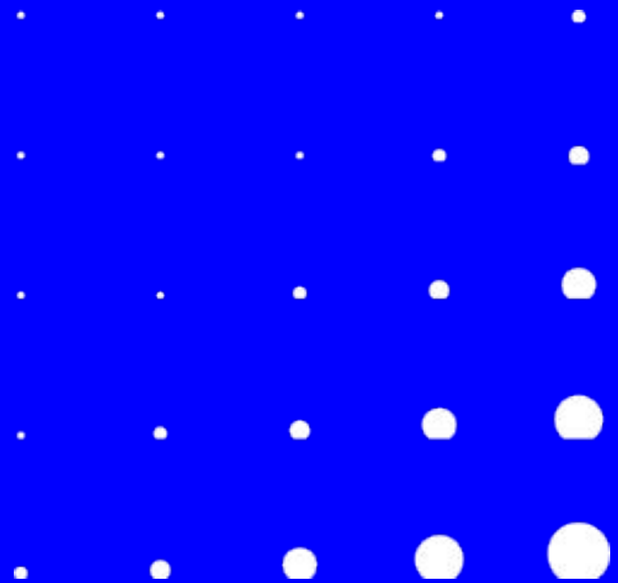
Technische Hochschule
mannheim

API-Design und Dokumentation mit OpenAPI

Béla Gallin
Serhat Göl
Fabian Hoppe
David Kubek
Jasmin Tschernoch



10.02.2026 | Team OpenAPI | Informatik-Workshop

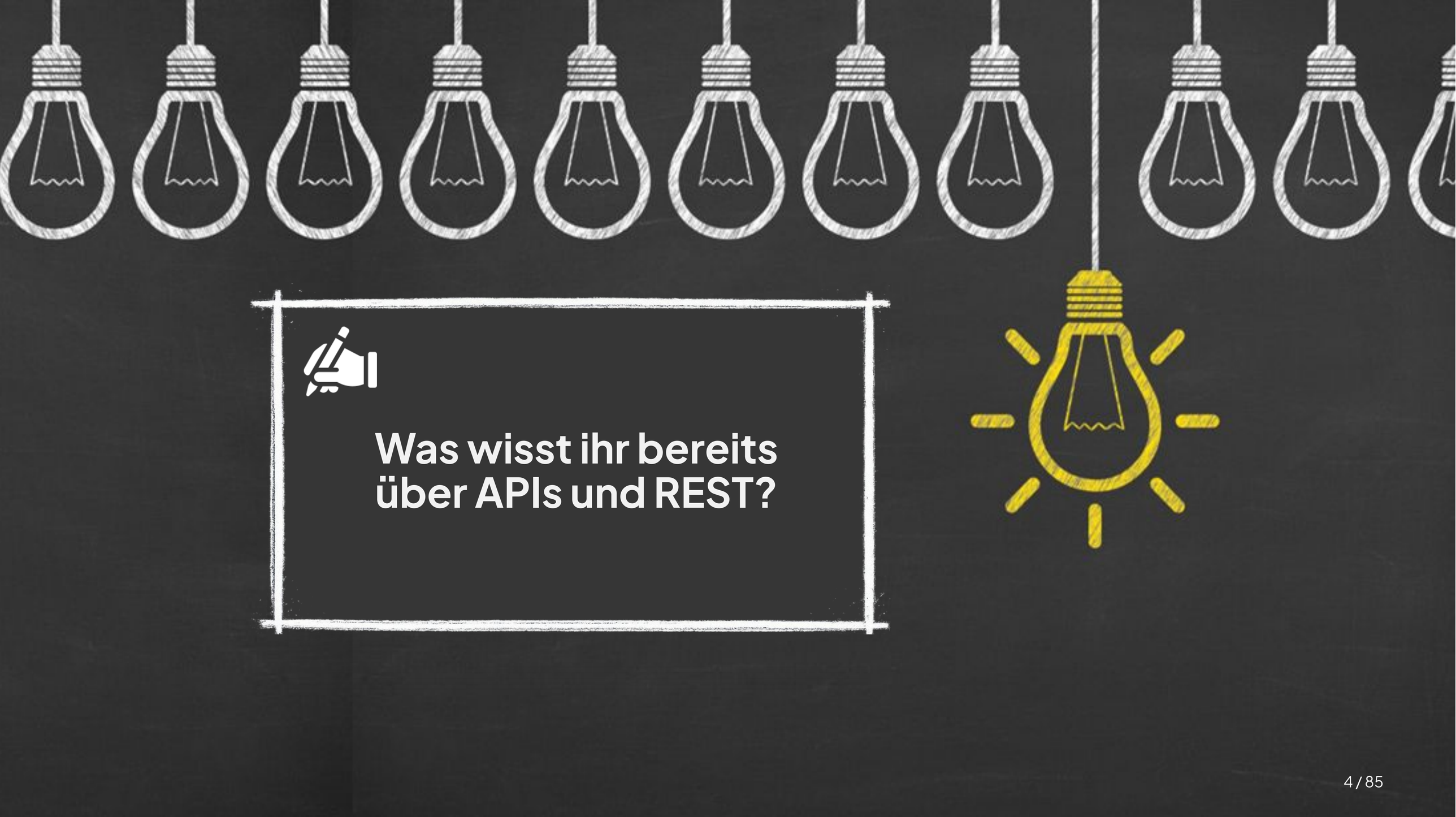
Agenda

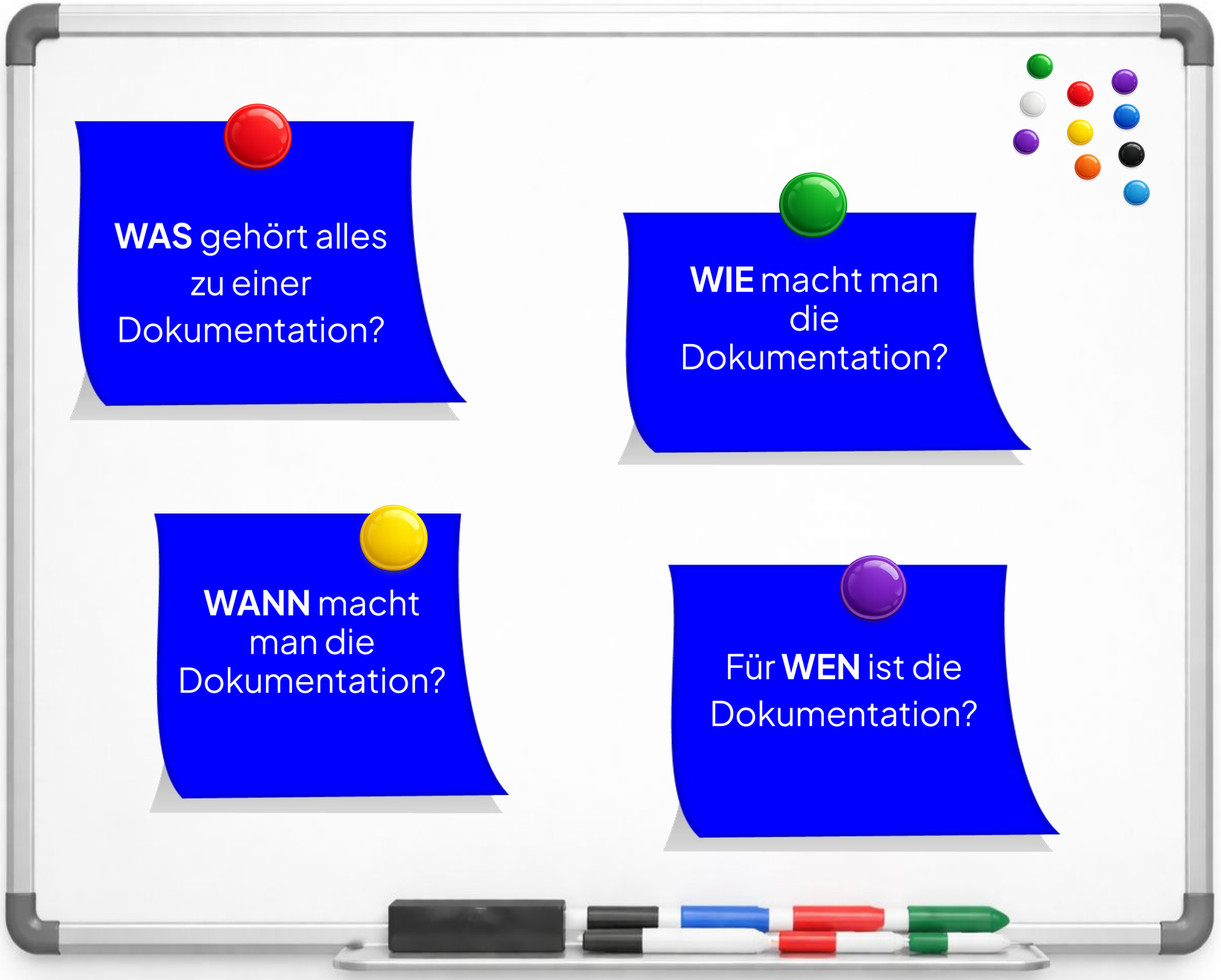
1. Wissensstand
2. Szenario und Erste Dokumentation
3. OpenAPI Einführung
4. Fertige OpenAPI-Dokumentation - Da wollen wir hin
5. Voraussetzungen und Tools
6. OpenAPI Spezifikation
7. Design First
8. Code First
9. Ausblick
10. Zusammenfassung

Technische Hochschule mannheim

Technische Hochschule
mannheim

1. Wissensstand


Was wisst ihr bereits
über APIs und REST?



Technische Hochschule mannheim

Wie könnt ihr eine (REST)API dokumentieren?

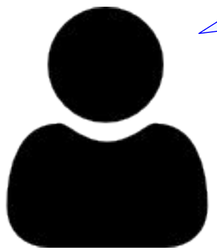
Technische Hochschule
mannheim

2. Szenario und Erste Dokumentation

Technische Hochschule
mannheim

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.



Definiert diese API schriftlich!

Restaurants	Bestellungen	Kunden
Verwaltung und Anzeige von Restaurants und deren Speisekarten	Erstellen, Abrufen und Verwaltung von Bestellungen	Kundenverwaltung und Profilinformationen

Technische Hochschule
mannheim

3. OpenAPI Einführung

Was ist OpenAPI?

OpenAPI

Standardisiere 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

ca. 2010

Problem:

- ➔ REST-APIs wurden immer verbreiteter
- ➔ Keine einheitliche, maschinenlesbare Beschreibung
- ➔ Dokumentation oft manuell, inkonsistent oder veraltet

2011

Lösung:

- ➔ **Swagger** entstand als **Spezifikation** und **Tooling**
- ➔ Ziel: APIs beschreibbar, testbar und verständlich zu machen

2015 & 2016

Weiterentwicklung:

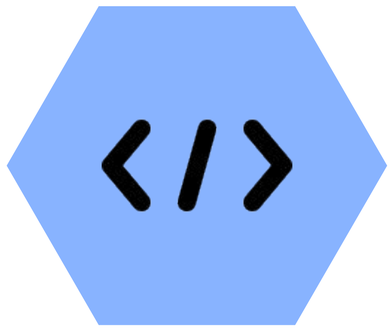
- ➔ Übergabe des Swagger Spezifikation an die **Linux Foundation**
- ➔ Umbenennung & Standardisierung als **OpenAPI Specification**

Heute

- ➔ **OpenAPI** = offizieller Standard
- ➔ **Swagger** = Tool-Ökosystem rund um OpenAPI

Was kann man mit OpenAPI machen?

