

OPENAPI SPECIFIKACE PRO ANALYTICKY



Kdo jsme?



Prokop Simek

Engineering, DevEx, CEO

12 let v IT, SWE,
Developer Experience,
Platform Engineering, AI



Jan Řičica

IT Analyst

IT a byznys analytik se
specializací na API
management a
energetiku.



Jakub Vacek

Engineering & AI Expert

Seniorní BE vývojář s
velkým přesahem do AI
technologií.

Co děláme?

Zlepšujeme kulturu a nástroje v IT pro rychlejší a kvalitnější delivery.

Naší doménou je především **API management**, tvorba **SDKs**, technická & uživatelská **dokumentace** a **vývojářské portály a platformy** (externí i interní).

- Praktické zkušenosti v API managementu v různých firmách.
- Tvorba SDK a dokumentace pro projekty.
- Transformace vývojářské kultury – API a dokumentace jako pilíř rychlejšího vývoje.
- Vylepšený onboarding a integrace pro tisíce vývojářů i uživatelů.

Pár případů z praxe

- Dev portál pro Erste Group
- Dev community portál pro Factset
- Automatizované generování SDK z API specifikace
- Interní dev portály pro self-service
- Automatizace komplexních IT procesů pomocí AI

revolgy

ERSTE Š
Group

C ConductorOne

 **wultra**

FACTSET®

 **skipPay**

DATEIO

 **ButterCMS**

 **MassPay**

 **tyntec**

TATUM

 **Bank iD**

 **foxentry**

Plán dnešního workshopu

01

Základy API & OAS

- Co je API?
- Co je REST?
- Jak fungují CRUD operace?
- Jak fungují chybové stavy?

02

OAS

- Co je to OAS?
- Jaké jsou formáty zápisu?
- Jaké jsou základní funkce?
- Co je sémantické verzování?

03

Pokročilé funkce OAS & nástroje

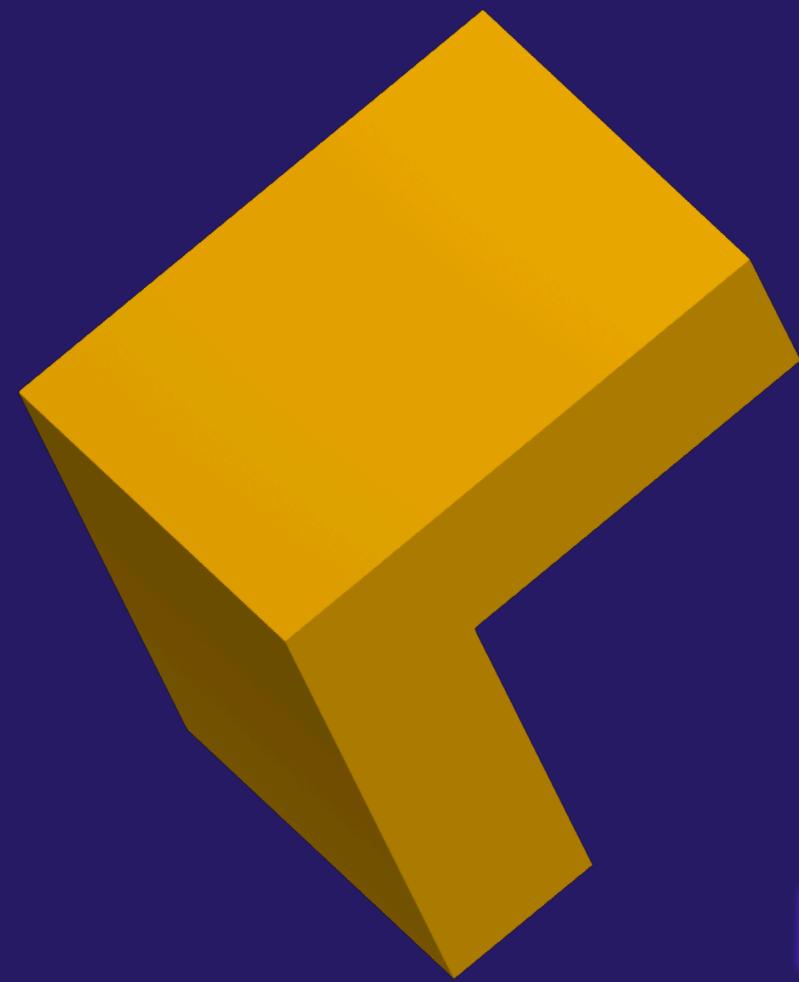
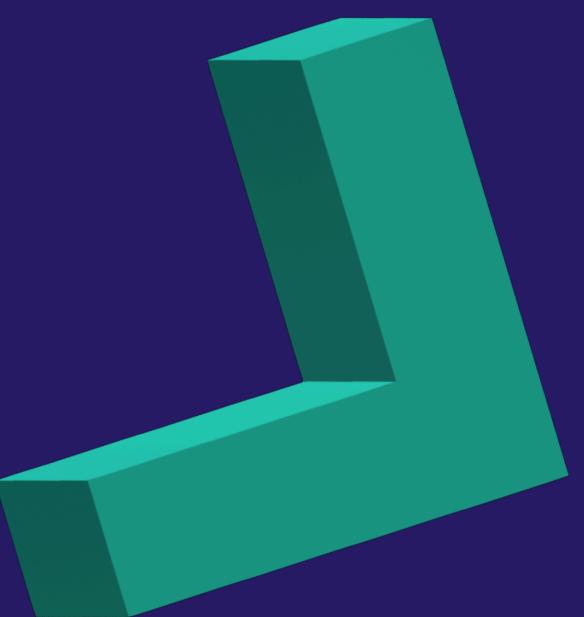
- Jak se liší oneOf, allOf, anyOf?
- Jak fungují reference?
- S jakými nástroji pracovat?
- Jaké jsou best practices?

04

Praktická část

- Praktická ukázka
- Validace OAS
- JSON / YAML schémata
- Testování & Mocking

OI
ÚVOD DO API

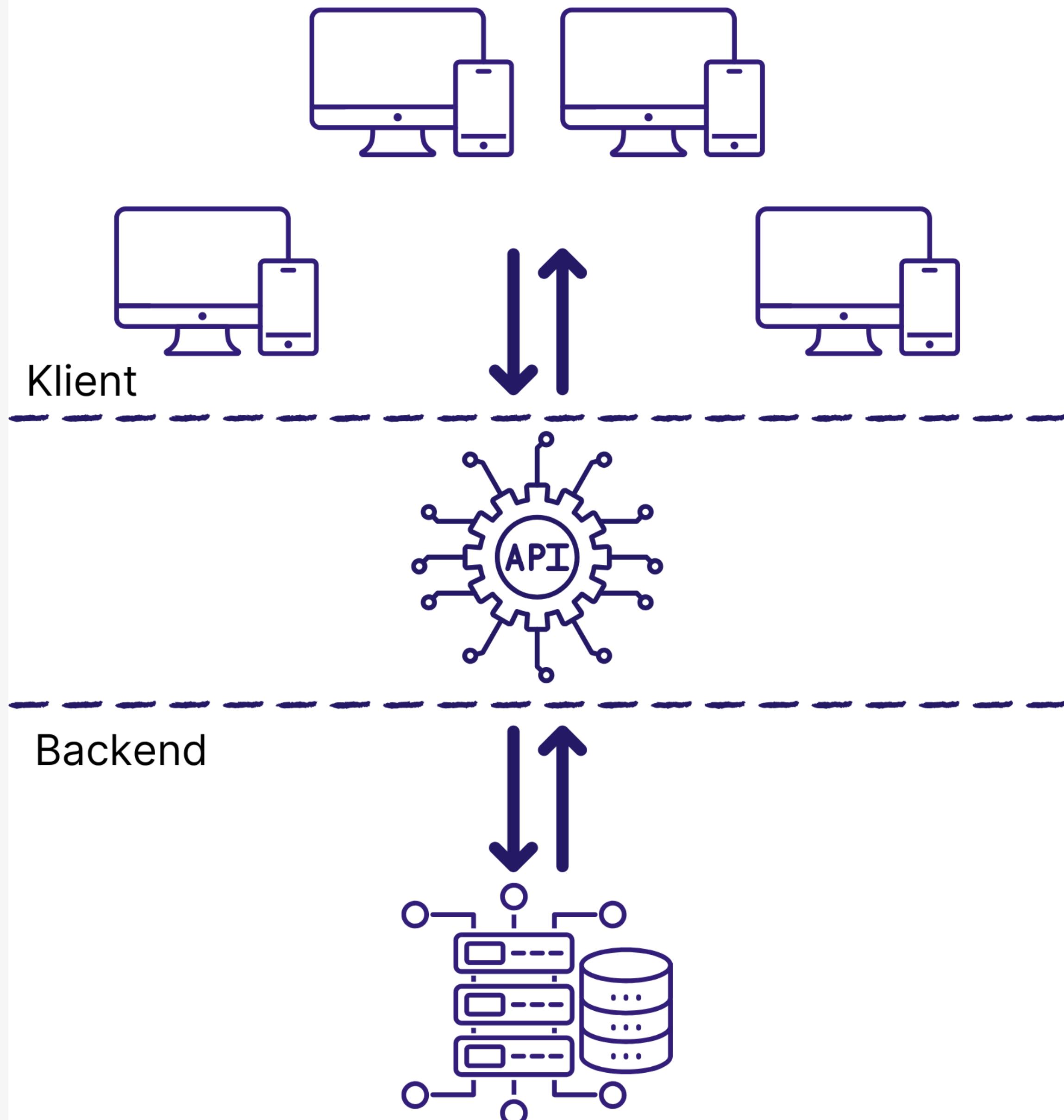


Co je API, REST, OAS,...?

API - Application Programming Interface: umožňuje různým systémům vzájemně komunikovat a vyměňovat data.

REST - Representational State Transfer: architektonický styl pro vytváření webových služeb, který klade důraz na jednoduchost, škálovatelnost a přenositelnost.

OAS - OpenAPI Specification: standardizovaný formát pro popis REST API, který usnadňuje vývoj, dokumentaci a testování.



Proč API?

Mikroservisová architektura

- Rozdělení velkých aplikací na menší, samostatně provozované služby.

Integrace systémů

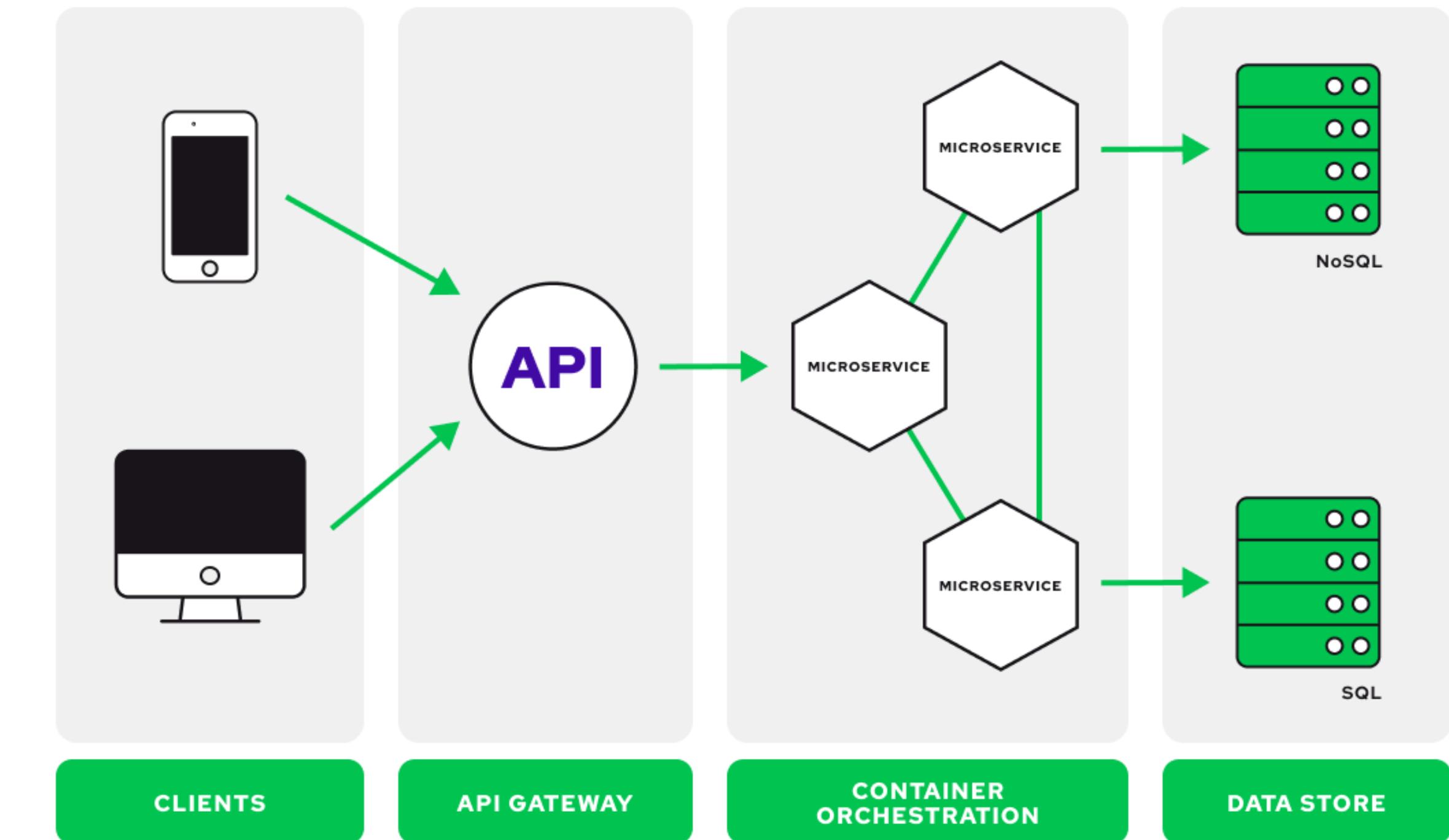
- Spojování různých systémů a služeb.

Snadná konzumace dat

- Dostupnost dat pro různé aplikace a uživatele.

Hlavní výhody:

- Škálovatelnost
- Menší šance na chybovost
- Rychlejší vývoj



Proč REST?

Bezstavovost

- Každý request obsahuje všechny informace potřebné pro úspěšné volání.

