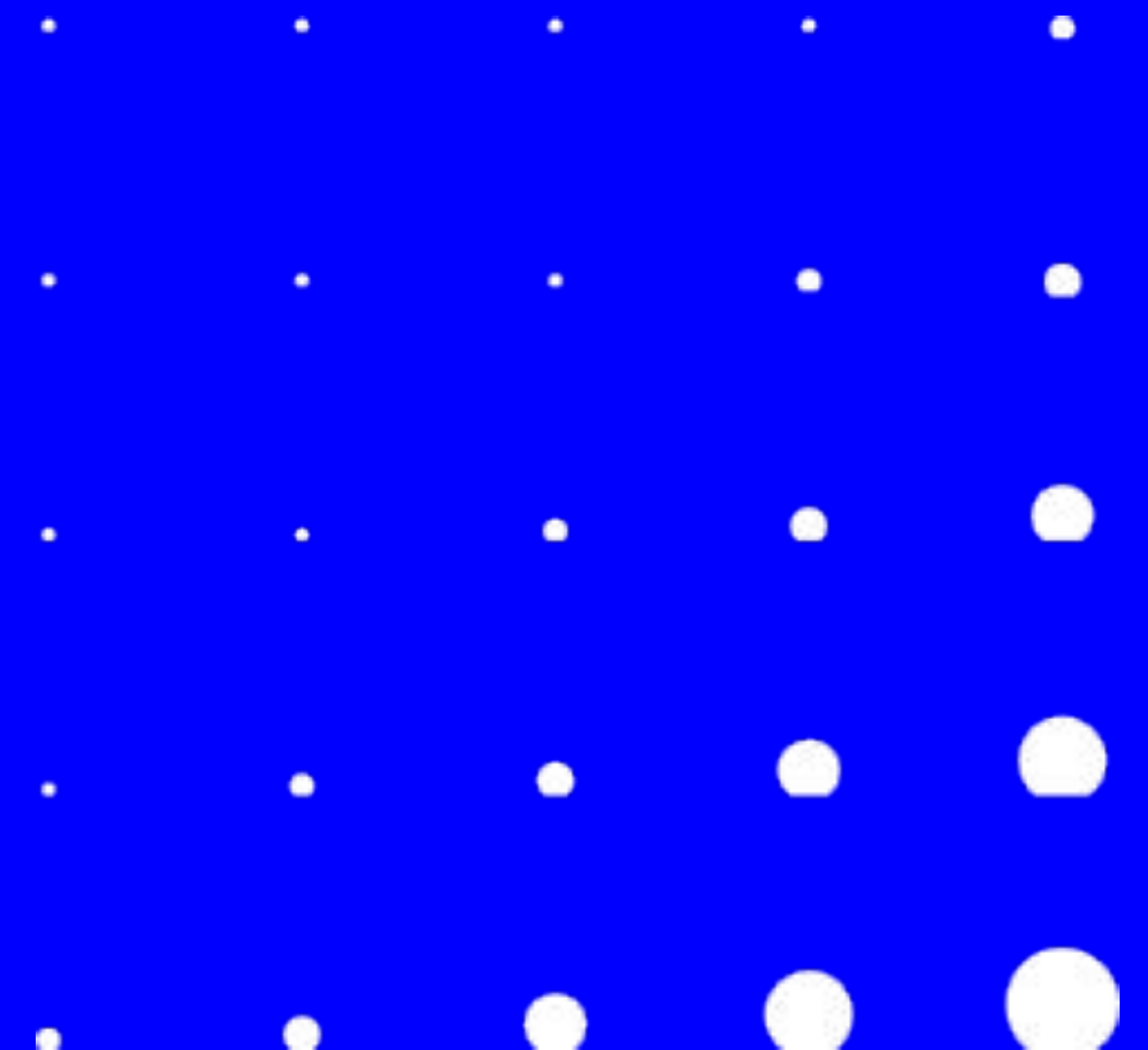


API-Design und Dokumentation mit OpenAPI

Béla Gallin
Serhat Göl
Fabian Hoppe
David Kubek
Jasmin Tschernoch



Agenda

1. Wissensstand
2. Szenario und Erste Dokumentation
3. OpenAPI Einführung
4. Fertige OpenAPI-Dokumentation - Da wollen wir hin
5. Voraussetzungen und Tools
6. OpenAPI Spezifikation
7. Design First
8. Code First
9. Ausblick
10. Zusammenfassung

1. Wissensstand



**Was wisst ihr bereits
über APIs und REST?**



WAS gehört alles
zu einer
Dokumentation?

WIE macht man
die
Dokumentation?

WANN macht
man die
Dokumentation?

Für **WEN** ist die
Dokumentation?

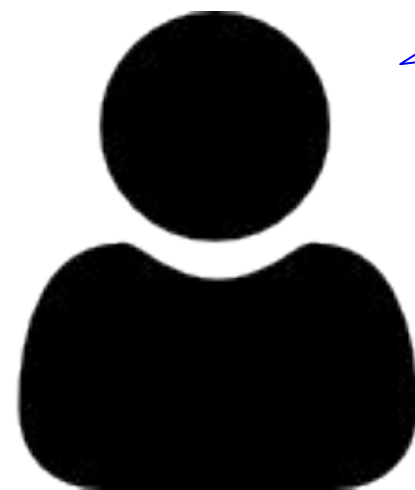


**Wie könnt ihr eine
(REST)API
dokumentieren?**

2. Szenario und Erste Dokumentation

Food Express

Ihr wurdet als Entwicklerteam beauftragt, die API-Dokumentation für FoodExpress zu erstellen – eine innovative Plattform zur Online-Essensbestellung. Die Plattform verbindet hungrige Kunden mit lokalen Restaurants und ermöglicht eine nahtlose Bestellabwicklung.



Restaurants

Verwaltung und
Anzeige von
Restaurants und
deren Speisekarten

Bestellungen

Erstellen, Abrufen und
Verwaltung von
Bestellungen

Kunden

Kundenverwaltung
und
Profilinformationen



**Definiert diese
API schriftlich!**

3. OpenAPI Einführung

Was ist OpenAPI?

OpenAPI

Standardisierte Spezifikation zur Beschreibung von HTTP-basierten APIs

- Beschreibt WAS eine API kann – nicht WIE sie intern implementiert ist
- Maschinen- und menschenlesbar
- Technologieneutral (Sprache, Framework egal)

Mit OpenAPI kann man u. a. definieren:

Endpunkte (URLs)

HTTP-Methoden
(GET, POST,...)

Parameter,
Request- &
Response-
Struktur

Statuscodes

Authentifizierung

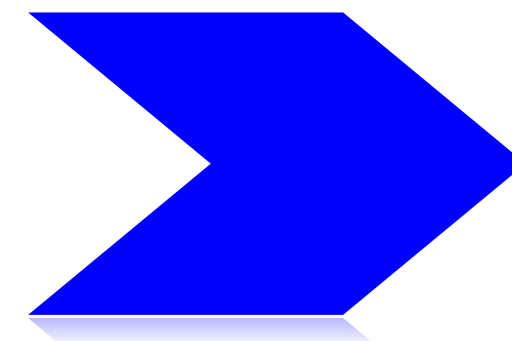
Beispiele &
Beschreibungen

Warum OpenAPI?



Zentrale Probleme ohne OpenAPI:

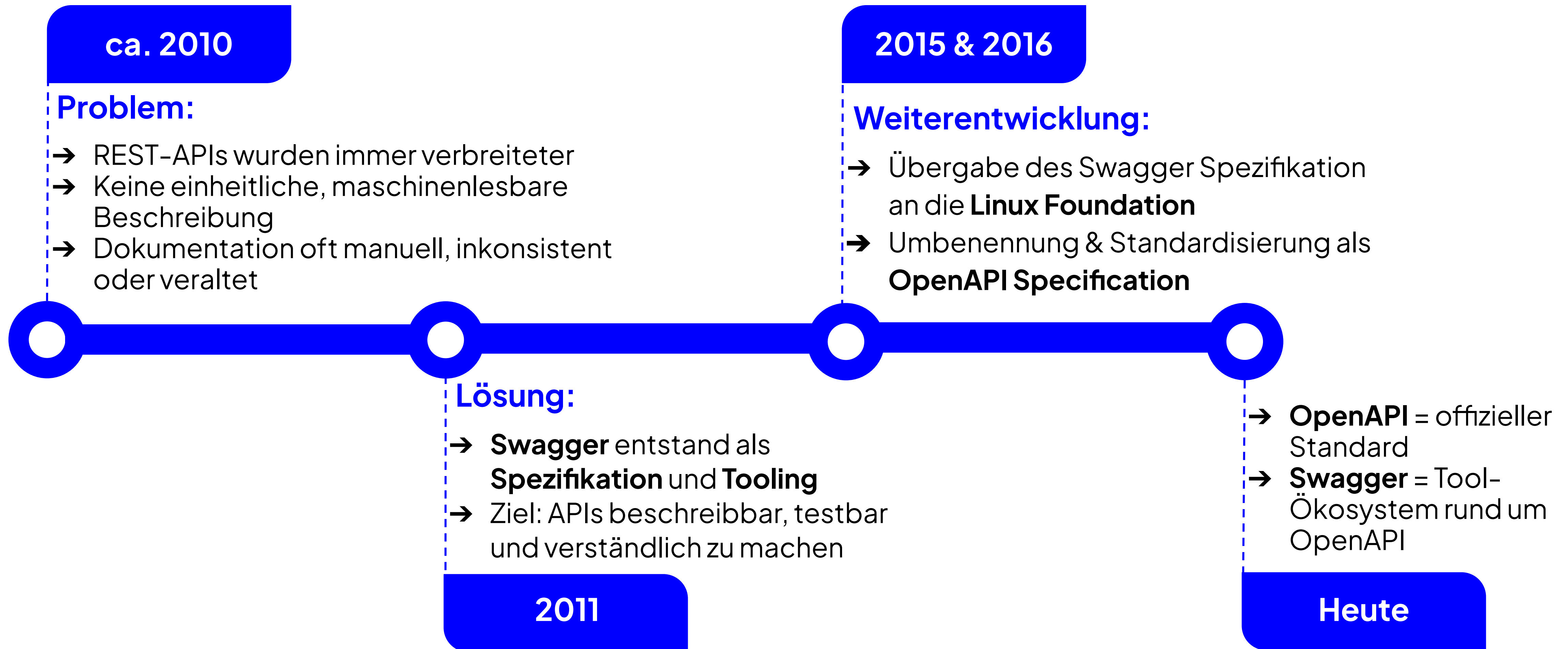
- ↯ Unklare API-Verträge
- ↯ Unterschiedliche Erwartung zwischen Teams
- ↯ Dokumentation veraltet oder fehlt
- ↯ Hoher Abstimmungsaufwand