Interaktive Doku (Swagger UI)

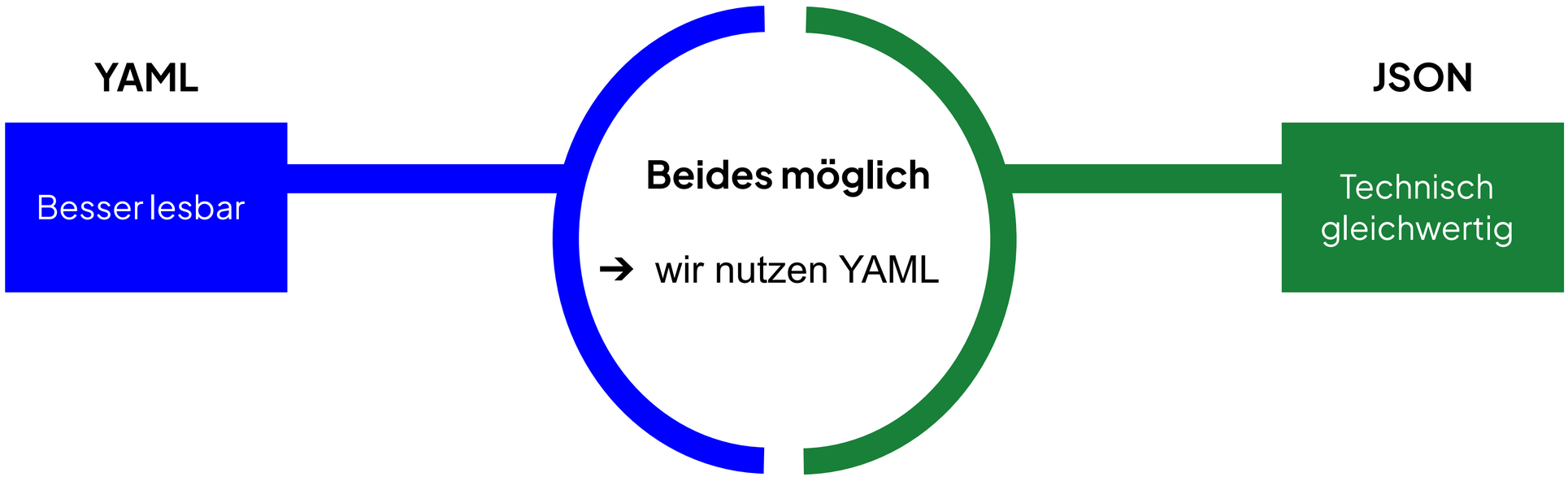


Client-Code generieren

Mock-Server & Tests

API designen & reviewen

YAML vs. JSON

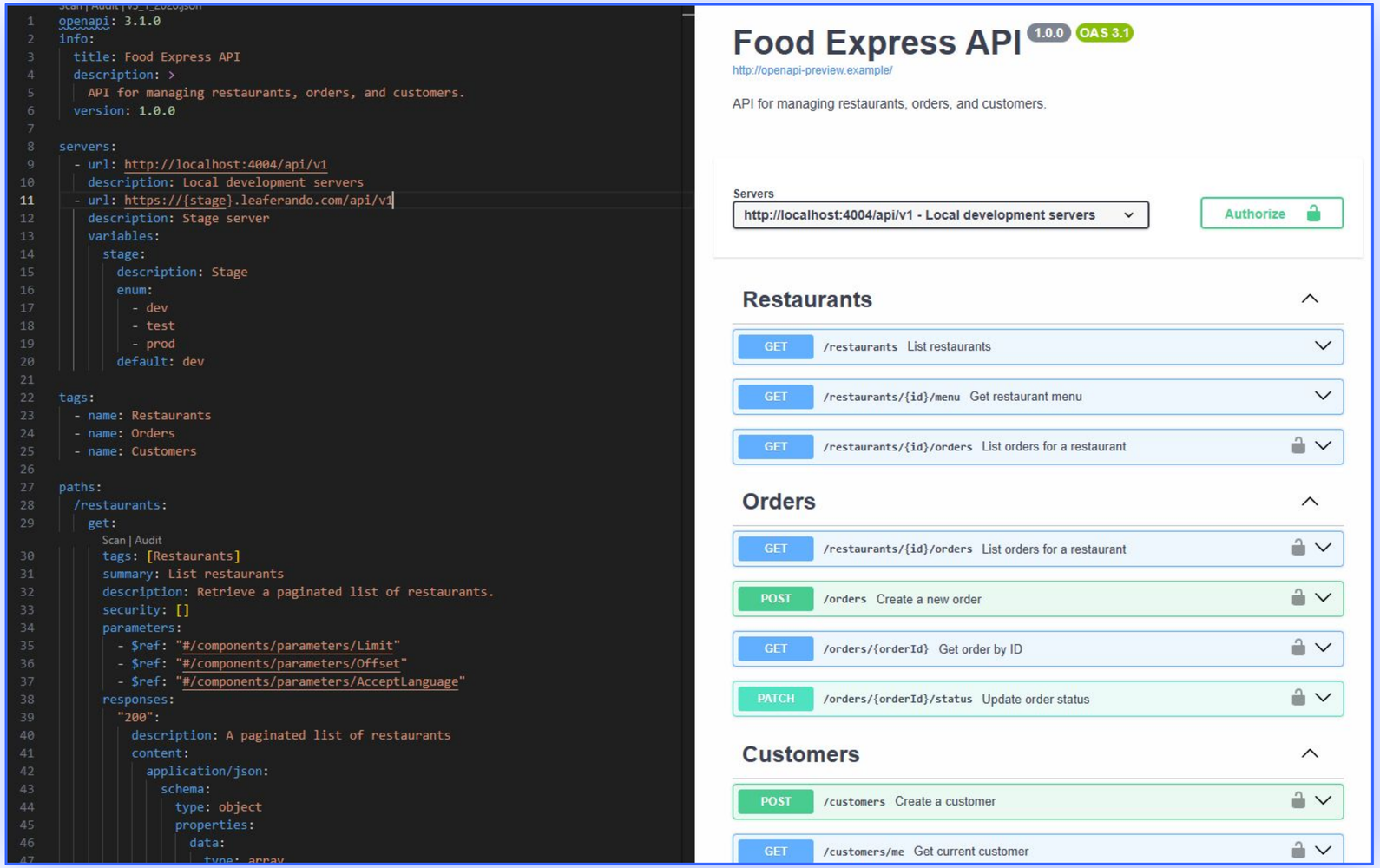

YAML
Besser lesbar
Beides möglich
➔ wir nutzen YAML
JSON
Technisch gleichwertig

Technische Hochschule
mannheim

4. Fertige OpenAPI-Dokumentation – Da wollen wir hin

API-Dokumentation für unser Szenario

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

servers:
  - url: http://localhost:4004/api/v1
    description: Local development servers
  - url: https://{stage}.leaferando.com/api/v1
    description: Stage server
    variables:
      stage:
        description: Stage
        enum:
          - dev
          - test
          - prod
        default: dev

tags:
  - name: Restaurants
  - name: Orders
  - name: Customers

paths:
  /restaurants:
    get:
      Scan | Audit
      tags: [Restaurants]
      summary: List restaurants
      description: Retrieve a paginated list of restaurants.
      security: []
      parameters:
        - $ref: "#/components/parameters/Limit"
        - $ref: "#/components/parameters/Offset"
        - $ref: "#/components/parameters/AcceptLanguage"
      responses:
        "200":
          description: A paginated list of restaurants
          content:
            application/json:
              schema:
                type: object
                properties:
                  data:
```



Technische Hochschule
mannheim

Pause (15 min)

Technische Hochschule
mannheim

5. Voraussetzungen und Tools

Einführung YAML

YAML = **Y**et **A**nother **M**arkup **L**anguage

Zweck

Datenserialisierung → Daten speichern und übertragen

Warum YAML?

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

```yaml
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

```yaml
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

Key-Value Paare

Grundelement ist Schlüssel: Wert

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)

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

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 ~)
```

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

YAML
Red Hat

OpenAPI (Swagger) Editor
42crunch

YAML
von Red Hat

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

Technische Hochschule
mannheim

6. OpenAPI Spezifikation

OpenAPI Spezifikation

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

Details in der Dokumentation nachlesen.

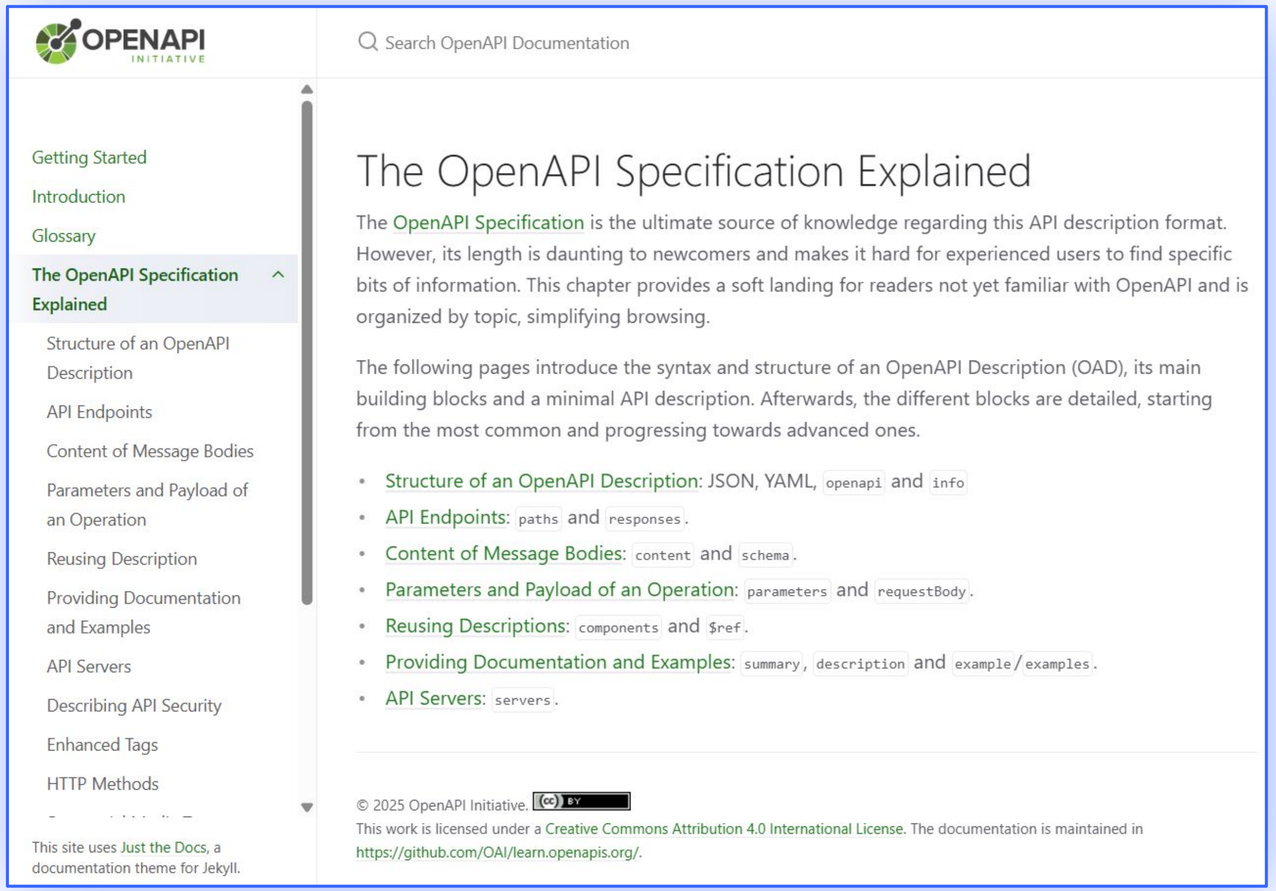
Ressourcen

Dokumentation mit Erklärung

The OpenAPI Specification Explained

Dokumentation

OpenAPI Specification versions 3.1.0



Struktur

- 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

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

Server

- 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

```yaml
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

- 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`