Klient-server architektura

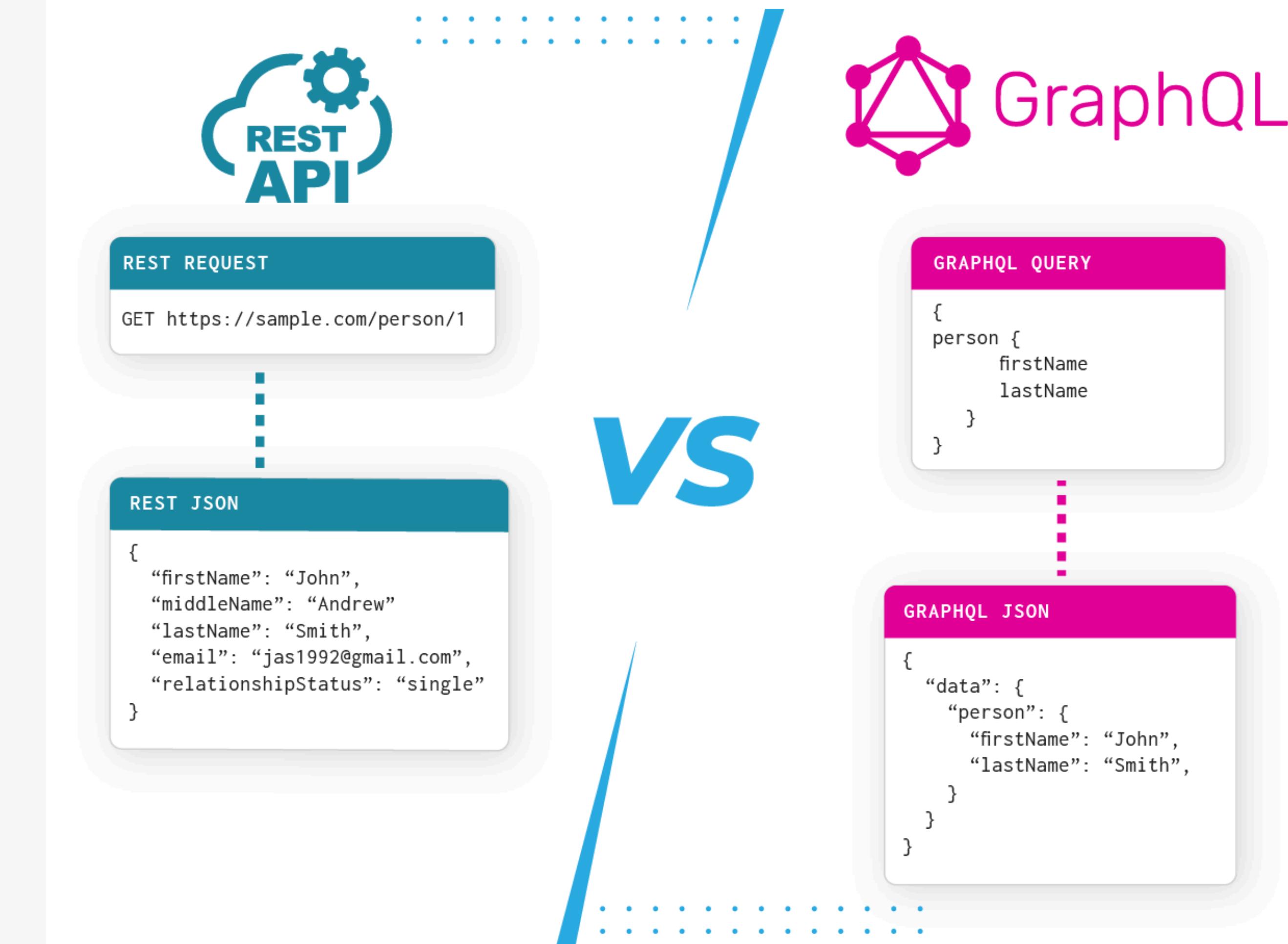
- Jasně oddělení klientské a server části.

Unifikace interakce

- Standardizace HTTP (GET, POST, PUT, DELETE) metod a formátů (JSON,...)

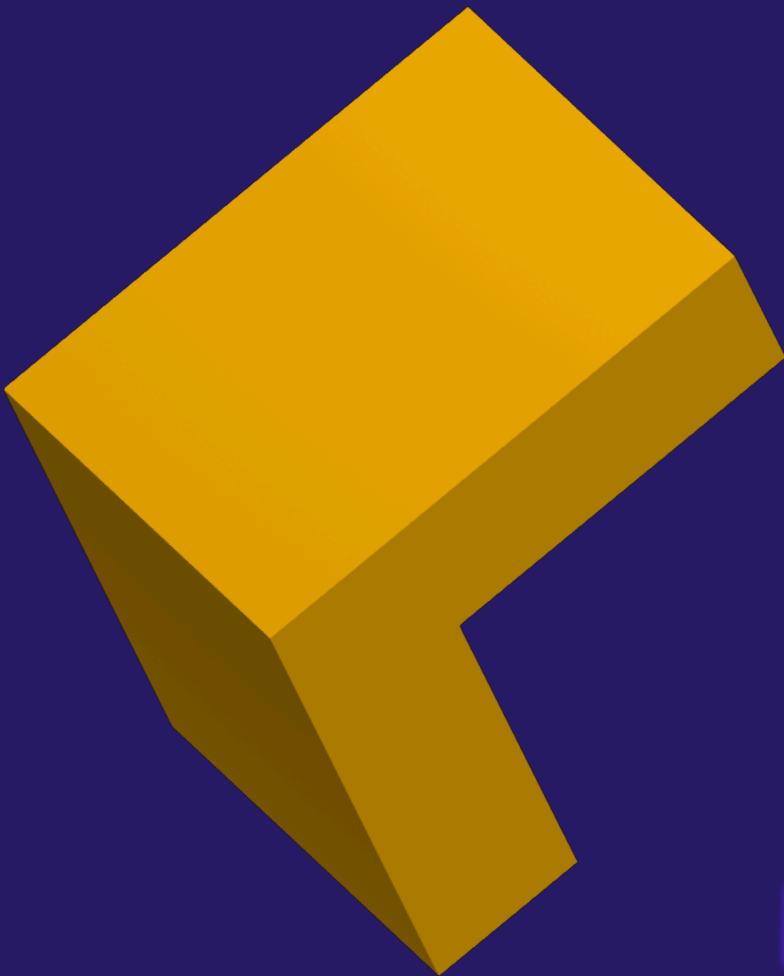
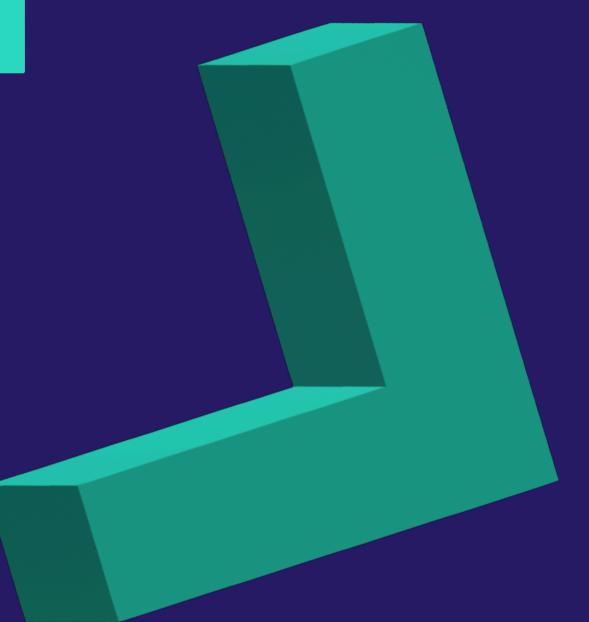
Hlavní výhody:

- Jednotná pravidla zápisu a použití
- Snadná čitelnost
- Rychlejší vývoj

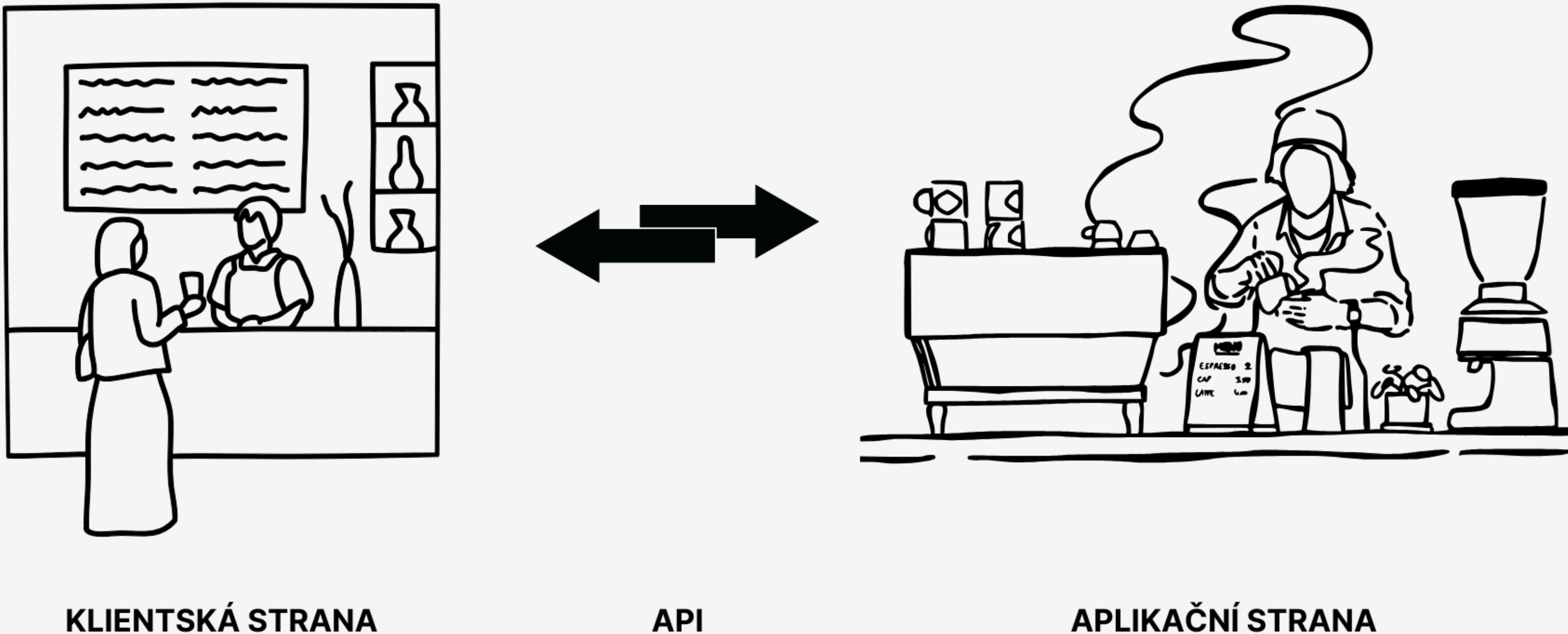


02

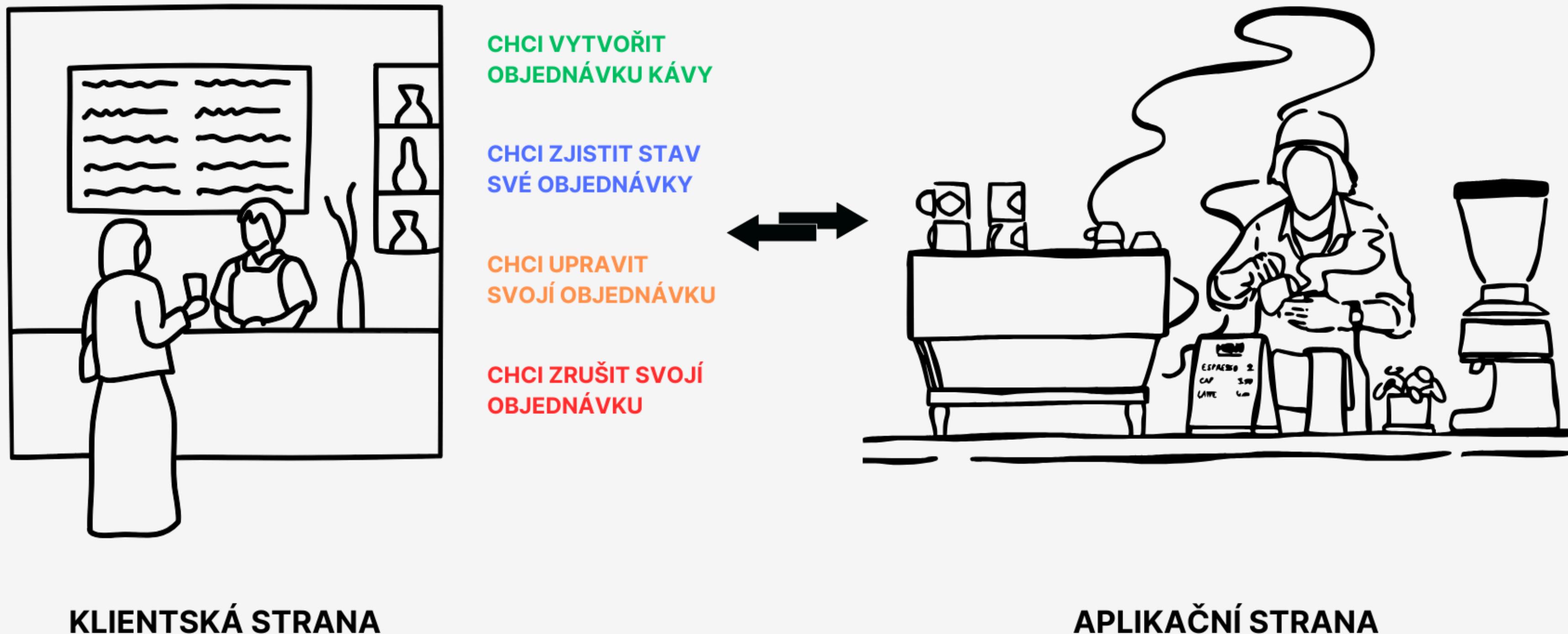
ZÁKLADY REST API



Jak funguje REST API?



Jak funguje REST API?



Co je CRUD, URL, HTTP,...?