OpenAPI löst Probleme durch:

- ✓ Eine **Single Source of Truth**
- ✓ Klare, versionierbare API-Beschreibung
- ✓ Automatisierung (Code, Tests, Docs)

Historie & Swagger – von Swagger zu OpenAPI

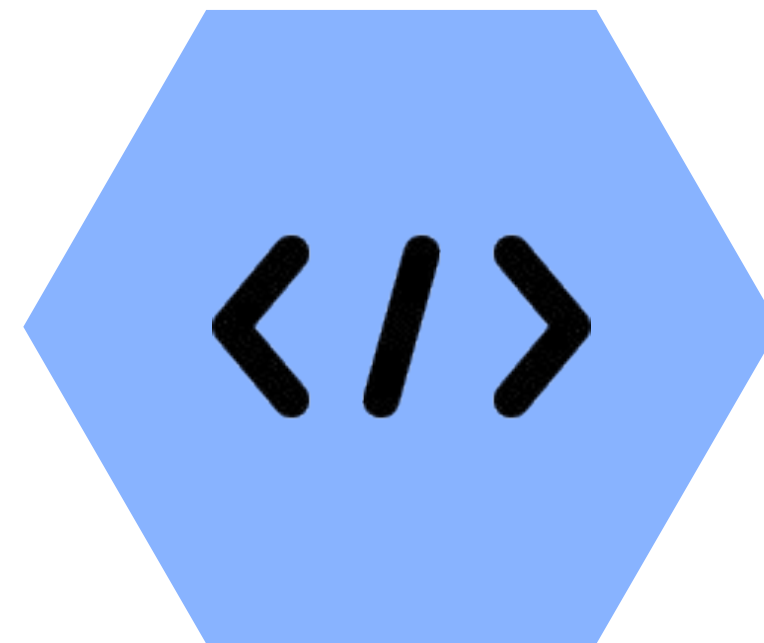


Was kann man mit OpenAPI machen?

Interaktive Doku (Swagger UI)



Client-Code generieren



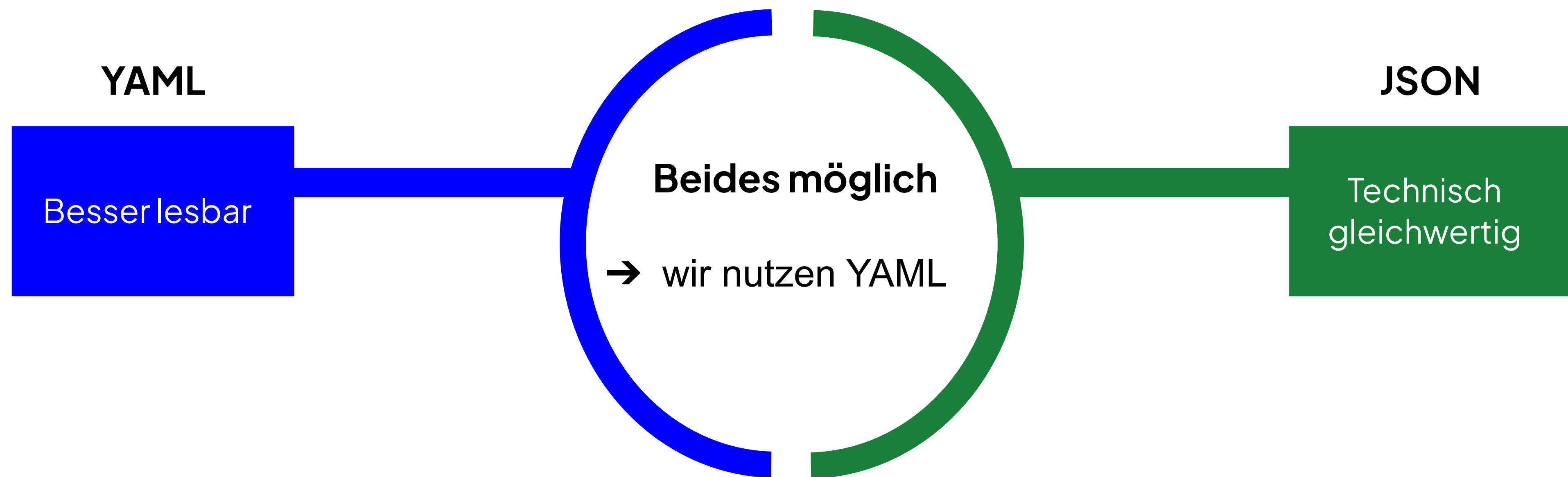
Mock-Server & Tests



API designen & reviewen



YAML vs. JSON



4. Fertige OpenAPI-Dokumentation – Da wollen wir hin

API-Dokumentation für unser Szenario

```
1  openapi: 3.1.0
2  info:
3    title: Food Express API
4    description: >
5      API for managing restaurants, orders, and customers.
6    version: 1.0.0
7
8  servers:
9    - url: http://localhost:4004/api/v1
10     description: Local development servers
11    - url: https://{stage}.leaferando.com/api/v1
12     description: Stage server
13     variables:
14       stage:
15         description: Stage
16         enum:
17           - dev
18           - test
19           - prod
20         default: dev
21
22  tags:
23    - name: Restaurants
24    - name: Orders
25    - name: Customers
26
27  paths:
28    /restaurants:
29      get:
30        Scan | Audit
31        tags: [Restaurants]
32        summary: List restaurants
33        description: Retrieve a paginated list of restaurants.
34        security: []
35        parameters:
36          - $ref: "#/components/parameters/Limit"
37          - $ref: "#/components/parameters/Offset"
38          - $ref: "#/components/parameters/AcceptLanguage"
39        responses:
40          "200":
41            description: A paginated list of restaurants
42            content:
43              application/json:
44                schema:
45                  type: object
46                  properties:
47                    data:
48                      type: array
```

Food Express API 1.0.0 OAS 3.1

<http://openapi-preview.example/>

API for managing restaurants, orders, and customers.

Servers

http://localhost:4004/api/v1 - Local development servers

Authorize

Restaurants

GET

/restaurants

List restaurants

GET

/restaurants/{id}/menu

Get restaurant menu

GET

/restaurants/{id}/orders

List orders for a restaurant

Orders

GET

/restaurants/{id}/orders

List orders for a restaurant

POST

/orders

Create a new order

GET

/orders/{orderId}

Get order by ID

PATCH

/orders/{orderId}/status

Update order status

Customers

POST

/customers

Create a customer

GET

/customers/me

Get current customer

Pause (15 min)

5. Voraussetzungen und Tools

Einführung YAML

YAML = Yet Another Markup Language

Zweck

Datenserialisierung → Daten speichern und übertragen

Warum YAML?

- ✓ **Lesbarkeit:** Gut für Menschen lesbar
- ✓ **Nutzung:** Standard für Konfigurationsdateien (Docker, Kubernetes, GitHub Actions, OpenAPI)

```
einkaufsliste:
- Äpfel
- Bananen

mitarbeiter:
- name: Hans
  abteilung: IT
  skills:           # Eine Liste innerhalb eines Objekts in einer Liste
    - Python
    - Docker
- name: Sarah
  abteilung: HR
  skills:
    - Recruiting

string_text: "%Hallo Welt" # Text
integer_zahl: 42           # Ganzzahl
float_komma: 3.14          # Fließkommazahl
boolean_wahr: true        # Wahrheitswert
null_wert: null            # Leerer Wert (oder ~)
```

Einführung YAML

Einrückung

Struktur durch Leerraum

Indentation um Hierarchien zu bilden

Key-Value Paare

Grundelement ist
Schlüssel: Wert

```
parent_element:
|  child_element:      # 2 Leerzeichen Einrückung
|  |  grandchild: Wert # 4 Leerzeichen Einrückung

# Dies ist ein Kommentar für die ganze Datei
name: Max Mustermann # String (Text)
version: 1.0          # Das Leerzeichen nach dem Doppelpunkt ist Pflicht!
# falsch: version:1.0
```

Regel

Nutze immer Leerzeichen, keine Tabs!
→ 2 Leerzeichen pro Ebene

Wichtig

Nach dem Doppelpunkt muss ein
Leerzeichen folgen

Kommentare

Alles nach einem # wird
ignoriert

Einführung YAML

Einrückung

Objekte (Dictionaries)

Sammlung von **Key-Value** Paaren.

Listen (Arrays)

Mit einem Bindestrich - und Leerzeichen eingeleitet

Verschachtelung

Listen können Objekte enthalten und umgekehrt

Datentypen

Automatisch erkannt (String, Integer, Boolean, Float)

```
kunden_profil:
  vorname: Lisa
  nachname: Müller
  adresse:
    stadt: Berlin
    plz: 10115

einkaufsliste:
- Äpfel
- Bananen

mitarbeiter:
- name: Hans
  abteilung: IT
  skills:
    - Python
    - Docker
- name: Sarah
  abteilung: HR
  skills:
    - Recruiting

string_text: "%Hallo Welt" # Text
integer_zahl: 42           # Ganzzahl
float_komma: 3.14          # Fließkommazahl
boolean_wahr: true        # Wahrheitswert
null_wert: null           # Leerer Wert (oder ~)
```

Eine Liste innerhalb eines Objekts in einer Liste

Anführungszeichen bei
Sonderzeichen benötigt

Swagger Tools

Editor, UI, Codegen

Swagger Editor (Design & Entwurf)

Funktion: Browser-basierter Editor

Vorteil: Feedback und Fehleranzeige, live Vorschau

Nutzer: API-Architekten & Backend-Entwickler

Swagger UI (Visualisierung & Testen)

Funktion: Rendert die Spezifikation in interaktive HTML-Seite

Vorteil: Ermöglicht direktes Testen (Requests im Browser)

Nutzer: Frontend-Entwickler, Tester & Stakeholder

Swagger Codegen (Automatisierung)

Funktion: Generiert Code aus Spezifikation

Output:

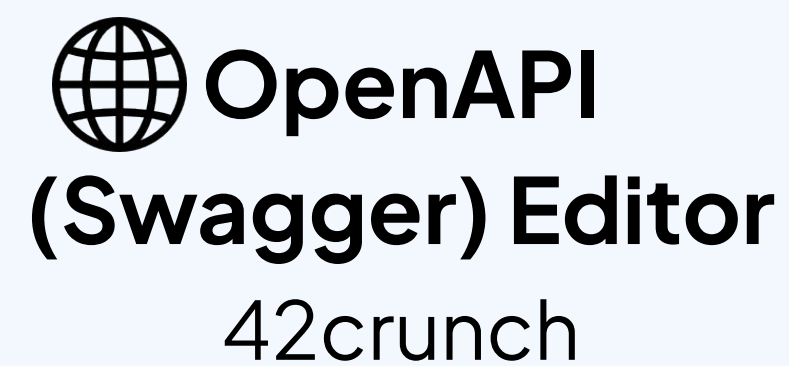
- **Server Stubs:** Grundgerüst Backend
- **Client SDKs:** Bibliotheken Frontend

Vorteil: Spart Zeit, verhindert Fehler zwischen Doku und Code

VS Code Extensions

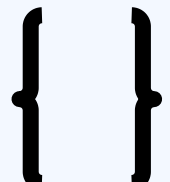


Funktionen der Extension



- **Validierung:** Syntaxfehler, strukturelle Probleme
- **Autovervollständigung:** Kontextbezogene Vorschläge
- **Hover-Informationen:** Dokumentationen und Beschreibungen
- **Dokumentgliederung:** Hierarchische Baumstruktur des Dokuments → schnelle Navigation in großen Dateien
- **Formatierung:** Formatierung des Codes

VS Code Extensions

 **YAML**
Red Hat

 **OpenAPI
(Swagger) Editor**
42crunch



OpenAPI (Swagger) Editor

von 42crunch

Funktionen der Extension

- **Live-Preview:** Rendert Swagger/OpenAPI-Dateien als grafische Benutzeroberfläche (Swagger UI)
- **Autovervollständigung:** Für Swagger 2.0, OpenAPI 3.0 Tags und Keywords
- **Echtzeit-Validierung:** Syntaxfehler gemäß OpenAPI-Spezifikation
- **Auto-Refresh:** Vorschau aktualisiert bei Änderungen der Datei

6. OpenAPI Spezifikation

OpenAPI Spezifikation

Informationen hier sind nur ein Überblick über grundlegende Funktionen und Anwendung.

Details in der Dokumentation nachlesen.



Ressourcen

Dokumentation mit Erklärung

[The OpenAPI Specification Explained](#)

Dokumentation

[OpenAPI Specification versions 3.1.0](#)

The screenshot shows the OpenAPI Initiative website. The header includes the OpenAPI logo and a search bar. The left sidebar contains a navigation menu with items like 'Getting Started', 'Introduction', 'Glossary', and 'The OpenAPI Specification Explained'. The main content area is titled 'The OpenAPI Specification Explained' and contains an introductory paragraph, a list of topics, and a list of links to specific sections. The footer includes copyright information and a Creative Commons license.

OPENAPI INITIATIVE

Search OpenAPI Documentation

The OpenAPI Specification Explained

The **OpenAPI Specification** is the ultimate source of knowledge regarding this API description format. However, its length is daunting to newcomers and makes it hard for experienced users to find specific bits of information. This chapter provides a soft landing for readers not yet familiar with OpenAPI and is organized by topic, simplifying browsing.

The following pages introduce the syntax and structure of an OpenAPI Description (OAD), its main building blocks and a minimal API description. Afterwards, the different blocks are detailed, starting from the most common and progressing towards advanced ones.

- **Structure of an OpenAPI Description:** JSON, YAML, `openapi` and `info`
- **API Endpoints:** `paths` and `responses`.
- **Content of Message Bodies:** `content` and `schema`.
- **Parameters and Payload of an Operation:** `parameters` and `requestBody`.
- **Reusing Descriptions:** `components` and `$ref`.
- **Providing Documentation and Examples:** `summary`, `description` and `example/examples`.
- **API Servers:** `servers`.

© 2025 OpenAPI Initiative. This work is licensed under a [Creative Commons Attribution 4.0 International License](https://creativecommons.org/licenses/by/4.0/). The documentation is maintained in <https://github.com/OAI/learn.openapis.org/>.

This site uses [Just the Docs](#), a documentation theme for Jekyll.

Struktur

Struktur

Server

Endpunkte
und HTTP
Operationen

Responses

Contents

Path
Parameters

Dokumen-
tation

- OpenAPI Dokument repräsentiert ein Objekt in YAML oder JSON
- Felder, die vorhanden sein müssen: `openapi, info`
 - Und eins dieser: `paths, components, webhooks`
- `openapi:` String, der Version der Spezifikation angibt
 - Wird zum Interpretieren des Dokuments genutzt
 - Aktuelle Major-Version: 3
- `info:` Allgemeine Infos zur API
 - Objekt mit den Feldern: `title, version, description` (optional)
- `paths:` Beschreibt Endpunkte der API

```
openapi: 3.1.0
info:
  title: Students API
  version: 1.0.0
paths: {}
```

Server

Struktur

Server

Endpunkte
und HTTP
Operationen

Responses

Contents

Path
Parameters

Dokumen-
tation

- Basis-URL der API definieren
- Ist optional
- Relevantes Feld: `servers`
- Array im OpenAPI-Objekt
- Element im Array definiert ein Objekt mit Infos zum Server
- Felder im Objekt: `url` (MUSS), `description`, `variables`
- Endpunkte werden an Basis-URL angehängt
- `variables`: URL kann Variablen enthalten; hier definiert
 - Objekt mit Variablennamen als Feldern
 - Pro Variable min. `default` definiert

```
openapi: 3.1.0
info:
  title: Students API
  version: 1.0.0
servers:
  - url: http://localhost:4004/api/v1
    description: Local development server
  - url: https://{stage}.students.com/api/v1
    description: Stage server
    variables:
      stage:
        description: Stage
        enum:
          - dev
          - test
          - prod
        default: dev
```

Endpunkte und HTTP Operationen

Struktur

Server

**Endpunkte
und HTTP
Operationen**

Responses

Contents

Path

Parameters

Dokumen-
tation

- Endpunkte in OpenAPI `paths`
- In `paths` werden die Endpunktnamen angegeben
- Für jeden Pfad alle unterstützten HTTP-Operationen aufgeführt
- Pfad muss mit / starten
- Zwei gleiche Operationen pro Pfad **NICHT** erlaubt
- Für Operation ein Objekt mit Infos über Verwendung
- Pro Operation: `parameters, requestBody, responses`

```
openapi: 3.1.0
info:
  title: Students API
  version: 1.0.0
paths:
  /students:
    get:
      summary: Get all students
      description: Retrieves all the students in
the system
      parameters: [...]
      requestBody: ...
      responses: ...
```

Responses

Struktur

Server

Endpunkte
und HTTP
Operationen

Responses

Contents

Path

Parameters

Dokumen-
tation

- In `responses` Reaktion der Operation definieren
- Pro Response ein Feld mit einem Objekt
- Felder müssen HTTP Response Codes sein
- Min. eine Response pro Operation
- Response definiert mit u. a.
 - `description:` Beschreibt Bedeutung (MUSS)
 - `content:` Möglicher Inhalt der Response beschrieben

```
openapi: 3.1.0
info:
  title: Students API
  version: 1.0.0
paths:
  /students:
    get:
      summary: Get all students
      description: Retrieves all the students in
the system
      parameters: [...]
      requestBody: ...
      responses:
        "200":
          description: OK
          content: ...
        "400":
          description: Bad Request
```

Contents

- `content` Feld beschreibt Inhalt von Response-Body oder Request-Body
- Mehrere Contents (Ein Feld pro Content mit Objekt)
- Name des Feldes hat Name eines Media-Types (z. B. `application/json`, `text/html`, `text/plain`)
- Objekt beschreibt die Struktur des Inhalts mit `schema`
- In `schema` wird ein `type` definiert: `number`, `integer`, `string`, `boolean`, `array`, `object`
- Basierend auf `type` weitere Angaben (siehe Beispiel)
- Für `object` braucht es `properties` für die Struktur
- Mit `examples` Beispiele angeben → gut für Dokumentation

```
content:
  application/json:
    schema:
      type: object
      properties:
        id:
          type: string
          examples: [11111111]
        name:
          type: string
          minLength: 1
          examples: [Max Mustermann]
        semester:
          type: integer
          minimum: 1
          examples: [1]
        vacationSemester:
          type: boolean
          examples: [false]
        courseOfStudy:
          type: string
          enum:
            - Informatik
            - Maschinenbau
            - Soziale Arbeit
          examples: [Informatik]
        classes:
          type: array
          maxItems: 10
          items:
            type: string
            examples: [[PR1, MA1]]
```