```yaml
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

- 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

```yaml
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: [1111111]
       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]]
```

Technische Hochschule mannheim

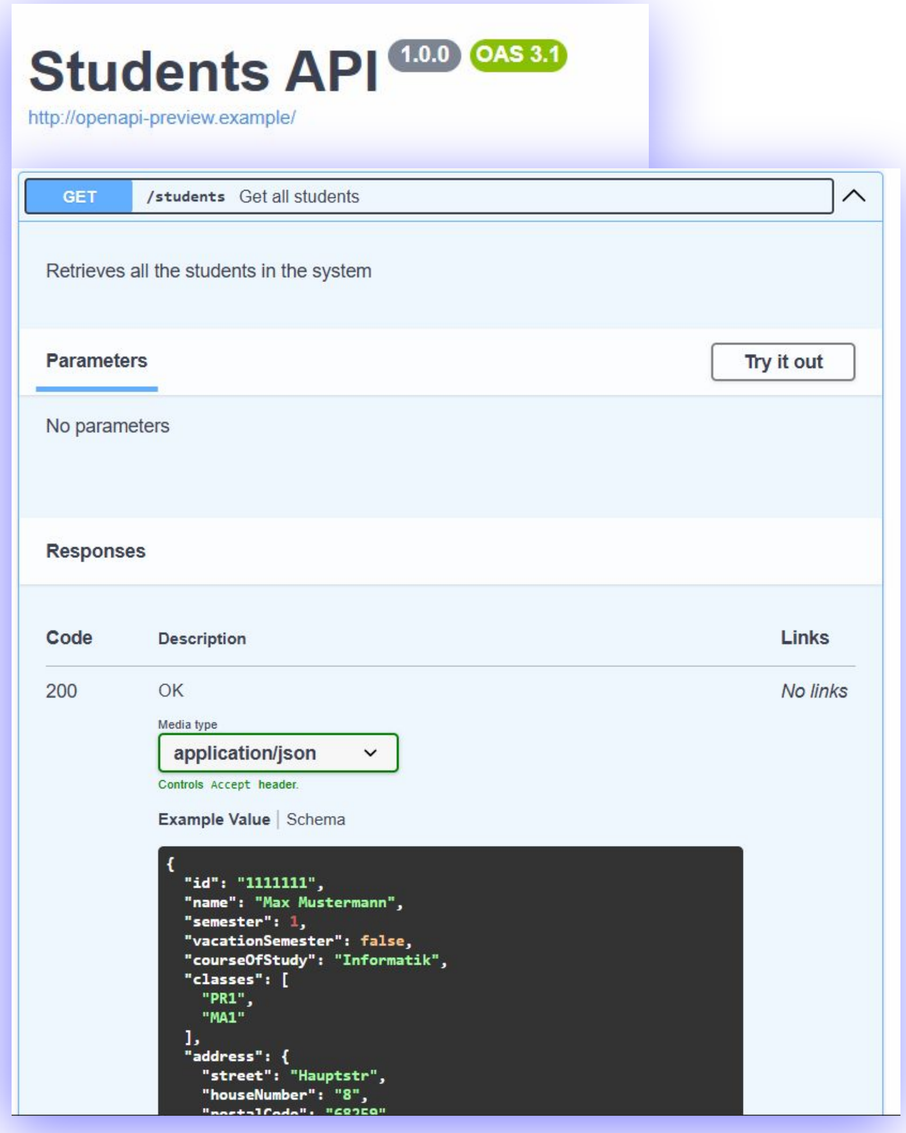
Path Parameters

- 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 `{}`)

```yaml
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

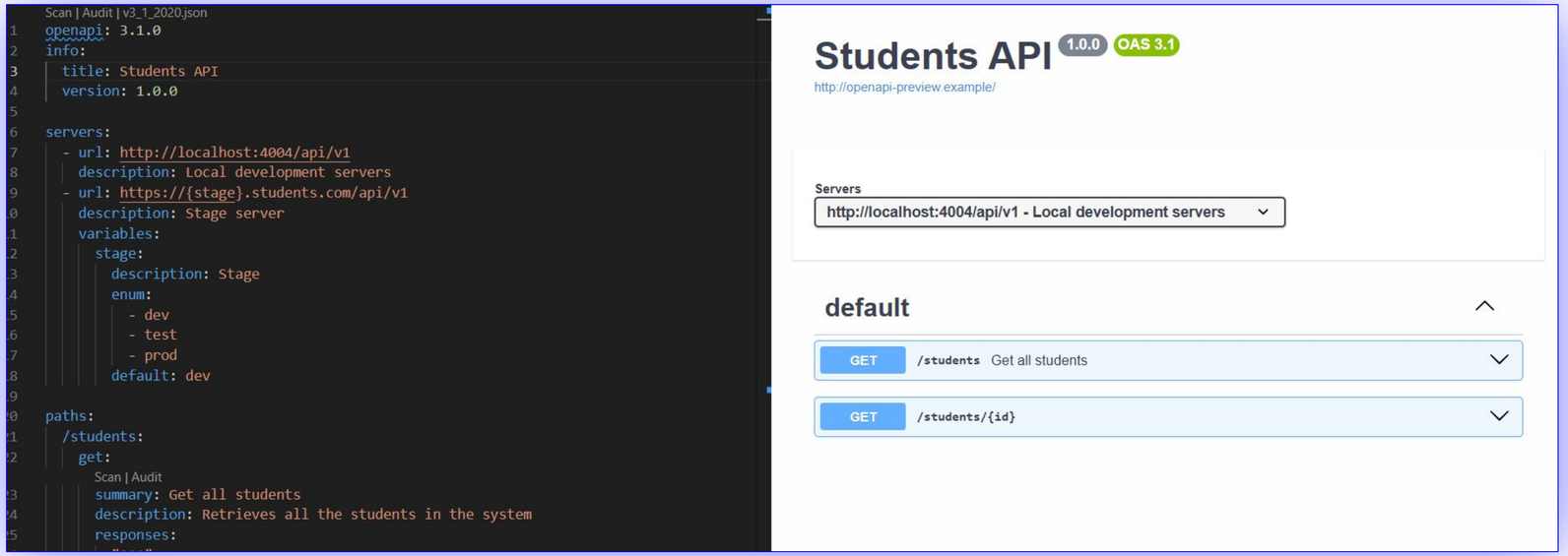
- Bei vielen Objekten sind `summary` und `description` vorhanden
- Gut zur Dokumentation des jeweiligen Elements
- `summary` für eine Kurzbeschreibung des Elements
- `description` für ausführlichere Beschreibung
  - Hier auch Markdown anwendbar für Formatierung usw.
- In `schema` auch `examples` vorhanden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



Pause (10 min)

Weitere Parameter

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

```yaml
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

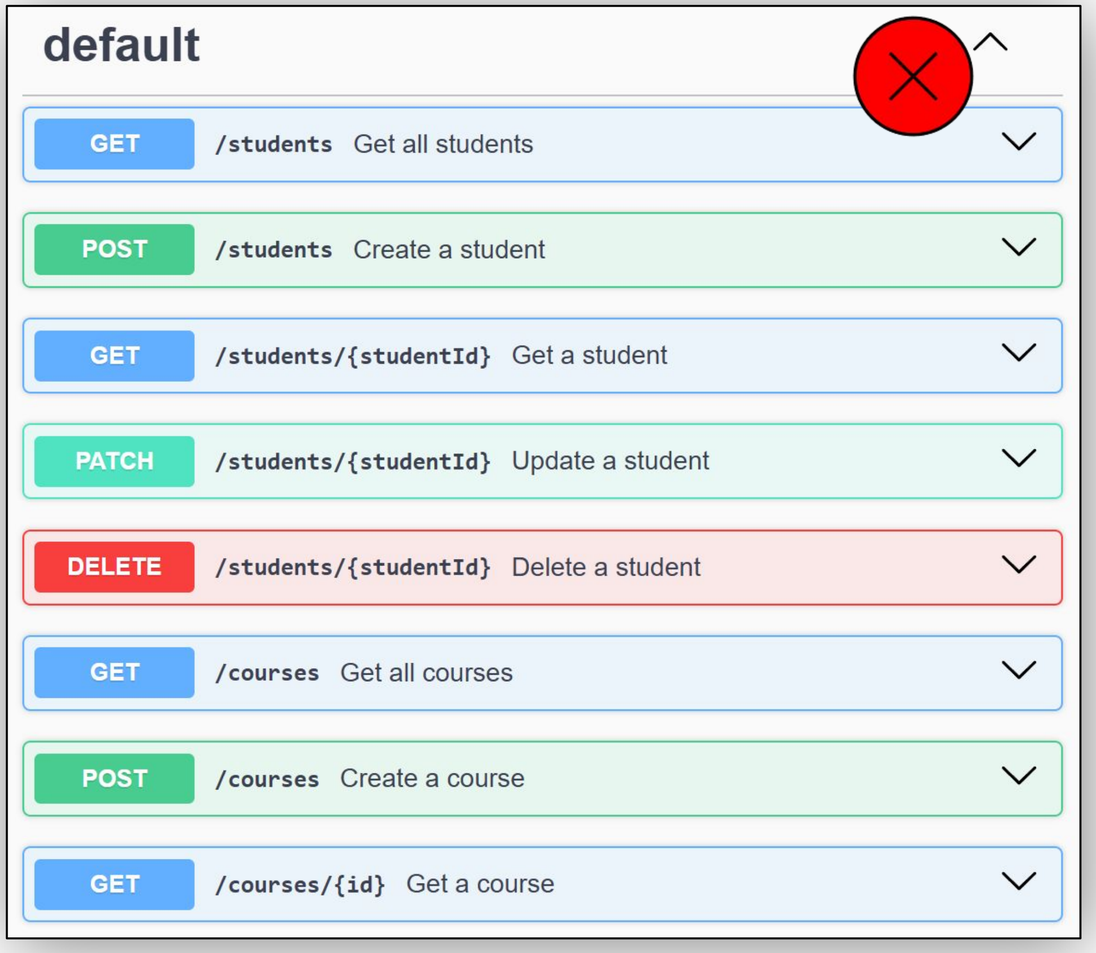
- Operationen können einen **Request Body (Payload)** haben
- typisch für **POST, PUT und PATCH**
- definiert durch `schema`
- klar getrennt von `parameters`