CRUD - Create, Read, Update, Delete - **základní** metody pro REST

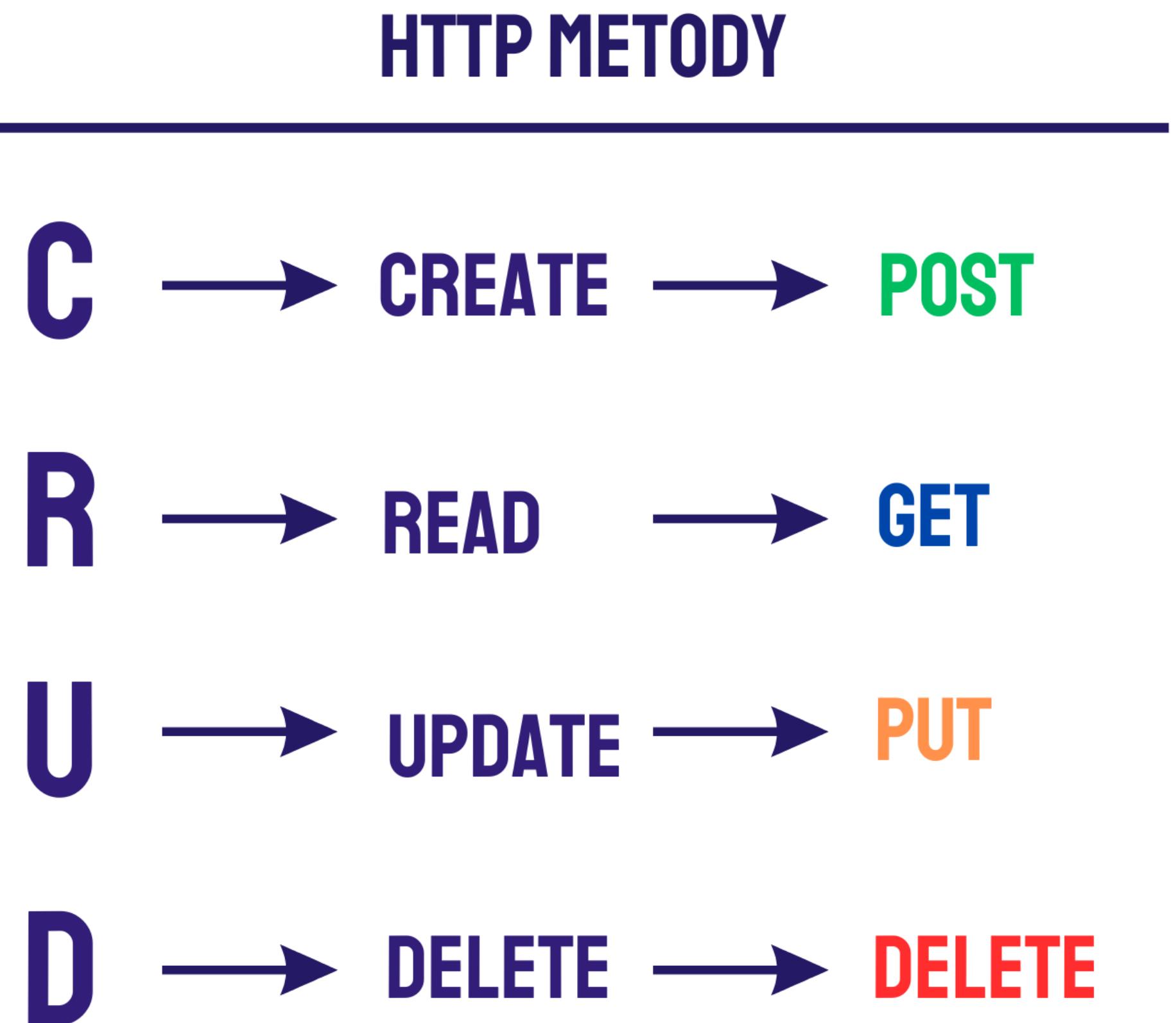
API operace

Rozšířené REST metody:

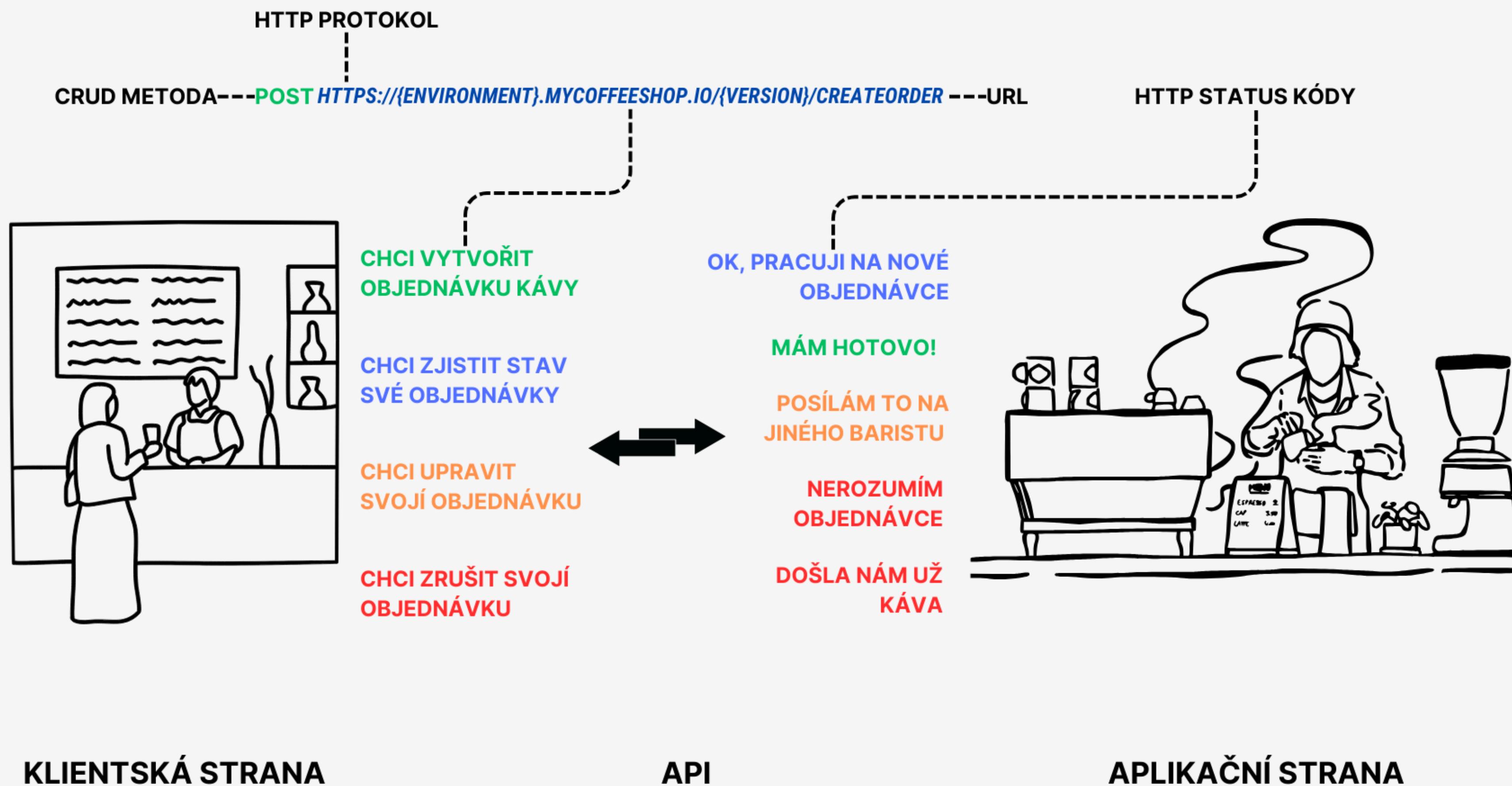
- **PATCH** - Částečně aktualizuj, ale nenahrazuj celý záznam.
- **HEAD** - Vrat' jen hlavičku odpovědi.
- **OPTIONS** - Vrat' možnosti serveru.
- **TRACE** - Debug a diagnostika
- **CONNECT** - Vytvoř tunel client-to-server.

Co je dále důležité znát?

- **URL** - Uniform Resource Locator - specifikace umístění zdroje
- **HTTP** - HyperText Transfer Protocol - protokol pro komunikaci s WWW servery



Jak funguje REST API?



Jaké existují HTTP status kódy?

1xx – Informační

Tyto kódy naznačují, že server přijal požadavek a pokračuje ve zpracování.

Používají se zřídka.

- 100 Continue – Klient může pokračovat v odesílání dalších částí požadavku.
- 101 Switching Protocols – Server přepíná na jiný protokol požadovaný klientem.

2xx – Úspěšné

Požadavek byl úspěšně přijat, pochopen a zpracován.

- 200 OK – Požadavek byl úspěšný.
- 201 Created – Nový zdroj byl úspěšně vytvořen.

3xx – Přesměrování

Požadavek vyžaduje další akci ze strany klienta, obvykle přesměrování na jiný URL.

- 301 Moved Permanently – Požadovaný zdroj byl trvale přesunut na jinou adresu.
- 304 Not Modified – Klient může použít uloženou kopii (cache), protože obsah se nezměnil.

4xx – Chyby na straně klienta

Kódy této kategorie značí, že chyba byla způsobena chybným požadavkem klienta.

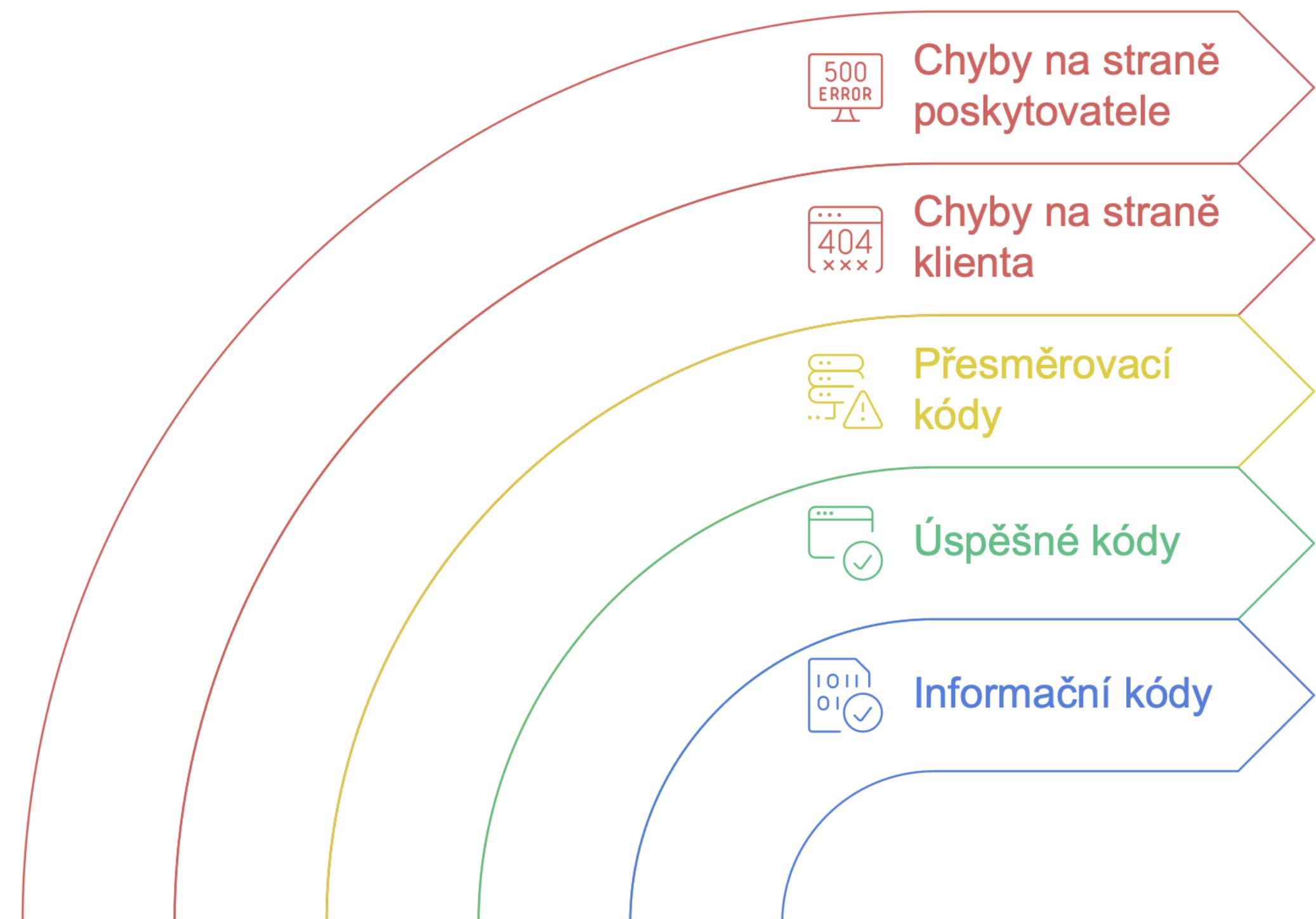
- 400 Bad Request – Požadavek je špatně formulován nebo neplatný.
- 401 Unauthorized – Požadavek vyžaduje ověření (např. přihlášení).

5xx – Chyby na straně serveru

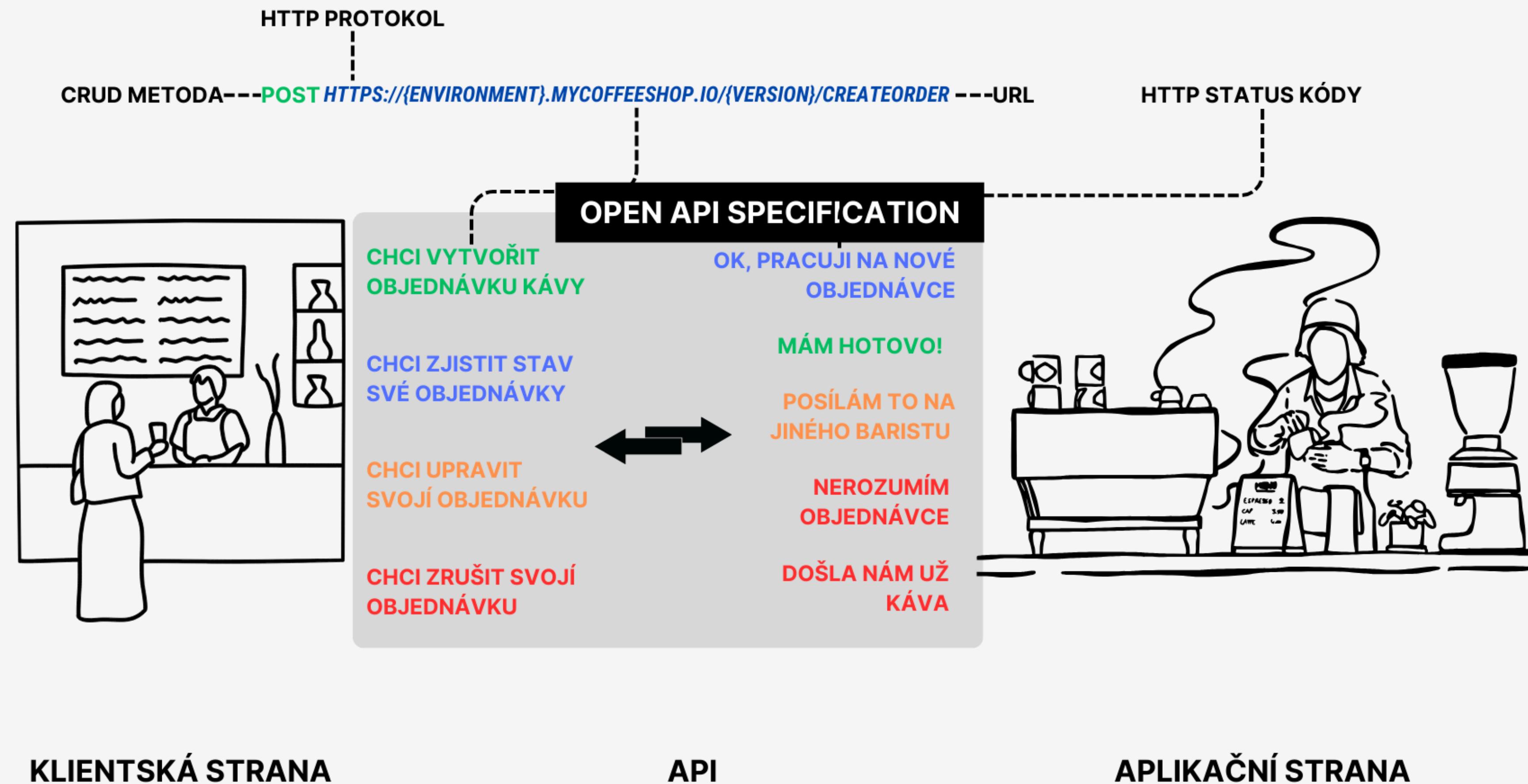
Tyto kódy signalizují, že problém nastal na serveru při zpracování požadavku.