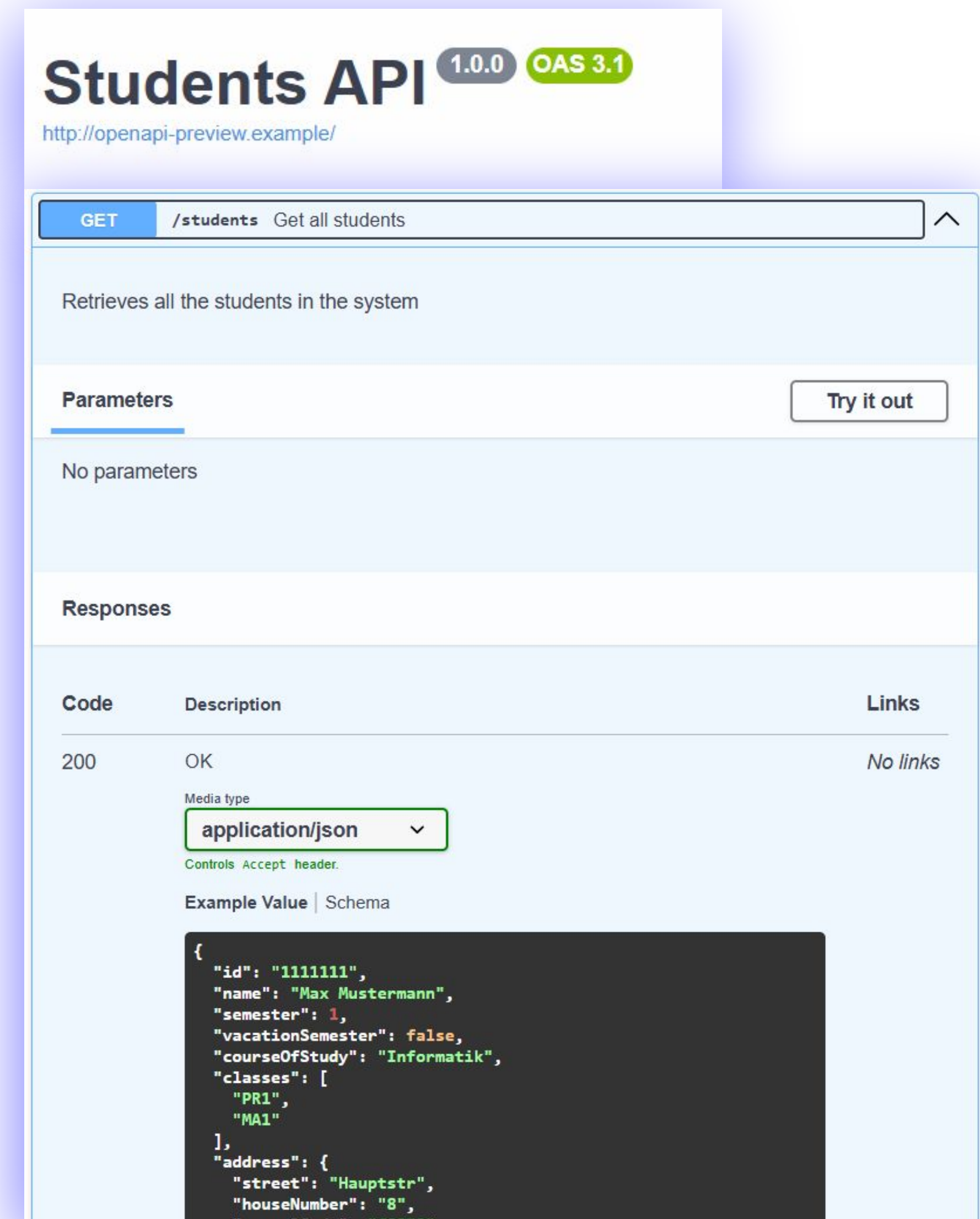
Path Parameters

- Für GET Requests häufig verwendet
- Gibt auch noch weitere Parameter: Query, Header
- Für die HTTP Operation das `parameters` Feld definieren
- `parameters` ist ein Array von Parametern
- Pro Parameter folgende Felder:
 - `name`: Eindeutiger Name
 - `in`: Ort des Parameters (Pfad, Query, Header)
 - `required`: Ob vorhanden sein muss
 - `schema`: Struktur des Parameters
- Für `schema` gleiche Definition wie bei Contents (Typ, Beispiele, ...)
- Pfad Parameter: `in: path` angeben
 - Muss immer vorhanden sein (`required: true`)
 - `name` des Parameters in Pfad-URL angeben (mit { })

```
openapi: 3.1.0
info:
  title: Students API
  version: 1.0.0
paths:
  /students/{id}:
    get:
      parameters:
        - in: path
          name: id
          required: true
          schema:
            type: string
            examples:
              [c0f89721-f331-4470-b160-d61c205beac8]
      responses: ...
```

Dokumentation

- Bei vielen Objekten sind `summary` und `description` vorhanden
- Gut zur Dokumentation des jeweiligen Elements
- `summary` für eine Kurzbeschreibung des Elements
- `description` für ausführlichere Beschreibung
 - Hier auch Markdown anwendbar für Formatierung usw.
- In `schema` auch `examples` vorhanden
 - Passende Beispiele angeben
- `externalDocs` für Referenz auf externe Dokumentation



Praxis: Spezifikation Teil 1



Szenario auf Gitty <https://gitty.informatik.hs-mannheim.de/2121190/IWS-OpenAPI>

- OpenAPI Dokumentation in VSCode
- Info ausfüllen
- Server definieren (mehrere sinnvolle)
- GET requests für Szenario
- Responses mit Contents definieren
- Sinnvolle summaries, descriptions und Beispiele

The image shows a side-by-side comparison of an OpenAPI specification in VS Code and its rendered Swagger UI.

VS Code Editor (Left): Displays the OpenAPI 3.1.0 JSON specification for 'Students API'. The spec includes:

- info:** title: Students API, version: 1.0.0
- servers:** Two servers are defined: a local development server at `http://localhost:4004/api/v1` and a stage server at `https://{stage}.students.com/api/v1`. The stage variable is defined with values `dev`, `test`, and `prod`, with `dev` as the default.
- paths:** A `/students` endpoint with a `get` method. The description is 'Retrieves all the students in the system'.

Swagger UI (Right): The rendered interface for 'Students API 1.0.0 OAS 3.1'. It features a 'Servers' dropdown menu currently set to 'http://localhost:4004/api/v1 - Local development servers'. Under the 'default' section, two GET endpoints are listed: `/students` (description: 'Get all students') and `/students/{id}`.

Pause (10 min)

Weitere Parameter

Weitere
Parameter

Payload

Tags

Komponenten

API Security

	Query	Header	Cookie
in	query	header	cookie
Typische Nutzung	Filter, Sortierung, Pagination	Sprache, Content-Type, Tokens	Sitzungs-/ Tracking- informationen
steuert	Antwort	Metadaten	Zustand/Ses- sion-Kontext

```
paths:
  /students:
    get:
      parameters:
        - name: limit
          in: query
          description: Number of Students to return
          required: true
          schema:
            type: integer
            minimum: 1
            maximum: 100
            default: 25
        - name: Accept-Language
          in: header
          description: Preferred language for the
            response
          required: false
          schema:
            type: string
            example: "de-DE"
        - name: sessionId
          in: cookie
          description: Session identifier
          required: false
          schema:
            type: string
            example: "abc1234"
```

Payload

- Operationen können einen **Request Body (Payload)** haben
- typisch für **POST, PUT und PATCH**
- definiert durch `schema`
- klar getrennt von `parameters`

```
requestBody:
  description: Create a student
  required: true
  content:
    application/json:
      schema:
        type: object
        required: [id, name ...]
        properties:
          id:
            type: string
          name:
            type: string
          ...
```

```
paths:
  /students:
    post:
      ...
      requestBody: }
      ...
    patch:
      ...
      requestBody: }
      ...
```

```
requestBody:
  description: Update fields for a student
  required: true
  content:
    application/json:
      schema:
        type: object
        properties:
          id:
            type: string
          name:
            type: string
          ...
```

Tags

Weitere
Parameter

Payload

Tags

Komponenten

API Security

default

GET

/students

Get all students

POST

/students

Create a student

GET

/students/{studentId}

Get a student

PATCH

/students/{studentId}

Update a student

DELETE

/students/{studentId}

Delete a student

GET

/courses

Get all courses

POST

/courses

Create a course

GET

/courses/{id}

Get a course

Students

GET

/students

Get all students

POST

/students

Create a student

GET

/students/{studentId}

Get a student

PATCH

/students/{studentId}

Update a student

DELETE

/students/{studentId}

Delete a student

GET

/notes/{studentId}

Get student notes

Courses

GET

/courses

Get all courses

POST

/courses

Create a course

GET

/courses/{id}

Get a course

37 / 85

Tags

- Tags **gruppieren** einzelne **Operationen**
- jeder API-Operation kann eine Liste von Tags zugewiesen werden
- **Übersichtlichkeit** und schnelles Finden von Endpunkten

```
tags:
  - Students
  - Courses
  - Notes

paths:
  /students/{id}:
    get:
      tags: [Students]
      ...
    patch:
      tags: [Students]
      ...
  /courses:
    get:
      tags: [Courses]
      ...
    post:
      tags: [Courses]
      ...
  /notes{id}:
    get:
      tags: [Notes, Students]
      ...
```

Komponenten

- `components` = wiederverwendbare Bausteine
- per Referenz `$ref` genutzt

Vorteile:

- Wiederverwendung
- Weniger Duplikate
- Konsistenz

```
paths:
  /student/{id}:
    get:
      ...
      responses:
        "200":
          $ref: "#/components/responses/Student"
      ...
components:
  parameters:
    ...
  schemas:
    ...
  responses:
    Student:
      ...
  requestBodies:
    ...
  examples:
    ...
  securitySchemes:
    ...
```

Komponenten

Weitere
Parameter

Payload

Tags

Komponenten

API Security

```
paths:
  /students:
    get:
      ...
      parameters:
        - in: query
          name: id
          schema:
            type: string
            description: Filter by course id
        - in: query:
          name: courseOfStudy
      ...
      responses:
        "200":
          description: Successful
          content:
            application/json:
              schema:
                type: array
                items:
                  type: object
                  properties:
                    id:
                      type: string
                  ...
```

ohne components

```
paths:
  /students:
    get:
      ...
      parameters:
        - $ref: "#/components/parameters/FilterByCourse"
        - $ref: "#/components/parameters/FilterByCourseOfStudy"
      responses:
        "200":
          description: Successful
          content:
            application/json:
              schema:
                type: array
                items:
                  $ref: "#/components/schemas/Student"
      ...
  components:
    parameters:
      FilterByCourse:
        description: Filter by course name
        name: courseName
        in: query
        schema:
          type: string
      ...
    schemas:
      Student:
        type: object
        properties:
          id:
            type: string
        ...
```

mit components

Komponenten

Weitere
Parameter

Payload

Tags

Komponenten

API Security

```
paths:
  /students:
    post:
      ...
      requestBody:
        required: true
        content:
          application/json:
            schema:
              type: object
              properties:
                id:
                  type: string
              ...
      responses:
        "201":
          description: Successful
          content:
            application/json:
              schema:
                type: object
                properties:
                  id:
                    type: string
                examples: [111111]
              ...
```

ohne components

```
paths:
  /students:
    post:
      requestBody:
        $ref: "#/components/requestBodies/CreateStudent"
      responses:
        "201":
          $ref: "#/components/responses/Student"
components:
  responses:
    Student:
      description: Successful
      content:
        application/json:
          schema:
            $ref: "#/components/schemas/Student"
          examples:
            newStudent:
              $ref: "#/components/examples/StudentExample"
  requestBodies:
    CreateStudent:
      required: true
      content:
        application/json:
          schema:
            $ref: "#/components/schemas/CreateStudent"
  examples:
    StudentExample:
      value:
        id: 111111
        ...
```

mit components

API Security

- **API Security** = Authentifizierung und Autorisierung
- verschiedene Sicherheitsschemata unterstützt
- zentrale **Begriffe**:
 - `securitySchemes:` Was ist möglich?
 - `security:` Was wird genutzt?
- **Geltungsbereich**: global oder pro Operation

```
paths:
  /students:
    post:
      ...
      security:
        - BasicAuth[]
  /notes/{studentId}:
    ...
    get:
      ...
      security:
        - AuthBearer[]

components:
  ...
  securitySchemes:
    BasicAuth:
      scheme: basic
      type: http

    AuthBearer:
      scheme: bearer
      type: http
      bearerFormat: jwt

    ApiKeyAuth:
      type: apiKey
      in: header
      name: X-API-Key
```