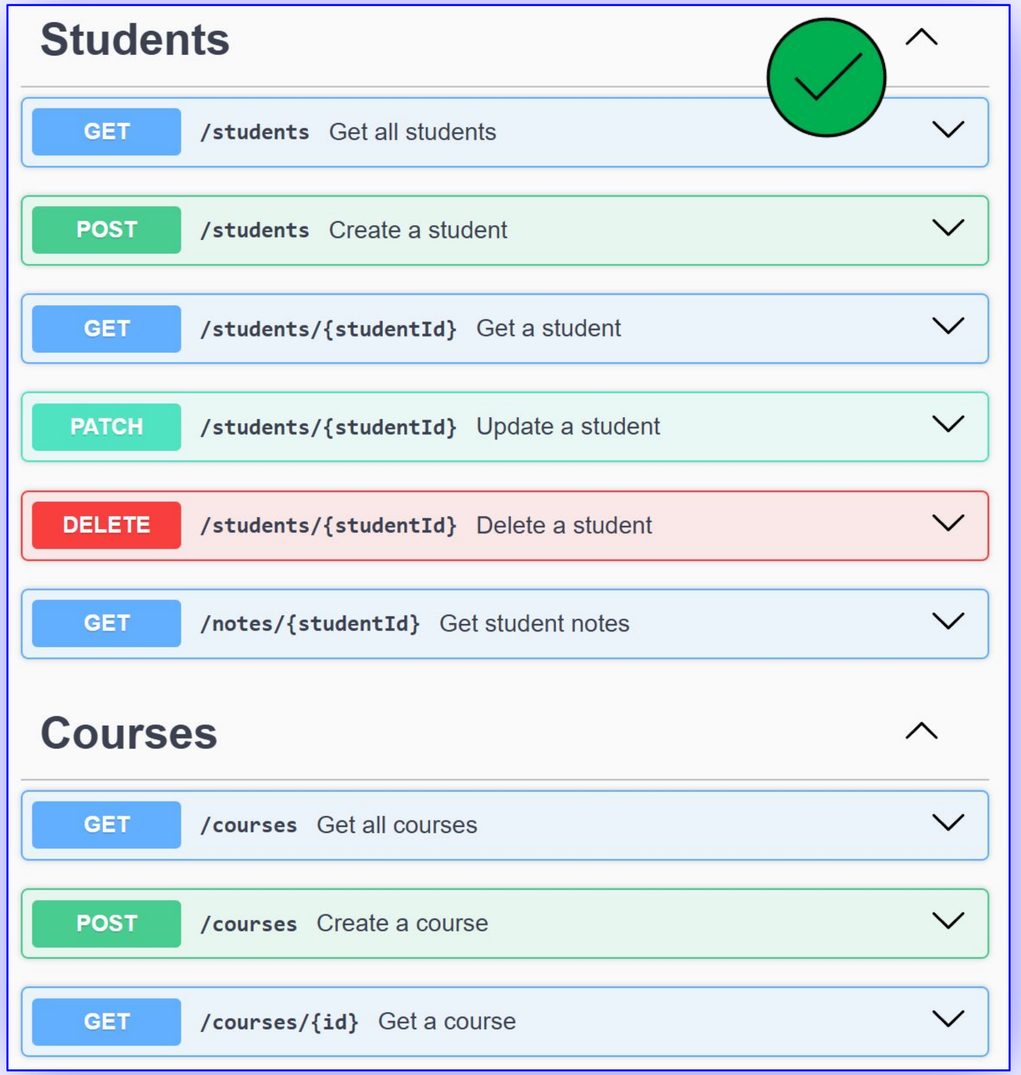
```
paths:
  /students:
    post:
      …
      requestBody:
      …
    patch:
      …
      requestBody:
      …
```

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

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

Tags





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

ohne components

```yaml
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
                            …
```

mit components

```yaml
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
        …
```

Komponenten

ohne components

```yaml
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]
                  …
```

mit components

```yaml
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
        …
```

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

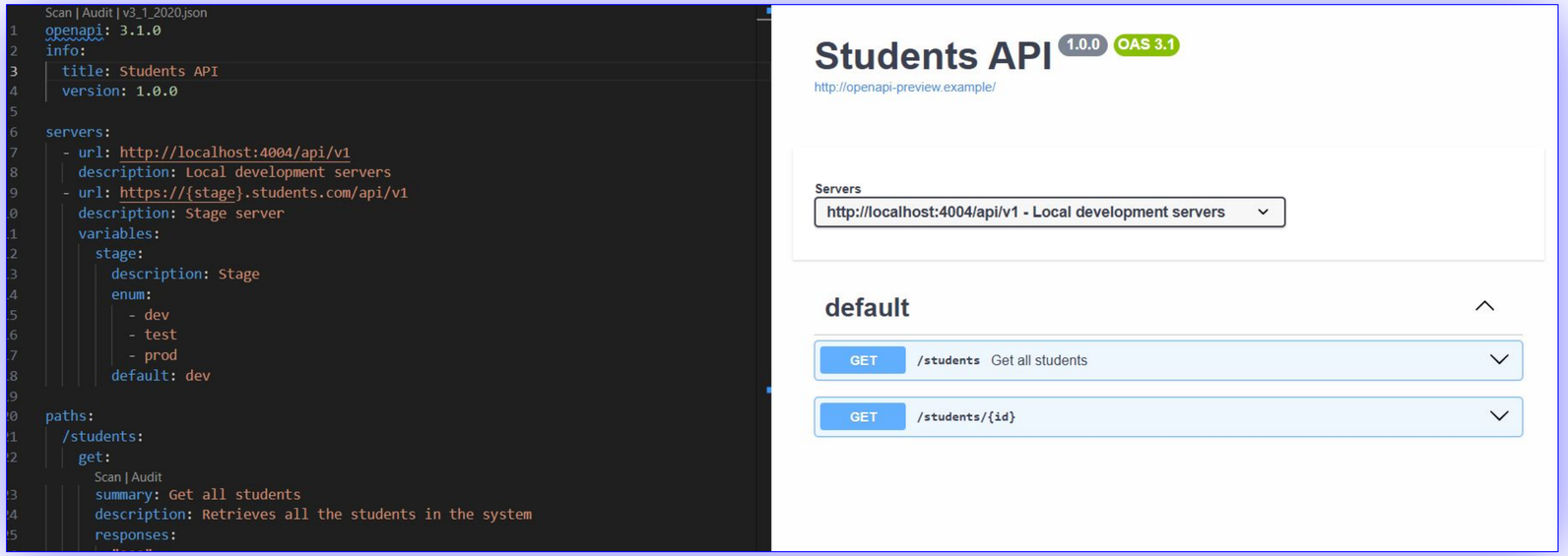
```yaml
paths:
  /students:
    post:
      …
      security:
        - BasicAuth[]
  /notes/{studentId}:
    …
    get:
      …
      security:
        - AuthBearer[]

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

    AuthBearer:
      scheme: bearer
      type: http
      bearerFormat: jwt

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

Technische Hochschule mannheim

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



Technische Hochschule
mannheim

Mittagspause (45 min)

Technische Hochschule
mannheim

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/distro/docker-compose/in-memory-with-studio/docker-compose.yml

$ docker compose up
```