- 500 Internal Server Error – Obecná chyba na straně serveru.
- 503 Service Unavailable – Server není připraven požadavek zpracovat.

Obvyklou příčinou je výpadek serveru z důvodu údržby nebo jeho přetížení.

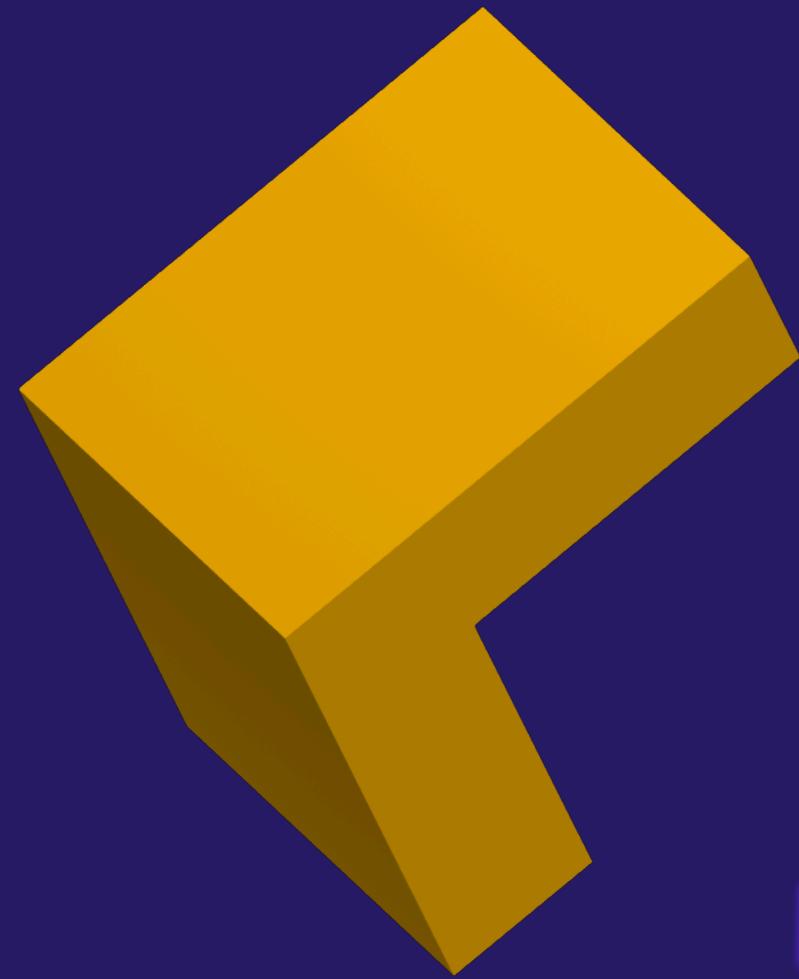


Jak funguje REST API?



03

OPENAPI SPECIFIKACE



Jak funguje OAS?

OpenAPI Specification (OAS) je standardizovaný formát popisu RESTful API. Umožňuje vývojářům definovat rozhraní API tak, aby bylo čitelné jak pro lidi, tak pro stroje.

K čemu slouží?

- **Dokumentace API:** Poskytuje přehledné informace o endpointy, parametrech a odpovědích.
- **Automatizace:** Generování kódu klientů, serverů a testovacích sad.
- **Standardizace:** Usnadňuje sdílení a spolupráci mezi vývojáři.

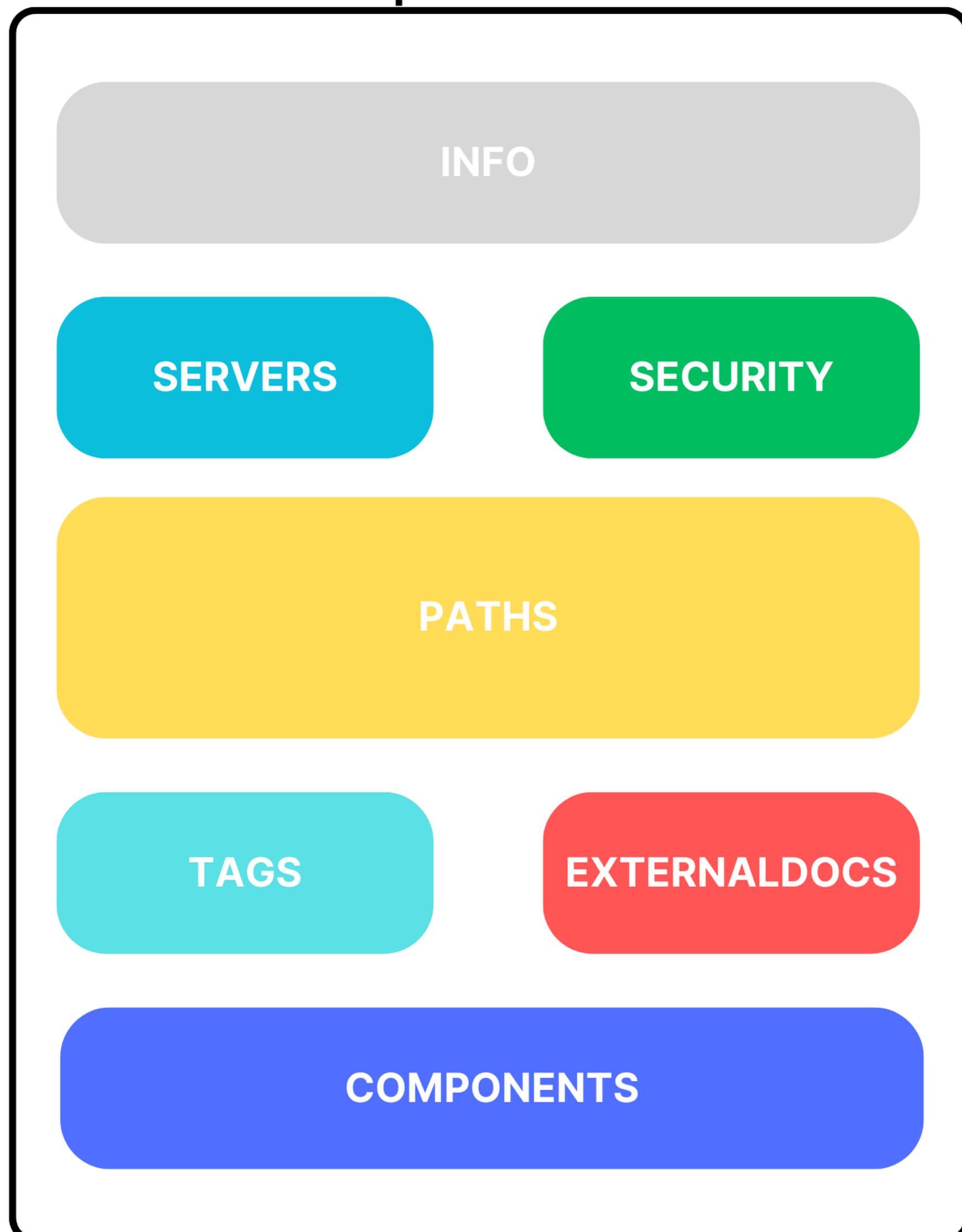
Jaká je historie OAS?

2010: Swagger vyvinut firmou Wordnik jako nástroj pro tvorbu API dokumentace.

2015: Swagger byl přejmenován na OpenAPI Specification a předán pod správu OpenAPI Initiative.

Dnes: OAS je široce uznávaným standardem pro popis API, s podporou mnoha nástrojů (Swagger UI, Postman, Redoc).

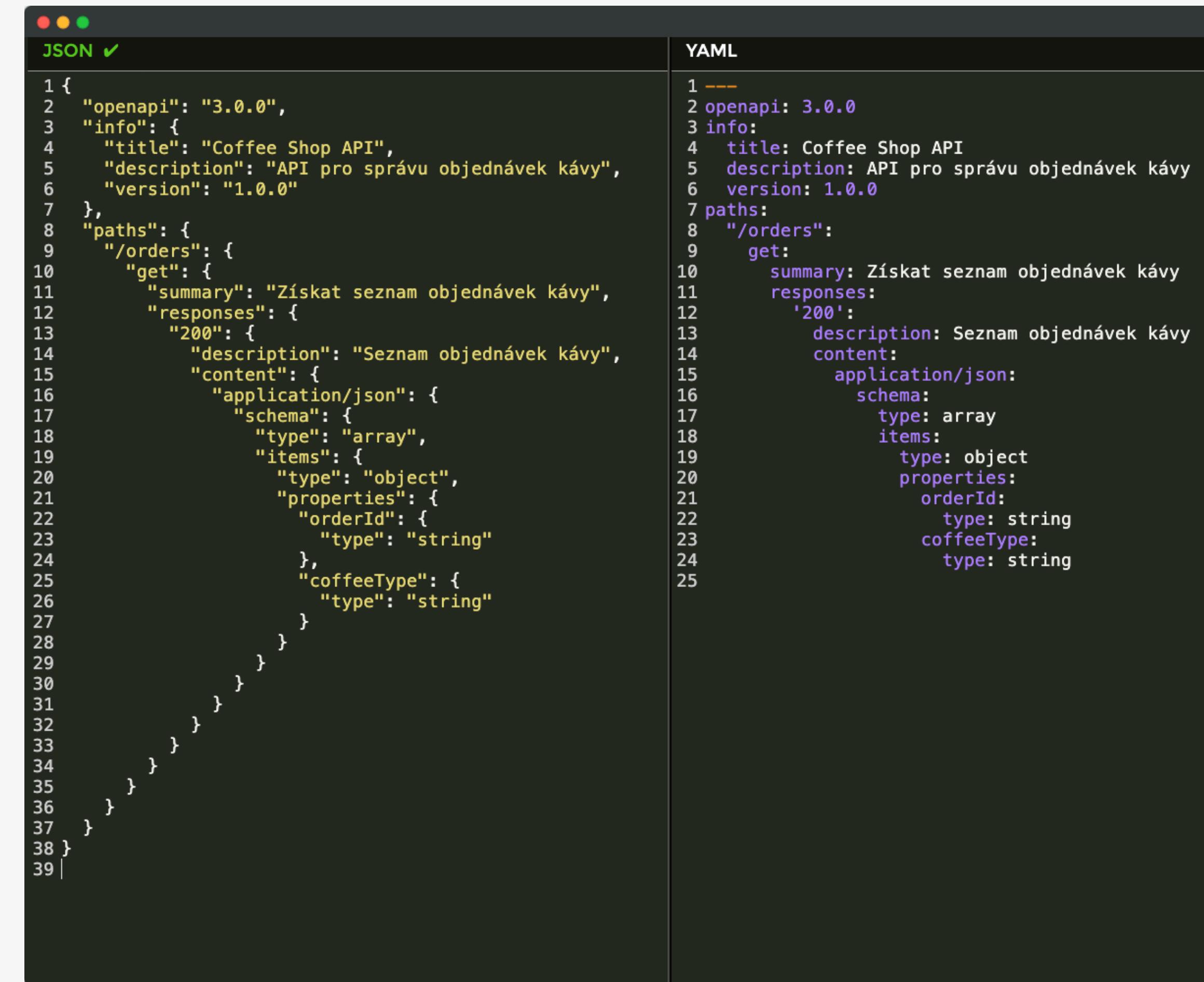
OpenAPI 3.0



Jak se zapisuje OAS?

Pro zápis v OAS se používají dva hlavní formáty: **JSON** a **YAML**. OAS staví na JSON Schema specifikaci, což znamená, že definice datových typů, struktur a validačních pravidel v OAS využívají prvky JSON Schema.

Aspekt	JSON	YAML
Čitelnost	Méně čitelný pro lidi díky závorkám a uvozovkám	Lepší čitelnost díky odsazení a absenci uvozovek
Syntaxe	Vyžaduje čárky, závorky {} pro objekty, [] pro pole	Flexibilní struktura založená na odsazení
Reprezentace dat	Používá závorky {} pro objekty, hranaté závorky [] pro pole	Používá odsazení a pomlčky - pro seznamy
Velikost souboru	Často větší kvůli syntaxi (např. uvozovky)	Kompaktnější a úspornější
Případ užití	Běžné v komunikaci mezi stroji, snadno parsovatelné programy	Preferováno v dokumentaci, kde je důležitá čitelnost pro lidi



The screenshot shows a code editor with two tabs: "JSON" and "YAML". Both tabs contain the same OpenAPI specification for a "Coffee Shop API" that lists an endpoint for getting coffee orders. The JSON tab on the left is in dark mode and shows the JSON structure with line numbers. The YAML tab on the right is also in dark mode and shows the YAML representation of the same API definition. The code is identical in both tabs, demonstrating how the JSON Schema is represented in both formats.

```
1 {  
2   "openapi": "3.0.0",  
3   "info": {  
4     "title": "Coffee Shop API",  
5     "description": "API pro správu objednávek kávy",  
6     "version": "1.0.0"  
7   },  
8   "paths": {  
9     "/orders": {  
10       "get": {  
11         "summary": "Získat seznam objednávek kávy",  
12         "responses": {  
13           "200": {  
14             "description": "Seznam objednávek kávy",  
15             "content": {  
16               "application/json": {  
17                 "schema": {  
18                   "type": "array",  
19                   "items": {  
20                     "type": "object",  
21                     "properties": {  
22                       "orderId": {  
23                         "type": "string"  
24                       "coffeeType": {  
25                         "type": "string"  
26                       }  
27                     }  
28                   }  
29                 }  
30               }  
31             }  
32           }  
33         }  
34       }  
35     }  
36   }  
37 }  
38 |  
39 |
```

```
1 ---  
2 openapi: 3.0.0  
3 info:  
4   title: Coffee Shop API  
5   description: API pro správu objednávek kávy  
6   version: 1.0.0  
7 paths:  
8   "/orders":  
9     get:  
10       summary: Získat seznam objednávek kávy  
11       responses:  
12         '200':  
13           description: Seznam objednávek kávy  
14           content:  
15             application/json:  
16               schema:  
17                 type: array  
18                 items:  
19                   type: object  
20                   properties:  
21                     orderId:  
22                       type: string  
23                     coffeeType:  
24                       type: string
```

Jaké datové typy se používají v OAS?

Pro zápis v OAS se používá několik datových typů.

