dass wir die `constraintOption`-Serialisierungsgruppe benutzen, um zugehörige Konfigurationsoptionen ergänzend serialisieren zu lassen:

```
curl -X GET "https://localhost/aaas/field/constraints?field=1&
↪groups[]=constraintOption"
```


```
[
    {
        "id": 2,
        "name": "NotBlank",
        "field": "/aaas/service/fields/1",
        "constraintOptions": [
            {
                "id": 2,
                "name": "message",
                "value": "This value should be not blank.",
                "constraint": "/aaas/field/constraints/2"
            }
        ]
    },
    {
        "id": 3,
        "name": "GreaterThan",
        "field": "/aaas/service/fields/1",
        "constraintOptions": [
            {
                "id": 3,
                "name": "message",
                "value": "This value must be greater than 10.",
                "constraint": "/aaas/field/constraints/3"
            },
            {
                "id": 4,
                "name": "value",
                "value": "10",
                "constraint": "/aaas/field/constraints/3"
            }
        ]
    }
]
```


10.6 Serialisierungsgruppen

Sie finden im Folgenden eine Übersicht der verwendeten Serialisierungsgruppen von Feldvalidierungen:

Gruppe	Definition
field	Serialisiert/Deserialisiert `field`-Attribute
constraintOption	Serialisiert/Deserialisiert `constraintOption`-Attribute

10.6.1 Serialiserung von Feldern

Um zusätzlich zu den Feldvalidierungen die zugehörigen Felder serialisieren zu lassen, senden Sie einen entsprechenden HTTP-Query-Paramater und setzen ihn auf `field`:

```
curl -X GET https://localhost/aaas/field/constraints?groups[]=field ...
```


```
[
    {
        "id": 1,
        "name": "NotBlank",
        "field": {
            "id": 1,
            "name": "name",
            "description": null,
            "dataType": "string",
            "length": 100,
            "dataTypePrecision": null,
            "dataTypeScale": null,
            "isUnique": false,
            "isNullable": false,
            "service": "/aaas/repository/services/2",
            "fieldOptions": [
            ],
            "constraints": [
                "/aaas/field/constraints/1"
            ],
            "relation": null
        },
        "constraintOptions": [
            "/aaas/field/constraint/options/1"
        ]
    }
]
```


10.6.2 Serialiserung von Validierungsoptionen

Sie setzen den HTTP-Query-Parameter für Serialisierungsgruppen auf `constraintOption`:

```
curl -X GET https://localhost/aaas/field/constraints?groups[]=constraintOption ...
```


```
[
    {
        "id": 1,
        "name": "NotBlank",
        "field": "/aaas/service/fields/1",
        "constraintOptions": [
            {
                "id": 1,
                "name": "message",
                "value": "This value should be not blank.",
                "constraint": "/aaas/field/constraints/1"
            }
        ]
    }
]
```

10.6.3 Deserialiserung von Validierungsoptionen

Sie setzen den HTTP-Query-Parameter für Serialisierungsgruppen auf `constraintOption` und senden valide Objekattribute:

```
curl -X POST https://localhost/aaas/field/constraints?groups[]=constraintOption ...
```

```
{
  "name": "Isbn",
  "field": "/aaas/service/fields/1",
  "constraintOptions": [
    {
      "name": "type",
      "value": "isbn10"
    },
    {
      "name": "message",
      "value": "This is not a valid isbn10 book number."
    }
  ]
}
```

```
{
    "id": 2,
    "name": "Isbn",
    "field": "/aaas/service/fields/1",
    "constraintOptions": [
        {
            "id": 2,
            "name": "type",
            "value": "isbn10",
            "constraint": "/aaas/field/constraints/2"
        },
        {
            "id": 3,
            "name": "message",
            "value": "This is not a valid isbn10 book number.",
            "constraint": "/aaas/field/constraints/2"
        }
    ]
}
```

KAPITEL 11

Feldrelationen

Über die von uns zur Verfügung gestellten Feldrelationen können Sie Beziehungen unter Ihren Services abbilden. Wenn Sie eine Anzahl an Produkten persistieren möchten und jedes Produkt ist dabei einer Kategorie zugeordnet, dann macht es Sinn, das Datenmodell für Kategorien vom Datenmodell für Produkte zu separieren. Sie hätten dann sowohl einen Service für Produkte, als auch für Kategorien. Die Verknüpfung der Services untereinander können Sie mit Feldrelationen realisieren.

11.1 Relationstypen

Es existieren vier unterschiedliche Relationstypen:

Typ	Definition	Beispiel
OneToOne	Ein Datensatz aus Ihrem Service ist genau einem Datensatz aus einem anderen Service zugeordnet.	• Kontaktdaten einer Person • Produktdetails eines Produkts
OneToMany / ManyToOne	Ein Datensatz aus Ihrem Service ist mehreren Datensätzen aus einem anderen Service zugeordnet.	• Bestellvorgänge eines Kunden • Produkte in einer Kategorie
ManyToMany	Mehrere Datensätze aus Ihrem Service sind mehreren Datensätzen aus einem anderen Service zugeordnet.	• Produkte in mehreren Kategorien

Hinweis: Die Generierung Ihrer API bricht momentan vorzeitig ab, wenn Sie an einer Stelle eine `OneToMany`-Relation definieren. Wir rüsten die Funktionalität später nach. Wenn Sie eine `OneToMany`-Beziehung implementieren möchten, dann gehen Sie vorläufig bitte den umgekehrten Weg und implementieren am inversen Service eine `ManyToOne`-Relation. Die Funktionalität Ihrer API ist später dieselbe.