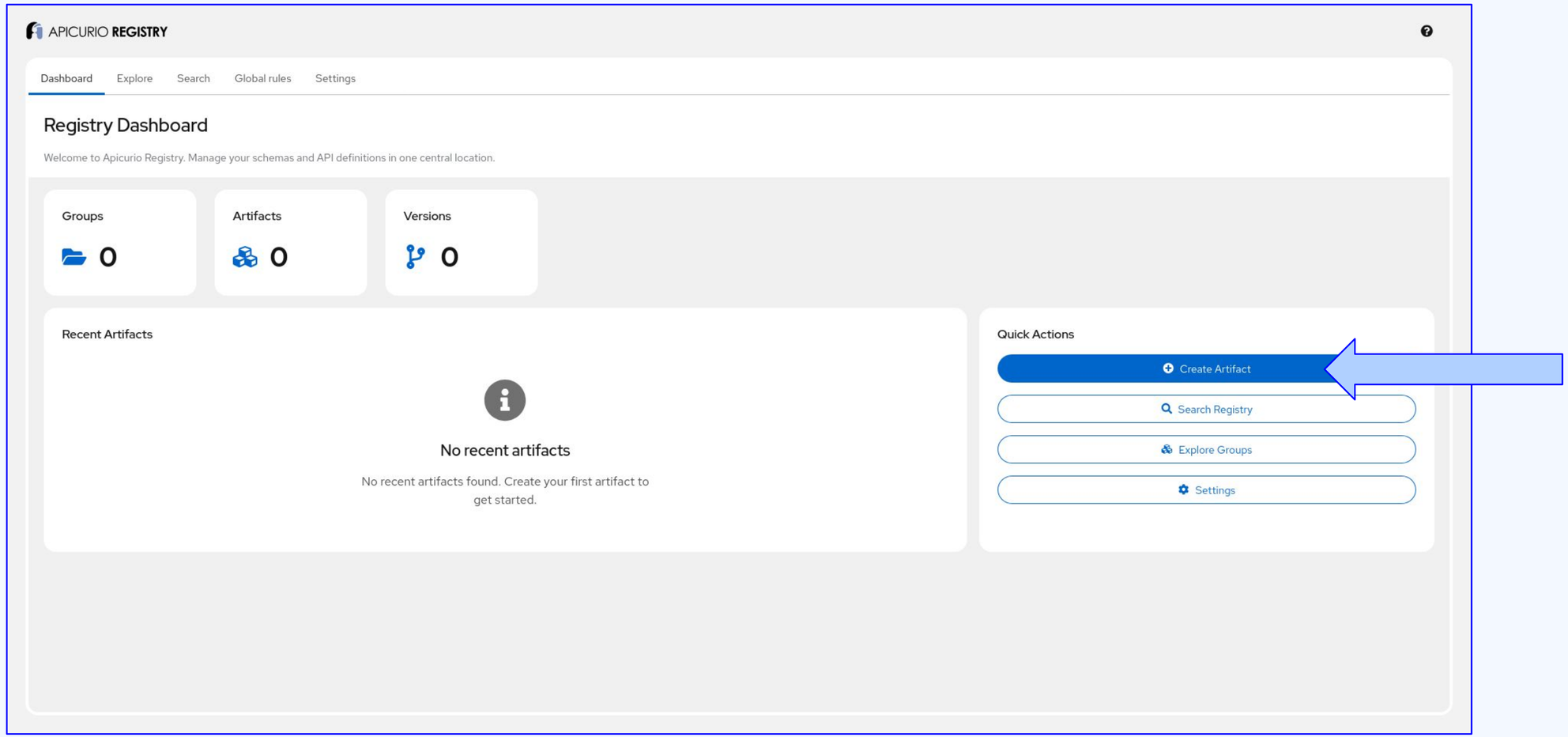
GUI Editoren

Apicurio Registry

APIGit

OpenAPI Designer

OpenAPI Specifications Plugin



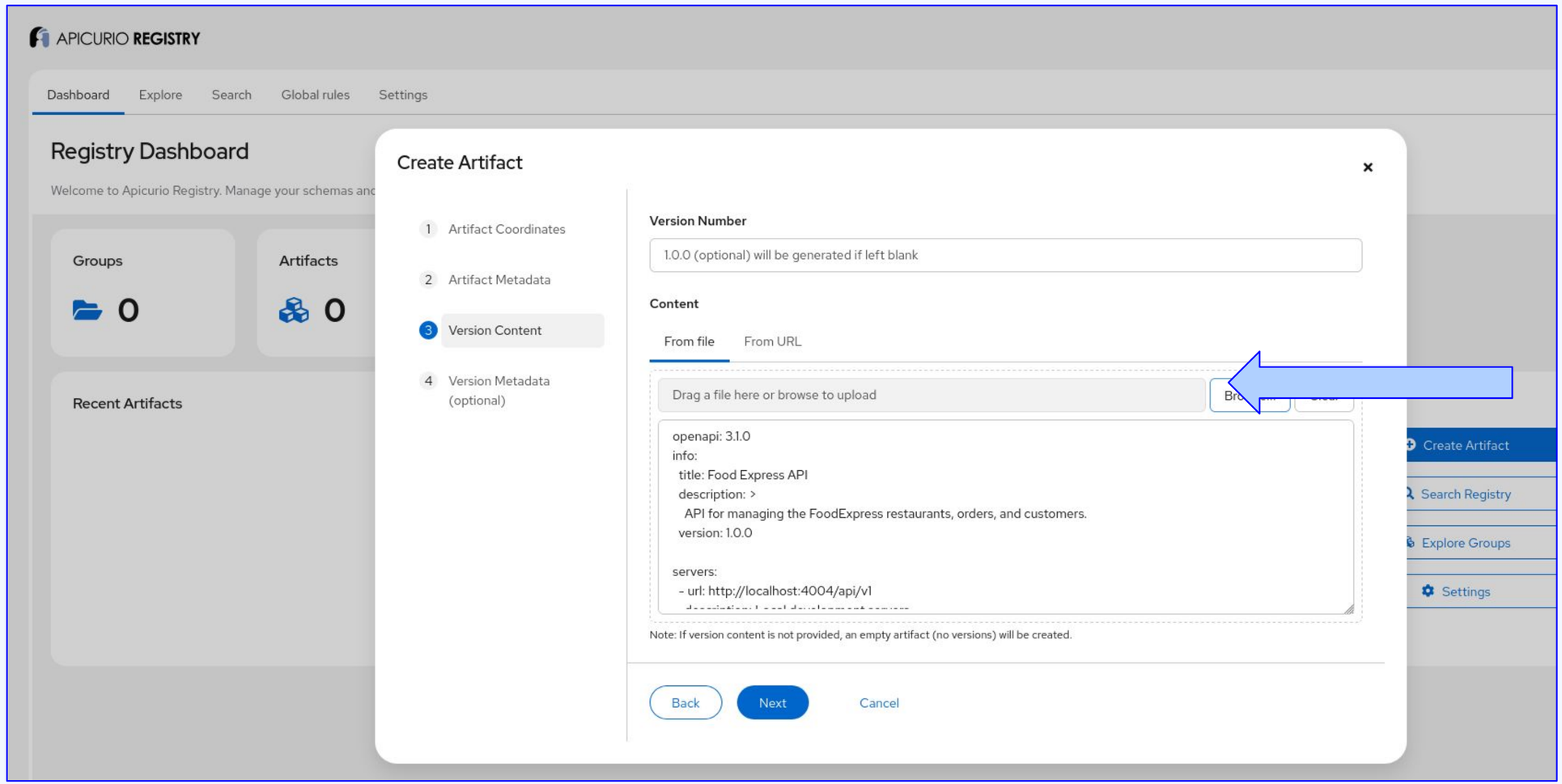
GUI Editoren

Apicurio Registry

APIGit

OpenAPI Designer

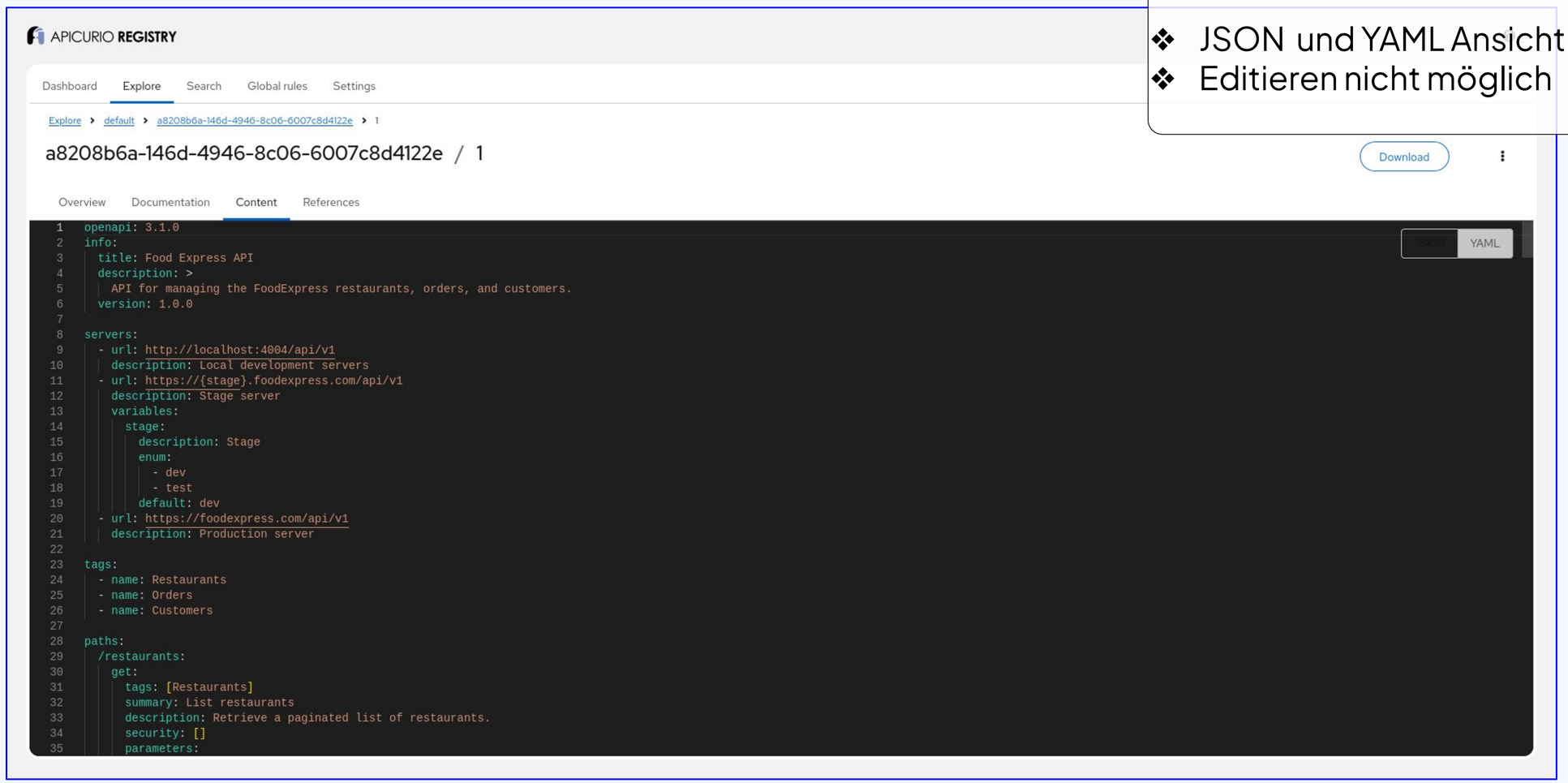
OpenAPI Specifications Plugin



GUI Editoren

Apicurio Registry

- JSON und YAML Ansicht
- Editieren nicht möglich



APIGit

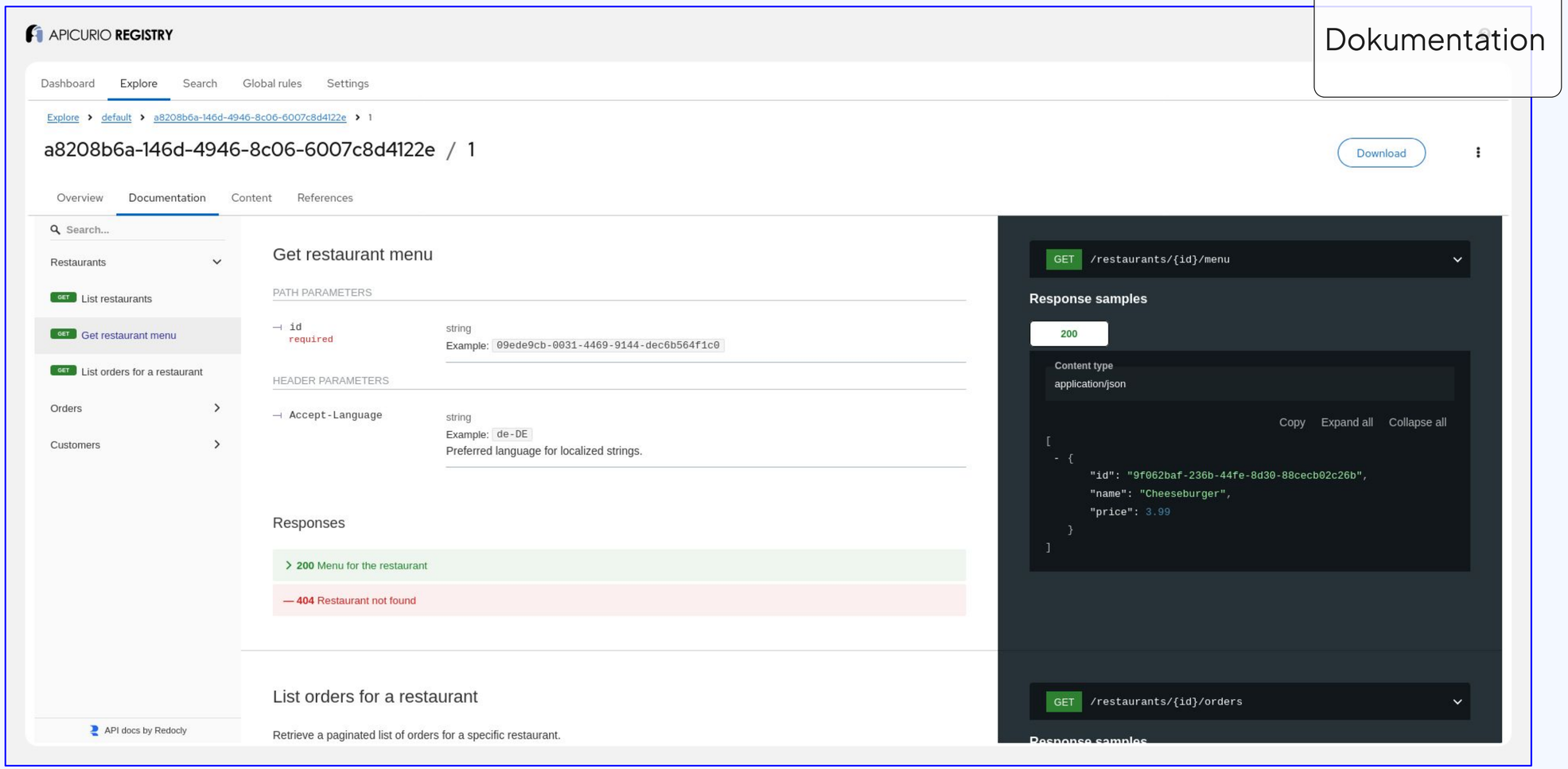
OpenAPI Designer

OpenAPI Specifications Plugin

GUI Editoren

Apicurio Registry

- APIGit
- OpenAPI Designer
- OpenAPI Specifications Plugin



Dokumentation

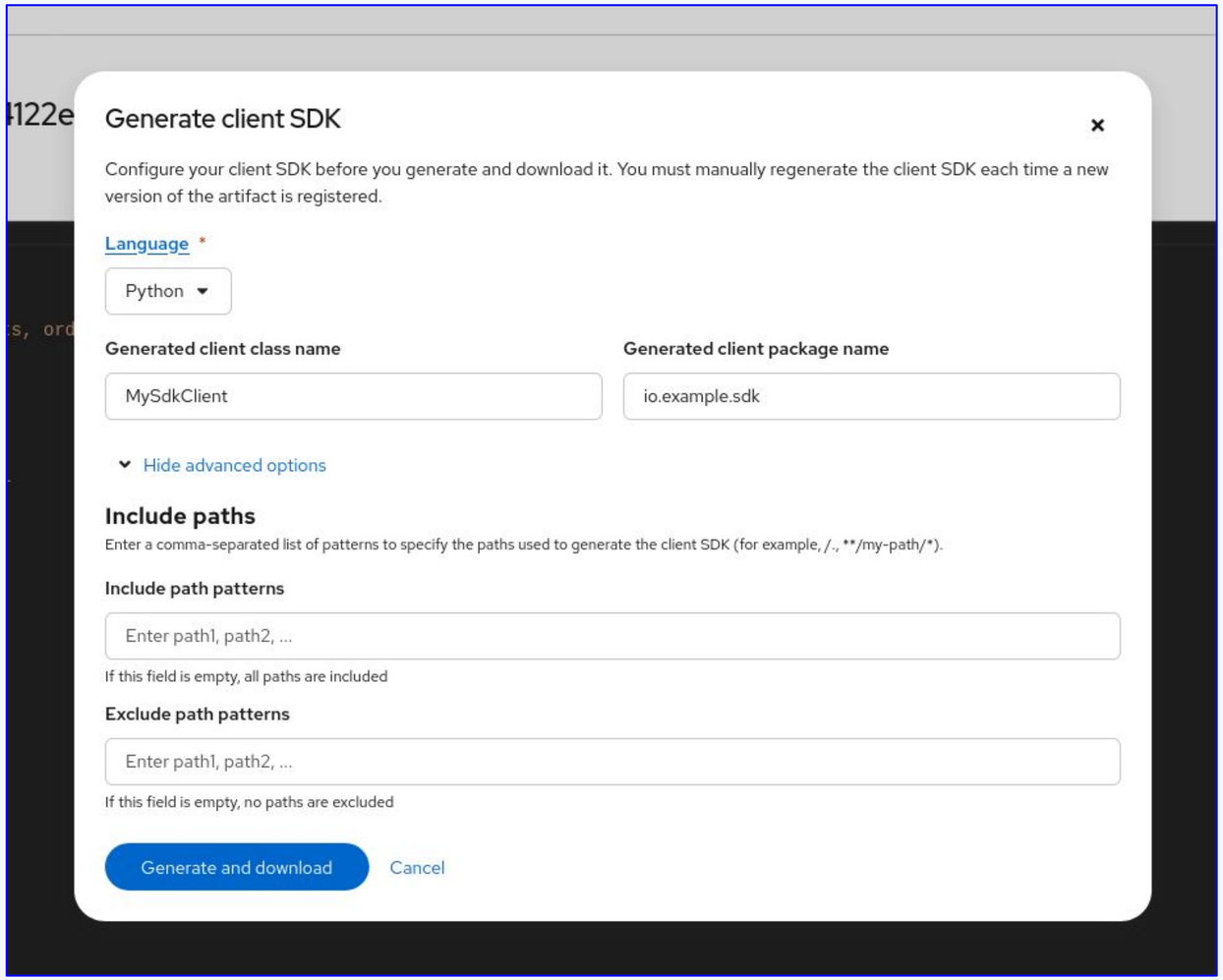
GUI Editoren

Apicurio Registry

APIGit

OpenAPI Designer

OpenAPI Specifications Plugin

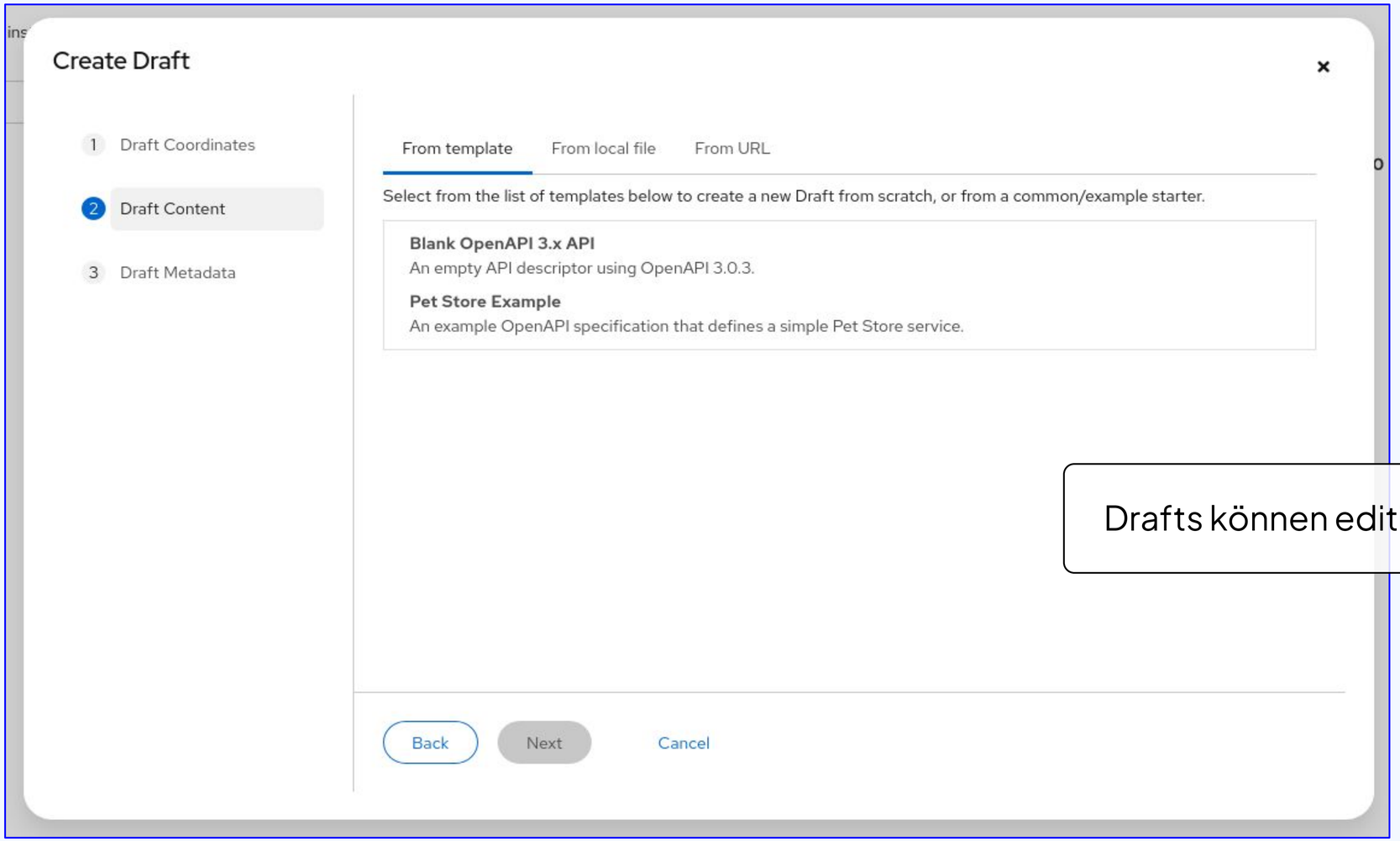


- Generierung des Clients
- Unterstützung für verschiedene Programmiersprachen