Praxis: Spezifikation Teil 2



Szenario auf Gitty <https://gitty.informatik.hs-mannheim.de/2121190/IWS-OpenAPI>

- GET, POST und PATCH Operationen
- Weitere Parameter definieren (siehe Szenario)
- Payload bei POST, PATCH
- Tags verwenden
- Sinnvolle Komponenten einfügen
- Security für API definieren

Scan | Audit | v3_1_2020.json
1 openapi: 3.1.0
2 info:
3 title: Students API
4 version: 1.0.0
5
6 servers:
7 - url: http://localhost:4004/api/v1
8 description: Local development servers
9 - url: https://{stage}.students.com/api/v1
10 description: Stage server
11 variables:
12 stage:
13 description: Stage
14 enum:
15 - dev
16 - test
17 - prod
18 default: dev
19
20 paths:
21 /students:
22 get:
23 Scan | Audit
24 summary: Get all students
25 description: Retrieves all the students in the system
26 responses:
27

Students API 1.0.0 OAS 3.1
<http://openapi-preview.example/>

Servers
http://localhost:4004/api/v1 - Local development servers

default
GET /students Get all students
GET /students/{id}

Mittagspause (45 min)

7. Design First

Was ist Design First?

Design First

- OpenAPI Spezifikation wird in einem Texteditor geschrieben
- Aus der Spezifikation wird ein REST Api Client erstellt
- Codegenerierung von Serverrouten möglich

Felder

- ◆ OpenAPI version
Version der Spezifikation
- ◆ info (title, description, version)
Name der API-Schnittstelle (Beschreibung, Version)
- ◆ servers (url)
- ◆ paths / Endpunkte
- ◆ components (SecuritySchemes)
- ◆ schemas / Formatierung der Parameter

Vor- und Nachteile

Vorteile

- ✓ Technologieunabhängigkeit
- ✓ Keine Programmierkenntnisse erforderlich
- ✓ Implementierung an API gebunden

Nachteile

- Projektplanung

GUI Editoren

Apicurio Registry

APIGit

OpenAPI
Designer

OpenAPI
Specifications
Plugin

- ❖ Apicurio Studio wird nicht mehr unterstützt
- ❖ Funktionalität zur Bearbeitung von API Dokumenten über “Draft”-Feature

```
curl -o docker-compose.yml  
https://raw.githubusercontent.com/Apicurio/apicurio-registry/main/distro/doc  
ker-compose/in-memory-with-studio/docker-compose.yml
```

The Solution: Studio Features Built into Registry 3.1.0

Rather than leaving Studio users without a path forward, we’ve directly incorporated Studio’s editing capabilities into the Apicurio Registry user experience. Starting with **Apicurio Registry 3.1.0**, all of the core Studio functionality is available as an opt-in feature within Registry itself.

This means you get the best of both worlds:

- The robust schema/API management capabilities of Registry
- The intuitive editing experience of Studio
- A single project to install, configure, and maintain
- Unified lifecycle management for your API designs

How It Works: The New “Drafts” Feature

The Studio functionality in Registry is built on top of a powerful new feature called **Drafts**. Let me explain how this works.

GUI Editoren

Apicurio Registry

APIGit

OpenAPI
Designer

OpenAPI
Specifications
Plugin

Installation über Docker

```
$ curl -o docker-compose.yml  
https://raw.githubusercontent.com/Apicurio/apicurio-registry/main/dist  
ro/docker-compose/in-memory-with-studio/docker-compose.yml  
  
$ docker compose up
```

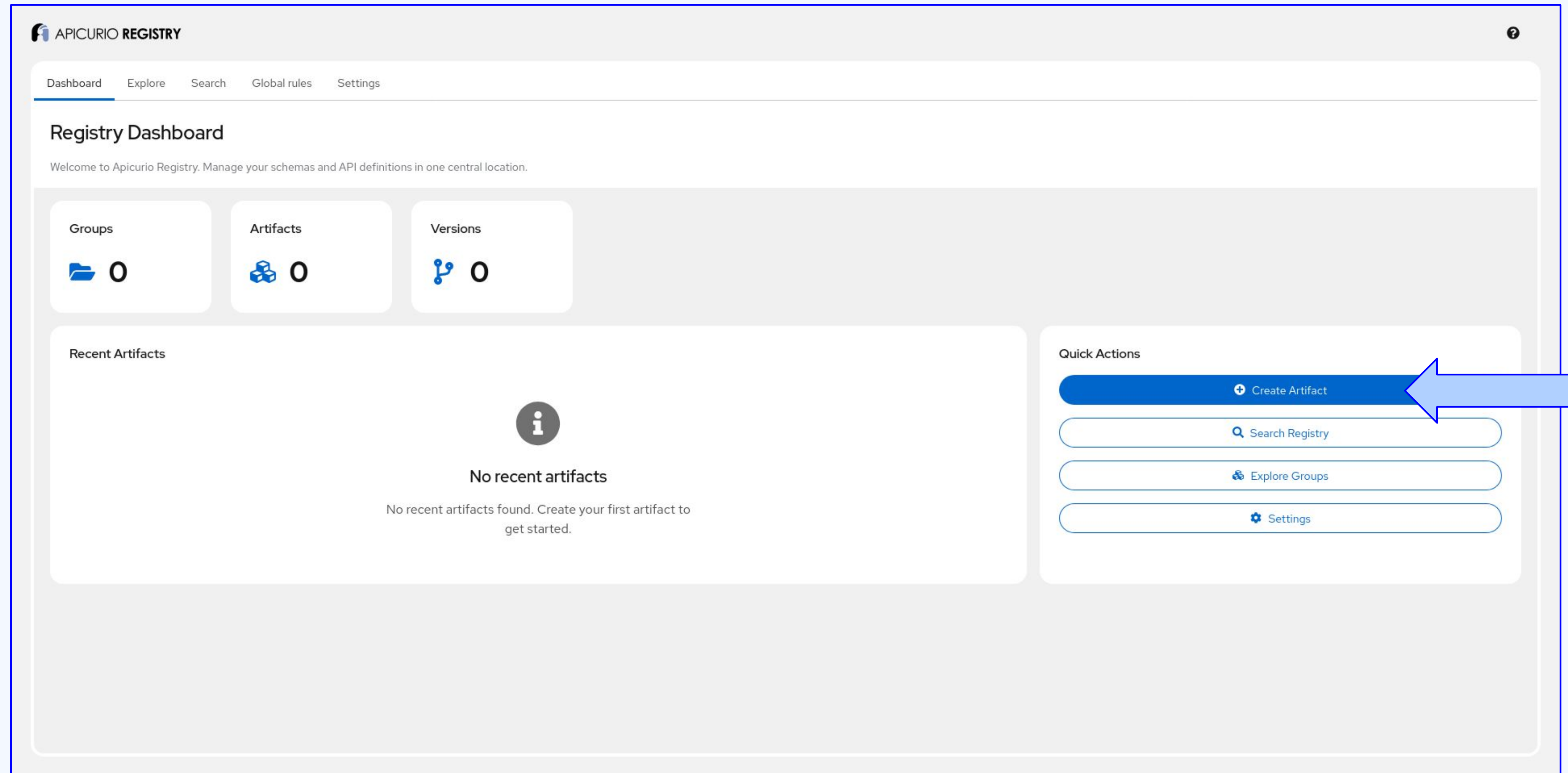
GUI Editoren

Apicurio Registry

APIGit

OpenAPI
Designer

OpenAPI
Specifications
Plugin



GUI Editoren

Apicurio Registry

APIGit

OpenAPI
Designer

OpenAPI
Specifications
Plugin

APICURIO REGISTRY

Dashboard Explore Search Global rules Settings

Registry Dashboard

Welcome to Apicurio Registry. Manage your schemas and artifacts.

Groups

0

Artifacts

0

Recent Artifacts

Create Artifact

- Artifact Coordinates
- Artifact Metadata
- Version Content**
- Version Metadata (optional)

Version Number

1.0.0 (optional) will be generated if left blank

Content

From file From URL

Drag a file here or browse to upload

```
openapi: 3.1.0
info:
  title: Food Express API
  description: >
    API for managing the FoodExpress restaurants, orders, and customers.
  version: 1.0.0

servers:
  - url: http://localhost:4004/api/v1
```

Note: If version content is not provided, an empty artifact (no versions) will be created.

Back Next Cancel

GUI Editoren

Apicurio Registry

APIGit

OpenAPI
Designer

OpenAPI
Specifications
Plugin

The screenshot displays the Apicurio Registry web interface. The top navigation bar includes 'Dashboard', 'Explore', 'Search', 'Global rules', and 'Settings'. The breadcrumb trail shows 'Explore > default > a8208b6a-146d-4946-8c06-6007c8d4122e > 1'. The main content area shows the API ID 'a8208b6a-146d-4946-8c06-6007c8d4122e / 1' with a 'Download' button. Below this, there are tabs for 'Overview', 'Documentation', 'Content', and 'References'. The 'Content' tab is active, displaying a YAML OpenAPI specification. On the right side of the code editor, there are tabs for 'JSON' and 'YAML', with 'YAML' currently selected.

```
1 openapi: 3.1.0
2 info:
3   title: Food Express API
4   description: >
5     API for managing the FoodExpress restaurants, orders, and customers.
6   version: 1.0.0
7
8 servers:
9   - url: http://localhost:4004/api/v1
10     description: Local development servers
11   - url: https://{stage}.foodexpress.com/api/v1
12     description: Stage server
13     variables:
14       stage:
15         description: Stage
16         enum:
17           - dev
18           - test
19         default: dev
20   - url: https://foodexpress.com/api/v1
21     description: Production server
22
23 tags:
24   - name: Restaurants
25   - name: Orders
26   - name: Customers
27
28 paths:
29   /restaurants:
30     get:
31       tags: [Restaurants]
32       summary: List restaurants
33       description: Retrieve a paginated list of restaurants.
34       security: []
35       parameters:
```

- ❖ JSON und YAML Ansicht
- ❖ Editieren nicht möglich

GUI Editoren

Apicurio Registry

APIGit

OpenAPI
Designer

OpenAPI
Specifications
Plugin

APICURIO REGISTRY

Dashboard

Explore

Search

Global rules

Settings

Explore

default

a8208b6a-146d-4946-8c06-6007c8d4122e

> 1

a8208b6a-146d-4946-8c06-6007c8d4122e

/ 1

Download

Overview

Documentation

Content

References

Search...