11.2 Feldrelationen anlegen

Wenn Sie ein Feld vom Datentyp `relation` angelegt haben, dann können Sie diesem Feld eine entsprechende Beziehung zuordnen. Sie spezifizieren dabei lediglich den Relationstyp und den Servive, zu dem Sie die Beziehung herstellen möchten.

Sie könnten beispielsweise dem Service `Article` ein `author`-Feld zuordnen, in welchem der Autor des entsprechenden Artikels persistiert wird. Das `author`-Feld wäre dann eine Relation zu Ihrem `Author`-Service, in dem Sie eine Reihe an Autoren vorrätig halten:

```
curl -X POST "https://localhost/aaas/service/fields" ...
```

```
{
  "name": "author",
  "dataType": "relation",
  "service": "/aaas/repository/services/1"
}
```

Im obigen Beispiel legen Sie ein Feld vom Typ `relation` an und ordnen es dem entsprechenden Service zu. Anschließend legen Sie die Details zugehörigen Relation fest:

```
curl -X POST "https://localhost/aaas/field/relations" ...
```

```
{
  "type": "ManyToOne",
  "field": "/aaas/service/fields/1",
  "service": "/aaas/repository/services/2"
}
```

Der Server antwortet mit der angelegten Ressource:

```
{
    "id": 1,
    "type": "ManyToOne",
    "mappedBy": null,
    "inversedBy": null,
    "orphanRemoval": false,
    "joinColumnName": null,
    "joinColumnReferencedColumnName": "id",
    "joinColumnIsUnique": false,
    "joinColumnIsNullable": true,
    "field": "/aaas/service/fields/1",
    "service": "/aaas/repository/services/2"
}
```

Wenn Sie später Ihre API generieren lassen, dann erhält auf Datenbankebene die `Api_Article`-Tabelle den Fremdschlüssel `author_id`.

Sie können natürlich auch Felder und gleichzeitig Relationen anlegen. Benutzen Sie dafür die Serialisierungsgruppe `relation`:

```
curl -X POST  "https://localhost/aaas/service/fields?groups[]=relation" ...
```

```
{
  "name": "author",
  "dataType": "relation",
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
  "service": "/aaas/repository/services/1",
  "relation": {
    "service": "/aaas/repository/services/2",
    "type": "ManyToOne"
  }
}
```


```
{
    "id": 1,
    "name": "author",
    "description": null,
    "dataType": "relation",
    "length": null,
    "dataTypePrecision": null,
    "dataTypeScale": null,
    "isUnique": false,
    "isNullable": false,
    "service": "/aaas/repository/services/1",
    "fieldOptions": [
    ],
    "constraints": [
    ],
    "relation": {
        "id": 1,
        "type": "ManyToOne",
        "mappedBy": null,
        "inversedBy": null,
        "orphanRemoval": false,
        "joinColumnName": null,
        "joinColumnReferencedColumnName": "id",
        "joinColumnIsUnique": false,
        "joinColumnIsNullable": true,
        "field": "/aaas/service/fields/1",
        "service": "/aaas/repository/services/2"
    }
}
```


Wenn Sie keinen Relationstypen angeben, dann wird standardmäßig eine Relation vom Typ `ManyToOne` erzeugt.

11.3 Feldrelationen bearbeiten

Obschon die Bearbeitung von Feldrelationen wahrscheinlich selten vorkommt, geben wir Ihnen an der Stelle trotzdem einen entsprechenden Überblick. Sie senden dafür einen HTTP-Put-Request an den korrespondierenden Endpunkt:

```
curl -X PUT  "https://localhost/aaas/field/relations/1" ...
```

```
{
  "type": "ManyToMany"
}
```

Sie ändern im obigen Beispiel den Relationstyp auf `ManyToMany`.

11.4 Feldrelationen löschen

Beim Löschen von Feldrelationen senden Sie einen HTTP-Delete-Request an den entsprechenden Endpunkt:

```
curl -X DELETE  "https://localhost/aaas/field/relations/1" ...
```

Der API-Server antwortet mit dem HTTP-Statuscode 204 (No Content).

11.5 Feldrelationen anzeigen

Um sich alle Relationen anzeigen zu lassen, die einem bestimmten Service zugeordnet sind, senden Sie einen HTTP-Get-Request an den entsprechenden Endpunkt und setzen den HTTP-Query-Parameter `service` auf den zugehörigen Wert:

```
curl -X GET  "https://localhost/aaas/field/relations?service=2" ...
```

```
[
    {
        "id": 1,
        "type": "ManyToOne",
        "mappedBy": null,
        "inversedBy": null,
        "orphanRemoval": false,
        "joinColumnName": null,
        "joinColumnReferencedColumnName": "id",
        "joinColumnIsUnique": false,
        "joinColumnIsNullable": true,
        "field": "/aaas/service/fields/1",
        "service": "/aaas/repository/services/2"
    }
]
```

11.6 Serialisierungsgruppen

Es folgt eine Übersicht der verwendeten Serialisierungsgruppen bei Feldrelationen:

Gruppe	Definition
field	Serialisiert/Deserialisiert `field`-Attribute

11.6.1 Serialisierung von Feldern

Wenn Sie zusätzlich zu den Relationen noch die entsprechenden Felder serialisieren lassen möchten, dann senden Sie als HTTP-Query-Parameter die entsprechende Serialisierungsgruppe und setzen den Wert auf `field`:

```
curl -X GET  "https://localhost/aaas/field/relations/1?groups[]=field" ...
```

```
{
    "id": 1,
    "type": "ManyToOne",
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
    "mappedBy": null,
    "inversedBy": null,
    "orphanRemoval": false,
    "joinColumnName": null,
    "joinColumnReferencedColumnName": "id",
    "joinColumnIsUnique": false,
    "joinColumnIsNullable": true,
    "field": {
        "id": 1,
        "name": "author",
        "description": null,
        "dataType": "relation",
        "length": null,
        "dataTypePrecision": null,
        "dataTypeScale": null,
        "isUnique": false,
        "isNullable": false,
        "service": "/aaas/repository/services/1",
        "fieldOptions": [
        ],
        "constraints": [
        ],
        "relation": "/aaas/field/relations/1"
    },
    "service": "/aaas/repository/services/2"
}
```

KAPITEL 12

Servicefilter

Mit den von uns zur Verfügung gestellten Servicefiltern können Sie Ihre API-Konsumenten in die Lage versetzen, Antwort-Kollektionen Ihrer API nach bestimmten Kriterien zu filtern. Bei einem Autoren-Service wäre es so beispielsweise möglich, sich alle Autoren ausgeben zu lassen, die mit Vornamen `Marcel` heißen.

12.1 Filtertypen

Wir implementieren eine ganze Reihe an verschiedenen Filtertypen:

Typ	**Definition**
string	Ermöglicht das Filtern nach Zeichenketten.
date	Ermöglicht das Filtern nach Datumsformaten.
boolean	Ermöglicht das Filtern nach `true`- und `false`-Werten
numeric	Ermöglicht das Filtern nach numerischen Angaben.
range	Ermöglicht das Filtern nach Wertebereichen.
exists	Ermöglicht das Filtern nach ungleich `null`-Werten.
order	Ermöglicht auf- und absteigende Sortierung nach entsprechenden Kriterien.

12.2 Anlegen von Servicefiltern

Wenn Sie einen Servicefilter anlegen möchten, dann spezifizieren Sie den entsprechenden Filtertyp und geben eine Reihe an Servicefeldern (Filtereigenschaften) an, nach denen gefiltert werden soll. Wir geben Ihnen im Folgenden je Filter ein Beispiel zur Erstellung und ein Beispiel zur späteren Verwendung.

Hinweis: Beachten Sie bitte, dass wir die Filtereigenschaften datenbanktechnisch von den jeweiligen Filtern separiert haben. Der einfachheitshalber benutzen wir daher in den folgenden Beispielen die `filterProperty`-Serialisierungsgruppe und sparen uns damit das Absetzen eines zusätzlichen HTTP-Requests.

12.2.1 Stringfilter

Der Stringfilter ermöglicht das Filtern nach bestimmten Zeichenketten. Die zu filternden Zeichenketten können Sie dabei auf unterschiedliche Art und Weise definieren. Man spricht dabei von sog. *Matching Strategies*.

Strategie	Definition
exact	Der Feldwert muss exakt mit der Zeichenkette übereinstimmen. `LIKE text`
partial	Im Feldwert muss die Zeichenkette vorkommen. `LIKE %text%`
start	Der Feldwert startet mit der Zeichenkette. `LIKE text%`
end	Der Feldwert endet mit der Zeichenkette. `LIKE %text`
word_start	Sucht nach Wörtern, die mit der Zeichenkette beginnen. `LIKE text% OR LIKE % text%`

Setzen Sie bitte noch jeweils ein `i` vor die jeweilige Strategie, wenn Sie Unempfindlichkeit gegenüber Groß- und Kleinschreibung herstellen möchten. Beachten Sie aber bitte, dass Kollationen für unsere Datenbanktabellen standardmäßig auf `utf8mb4_unicode_ci` gesetzt sind, welche bereits Unempflichkeit gegenüber Groß- und Kleinschreibung implementieren (_ci = case-insensitive).

Sie senden einen HTTP-POST-Request an den entsprechenden Endpunkt:

```
curl -X POST "https://localhost/aaas/service/filters?groups[]=filterProperty" ...
```

```
{
  "type": "string",
  "service": "/aaas/repository/services/3",
  "properties": [
    {
      "field": "/aaas/service/fields/1",
      "value": "start"
    }
  ]
}
```

Im obigen Beispiel legen wir einen `String`-Filter für den entsprechenden Service an und belegen dabei ein Feld mit der `start`-Match-Strategie.

Beispiel zur Verwendung (Alle Produkte, deren Name mit „kern“ anfangen):

```
curl -X GET "https://localhost/api/shop/products?name=kern" ...
```

```
[
    {
        "id": 6,
        "name": "Kernbohrer",
        "price": "250.00",
        "isInStock": true,
        "createdAt": "2020-08-21T00:00:00+02:00",
        "transportFees": null
    }
]
```

12.2.2 Datefilter

Der Datefilter erlaubt das Filtern nach Datumsformaten. Wenn Sie ein Servicefeld vom Datentyp `date`, `time` oder `datetime` definiert haben, dann können Sie mithilfe des Datefilters beispielsweise alle Datensätze abfragen, deren Feldwert nach oder vor einem bestimmten Datum liegt. Selbstverständlich können Abfragen auch kombiniert werden.