Typ	Popis	Příklad
string	Textový řetězec	"Hello World"
number	Číslo (desetinné)	3.14, -99.99
integer	Celé číslo	42, -10
boolean	Pravdivostní hodnota	true, false
array	Pole hodnot	["apple", "banana", "cherry"]
object	Objekt	{"name": "John", "age": 30}
null	Prázdná hodnota	null

The screenshot shows a code editor window with a dark theme. The title bar says "Welcome" and "test.yaml 9+". The sidebar has icons for search, refresh, and other file operations. The main area displays the following YAML code:

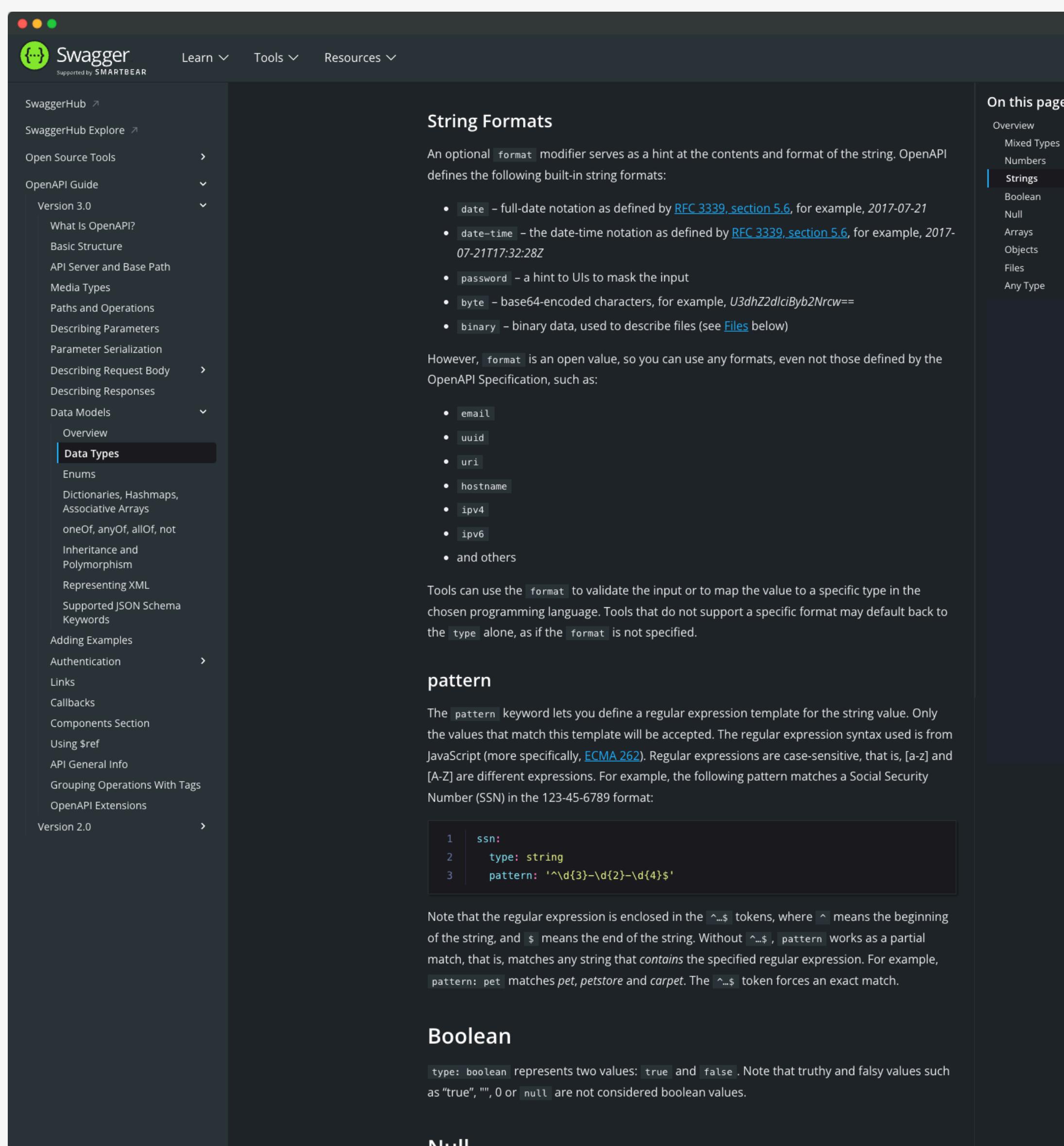
```
1   stringType:
2     type: string
3     example: "Hello World"
4   dateTimeType:
5     type: string
6     format: date-time
7     example: "2025-01-31T14:00:00Z"
8
9   numberType:
10    type: number
11    example: 3.14
12   integerType:
13    type: integer
14    format: int32
15    example: 42
16   booleanType:
17    type: boolean
18    example: true
19   arrayType:
20    type: array
21    items:
22      type: string
23      example: ["apple", "banana", "cherry"]
24   objectType:
25    type: object
26    properties:
27      name:
28        type: string
29        age:
30          type: integer
31          example:
32            name: "Alice"
33            age: 30
34   nullableString:
35     type: string
36     nullable: true
37     example: null
38
```

The status bar at the bottom shows "16 △ 0 ⌂ 0 Live Share".

Jaké datové formáty se používají v OAS?

Pro zpřesnění datového typu se používají formáty. Příklad formátů pro **string**:

Atribut	Popis	Příklad
format: date	Datum ve formátu YYYY-MM-DD	"2025-01-31"
format: date-time	Datum a čas podle ISO 8601	"2025-01-31T10:15:30Z"
format: password	Maskované heslo	"mySuperSecret"
format: byte	Base64 kódovaná data	"U29tZVRleHQ="
format: binary	Binární data (soubor)	(používá se pro nahrávání souborů)



The screenshot shows the Swagger UI interface with the sidebar open. The 'Data Types' section is selected under 'String'. The main content area is titled 'String Formats' and explains that an optional `format` modifier serves as a hint for string contents and format. It defines built-in string formats like `date`, `date-time`, `password`, `byte`, and `binary`. It also notes that `format` is an open value allowing for any format. Below this, the 'pattern' section is introduced, explaining how it defines a regular expression template for string values. A code example for a Social Security Number (SSN) is shown:

```
1 | ssn:  
2 |   type: string  
3 |   pattern: '^\\d{3}-\\d{2}-\\d{4}$'
```

Note that the regular expression is enclosed in the `^$` tokens, where `^` means the beginning of the string and `$` means the end of the string. Without `^$`, `pattern` works as a partial match, matching any string that contains the specified regular expression. For example, `pattern: pet` matches `pet`, `petstore` and `carpet`. The `^$` token forces an exact match.

Boolean
`type: boolean` represents two values: `true` and `false`. Note that truthy and falsy values such as `"true"`, `""`, `0` or `null` are not considered boolean values.

Null

Jak funguje OAS?

OAS se z pohledu zápisu dělá na několik kategorií:

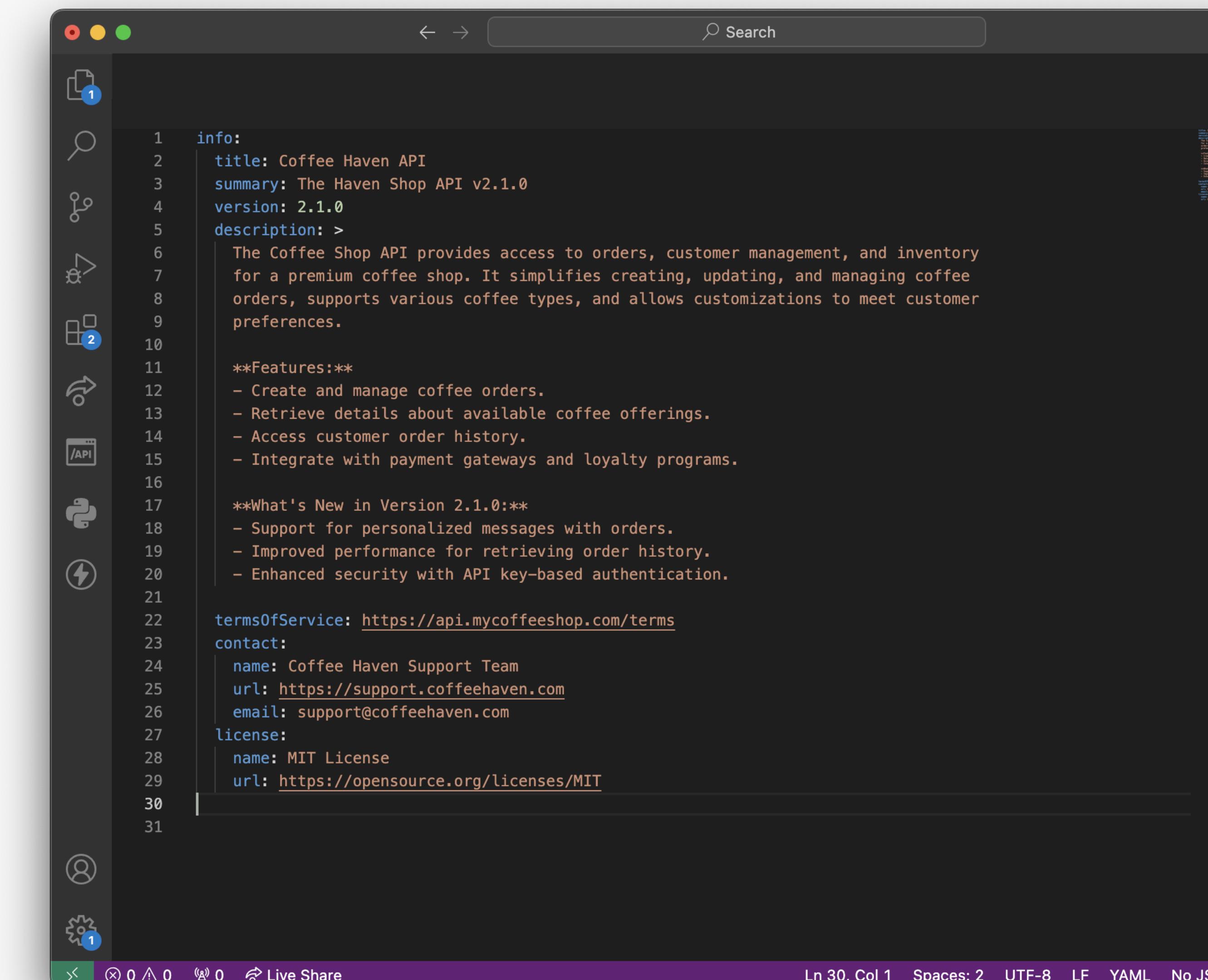
- 1. Info** - Obsahuje metadata o API (název, verze, popis).
- 2. Paths** - Definuje jednotlivé endpointy API a jejich operace (GET, POST, PUT, DELETE).
- 3. Components** - Umožňuje opětovné použití definic (např. schémat dat).
- 4. Security a x-parametrizace** - Definuje použití zabezpečení a custom parametrizace.

The screenshot shows the OpenAPI Editor extension in Visual Studio Code. The left sidebar displays the 'OPENAPI: OUTLINE' tree view, which includes sections for General, Servers, Security, Tags, Operation ID, Paths, /orders (with post, get, patch operations), Components, Security Schemes, and Schemas. The main code editor area shows the corresponding JSON code for an OpenAPI 3.0 specification. The code defines a 'Coffee Shop API' with version 1.0.0, a contact person, and two servers: a production server and a sandbox server. It also defines a '/orders' path with methods post, get, and patch, each with their respective requestBody and responses. The 'Components' section contains a schema for an order. The 'Security Schemes' section is currently empty.

```
OpenAPI 3.0.X (v3.json) | Scan | Audit
openapi: 3.0.0
info:
  title: Coffee Shop API
  version: 1.0.0
  description: API for my coffee shop.
  contact:
    name: Coffee Shop Support
    email: support@coffeeshop.com
servers:
  - url: https://api.coffeeshop.com/v1
    description: Production server
  - url: https://sandbox.api.coffeeshop.com/v1
    description: Sandbox server
paths:
  /orders:
    post:
      requestBody:
      responses:
    get:
      parameters:
      responses:
    patch:
      parameters:
      requestBody:
      responses:
      delete
    Components:
      Security Schemes:
      Schemas:
        application/json:
          schema:
            $ref: '#/components/schemas/Order'
responses:
  201:
    description: Order was created.
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/Order'
  400:
    description: Bad request.
  get:
    summary: Get current order status.
    operationId: getOrders
    parameters:
      - name: status
        in: query
        description: Filter by current status.
        required: false
        schema:
          type: string
```

Základní části OAS - Info

Info objekt poskytuje metadata o rozhraní API. Tato metadata mohou být v případě potřeby použita klienty. Dat také mohou být dále zobrazena v různých editorech, případně developer portálech.



§ 4.8.2.1 Fixed Fields

Field Name	Type	Description
title	string	REQUIRED. The title of the API.
summary	string	A short summary of the API.
description	string	A description of the API. [CommonMark] syntax <i>MAY</i> be used for rich text representation.
termsOfService	string	A URI for the Terms of Service for the API. This <i>MUST</i> be in the form of a URI.
contact	Contact Object	The contact information for the exposed API.
license	License Object	The license information for the exposed API.
version	string	REQUIRED. The version of the OpenAPI Document (which is distinct from the OpenAPI Specification version or the version of the API being described or the version of the OpenAPI Description).

This object *MAY* be extended with [Specification Extensions](#).