GUI Editoren

Apicurio Registry

- APIGit
- OpenAPI Designer
- OpenAPI Specifications Plugin



Drafts können editiert werden

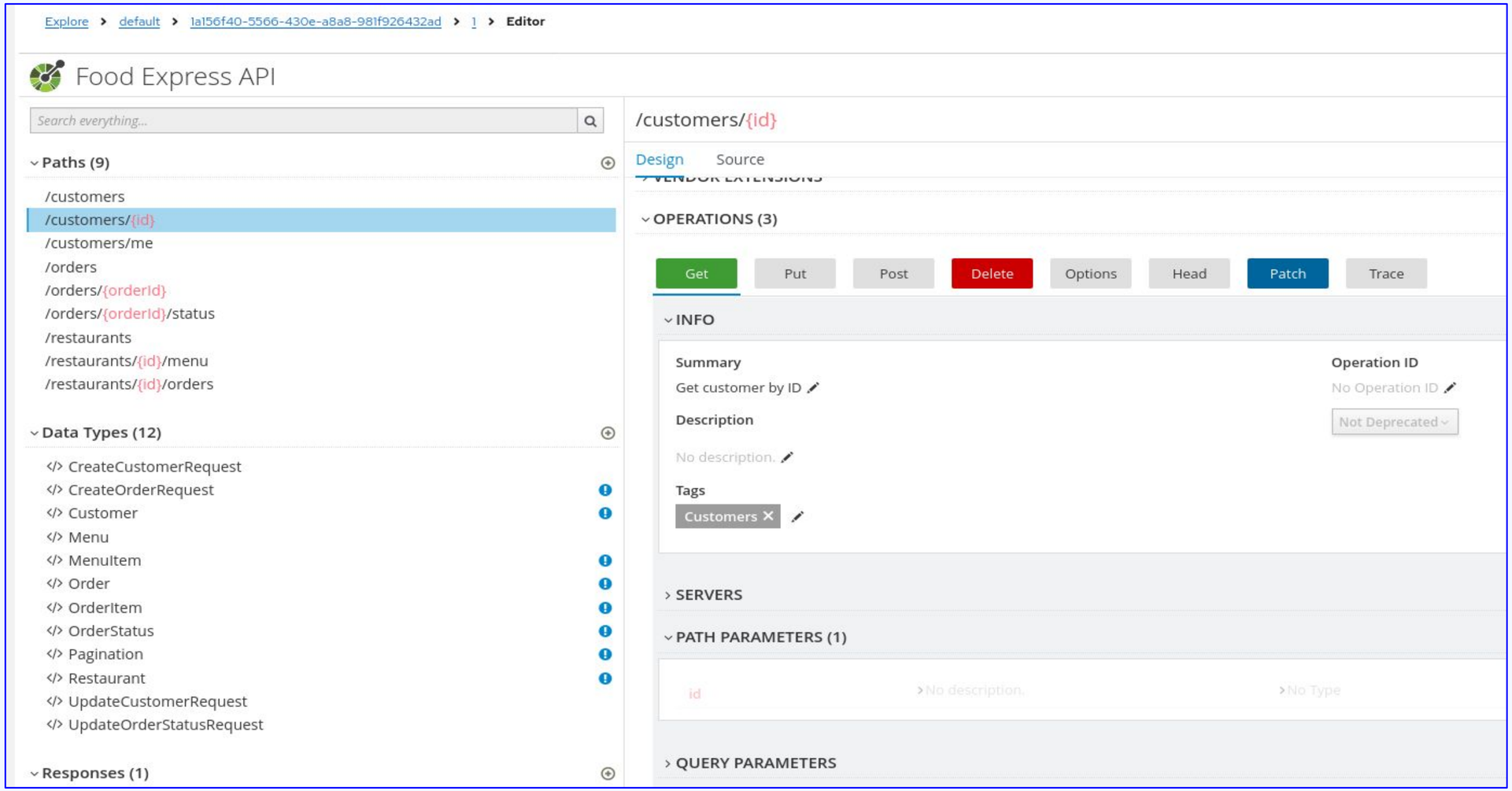
GUI Editoren

Apicurio Registry

APIGit

OpenAPI Designer

OpenAPI Specifications Plugin



Technische Hochschule mannheim

GUI Editoren

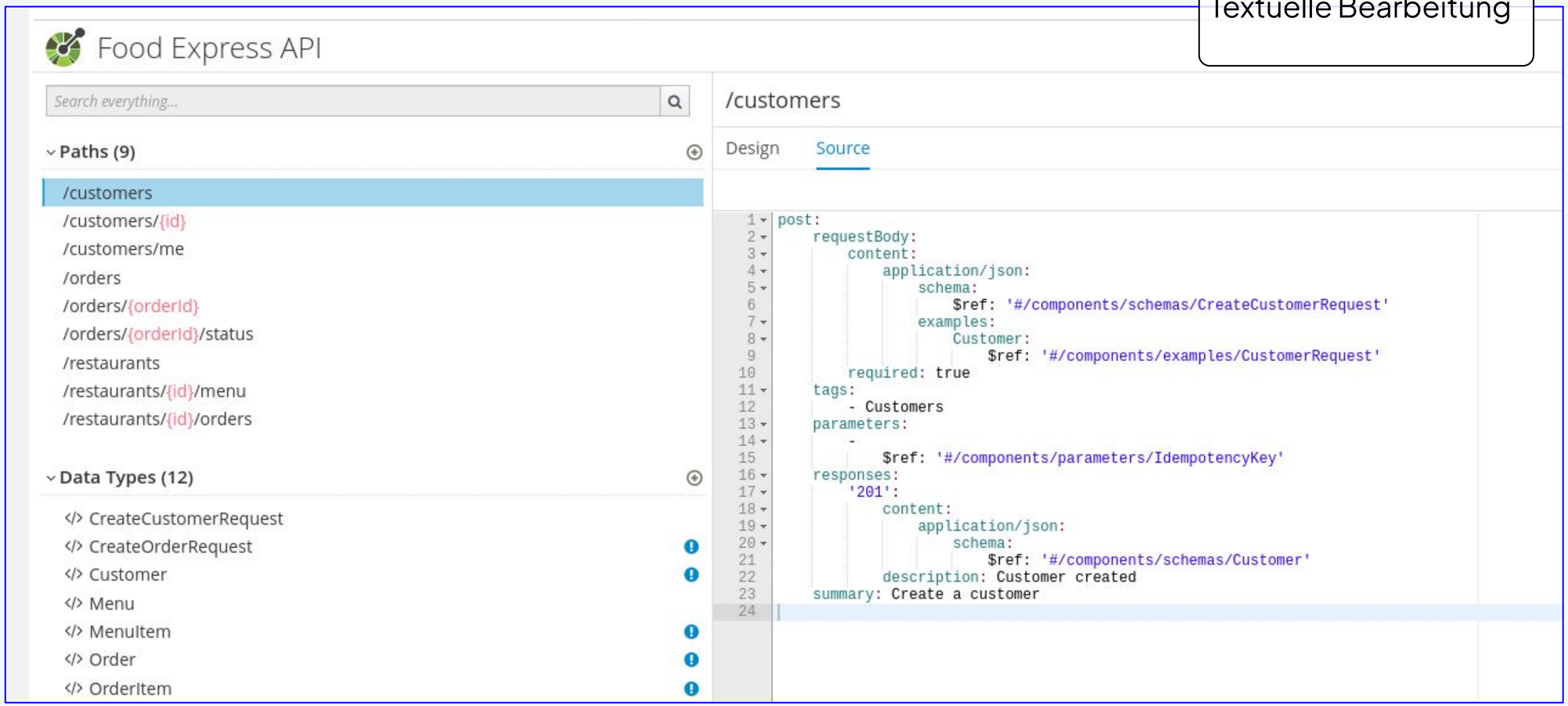
Apicurio Registry

APIGit

OpenAPI Designer

OpenAPI Specifications Plugin

Textuelle Bearbeitung



Technische Hochschule
mannheim

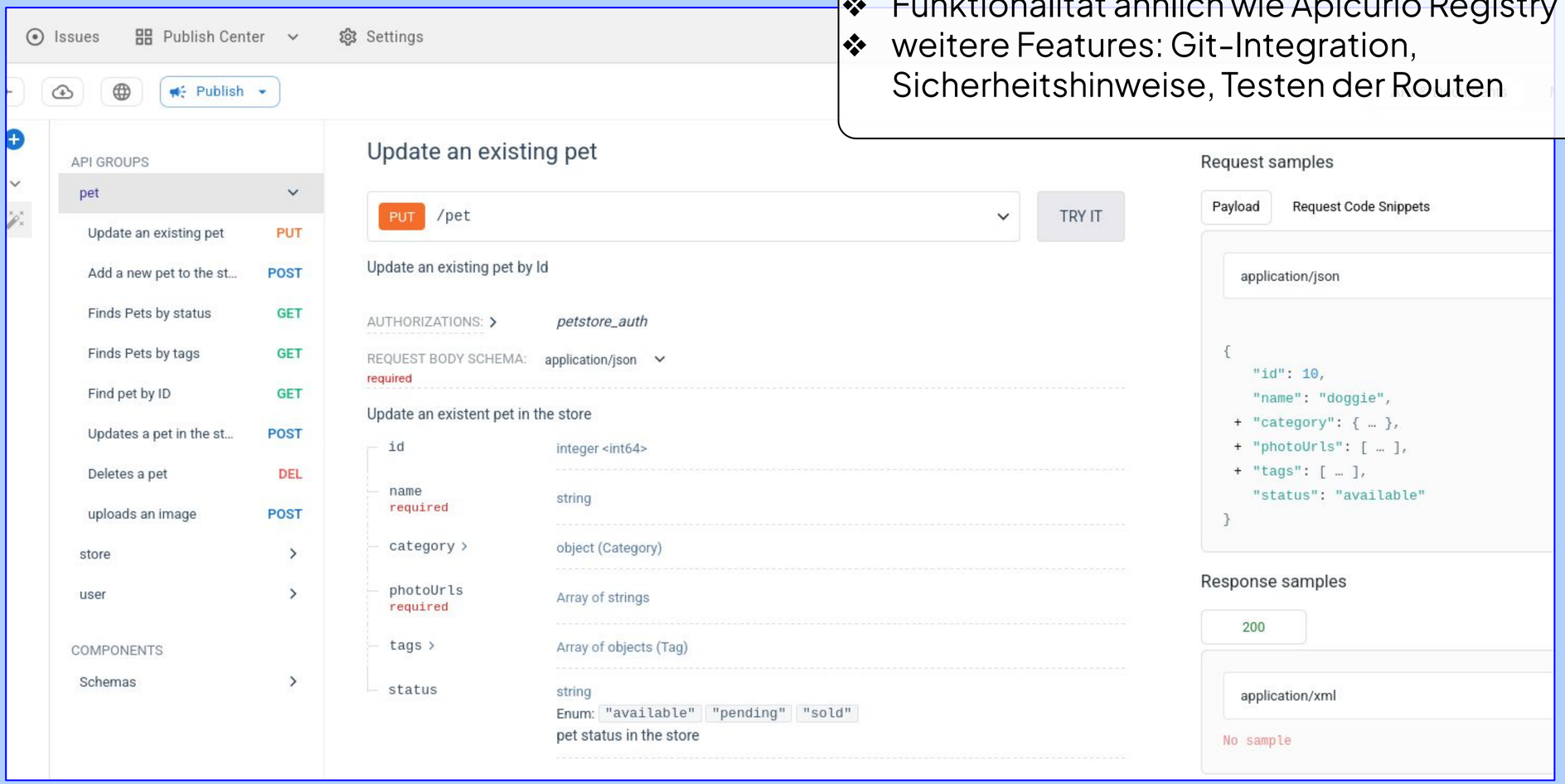
GUI Editoren

Apicurio Registry

APIGit

OpenAPI Designer

OpenAPI Specifications Plugin



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

Technische Hochschule mannheim

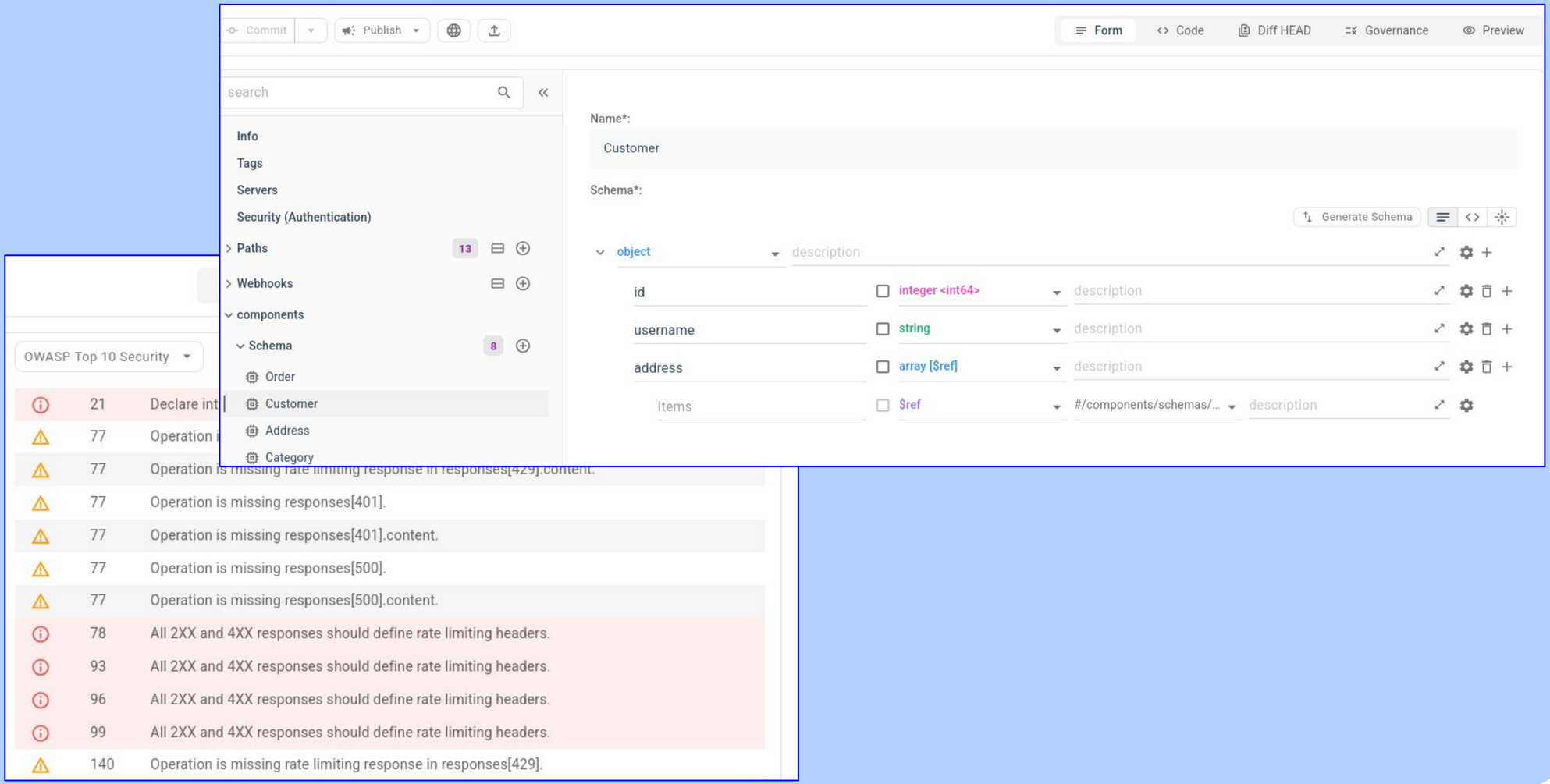
GUI Editoren

Apicurio Registry

APIGit

OpenAPI Designer

OpenAPI Specifications Plugin



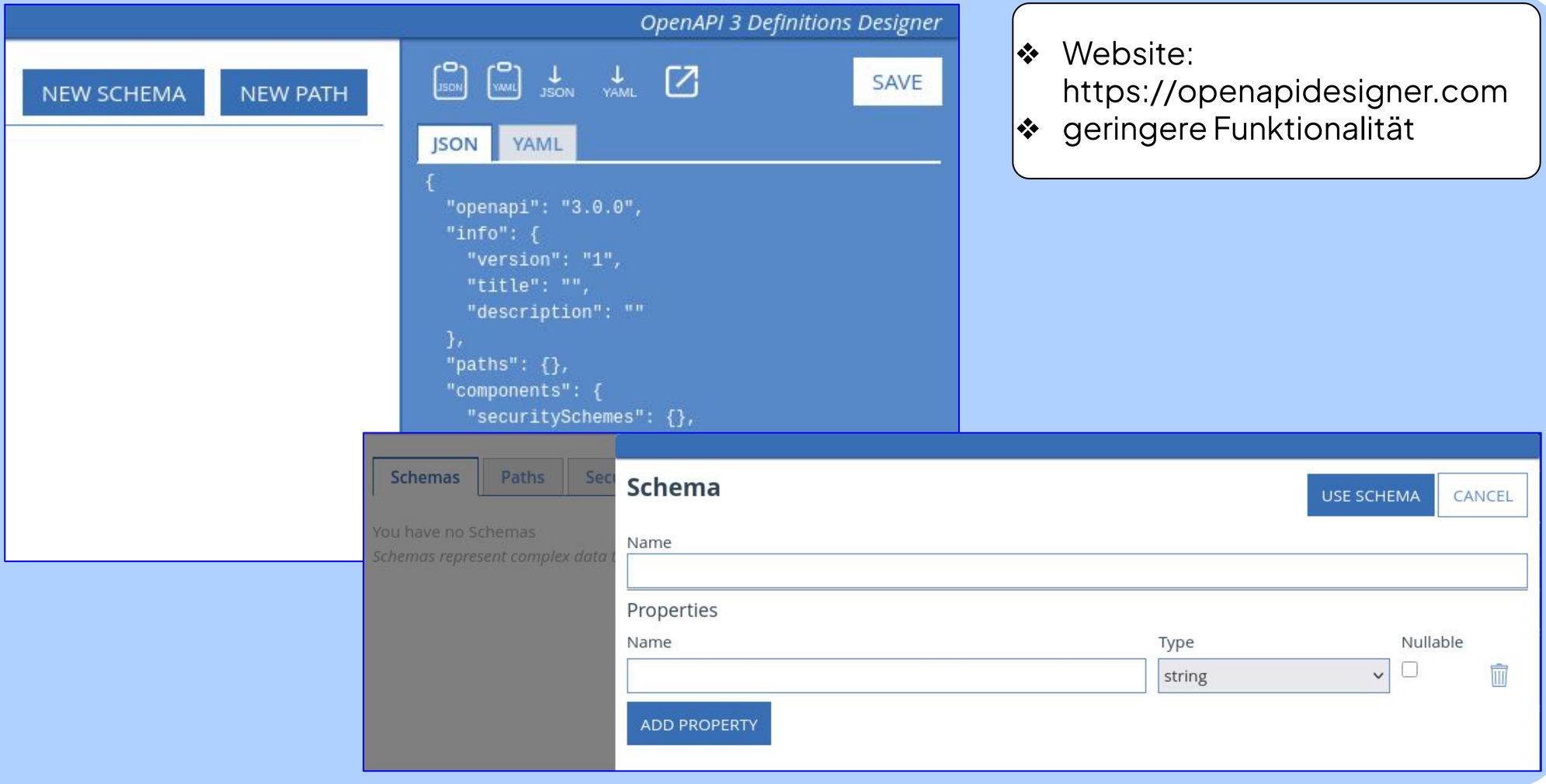