Restaurants

GET List restaurants

GET Get restaurant menu

GET List orders for a restaurant

Orders

Customers

Get restaurant menu

PATH PARAMETERS

id

required

string

Example: 09ede9cb-0031-4469-9144-dec6b564f1c0

HEADER PARAMETERS

Accept - Language

string

Example: de-DE

Preferred language for localized strings.

Responses

> 200 Menu for the restaurant

— 404 Restaurant not found

List orders for a restaurant

Retrieve a paginated list of orders for a specific restaurant.

GET /restaurants/{id}/menu

Response samples

200

Content type

application/json

Copy Expand all Collapse all

[
- {
 "id": "9f062baf-236b-44fe-8d30-88cecb02c26b",
 "name": "Cheeseburger",
 "price": 3.99
}]

GET /restaurants/{id}/orders

Response samples

Dokumentation

GUI Editoren

Apicurio Registry

APIGit

OpenAPI
Designer

OpenAPI
Specifications
Plugin

The screenshot shows a 'Generate client SDK' dialog box with the following fields and options:

- Language:** A dropdown menu currently set to 'Python'.
- Generated client class name:** A text input field containing 'MySdkClient'.
- Generated client package name:** A text input field containing 'io.example.sdk'.
- Hide advanced options:** A toggle switch currently turned off.
- Include paths:** A section with a description: 'Enter a comma-separated list of patterns to specify the paths used to generate the client SDK (for example, /., **/my-path/*)'. It includes a text input field with the placeholder 'Enter path1, path2, ...' and a note: 'If this field is empty, all paths are included'.
- Exclude path patterns:** A section with a description: 'Enter path1, path2, ...'. It includes a text input field with the placeholder 'Enter path1, path2, ...' and a note: 'If this field is empty, no paths are excluded'.
- Buttons:** 'Generate and download' (blue) and 'Cancel' (grey).

- ❖ Generierung des Clients
- ❖ Unterstützung für verschiedene Programmiersprachen

GUI Editoren

Apicurio Registry

APIGit

OpenAPI
Designer

OpenAPI
Specifications
Plugin

The screenshot shows a 'Create Draft' dialog box with a close button (x) in the top right corner. On the left, there is a vertical list of steps: '1 Draft Coordinates', '2 Draft Content' (which is highlighted with a blue circle and background), and '3 Draft Metadata'. The main area of the dialog has three tabs: 'From template' (selected with a blue underline), 'From local file', and 'From URL'. Below the tabs, there is a text prompt: 'Select from the list of templates below to create a new Draft from scratch, or from a common/example starter.' A list of templates is shown in a box: 'Blank OpenAPI 3.x API' with the description 'An empty API descriptor using OpenAPI 3.0.3.', and 'Pet Store Example' with the description 'An example OpenAPI specification that defines a simple Pet Store service.' At the bottom of the dialog, there are three buttons: 'Back' (outlined), 'Next' (solid grey), and 'Cancel' (outlined).

Drafts können editiert werden

GUI Editoren

Apicurio Registry

APIGit

OpenAPI
Designer

OpenAPI
Specifications
Plugin

Explore > default > 1a156f40-5566-430e-a8a8-981f926432ad > 1 > Editor

Food Express API

Search everything...

Paths (9)

/customers

/customers/{id}

/customers/me

/orders

/orders/{orderId}

/orders/{orderId}/status

/restaurants

/restaurants/{id}/menu

/restaurants/{id}/orders

Data Types (12)

</> CreateCustomerRequest

</> CreateOrderRequest

</> Customer

</> Menu

</> MenuItem

</> Order

</> OrderItem

</> OrderStatus

</> Pagination

</> Restaurant

</> UpdateCustomerRequest

</> UpdateOrderStatusRequest

Responses (1)

/customers/{id}

Design Source

VENDOR EXTENSIONS

OPERATIONS (3)

Get

Put

Post

Delete

Options

Head

Patch

Trace

INFO

Summary

Get customer by ID

Description

No description.

Tags

Customers

Operation ID

No Operation ID

Not Deprecated

SERVERS

PATH PARAMETERS (1)

id

No description.

No Type

QUERY PARAMETERS

57 / 85

GUI Editoren

Apicurio Registry

APIGit

OpenAPI
Designer

OpenAPI
Specifications
Plugin

Textuelle Bearbeitung

The screenshot displays the OpenAPI Designer interface for the 'Food Express API'. On the left sidebar, there is a search bar and two expandable sections: 'Paths (9)' and 'Data Types (12)'. The 'Paths' section lists endpoints such as '/customers', '/customers/{id}', and '/orders'. The 'Data Types' section lists types like 'CreateCustomerRequest', 'CreateOrderRequest', 'Customer', 'Menu', 'MenuItem', 'Order', and 'OrderItem'. The main area on the right is titled '/customers' and has tabs for 'Design' and 'Source'. The 'Source' tab is active, showing a JSON OpenAPI specification for a POST endpoint. The specification includes details for the request body (application/json), a schema reference to 'CreateCustomerRequest', an example reference to 'CustomerRequest', a required flag, a tag 'Customers', a parameter reference to 'IdempotencyKey', and a 201 response with a schema reference to 'Customer' and a summary 'Create a customer'.

```
1 post:
2   requestBody:
3     content:
4       application/json:
5         schema:
6           $ref: '#/components/schemas/CreateCustomerRequest'
7         examples:
8           Customer:
9             $ref: '#/components/examples/CustomerRequest'
10        required: true
11      tags:
12        - Customers
13      parameters:
14        -
15          $ref: '#/components/parameters/IdempotencyKey'
16      responses:
17        '201':
18          content:
19            application/json:
20              schema:
21                $ref: '#/components/schemas/Customer'
22              description: Customer created
23      summary: Create a customer
24
```

GUI Editoren

Apicurio Registry

APIGit

OpenAPI
Designer

OpenAPI
Specifications
Plugin

- ❖ Funktionalität ähnlich wie Apicurio Registry
- ❖ weitere Features: Git-Integration, Sicherheitshinweise, Testen der Routen

Issues

Publish Center

Settings

+

API GROUPS

pet

Update an existing pet PUT

Add a new pet to the st... POST

Finds Pets by status GET

Finds Pets by tags GET

Find pet by ID GET

Updates a pet in the st... POST

Deletes a pet DEL

uploads an image POST

store >

user >

COMPONENTS

Schemas >

Update an existing pet

PUT /pet TRY IT

Update an existing pet by Id

AUTHORIZATIONS: > petstore_auth

REQUEST BODY SCHEMA: application/json required

Update an existent pet in the store

id integer <int64>

name required string

category > object (Category)

photoUrls required Array of strings

tags > Array of objects (Tag)

status string
Enum: "available" "pending" "sold"
pet status in the store

Request samples

Payload Request Code Snippets

application/json

```
{  "id": 10,  "name": "doggie",  + "category": { ... },  + "photoUrls": [ ... ],  + "tags": [ ... ],  "status": "available"}
```

Response samples

200

application/xml

No sample

GUI Editoren

Apicurio Registry

APIGit

OpenAPI
Designer

OpenAPI
Specifications
Plugin

Commit Publish

Form Code Diff HEAD Governance Preview

search

Info

Tags

Servers

Security (Authentication)

Paths 13

Webhooks

components

Schema 8

Order

Customer

Address

Category

Name*: Customer

Schema*:

Generate Schema

object

description

id integer <int64> description

username string description

address array [\$ref] description

Items \$ref #/components/schemas/... description

OWASP Top 10 Security

21 Declare int

77 Operation i

77 Operation is missing rate limiting response in responses[429].content.

77 Operation is missing responses[401].

77 Operation is missing responses[401].content.

77 Operation is missing responses[500].

77 Operation is missing responses[500].content.

78 All 2XX and 4XX responses should define rate limiting headers.

93 All 2XX and 4XX responses should define rate limiting headers.

96 All 2XX and 4XX responses should define rate limiting headers.

99 All 2XX and 4XX responses should define rate limiting headers.

140 Operation is missing rate limiting response in responses[429].

GUI Editoren

Apicurio Registry

APIGit

OpenAPI Designer

OpenAPI
Specifications
Plugin

OpenAPI 3 Definitions Designer

NEW SCHEMANEW PATH

JSONYAMLSAVE

JSONYAML

```
{
  "openapi": "3.0.0",
  "info": {
    "version": "1",
    "title": "",
    "description": ""
  },
  "paths": {},
  "components": {
    "securitySchemes": {}
  }
}
```