Beim Abfragen können Sie später folgende HTTP-Query-Parameter setzen:

Parameter	Definition
after	Feldwert liegt nach dem Datum (Inkl. Datum). `?field[after]=date`
before	Feldwert liegt vor dem Datum (Inkl. Datum). `?field[before]=date`
strictly_after	Feldwert liegt nach dem Datum (Exkl. Datum). `?field[strictly_after]=date`
strictly_before	Feldwert liegt vor dem Datum (Exkl. Datum). `?field[strictly_before]=date`

Wenn das Servicefeld `null` beinhalten kann, dann können Sie entsprechende Datensätze inkludieren oder exkludieren. Setzen Sie dafür beim Anlegen der entsprechenden `filterProperty` den Wert für `value` einfach auf einen der folgenden Werte:

Value	Definition
exclude_null	`null`-Werte werden exkludiert.
include_null_before	Betrachtet Datensätze als Älteste.
include_null_after	Betrachtet Datensätze als Jüngste.
include_null_before_and_after	`null`-Werte werden inkludiert.

Sie senden einen HTTP-POST-Request an den entsprechenden Endpunkt:

```
curl -X POST "https://localhost/aaas/service/filters?groups[]=filterProperty" ...
```

```
{
  "type": "date",
  "service": "/aaas/repository/services/3",
  "properties": [
    {
      "field": "/aaas/service/fields/4"
    }
  ]
}
```

Beispiel zur Verwendung (Alle Produkte, die vor dem 15.08.2020 erstellt wurde):

```
curl -X GET "https://localhost/api/shop/products?createdAt[before]=2020-08-15" ...
```

```
[
    {
        "id": 1,
        "name": "Tieflochbohrer",
        "price": "199.00",
        "isInStock": true,
        "createdAt": "2020-08-11T00:00:00+02:00",
        "transportFees": null
    },
    {
        "id": 2,
        "name": "Spiralbohrer",
        "price": "99.99",
        "isInStock": true,
        "createdAt": "2020-08-07T00:00:00+02:00",
        "transportFees": "10"
    }
]
```

Beispiel zur Verwendung (Alle Produkte, die nach dem 20.08.2020 und einschließlich vor dem 01.09.2020 erstellt wurde):

```
curl -X GET "https://localhost/api/shop/products?createdAt[after]=2020-08-20&
↪createdAt[before_strictly]=2020-09-01" ...
```


```
[
    {
        "id": 3,
        "name": "Aufbohrer",
        "price": "40.00",
        "isInStock": false,
        "createdAt": "2020-08-20T00:00:00+02:00",
        "transportFees": null
    },
    {
        "id": 5,
        "name": "Festmaß-Aufbohrer",
        "price": "20.00",
        "isInStock": true,
        "createdAt": "2020-09-01T00:00:00+02:00",
        "transportFees": "35"
    },
    {
        "id": 6,
        "name": "Kernbohrer",
        "price": "250.00",
        "isInStock": true,
        "createdAt": "2020-08-21T00:00:00+02:00",
        "transportFees": null
    }
]
```


12.2.3 Booleanfilter

Der Booleanfilter ermöglicht das Filtern nach `true`- oder `false`-Werten. Wenn Sie ein Servicefeld vom Typ `boolean` definiert haben, dann können Sie später alle Datensätze abfragen, für die der entsprechende Feldwert auf `true` oder `false` gesetzt ist:

```
curl -X POST "https://localhost/aaas/service/filters?groups[]=filterProperty" ...
```


```
{
  "type": "boolean",
  "service": "/aaas/repository/services/3",
  "properties": [
    {
      "field": "/aaas/service/fields/3"
    }
  ]
}
```


Beispiel zur Verwendung (Alle Produkte, für die `isInStock` auf `false` gesetzt ist):

```
curl -X GET "https://localhost/api/shop/products?isInStock=false" ...
```

```
[
    {
        "id": 3,
        "name": "Aufbohrer",
        "price": "40.00",
        "isInStock": false,
        "createdAt": "2020-08-20T00:00:00+02:00",
        "transportFees": null
    },
    {
        "id": 4,
        "name": "VHM-Stufenbohrer",
        "price": "20.00",
        "isInStock": false,
        "createdAt": "2020-08-18T00:00:00+02:00",
        "transportFees": null
    }
]
```


12.2.4 Numericfilter

Der Numericfilter ermöglicht das Filtern nach Werten auf numerischen Feldern:

```
curl -X POST "https://localhost/aaas/service/filters?groups[]=filterProperty" ...
```


```
{
  "type": "numeric",
  "service": "/aaas/repository/services/3",
  "properties": [
    {
      "field": "/aaas/service/fields/2"
    }
  ]
}
```


Beispiel zur Verwendung (Alle Produkte, deren Produktpreis `20` beträgt):

```
curl -X GET "https://localhost/api/shop/products?price=20" ...
```


```
[
    {
        "id": 4,
        "name": "VHM-Stufenbohrer",
        "price": "20.00",
        "isInStock": false,
        "createdAt": "2020-08-18T00:00:00+02:00",
        "transportFees": null
    },
    {
        "id": 5,
        "name": "Festmaß-Aufbohrer",
        "price": "20.00",
        "isInStock": true,
        "createdAt": "2020-09-01T00:00:00+02:00",
        "transportFees": "35"
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    }
]
```

12.2.5 Rangefilter

Der Rangefilter ermöglicht das Filtern nach Wertebereichen auf numerischen Feldern. Wenn Sie in einem Servicefeld beispielsweise einen Produktpreis persistieren, dann können Sie mithilfe des Rangefilters alle Produkte abfragen, deren Preis in einem bestimmten Bereich liegt.