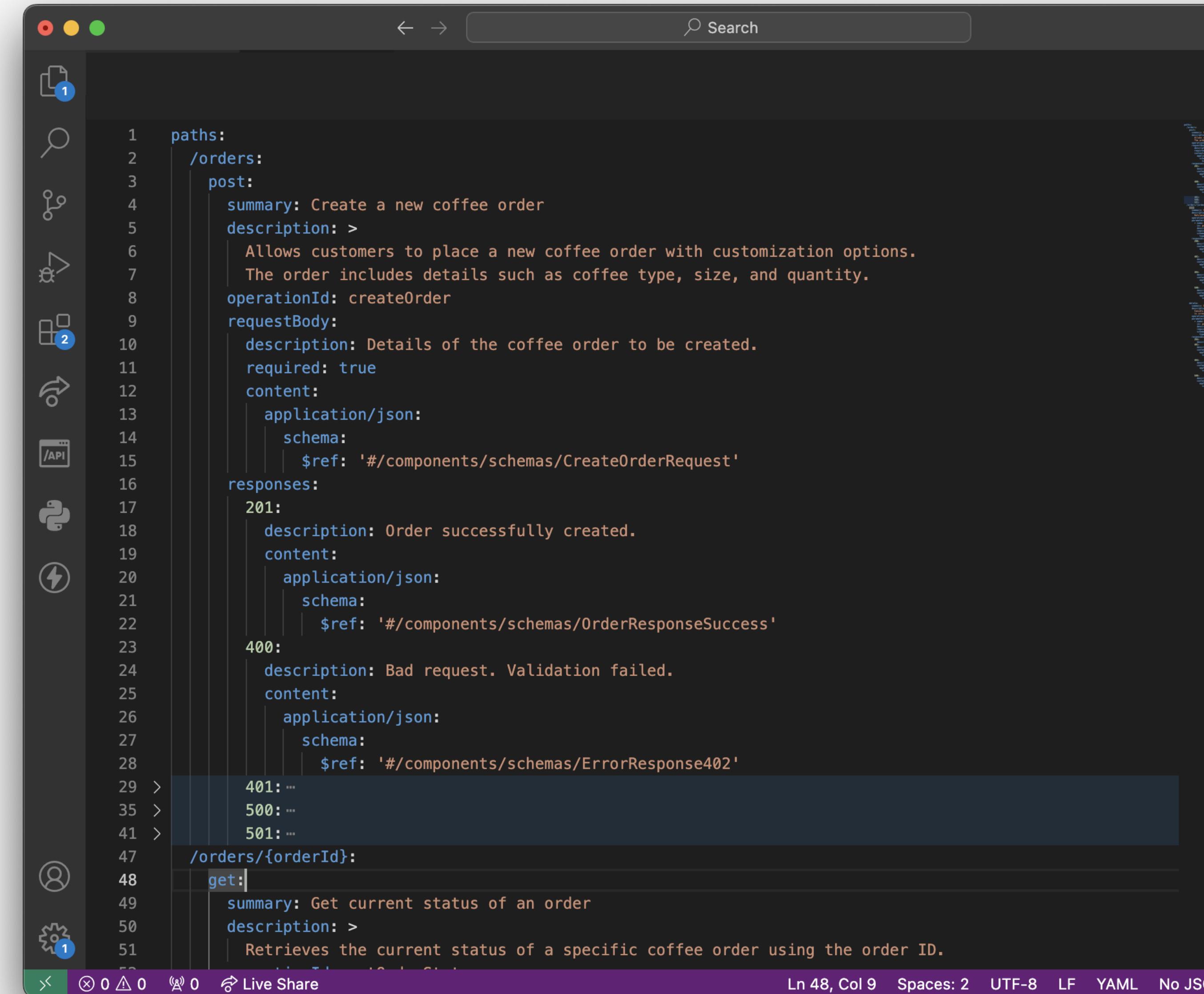
```
1 info:
2   title: Coffee Haven API
3   summary: The Haven Shop API v2.1.0
4   version: 2.1.0
5   description: >
6     The Coffee Shop API provides access to orders, customer management, and inventory
7     for a premium coffee shop. It simplifies creating, updating, and managing coffee
8     orders, supports various coffee types, and allows customizations to meet customer
9     preferences.
10
11 **Features:***
12   - Create and manage coffee orders.
13   - Retrieve details about available coffee offerings.
14   - Access customer order history.
15   - Integrate with payment gateways and loyalty programs.
16
17 **What's New in Version 2.1.0:***
18   - Support for personalized messages with orders.
19   - Improved performance for retrieving order history.
20   - Enhanced security with API key-based authentication.
21
22 termsOfService: https://api.mycoffeeshop.com/terms
23 contact:
24   name: Coffee Haven Support Team
25   url: https://support.coffeehaven.com
26   email: support@coffeehaven.com
27 license:
28   name: MIT License
29   url: https://opensource.org/licenses/MIT
30
31
```

Základní části OAS - Paths

Paths definuje všechny dostupné endpointy API a popisuje, jak mohou být použity. Tato sekce poskytuje strukturovaný přehled toho, jak komunikovat s API a jaké operace jsou podporovány.

Obsah Paths

- 1. Endpointy:** Každý klíč v sekci paths reprezentuje konkrétní cestu API.
- 2. HTTP metody:** Pro každou cestu jsou uvedeny dostupné HTTP metody.
- 3. Operace:** Každá metoda definuje jednu operaci, včetně parametrů, požadavků, odpovědí a případně dalších atributů.



```
1  paths:
2    /orders:
3      post:
4        summary: Create a new coffee order
5        description: >
6          Allows customers to place a new coffee order with customization options.
7          The order includes details such as coffee type, size, and quantity.
8        operationId: createOrder
9        requestBody:
10          description: Details of the coffee order to be created.
11          required: true
12          content:
13            application/json:
14              schema:
15                $ref: '#/components/schemas/CreateOrderRequest'
16        responses:
17          201:
18            description: Order successfully created.
19            content:
20              application/json:
21                schema:
22                  $ref: '#/components/schemas/OrderResponseSuccess'
23          400:
24            description: Bad request. Validation failed.
25            content:
26              application/json:
27                schema:
28                  $ref: '#/components/schemas/ErrorResponse402'
29        > 401: ...
35        > 500: ...
41        > 501: ...
47    /orders/{orderId}:
48      get:
49        summary: Get current status of an order
50        description: >
51          Retrieves the current status of a specific coffee order using the order ID.
```

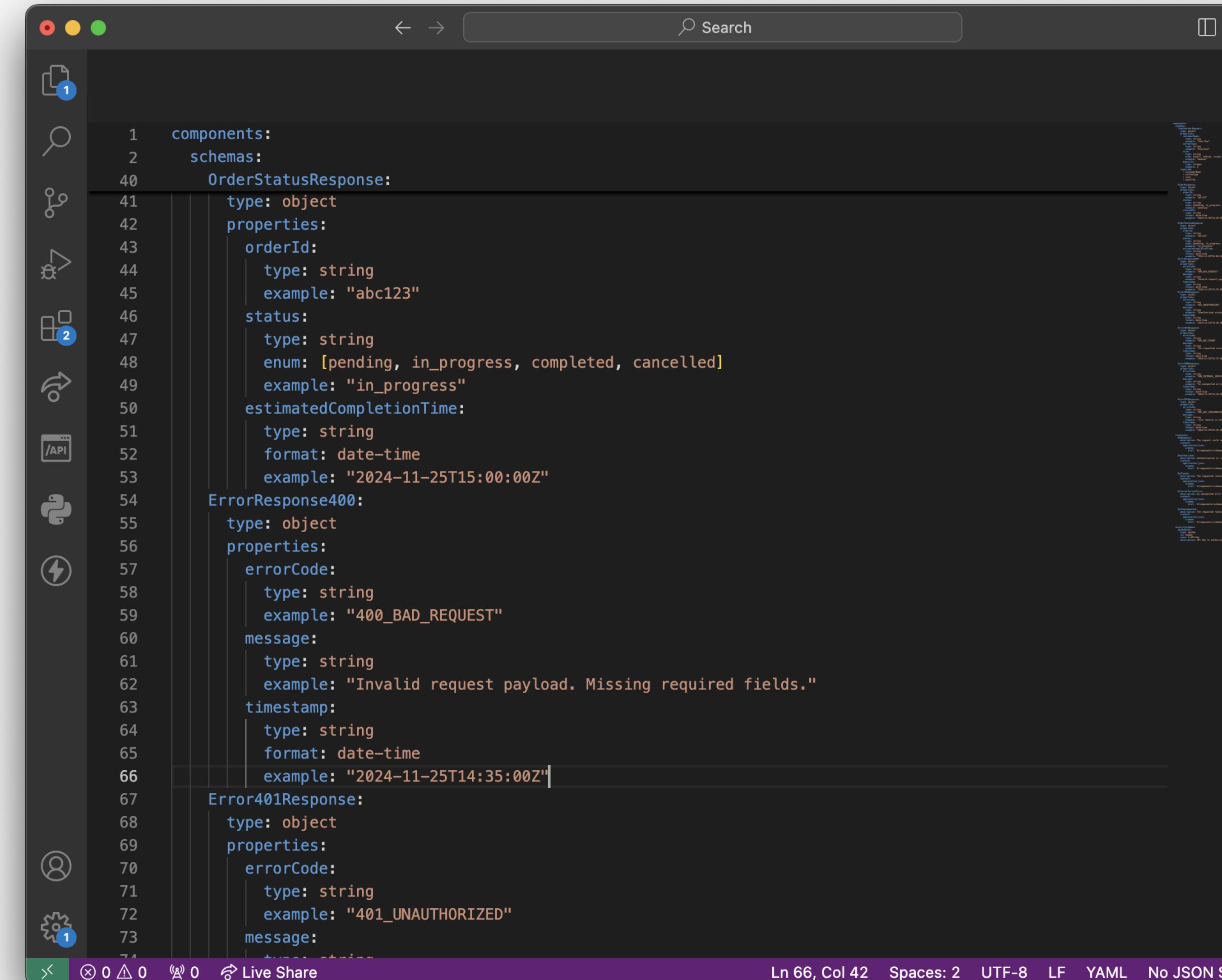
Ln 48, Col 9 Spaces: 2 UTF-8 LF YAML No JSON

Základní části OAS - Components

Components slouží jako centrální úložiště znovupoužitelných definic pro různé části specifikace API. Tato sekce umožňuje definovat často opakující se objekty, schémata, bezpečnostní schémata a další informace na jednom místě.

Obsah Components

1. **Schemas**
2. **Responses**
3. **Parameters**
4. **Requests**
5. **Security schemas**
6. **Examples**
7. **Headers**
8. **Links**
9. **Callbacks**



The screenshot shows a code editor window displaying an OpenAPI (OAS) specification in YAML format. The code is organized into sections under the 'components' key:

```
1 components:
2   schemas:
3     OrderStatusResponse:
4       type: object
5       properties:
6         orderId:
7           type: string
8           example: "abc123"
9         status:
10          type: string
11          enum: [pending, in_progress, completed, cancelled]
12          example: "in_progress"
13          estimatedCompletionTime:
14            type: string
15            format: date-time
16            example: "2024-11-25T15:00:00Z"
17
18     ErrorResponse400:
19       type: object
20       properties:
21         errorCode:
22           type: string
23           example: "400_BAD_REQUEST"
24         message:
25           type: string
26           example: "Invalid request payload. Missing required fields."
27         timestamp:
28           type: string
29           format: date-time
30           example: "2024-11-25T14:35:00Z"
31
32     Error401Response:
33       type: object
34       properties:
35         errorCode:
36           type: string
37           example: "401_UNAUTHORIZED"
38         message:
```

The editor interface includes a sidebar with icons for file operations, a search bar at the top right, and a status bar at the bottom indicating the current line (Ln 66), column (Col 42), and encoding (UTF-8). The status bar also shows options for LF, YAML, and No JSON.

Základní části OAS - Security & x-parametry

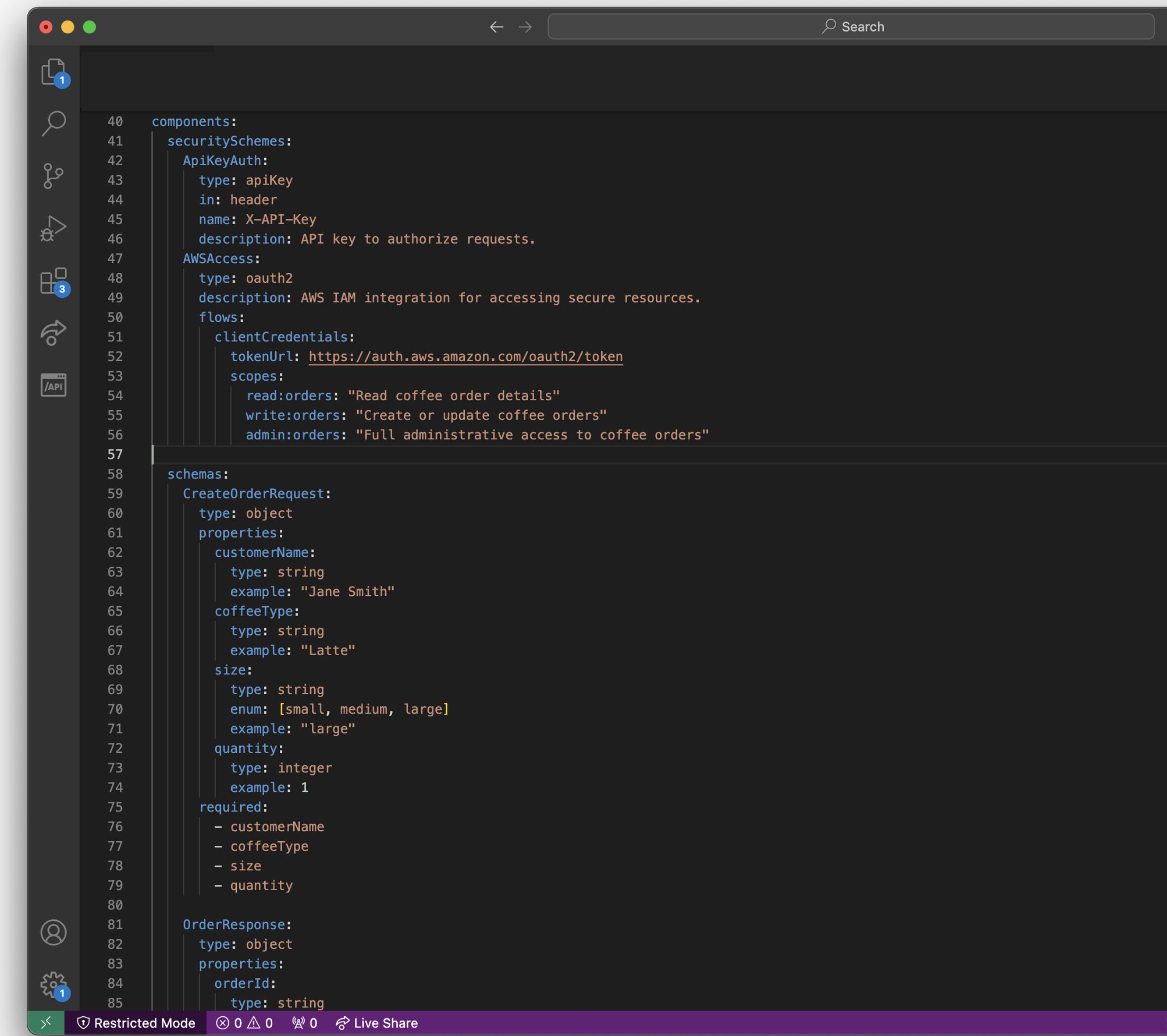
Security schémata slouží k definování způsobů, jakými se klienti autentizují a autorizují při přístupu k API. Umožňují popsat různé metody zabezpečení, které API podporuje, a integrují tyto mechanismy přímo do specifikace.

Způsoby autentizace podporované v OAS 3.0.0:

- API klíč
- HTTP Authentizace pomocí bearer tokenů
- OAuth 2.0
- OpenID Connect

x-parametry jsou rozšíření specifikace OAS, která umožňují vývojářům přidávat vlastní metadata nebo specifické konfigurace. Využívají prefix **x-** pro označení, že jde o uživatelsky definované atributy, a nejsou standardní součástí OAS.

```
1  paths:
2    /orders:
3      get:
4        x-rate-limit: 1000 # Maximum requests per hour
5        x-cache-control: "max-age=3600" # Caching configuration
6
```



```
components:
  securitySchemes:
    ApiKeyAuth:
      type: apiKey
      in: header
      name: X-API-Key
      description: API key to authorize requests.
    AWSAccess:
      type: oauth2
      description: AWS IAM integration for accessing secure resources.
      flows:
        clientCredentials:
          tokenUrl: https://auth.aws.amazon.com/oauth2/token
          scopes:
            read:orders: "Read coffee order details"
            write:orders: "Create or update coffee orders"
            admin:orders: "Full administrative access to coffee orders"

schemas:
  CreateOrderRequest:
    type: object
    properties:
      customerName:
        type: string
        example: "Jane Smith"
      coffeeType:
        type: string
        example: "Latte"
      size:
        type: string
        enum: [small, medium, large]
        example: "large"
      quantity:
        type: integer
        example: 1
    required:
      - customerName
      - coffeeType
      - size
      - quantity

  OrderResponse:
    type: object
    properties:
      orderId:
        type: string
```

Verzování API endpointů

Proč je verzování tak důležité?