SchemasPathsSecurity

You have no Schemas
Schemas represent complex data types

Schema

USE SCHEMACANCEL

Name

Properties

Name

Type

string

Nullable

☐

ADD PROPERTY

- ❖ Website: <https://openapidesigner.com>
- ❖ geringere Funktionalität

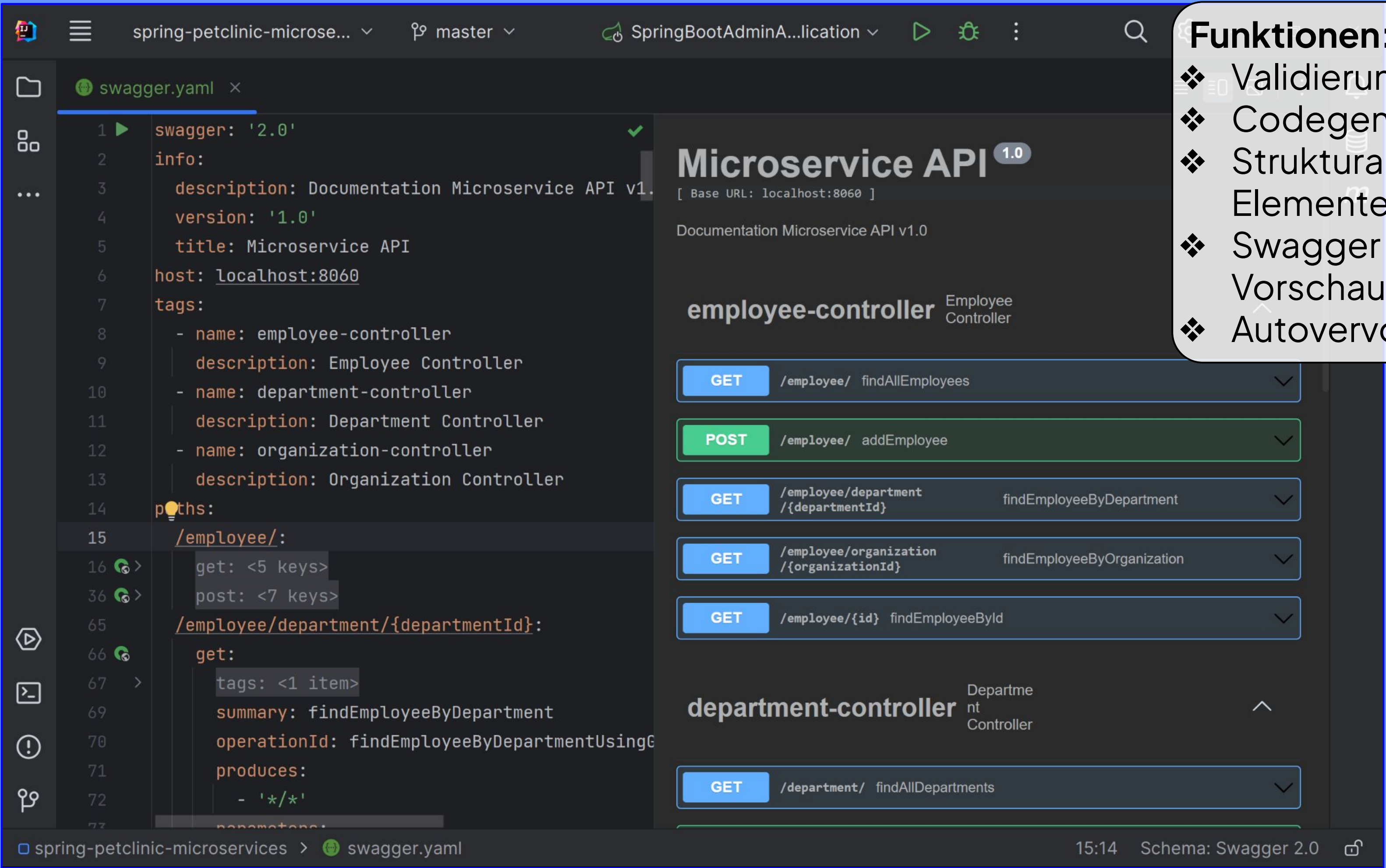
GUI Editoren

Apicurio Registry

APIGit

OpenAPI
Designer

OpenAPI
Specifications
Plugin



- Funktionen:**
- ❖ Validierung
 - ❖ Codegenerierung
 - ❖ Strukturansicht der Elemente (z. B. Routen)
 - ❖ Swagger UI / Redoc Vorschau
 - ❖ Autovervollständigung

Validierung

Überprüfung der Kompatibilität eines OpenAPI Dokument mit der Spezifikation

OpenAPI Spec Validator (PyPI)

```
openapi-spec-validator openapi.yaml
```

Openapi-schema-validator (PyPI)

```
validate({"name": "John", "city": "London"},  
schema)
```

Codegenerierung

OpenAPI Generator (<https://openapi-generator.tech/>)

- ❖ Unterstützung für über 40 verschiedene Server- und Clienttechnologien

Installation (alternativ):

```
npm install @openapitools/openapi-generator-cli -g
```

```
pip install openapi-generator-cli[jdk4py]
```

```
Docker: openapitools/openapi-generator-cli
```

Generatoren (Auswahl):

[python-aiohttp](#), [python-blueplanet](#), [python-fast api \(beta\)](#), [python-flask](#)

```
$ openapi-generator-cli generate -i input.yaml -g [generator] -o client
```



Beispiel Server/Client

```
openapi: 3.0.1
info:
  title: Book Management API
  description: API for managing books in a digital library
  version: 1.0.0

servers:
- url: http://localhost:8080
  description: Local development server

paths:
  /books:
    get:
      summary: Get all books
      responses:
        '200':
          description: A list of books
          content:
            application/json:
              schema:
                type: array
                items:
                  $ref: '#/components/schemas/Book'
```

```
/books/{id}:
  get:
    summary: Get a book by ID
    parameters:
      - in: path
        name: id
        schema:
          type: integer
          required: true
          description: ID of the book to retrieve
    responses:
      '200':
        description: Book found
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Book'
      '404':
        description: Book not found

components:
  schemas:
    Book:
      type: object
      properties:
        id:
          type: integer
          example: 1
        title:
          type: string
          example: Effective Java
        author:
          type: string
          example: Joshua Bloch
```

Praxis (Server/Client)



VS Code: Installation z. B. OpenAPI Toolkit (SpecLynx) für Syntax Highlighting

→ Terminal öffnen

```
$ uv run openapi-generator-cli generate -i openapi.yaml -g python-fastapi -o server
```

```
$ cd server
```

```
$ uv sync
```

```
$ uv add -dev fastapi uvicorn
```

README:

```
PYTHONPATH=src uv run uvicorn openapi_server.main:app --host 0.0.0.0 --port 8080
```

Danach im Browser <http://127.0.0.1:8080/docs> öffnen

Praxis (Server/Client)



VS Code: Installation z. B. OpenAPI Toolkit (SpecLynx) für Syntax Highlighting

→ Terminal öffnen

```
$ uv run openapi-generator-cli generate -i openapi.yaml -g python -o client  
$ cd client  
$ uv sync
```

README.md

Praxis (Server/Client)

```
import openapi_client
from openapi_client.rest import ApiException
from pprint import pprint

# Defining the host is optional and defaults to http://localhost:8080
# See configuration.py for a list of all supported configuration parameters.
configuration = openapi_client.Configuration(
    host = "http://localhost:8080"
)

# Enter a context with an instance of the API client
with openapi_client.ApiClient(configuration) as api_client:
    # Create an instance of the API class
    api_instance = openapi_client.DefaultApi(api_client)

    try:
        # Get all books
        api_response = api_instance.books_get()
        print("The response of DefaultApi->books_get:\n")
        pprint(api_response)
    except ApiException as e:
        print("Exception when calling DefaultApi->books_get: %s\n" % e)
```

Praxis (Server/Client)

uv run client.py

Exception when calling DefaultApi->books_get: (500)

Reason: Internal Server Error

HTTP response headers: HTTPHeaderDict({'date': 'Thu, 05 Feb 2026 23:00:52 GMT', 'server': 'uvicorn', 'content-length': '28', 'content-type': 'application/json'})

HTTP response body: {"detail": "Not implemented"}

➡ Dies liegt daran, dass die Funktion “books_get” nicht im Code implementiert wurde

Praxis (Server/Client)

```
@router.get(  
    "/books",  
    responses={  
        200: {"model": List[Book], "description": "A list of books"},  
    },  
    tags=["default"],  
    summary="Get all books",  
    response_model_by_alias=True,  
)  
  
async def books_get(  
    ) -> List[Book]:  
    if not BaseDefaultApi.subclasses:  
        raise HTTPException(status_code=500, detail="Not implemented")  
    return await BaseDefaultApi.subclasses[0]().books_get()
```

Praxis: Design First



Gegeben ist die API-Dokumentation für FoodExpress:

<https://gitty.informatik.hs-mannheim.de/2121190/IWS-OpenAPI>

- Generiere einen Python Client und einen Python FastAPI Server
- Es soll das Tool openapi-generator-cli verwendet werden
- Ergänze die Implementierung für die Routen des Python FastAPI Servers
- Überprüfe jeweils die Funktionalität des Servers und Clients mit Mockdaten

Pause (10 min)?

8. Code First

Code First – Grundidee & Einordnung

Was bedeutet Code First?

Code First beschreibt einen Ansatz, bei dem:

- die **API zuerst implementiert wird**
- die **API-Dokumentation automatisch aus dem Code entsteht**
- der **Code die Quelle der Wahrheit** ist

Die OpenAPI-Dokumentation wird dabei z. B. aus:

- Typannotationen
- Modellen
- Annotationen / Metadaten generiert.

Eigenschaften von Code First

- Schneller Einstieg
- Wenig initialer Overhead
- Sehr nah an der Implementierung
- Dokumentation und Code bleiben synchron

Typische Einsatzszenarien

- Prototypen
- Kleine bis mittlere Teams
- Bestehende APIs
- Backend-getriebene Entwicklung

FastAPI & Typannotationen

FASTAPI: Typen sind der Vertrag

FastAPI ist ein Python-Webframework, das stark auf **Typannotationen** setzt.

Was bedeutet das konkret?

- ➡ Python-Typen beschreiben:
 - Request-Parameter
 - Request-Body
 - Response-Strukturen

- ➡ Diese Typinformationen werden automatisch genutzt für:
 - **Validierung**
 - **OpenAPI-Dokumentation**
 - **Swagger UI**

Beispiel

```
@app.get("/users/{user_id}")
def get_user(user_id: int):
    ...
```



- ➡ user_id **muss** eine Zahl sein
- ➡ FastAPI validiert automatisch
- ➡ OpenAPI kennt den Typ
- ➡ Swagger zeigt es korrekt an

FastAPI App & CRUD (Code First)

app = FastAPI()

- app ist die zentrale API-Instanz
- Alle Endpoints werden daran registriert
- OpenAPI wird **automatisch aus der App erzeugt**

	Operation	HTTP	Beispiel
C	Create 	POST	/users
R	Read 	GET	/users, /users/{id}
U	Update 	PUT/PATCH	/users/{id}
D	Delete 	DELETE	/users/{id}

Beispiel: Create User

```
@app.post("/users")
def create_user(user: UserCreate):
    ...
```



- UserCreate = Pydantic-Model
- Definiert:
 - Pflichtfelder
 - Datentypen
 - Validierung
- OpenAPI übernimmt das Schema automatisch

Praxis: Code First



Szenario auf Gitty <https://gitty.informatik.hs-mannheim.de/2121190/IWS-OpenAPI>

- Server starten
- JSON prüfen
- User-Objekt anpassen.
- Testen

```
code_first.py > ...
1  from fastapi import FastAPI, HTTPException, Path, Query, status
2  from pydantic import BaseModel, Field
3  from typing import List, Optional
4
5  app = FastAPI(
6      title="User Service API",
7      description="Simple CRUD API to demonstrate Code First OpenAPI generation",
8      version="1.0.0"
9  )
10
11  # -----
12  # Models
13  # -----
14
15  class UserBase(BaseModel):
16      name: str = Field(..., example="Alice")
17      age: int = Field(..., example="22")
18      email: str = Field(..., example="alice@example.com")
19
20  class UserCreate(UserBase):
21      pass
22
```

User Service API 1.0.0 OAS 3.1

[/openapi.json](#)

Simple CRUD API to demonstrate Code First OpenAPI generation with FastAPI

Users

POST

/users

Create a new user

▼

GET

/users

List all users

▼

GET

/users/{user_id}

Get user by ID

▼

PUT

/users/{user_id}

Update a user

▼

DELETE

/users/{user_id}

Delete a user

▼

9. Ausblick

Was ist sonst noch möglich?