Bei der späteren Verwendung können Sie einen der folgenden Parameter setzen. Sie können Parameter auch kombinieren.

Parameter	Definition
lt	Feldwert ist kleiner als Wert. `?field[lt]=value`
gt	Feldwert ist größer als Wert. `?field[gt]=value`
lte	Feldwert ist kleiner gleich Wert. `?field[lte]=value`
gte	Feldwert ist größer gleich Wert. `?field[gte]=value`
between	Feldwert liegt im Wertebereich. `?field[between]=value1..value2`

Sie senden einen HTTP-Post-Request an den entsprechenden Endpunkt:

```
curl -X POST "https://localhost/aaas/service/filters?groups[]=filterProperty" ...
```

```
{
  "type": "range",
  "service": "/aaas/repository/services/3",
  "properties": [
    {
      "field": "/aaas/service/fields/2"
    }
  ]
}
```

Beispiel zur Verwendung (Alle Produkte mit einer Preisspanne zwischen 150 und 300):

```
curl -X GET "https://localhost/api/shop/products?price[gt]=150&price[lt]=300" ...
```

```
[
    {
        "id": 1,
        "name": "Tieflochbohrer",
        "price": "199.00",
        "isInStock": true,
        "createdAt": "2020-08-11T00:00:00+02:00",
        "transportFees": null
    },
    {
        "id": 6,
        "name": "Kernbohrer",
        "price": "250.00",
        "isInStock": true,
        "createdAt": "2020-08-21T00:00:00+02:00",
        "transportFees": null
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    }
]
```

12.2.6 Existsfilter

Der Existsfilter filtert eine Kollektion nach annullierbaren Feldern. Wenn einer Ihrer Services beispielsweise ein `transportFees`-Feld (Transportgebühren) implementiert, bei welchem `isNullable` auf `true` gesetzt ist, dann können Sie sich später mithilfe des Existsfilter alle Produkte ausgeben lassen, bei denen das entsprechende Feld auf ungleich `null` gesetzt ist.

Die Syntax bei der späteren Verwendung unterscheidet sich ein wenig von den vorangegangen Beispielen:

Parameter	Definition
exists	Sie übergeben noch den Feldnamen: `?exists[field]=<true|false|1|0>`

Sie senden einen HTTP-Post-Request an den entsprechenden Endpunkt:

```
curl -X POST "https://localhost/aaas/service/filters?groups[]=filterProperty" ...
```

```
{
  "type": "exists",
  "service": "/aaas/repository/services/3",
  "properties": [
    {
      "field": "/aaas/service/fields/5"
    }
  ]
}
```

Beispiel zur Verwendung (Alle Produkte, bei denen `transportFees` nicht auf `null` gesetzt ist):

```
curl -X GET "https://localhost/api/shop/products?exists[transportFees]=true" ...
```

```
[
    {
        "id": 2,
        "name": "Spiralbohrer",
        "price": "99.99",
        "isInStock": true,
        "createdAt": "2020-08-07T00:00:00+02:00",
        "transportFees": "10"
    },
    {
        "id": 5,
        "name": "Festmaß-Aufbohrer",
        "price": "20.00",
        "isInStock": true,
        "createdAt": "2020-09-01T00:00:00+02:00",
        "transportFees": "35"
    }
]
```

12.2.7 Orderfilter

Der Orderfilter ermöglicht das feldbasierte auf- und absteigende Sortieren Ihrer API-Kollektionen. Bei der späteren Verwendung müssen Sie, wenn Sie keinen Standard definiert haben, die Sortierreihenfolge angeben:

Parameter	Definition
asc	Sortiert aufsteigend. `?order[field]=asc`
desc	Sortiert absteigend. `?order[field]=desc`

Sie können auch eine Standard-Sortierreihenfolge festlegen, indem Sie als Wert für `value` entweder `ASC` oder `DESC` festlegen:

```
curl -X POST "https://localhost/aaas/service/filters?groups[]=filterProperty" ...
```

```
{
  "type": "order",
  "service": "/aaas/repository/services/3",
  "properties": [
    {
      "field": "/aaas/service/fields/1",
      "value": "ASC"
    }
  ]
}
```

Beachten Sie bitte, dass wir im Request oben `value` auf `ASC` gesetzt haben. `?order[field]` sortiert später also automatisch aufsteigend und erwartet dementsprechend keinen Wert für den Parameter.