- 1. Zachování zpětné kompatibility:** Zajišťuje, že existující klienti budou i nadále fungovat správně, i když dojde ke změnám v API.
- 2. Jasná komunikace:** Pomáhá vývojářům a uživatelům API pochopit, kterou verzi API používají.
- 3. Strategie zrušení podpory (deprecace funkce):** Umožňuje spravovat zrušení starších verzí a zavedení nových funkcí nebo nekompatibilních změn.

V OAS struktuře lze k verzování přistoupit třemi způsoby:

- 1. Verze v cestě URL**
- 2. Verze pomocí query parametrů**
- 3. Verze pomocí hlaviček**

Verzování API endpointů

Verzování v cestě URL:

- **Výhody:** Snadné implementování a pochopení.
- **Nevýhody:** Může způsobit duplicitu verzí a náročnější údržbu.

Verzování pomocí query parametrů:

- **Výhody:** Udržuje URL čisté, snadná úprava parametru pro každou žádost.
- **Nevýhody:** Méně intuitivní, složitější pro nástroje dokumentace.

Verzování pomocí hlaviček:

- **Výhody:** Udržuje URL čistou, není potřeba měnit endpointy.
- **Nevýhody:** Méně viditelné pro koncové uživatele, vyžaduje vlastní parsování hlaviček.

Při použití více verzí vždy myslte na to, že zvyšujete složitost na BE.

```
1 /v1/orders: # Versioned path
2   get:
3     summary: Get list of coffee orders
4     responses:
5       '200':
6         description: List of orders
7         content:
8           application/json:
9             schema:
10            type: array
11
12 /orders: # No version in the path
13   get:
14     summary: Get list of coffee orders
15     parameters:
16       - name: version # Query parameter for version
17         in: query
18         description: Version of the API
19         required: true
20         schema:
21           type: string
22           example: "1"
23
24 /orders: # No version in the path
25   get:
26     summary: Get list of coffee orders
27     parameters:
28       - name: Accept
29         in: header # Version specified in the Accept header
30         description: API version in Accept header
31         required: true
32         schema:
33           type: string
34           example: "application/vnd.coffeeshop.v1+json"
```

Sémantické verzování OAS

Sémantické verzování (SemVer) je standard pro verzování softwarových produktů, který usnadňuje správu změn a zajišťuje zpětnou kompatibilitu.

Sémantické verzování je rozděleno do tří částí:

- **MAJOR**: Změny, které nejsou zpětně kompatibilní (např. odstranění nebo změna API endpointu).
- **MINOR**: Přidání nových funkcí, které jsou zpětně kompatibilní.
- **PATCH**: Opravy chyb nebo malé změny, které neovlivní funkčnost API.

Verze API podle sémantického verzování se zapisuje ve formátu:

MAJOR.MINOR.PATCH

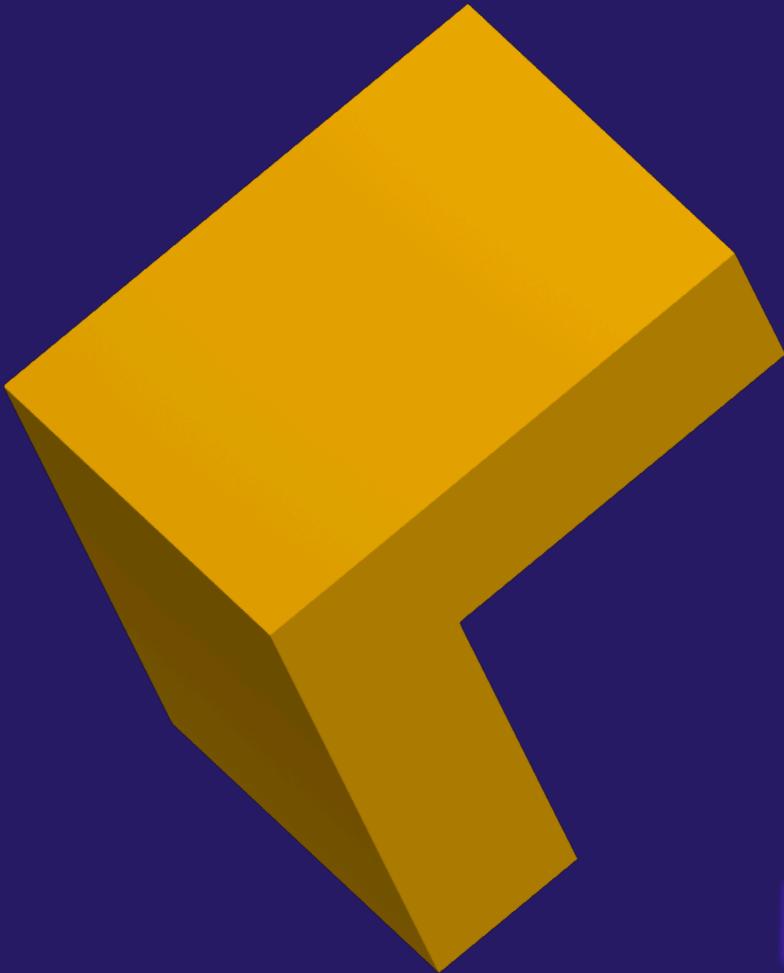
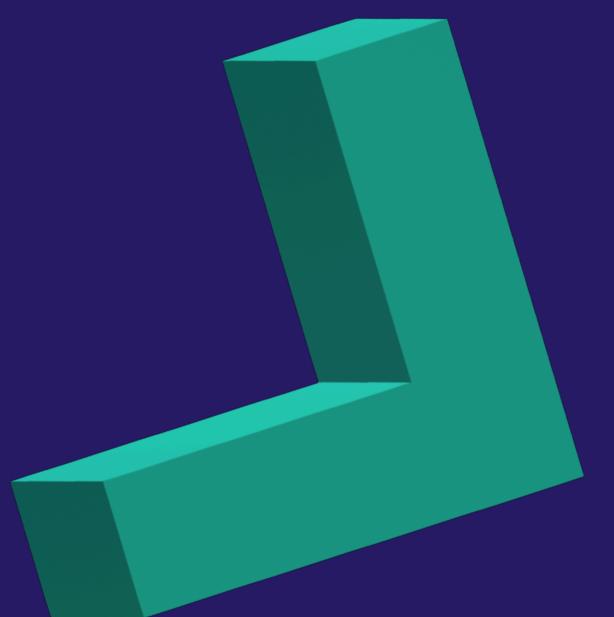
Příklad: 1.2.3

- **1: Major verze** (kompatibilita není zaručena)
- **2: Minor verze** (nové funkce, které jsou zpětně kompatibilní)
- **3: Patch verze** (opravy chyb)



04

POKROČILÉ FUNKCE OAS



Pokročilé funkce OAS

Rozšiřují možnosti popisu API nad rámec základních struktur.

Usnadňují práci s komplexními datovými strukturami a zpřehledňují dokumentaci.

Klíčové pokročilé funkce pro potřeby ČS

Kombinace schémat:

- **oneOf, allOf, anyOf**
- Umožňují vytvářet podmíněné nebo kombinované datové struktury.

Referenční schémata:

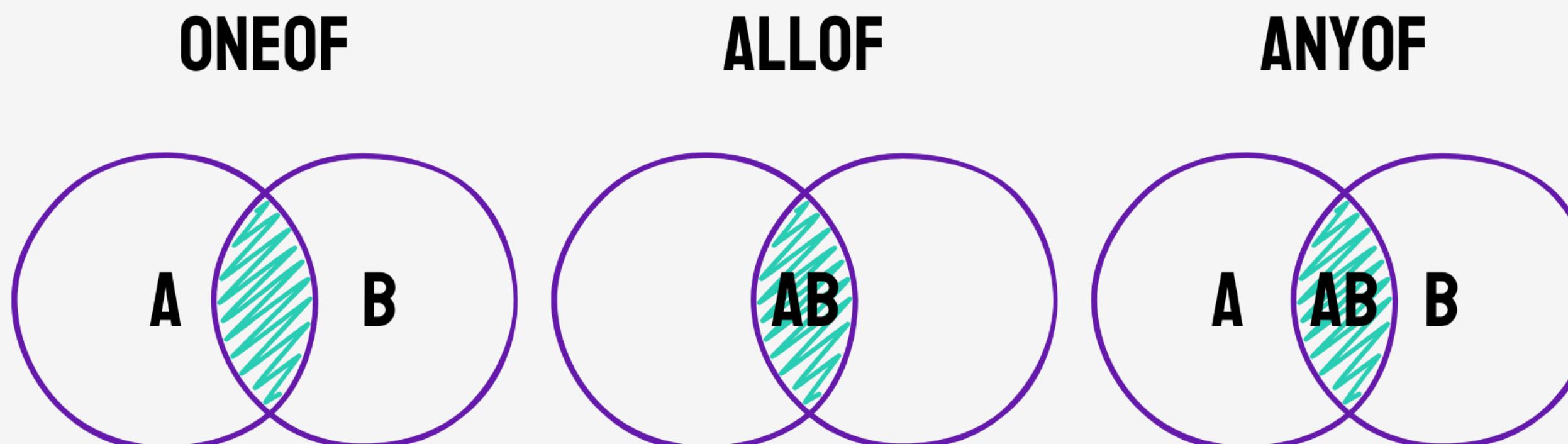
- Schémata lze definovat jednou a opakovaně používat pomocí **\$ref**.
- Zajišťují konzistenci a zjednoduší údržbu.

Příkladová data (examples):

- Poskytují příklady pro jednotlivé odpovědi, požadavky nebo parametry.
- Umožňují variabilitu a testování na různých datech.

oneOf, allOf, anyOf

Funkce	Popis	Příklad použití
oneOf	Objekt musí odpovídat přesně jednomu ze schémat uvedených v oneOf.	Validní zápis odpovídá buď osobě nebo firmě, ale ne oběma zároveň.
allOf	Objekt musí odpovídat všem uvedeným schématům najednou.	Kombinujeme základní vlastnosti osoby a specifických uživatelských práv.
anyOf	Objekt musí odpovídat alespoň jednomu z uvedených schémat.	Uživatel může mít buď základní profil, nebo může být zaměstnanec firmy.



A screenshot of a code editor interface showing a JSON schema definition. The schema includes sections for `oneOf`, `allOf`, and `anyOf`.

```
Scan | Audit
1 oneOf:
2   - type: string
3   - type: integer
4
5 allOf:
6   - $ref: '#/components/schemas/BaseOrder'
7   - $ref: '#/components/schemas/CoffeeOrder'
8
9 Scan | Try it | Audit
10 anyOf:
11   - type: string
12   - type: number
```

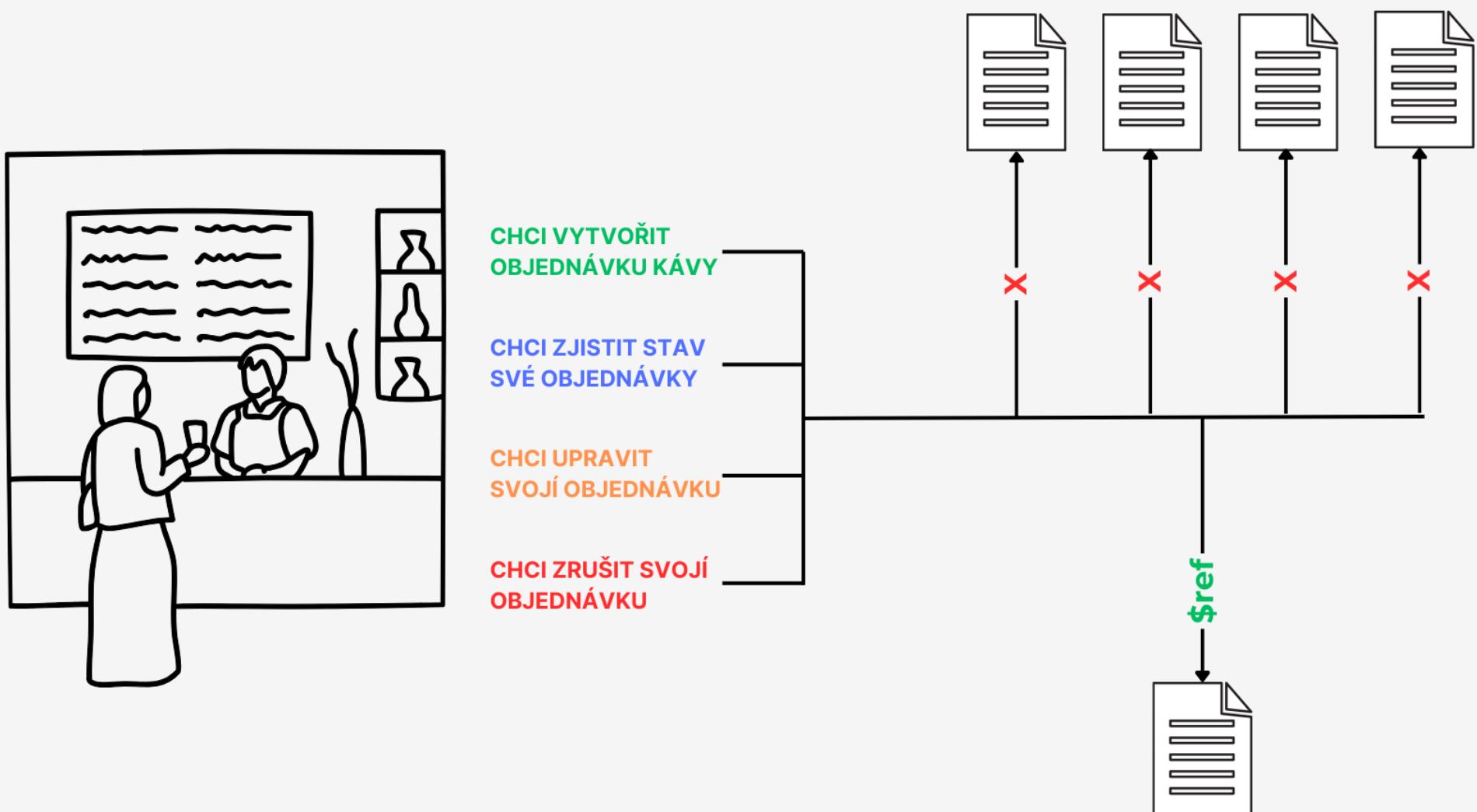
\$ref

Umožňují definovat schéma pouze jednou a používat jej opakovaně.

Pomocí **\$ref** odkazujeme na předdefinované části specifikace.

Výhody využití **\$ref**:

- **Konzistence:** Zajišťují jednotnost napříč specifikací.
- **Údržba:** Snadno upravíte všechny instance změnou jediného schématu.
- **Zjednodušení:** Zlepšují přehlednost a čitelnost OAS.