GUI Editoren

Apicurio Registry

APIGit

OpenAPI Designer

OpenAPI Specifications Plugin



- Website: https://openapidesigner.com
- geringere Funktionalität

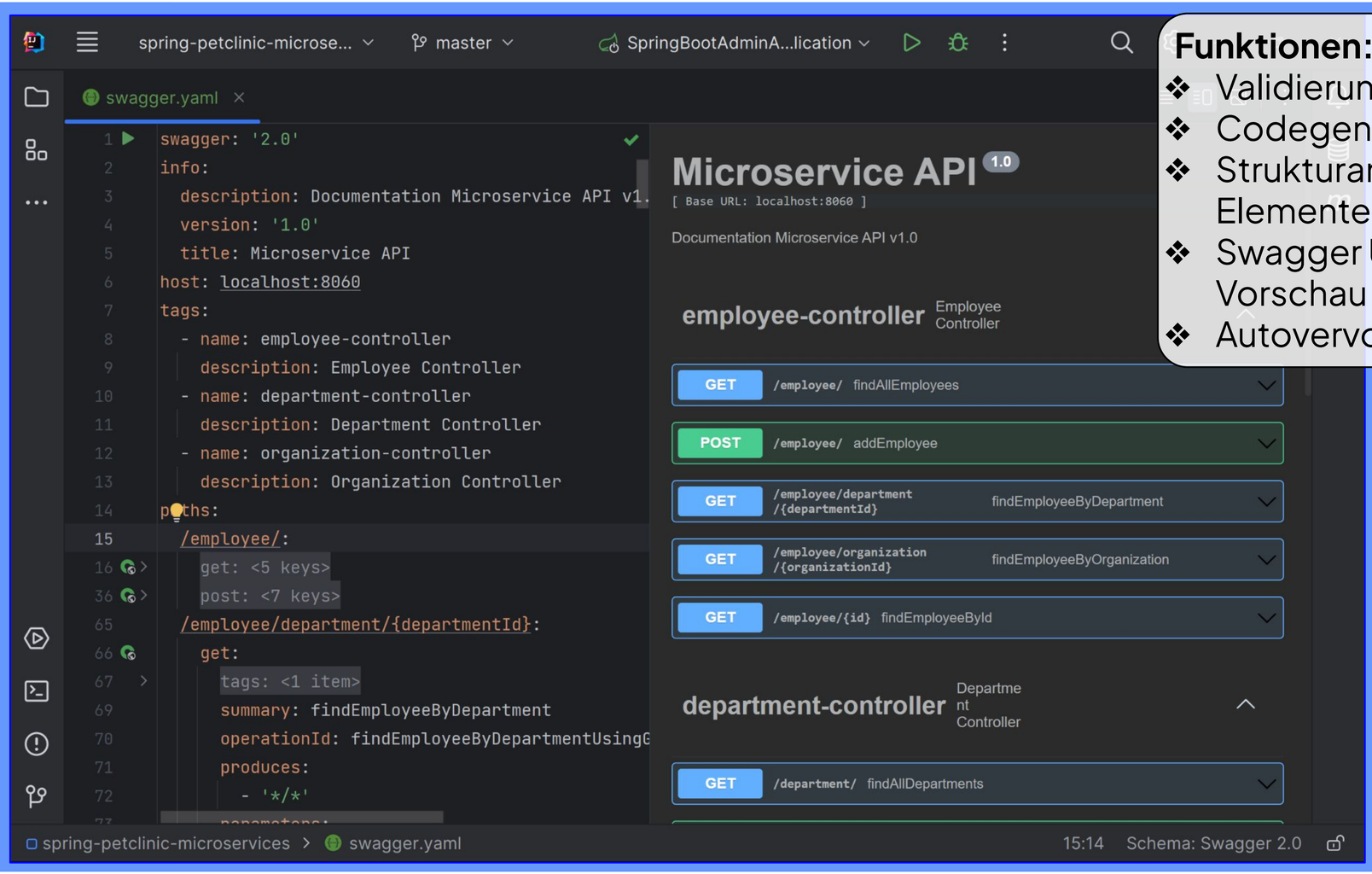
GUI Editoren

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

```yaml
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'
```

```yaml
 /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
```

Technische Hochschule mannheim

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)

```python
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)

```python
@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)?

Technische Hochschule
mannheim

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

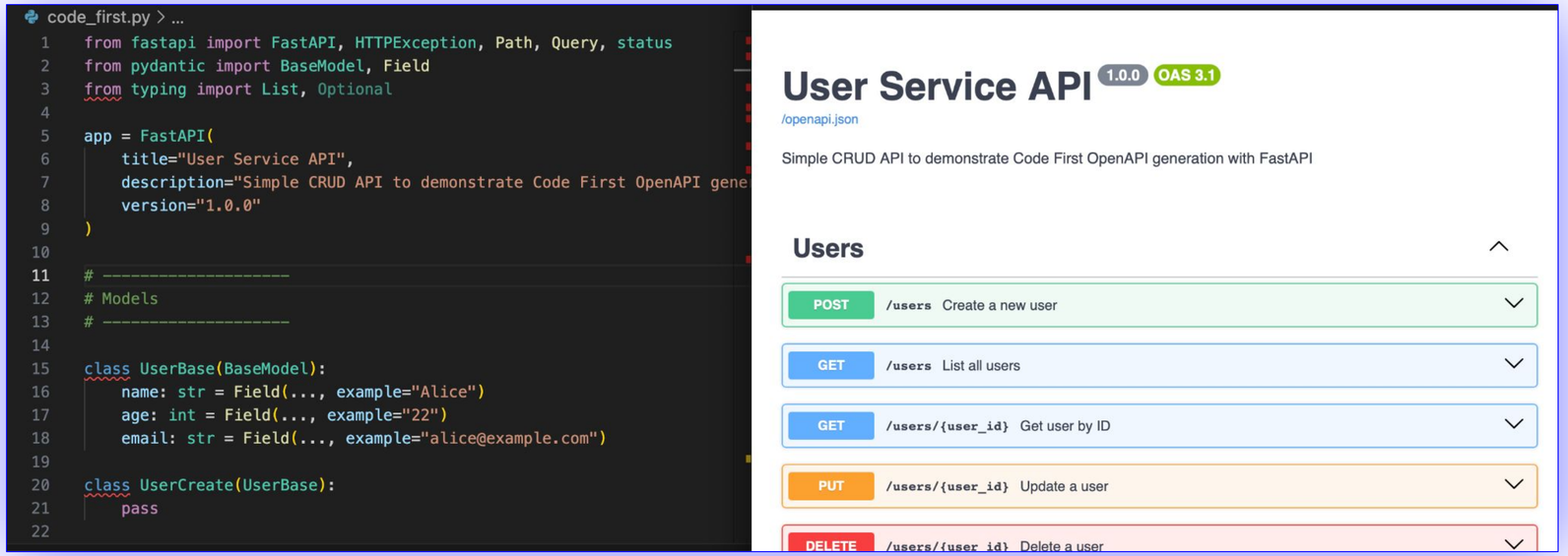
```
@app.post("/users")
def create_user(user: UserCreate):
    ...
```