Beispiel zur Verwendung (Kollektion aufsteigend nach `name` sortieren):

```
curl -X GET "https://localhost/api/shop/products?order[name]" ...
```

```
[
    {
        "id": 3,
        "name": "Aufbohrer",
        "price": "40.00",
        "isInStock": false,
        "createdAt": "2020-08-20T00:00:00+02:00",
        "transportFees": null
    },
    {
        "id": 5,
        "name": "Festmaß-Aufbohrer",
        "price": "20.00",
        "isInStock": true,
        "createdAt": "2020-09-01T00:00:00+02:00",
        "transportFees": "35"
    },
    {
        "id": 6,
        "name": "Kernbohrer",
        "price": "250.00",
        "isInStock": true,
        "createdAt": "2020-08-21T00:00:00+02:00",
        "transportFees": null
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
    },
    {
        "id": 2,
        "name": "Spiralbohrer",
        "price": "99.99",
        "isInStock": true,
        "createdAt": "2020-08-07T00:00:00+02:00",
        "transportFees": "10"
    },
    {
        "id": 1,
        "name": "Tieflochbohrer",
        "price": "199.00",
        "isInStock": true,
        "createdAt": "2020-08-11T00:00:00+02:00",
        "transportFees": null
    },
    {
        "id": 4,
        "name": "VHM-Stufenbohrer",
        "price": "20.00",
        "isInStock": false,
        "createdAt": "2020-08-18T00:00:00+02:00",
        "transportFees": null
    }
]
```


12.3 Servicefilter bearbeiten

Sie senden einen HTTP-Put-Request an den entsprechenden Endpunkt und editieren die zu bearbeitenden Felder:

```
curl -X PUT "https://localhost/aaas/service/filters/1" ...
```

```
{
  "type": "order"
}
```


```
{
    "id": 1,
    "type": "order",
    "service": "/aaas/repository/services/3",
    "properties": [
        "/aaas/service/filter/properties/1"
    ]
}
```


Im Beispiel oben ändern wir den Filtertyp auf `order`.

12.4 Servicefilter löschen

Sie senden einen HTTP-Delete-Request an den entsprechenden Endpunkt:

```
curl -X DELETE "https://localhost/aaas/service/filters/1" ...
```

Beachten Sie bitte, dass zugehörige Filtereigenschaften ebenfalls gelöscht werden.

12.5 Servicefilter anzeigen

Wenn Sie sich alle Filter anzeigen lassen möchten, die zu einem bestimmten Service gehören, dann können Sie einen entsprechenden HTTP-Query-Parameter verwenden:

```
curl -X GET "https://localhost/aaas/service/filters?service=3" ...
```


```
[
    {
        "id": 1,
        "type": "string",
        "service": "/aaas/repository/services/3",
        "properties": [
            "/aaas/service/filter/properties/1"
        ]
    },
    {
        "id": 2,
        "type": "date",
        "service": "/aaas/repository/services/3",
        "properties": [
            "/aaas/service/filter/properties/2"
        ]
    },
    {
        "id": 3,
        "type": "boolean",
        "service": "/aaas/repository/services/3",
        "properties": [
            "/aaas/service/filter/properties/3"
        ]
    },
    {
        "id": 4,
        "type": "numeric",
        "service": "/aaas/repository/services/3",
        "properties": [
            "/aaas/service/filter/properties/4"
        ]
    },
    {
        "id": 5,
        "type": "range",
        "service": "/aaas/repository/services/3",
        "properties": [
            "/aaas/service/filter/properties/5"
        ]
    },
    {
        "id": 6,
        "type": "exists",
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
        "service": "/aaas/repository/services/3",
        "properties": [
            "/aaas/service/filter/properties/6"
        ]
    },
    {
        "id": 7,
        "type": "order",
        "service": "/aaas/repository/services/3",
        "properties": [
            "/aaas/service/filter/properties/7"
        ]
    }
]
```


12.6 Serialisierungsgruppen

Sie finden im Folgenden eine Übersicht der verwendeten Serialisierungsgruppen von Servicefiltern:

Gruppe	Definition
service	Serialisiert/Deserialisiert `service`-Attribute
filterProperty	Serialisiert/Deserialisiert `filterProperty`-Attribute

12.6.1 Serialiserung von Services

Sie setzen `groups[]` auf `service`:

```
curl -X GET "https://localhost/aaas/service/filters/1?groups[]=service" ...
```


```
{
    "id": 1,
    "type": "string",
    "service": {
        "id": 3,
        "name": "Product",
        "description": null,
        "type": "list",
        "repository": "/aaas/project/repositories/2",
        "fields": [
            "/aaas/service/fields/1",
            "/aaas/service/fields/2",
            "/aaas/service/fields/3",
            "/aaas/service/fields/4",
            "/aaas/service/fields/5"
        ]
    },
    "properties": [
        "/aaas/service/filter/properties/1"
    ]
}
```

12.6.2 Serialiserung von Filtereigenschaften

Sie setzen `groups[]` auf `filterProperty`:

```
curl -X GET "https://localhost/aaas/service/filters/1?groups[]=filterProperty" ...
```


```
{
    "id": 1,
    "type": "string",
    "service": "/aaas/repository/services/3",
    "properties": [
        {
            "id": 1,
            "value": "start",
            "filter": "/aaas/service/filters/1",
            "field": "/aaas/service/fields/1"
        }
    ]
}
```