Beschleunigung &
Qualität (Dev)

Management &
Betrieb (Ops)

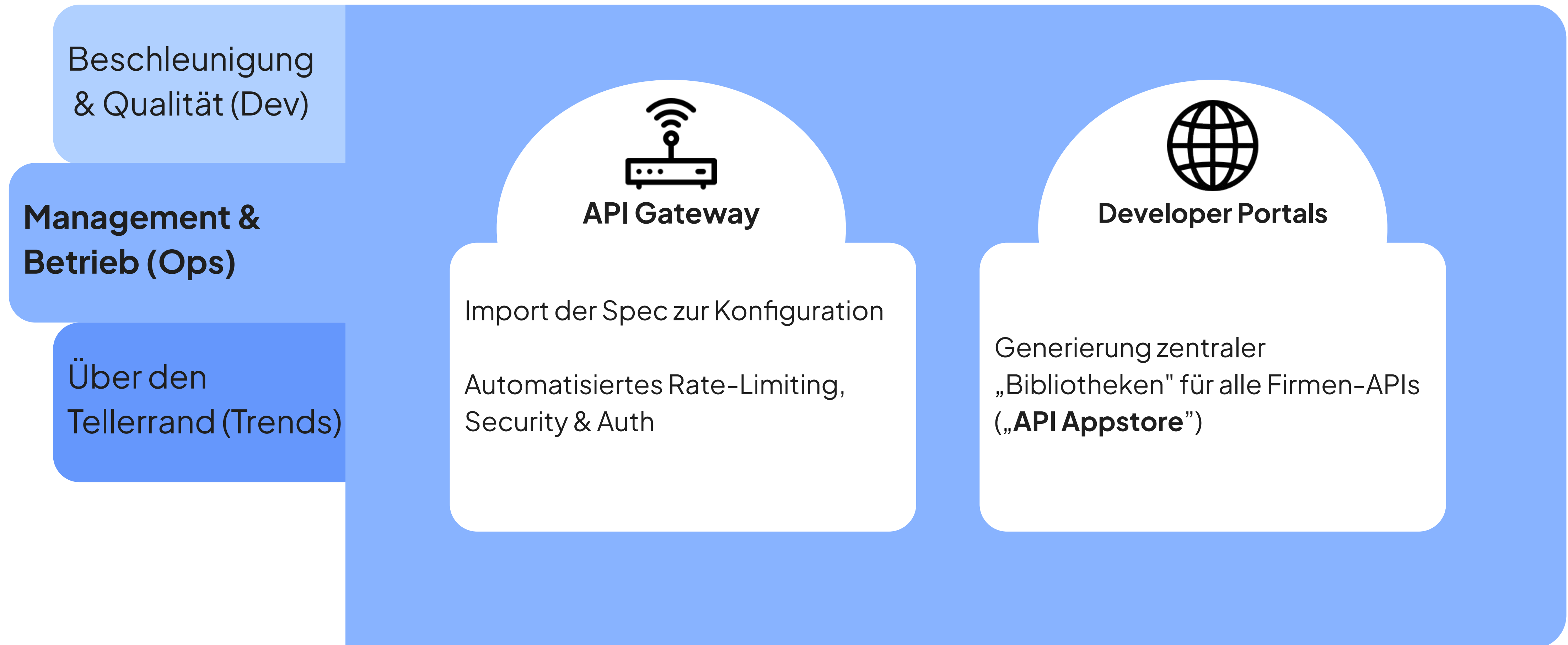
Über den
Tellerrand (Trends)



Contract Testing

Prüft automatisch: Stimmt der
Code noch mit der Doku überein?
(**Verhindert „Drift“**)

Was ist sonst noch möglich?

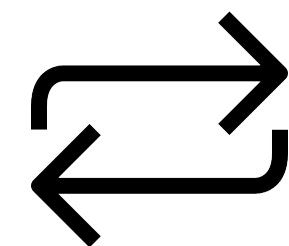


Was ist sonst noch möglich?

Beschleunigung
& Qualität (Dev)

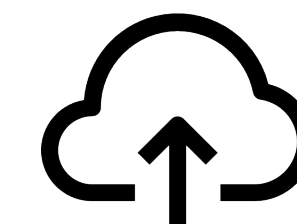
Management &
Betrieb (Ops)

**Über den Tellerrand
(Trends)**



AsyncAPI

Wie OpenAPI, aber für **asynchrone Events** (Kafka, WebSockets, RabbitMQ)



SwaggerHub & Cloud

Zentrale Plattform für Teams
(**Versionierung, Kollaboration,**
Style-Guides)

10. Zusammenfassung

Zusammenfassung

Standard für REST-APIs

Spezifikation als „**Vertrag**“ zwischen Frontend, Backend und externen Partnern

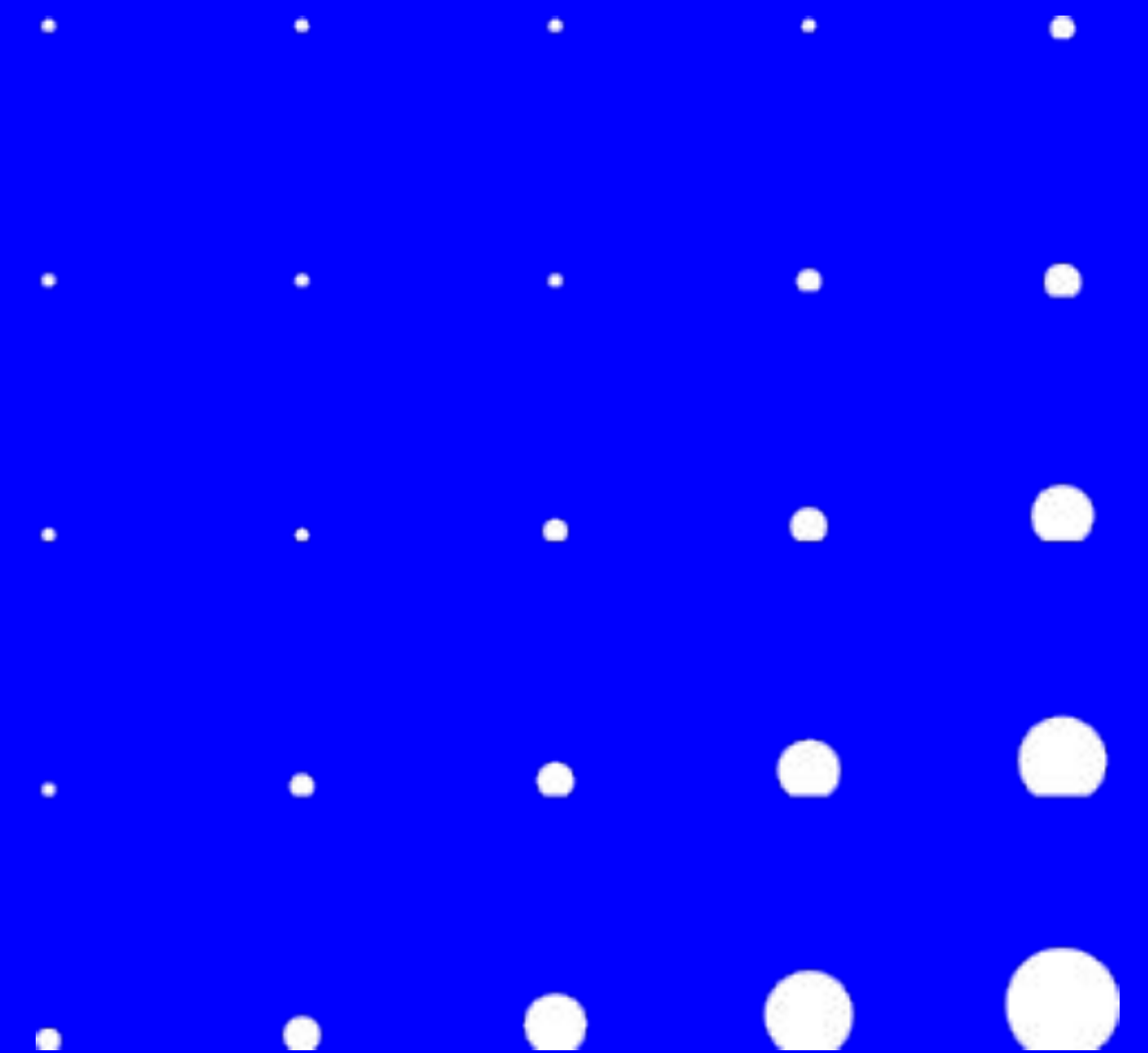
Workflow-Flexibilität:

Design First: Erst planen, dann bauen (Team-Kollaboration).

Code First: Dokumentation direkt aus dem Code generieren (schnelle Entwicklung).

Tools: **Kein** YAML **auswendig lernen** – nutzt Editoren, Linter und Plugins, um die Qualität eurer Spezifikation zu sichern.

Noch Fragen?



Quellen

- https://swagger.io/docs/specification/v3_0/about/
- <https://learn.openapis.org/specification/>
- https://idratherbewriting.com/learnapidoc/pubapis_openapi_tutorial_overview.html
- <https://github.com/OAI/OpenAPI-Specification/blob/main/versions/3.1.0.md>
- <https://www.javacodegeeks.com/openapi-documentation-yml-file-example.html>
- <https://yaml.org/spec/1.2.2/>