- 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



Technische Hochschule
mannheim

9. Ausblick

Technische Hochschule mannheim

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"**)

Technische Hochschule mannheim

Was ist sonst noch möglich?

Beschleunigung & Qualität (Dev)

Management & Betrieb (Ops)

Über den Tellerrand (Trends)

API Gateway

Import der Spec zur Konfiguration

Automatisiertes Rate-Limiting, Security & Auth

Developer Portals

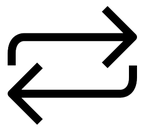
Generierung zentraler „Bibliotheken" für alle Firmen-APIs („**API Appstore**")

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)

Technische Hochschule
mannheim

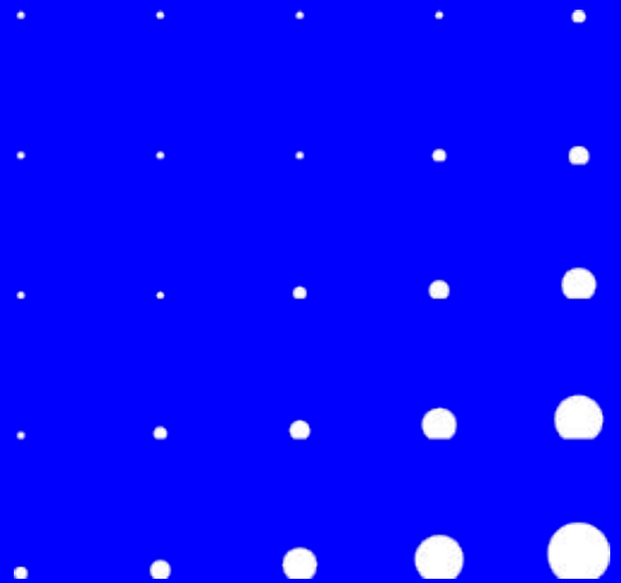
10. Zusammenfassung

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.

Technische Hochschule
mannheim

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/