12.6.3 Deserialiserung von Filtereigenschaften

Sie setzen `groups[]` auf `filterProperty` und senden valide Daten:

```
curl -X POST "https://localhost/aaas/service/filters/1?groups[]=filterProperty" ...
```


```
{
  "type": "string",
  "service": "/aaas/repository/services/3",
  "properties": [
    {
      "field": "/aaas/service/fields/1",
      "value": "partial"
    }
  ]
}
```



```
{
    "id": 1,
    "type": "string",
    "service": "/aaas/repository/services/3",
    "properties": [
        {
            "id": 1,
            "value": "start",
            "filter": "/aaas/service/filters/1",
            "field": "/aaas/service/fields/1"
        }
    ]
}
```

KAPITEL 13

Paginierung

Mithilfe der `itemsPerPage`- und `page`-Parameter können Sie API-Kollektionen paginieren und sich auf diese Art durch Ihre Datensätze bewegen. Die maximale Anzahl an Datensätzen pro Seite ist auf 100 festgelegt. Wenn Sie einen größeren Wert benötigen, dann bearbeiten Sie bitte die entsprechende Konfigurationsvariable in api_platform.yaml.

Strategie	Definition
page	Mit `page` geben Sie die aktuelle Seite an.
itemsPerPage	Mit `itemsPerPage` geben Sie die Anzahl an Datensätzen pro Seite an.

13.1 Beispiel

Sie setzen die entsprechenden HTTP-Query-Parameter:

```
curl -X POST "https://localhost/api/blog/articles?itemsPerPage=2" ...
```

```
[
    {
        "id": 1,
        "title": "Article 1"
    },
    {
        "id": 2,
        "title": "Article 2"
    }
]
```

Mit `page`-Parameter:

```
curl -X POST "https://localhost/api/blog/articles?itemsPerPage=2&page=2" ...
```

```
[
    {
        "id": 3,
        "title": "Article 3"
    },
    {
        "id": 4,
        "title": "Article 4"
    }
]
```


Nächste Seite:

```
curl -X POST "https://localhost/api/blog/articles?itemsPerPage=2&page=3" ...
```


```
[
    {
        "id": 5,
        "title": "Article 5"
    },
    {
        "id": 6,
        "title": "Article 6"
    }
]
```

KAPITEL 14

API-Generierung

Wenn Sie aus den von Ihnen erstellten Services und Felddefinitionen Ihre API generieren lassen möchten, dann müssen Sie drei Schritte durchführen. Anschließend können Sie Ihre API benutzen und Ihren API-Konsumenten zur Verfügung stellen.

14.1 APIs generieren

Im ersten Schritt generieren Sie aus Ihren Definitionen Ihre API. Dabei werden unter `/src/Aaas/Entity` und `/src/Aaas/Repository` die entsprechenden Entitäten und Repositorien erzeugt. Den erstellten Quellcode können Sie natürlich noch nach Belieben anpassen:

```
curl -X GET "https://localhost/aaas/projects/{id}/build" ...
```

14.2 Datenbank migrieren

Anschließend können Sie Ihre Datenbank migrieren, indem Sie den unten aufgeführten HTTP-Request absetzen. Wir erstellen dabei unter `/src/App/Migrations` eine entsprechende Migrations-Datei und vergleichen den aktuellen Stand Ihrer Datenbank mit Ihren API-Definitionen. Nachdem das geschehen ist, führen wir die in der Migrationsdatei befindlichen Datenbankoperationen aus und migrieren dabei Ihre Datenbank auf den neuen Stand:

```
curl -X GET "https://localhost/aaas/system/migrate" ...
```

14.3 Applikationscache leeren

Aus Anwendungssicht haben sich nach dem Erstellen Ihrer API grundlegende Konfigurationen (Caching, Routing, etc.) geändert. Es ist daher unbedingt notwendig den Applikationscache Ihrer Anwendung zu löschen bzw zu aktualisieren. Intern erfolgt dabei ein sog. `Cache-Warmup`, der grundlegende Anwendungskonfigurationen wiederherstellt und im Applikationscache vorrätig hält:

```
curl -X GET "https://localhost/aaas/system/clear-cache" ...
```

KAPITEL 15

Generisches Baumdiagramm

Mit einem durch AaaS-API erstellten generischen Baumdiagramm können Sie Stamm- oder Entscheidungsbäume in Ihrer Datenbank abbilden oder andere allgemeingültige Baumstrukturen entwerfen und implementieren.

Sie müssen dafür lediglich einen Service vom Typ `tree` erzeugen. Die von AaaS-API zur Verfügung gestellten Schnittstellen zum Umgang mit der dann generierten Baumstruktur sind einheitlich.

15.1 Tree-Service anlegen

Sie erzeugen ein Projekt samt Repositorium und einen Service vom Typ `tree`:

```
curl -X POST "https://localhost/aaas/projects?groups[]=repository&groups[]=service" ..
↪.
```

```
{
    "name":"My API Project",
    "repositories":[
        {
            "name":"Examples",
            "services":[
                {
                    "name":"Family",
                    "type":"tree"
                }
            ]
        }
    ]
}
```

Der API-Server antwortet mit dem HTTP-Statuscode 201 und der eben gerade erzeugten Ressource:

```
{
    "id": 1,
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
    "name": "My API Project",
    "description": null,
    "repositories": [
        {
            "id": 1,
            "name": "Examples",
            "description": null,
            "project": "/aaas/projects/1",
            "services": [
                {
                    "id": 1,
                    "name": "Family",
                    "description": null,
                    "type": "tree",
                    "repository": "/aaas/project/repositories/1",
                    "fields": [
                    ]
                }
            ]
        }
    ]
}
```


Sie ordnen dem Service anschließend noch zwei Felder zu:

```
curl -X POST  "https://localhost/aaas/service/fields" ...
```


```
{
    "name": "name",
    "service": "/aaas/repository/services/1"
}
```



```
{
  "name": "birthdate",
  "dataType": "date",
  "service": "/aaas/repository/services/1"
}
```


15.2 Migration des Systems

Sie erstellen nun aus dem angelegten Projekt eine PHP-API, migrieren die Datenbank und leeren den Applikationscache Ihrer Anwendung:

```
curl -X GET "https://localhost/aaas/projects/1/build"
curl -X GET "https://localhost/aaas/system/migrate"
curl -X GET "https://localhost/aaas/system/clear-cache"
```

Die von *AaaS-API* generierte PHP-API können Sie dann sofort nutzen.

15.3 Anlegen von Knoten

Sie können die erstellte PHP-API benutzen, um Ihre Baumstruktur zu befüllen:

```
curl -X POST "https://localhost/api/examples/families"
```

```
{
  "name" : "Leon Huber",
  "birthdate" : "2002-05-08"
}
```

Der API-Server antwortet dann mit der erstellten Ressource:

```
{
    "id": 1,
    "parent": null,
    "children": [
    ],
    "name": "Leon Huber",
    "birthdate": "2002-05-08T00:00:00+02:00"
}
```

Sie können dann Vorfahren von `Leon Huber` anlegen:

```
curl -X POST "https://localhost/api/examples/families"
```

```
{
  "name" : "Sara Huber",
  "birthdate" : "1987-04-15",
  "parent" : "/api/examples/families/1"
}

{
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
  "name" : "Erik Huber",
  "birthdate" : "1982-09-14",
  "parent" : "/api/examples/families/1"
}
```

Sie erstellen noch die Eltern von `Sara Huber`:

```
curl -X POST "https://localhost/api/examples/families"
```

```
{
  "name" : "Michelle Huber",
  "birthdate" : "1957-01-10",
  "parent" : "/api/examples/families/2"
}

{
  "name" : "Marcel Huber",
  "birthdate" : "1960-03-17",
  "parent" : "/api/examples/families/2"
}
```

15.4 Holen von Knoten

Sie können in gewohnter Weise die angelegten Ressourcen vom API-Server beziehen:

```
curl -X GET "https://localhost/api/examples/families"
```

```
[
    {
        "id": 1,
        "parent": null,
        "children": [
            "/api/examples/families/2",
            "/api/examples/families/3"
        ],
        "name": "Leon Huber",
        "birthdate": "2002-05-08T00:00:00+02:00"
    },
    {
        "id": 2,
        "parent": "/api/examples/families/1",
        "children": [
            "/api/examples/families/4",
            "/api/examples/families/5"
        ],
        "name": "Sara Huber",
        "birthdate": "1987-04-15T00:00:00+02:00"
    },
    {
        "id": 3,
        "parent": "/api/examples/families/1",
        "children": [
        ],
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
        "name": "Erik Huber",
        "birthdate": "1982-09-14T00:00:00+02:00"
    },
    {
        "id": 4,
        "parent": "/api/examples/families/2",
        "children": [
        ],
        "name": "Michelle Huber",
        "birthdate": "1957-01-10T00:00:00+01:00"
    },
    {
        "id": 5,
        "parent": "/api/examples/families/2",
        "children": [
        ],
        "name": "Marcel Huber",
        "birthdate": "1960-03-17T00:00:00+01:00"
    }
]
```


Wenn Sie nur ein Element beziehen möchten, dann spezifizieren Sie bitte noch den eineindeutigen Schlüssel der entsprechenden Ressource:

```
curl -X GET "https://localhost/api/examples/families/1"
```


```
{
    "id": 1,
    "parent": null,
    "children": [
        "/api/examples/families/2",
        "/api/examples/families/3"
    ],
    "name": "Leon Huber",
    "birthdate": "2002-05-08T00:00:00+02:00"
}
```


Wenn Sie sich die Vorfahren des entsprechenden Knotens anzeigen lassen möchten, dann hängen Sie bitte noch das Schlüsselwort `childrens` an die URI an. Sie können sich auf diese Art durch Ihre Baumstruktur hangeln:

```
curl -X GET "https://localhost/api/examples/families/1/childrens"
```


```
[
    {
        "id": 2,
        "parent": "/api/examples/families/1",
        "children": [
            "/api/examples/families/4",
            "/api/examples/families/5"
        ],
        "name": "Sara Huber",
        "birthdate": "1987-04-15T00:00:00+02:00"
    },
    {
        "id": 3,
        "parent": "/api/examples/families/1",
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        "children": [
        ],
        "name": "Erik Huber",
        "birthdate": "1982-09-14T00:00:00+02:00"
    }
]
```

```
curl -X GET "https://localhost/api/examples/families/2/childrens"
```

```
[
    {
        "id": 4,
        "parent": "/api/examples/families/2",
        "children": [
        ],
        "name": "Michelle Huber",
        "birthdate": "1957-01-10T00:00:00+01:00"
    },
    {
        "id": 5,
        "parent": "/api/examples/families/2",
        "children": [
        ],
        "name": "Marcel Huber",
        "birthdate": "1960-03-17T00:00:00+01:00"
    }
]
```