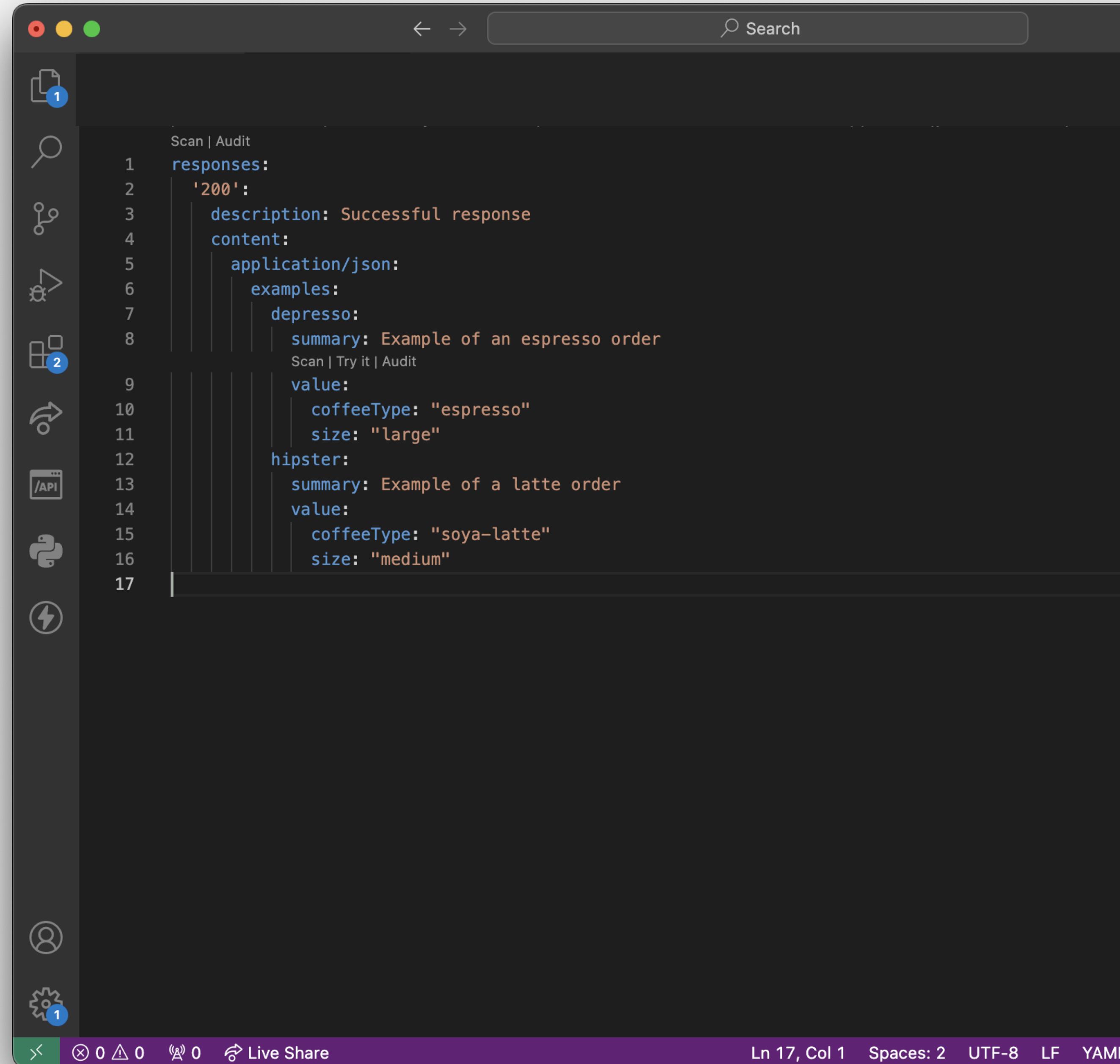
```
Scan | Audit
components:
  schemas:
    CoffeeOrder:
      type: object
      properties:
        coffeeType:
          type: string
        size:
          type: string
paths:
  /orders:
    post:
      summary: Create a coffee order
      requestBody:
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/CoffeeOrder'
```

Ln 19, Col 1 Spaces: 2 UTF-8 LF YAML OpenAPI 3.0.X

examples

Umožňují zobrazovat x různých verzí příkladů pro:

- **Parametry** (query, path, headers)
- **Tělo požadavku** (request body)
- **Odpovědi** (responses)



The screenshot shows a code editor interface with a dark theme. On the left, there's a vertical toolbar with icons for file operations, search, and other development tools. The main area displays a YAML configuration file. The file defines a 'responses' object with two entries: '200' and '404'. The '200' entry has a 'description' of 'Successful response', a 'content' type of 'application/json', and an 'examples' section. It contains two examples: 'depresso' and 'hipster'. Each example has a 'summary' and a 'value' object. The 'depresso' example summary is 'Example of an espresso order' and its value is {coffeeType: "espresso", size: "large"}. The 'hipster' example summary is 'Example of a latte order' and its value is {coffeeType: "soya-latte", size: "medium"}. The '404' entry has a 'description' of 'Not found' and a 'content' type of 'application/json'. The status bar at the bottom shows the file is 17 lines long, 1 column wide, and is saved in UTF-8 format.

```
responses:
  '200':
    description: Successful response
    content:
      application/json:
        examples:
          depresso:
            summary: Example of an espresso order
            Scan | Try it | Audit
            value:
              coffeeType: "espresso"
              size: "large"
          hipster:
            summary: Example of a latte order
            value:
              coffeeType: "soya-latte"
              size: "medium"
  '404':
    description: Not found
    content:
      application/json:
```

deprecated

Označuje funkce, endpointy nebo části API, které už **nejsou doporučeny k používání**.

Typicky naznačuje, že tato část API bude v budoucnu odstraněna nebo nahrazena.

V OAS se zapisuje pomocí tagu **deprecated: true**.

The screenshot shows a Swagger UI interface with the following details:

- Create a new coffee order**: Endpoint to place a new coffee order with details like coffee type and size.
- Request**:
 - Body**:
 - coffeeType**: string (Type of coffee (e.g., espresso, latte, cappuccino)).
 - size**: string (Size of the coffee (e.g., small, medium, large)).
 - extras**: array[string] (List of additional options (e.g., sugar, milk)).
- Responses**:
 - 201**: Order successfully created
 - 400**: Invalid request data
 - 401**: Unauthorized access
 - 500**: Internal server error

OAS JSON (Left Panel):

```
servers:
  - url: https://api.coffeeshop.com/v1
    description: Production server
  - url: https://staging.api.coffeeshop.com/v1
    description: Staging server

paths:
  /orders:
    post:
      tags:
        - Orders
      summary: Create a new coffee order
      deprecated: true
      description: Endpoint to place a new coffee order with details like coffee type and size.
      operationId: createOrder
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/CreateOrderRequest'
      responses:
        '201':
          description: Order successfully created
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/OrderResponse'
        '400':
          description: Invalid request data
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ErrorResponse'
        '401':
          description: Unauthorized access
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ErrorResponse'
        '500':
          description: Internal server error
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ErrorResponse'

  /orders/{orderId}:
    get:
      tags:
        - Status
      summary: Get the status of an order
      description: Retrieve the current status of an existing coffee order using its ID.
      operationId: getOrderStatus
      parameters:
        - name: orderId
          in: path
```

sunset headers

HTTP hlavička, která označuje datum ukončení endpointu nebo funkce.

Umožňuje klientům plánovat migraci na nové části API.

Hlavička Sunset obsahuje datum a čas v ISO 8601 formátu.

Příklad ISO 8601: **2024-11-28T08:36:42Z**

```
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177

  delete:
    tags:
      - Administration
    summary: Cancel an existing order
    description: Remove an existing coffee order from the system.
    operationId: cancelOrder
    parameters:
      - name: orderId
        in: path
        required: true
        schema:
          type: string
          description: The unique identifier of the coffee order.
    responses:
      '204':
        description: "Order successfully canceled. **Note: This endpoint is being deprecated.**"
        headers:
          Sunset:
            description: Indicates when this endpoint will be sunset.
            schema:
              type: string
              format: date-time
              example: 2024-12-31T23:59:59Z
      '400':
        description: Invalid order ID format
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/ErrorResponse'
      '401':
        description: Unauthorized access
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/ErrorResponse'
      '404':
        description: Order not found
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/ErrorResponse'
      '500':
        description: Internal server error
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/ErrorResponse'

  components:
    schemas:
      CreateOrderRequest:
        type: object
        required:
          - coffeeType
          - size
        properties:
          coffeeType:
            type: string
            description: Type of coffee (e.g., espresso, latte, cappuccino).
          size:
            type: string
            description: Size of the coffee (e.g., small, medium, large).
          extras:
            type: array
```

Cancel an existing order

DELETE `orders/{orderId}/`

Remove an existing coffee order from the system.

Request

Path Parameters

orderId string
The unique identifier of the coffee order.

Responses

204 400 401 404 500

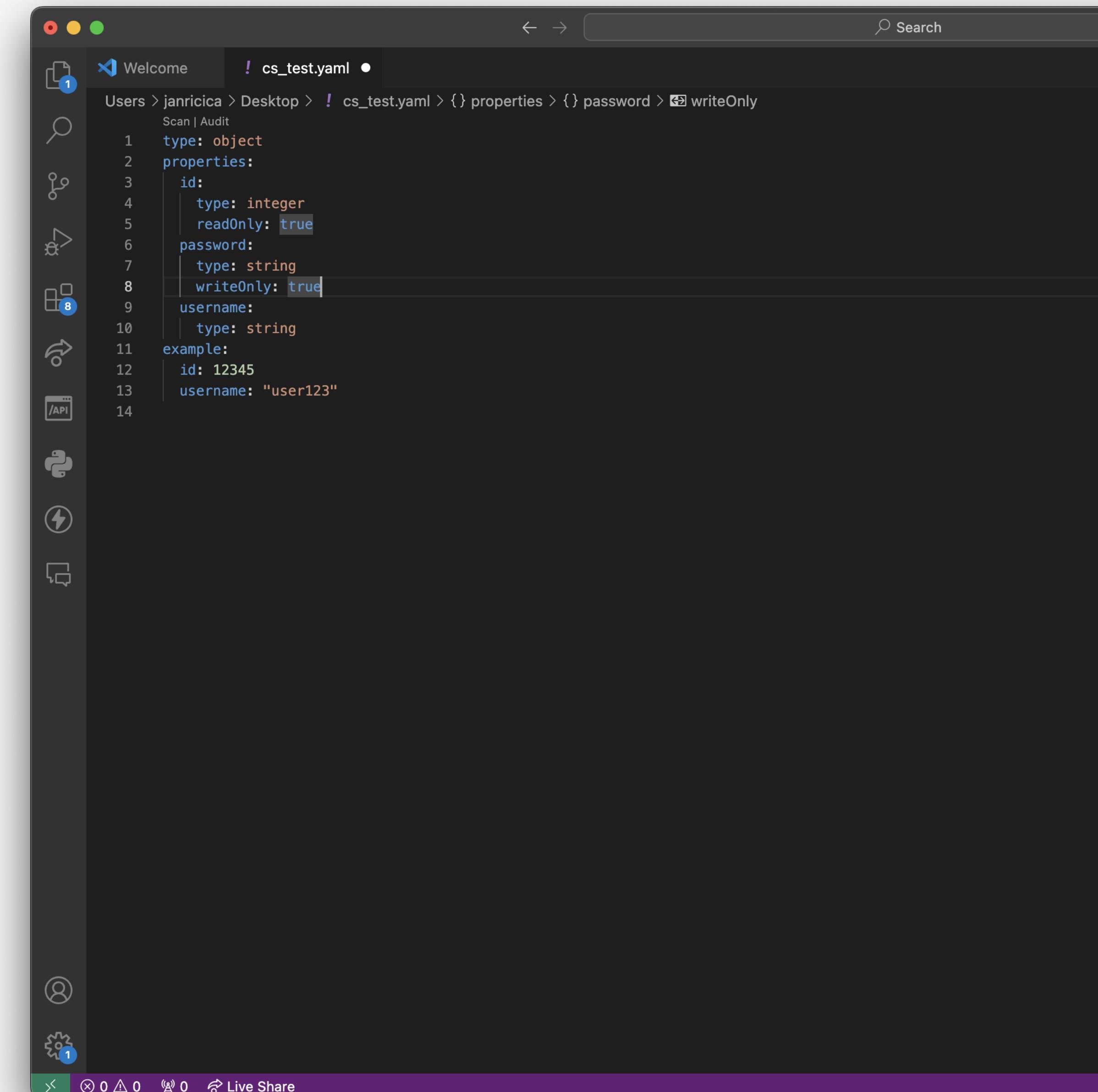
Order successfully canceled. Note: This endpoint is being deprecated.

Headers

Sunset string<date-time>
Indicates when this endpoint will be sunset.
Example: 2024-12-31T23:59:59Z

writeOnly & readOnly

Vlastnost	readOnly	writeOnly
Účel	Pro atributy, které jsou pouze ke čtení.	Pro atributy, které jsou pouze pro zápis.
Dostupnost v požadavku (Request)	Není povoleno – klient nesmí tuto hodnotu posílat.	Povinné nebo volitelné – klient tuto hodnotu posílá.
Dostupnost v požadavku (Response)	Server tuto hodnotu vrací.	Server tuto hodnotu nikdy nezahrnuje do odpovědi.
Typické použití	Automaticky generovaná data (ID, čas vytvoření, stav).	Citlivá data (hesla, tajné klíče).
Příklad hodnoty	id: 12345 (automaticky generované ID)	password: "secret123" (heslo zadané klientem)
Validace	Hodnota je ignorována, pokud je poslána klientem.	Hodnota je vyžadována nebo volitelná podle API specifikace.



The screenshot shows the Swagger UI interface with a YAML configuration file named `cs_test.yaml`. The file defines a schema for a user object with properties for id, password, and username. The `password` property is explicitly defined with `type: string` and `writeOnly: true`, indicating it is a sensitive value that should only be written by the client. The `id` and `username` properties are defined with `type: string` and `readOnly: true`, indicating they are automatically generated or provided by the server.

```
! cs_test.yaml
Users > janricica > Desktop > ! cs_test.yaml > {} properties > {} password > writeOnly
Scan | Audit
1 type: object
2 properties:
3   id:
4     type: integer
5     readOnly: true
6   password:
7     type: string
8     writeOnly: true
9   username:
10    type: string
example:
12      id: 12345
13      username: "user123"
```

OAS 3.1.0

Aspekt	OAS 3.0.x	OAS 3.1.x
Podpora JSON	JSON Schema draft-04	JSON Schema draft 2020-12 - širší validace a flexibilita
Nullabe atribut	Samostatná vlastnost nullable (boolean).	nullable je odstraněno. Používá se null pomocí type pole
Podpora webhooks	NE	ANO
Výchozí hodnota servers	servers je povinné na úrovni specifikace	servers není vyžadované pro podporu asynchronních specifikací
Možnost kombinace více typů	NE	ANO - např. type: [string, null]
Porpora identifikátorů	ANO - \$ref	ANO - \$id, \$ref

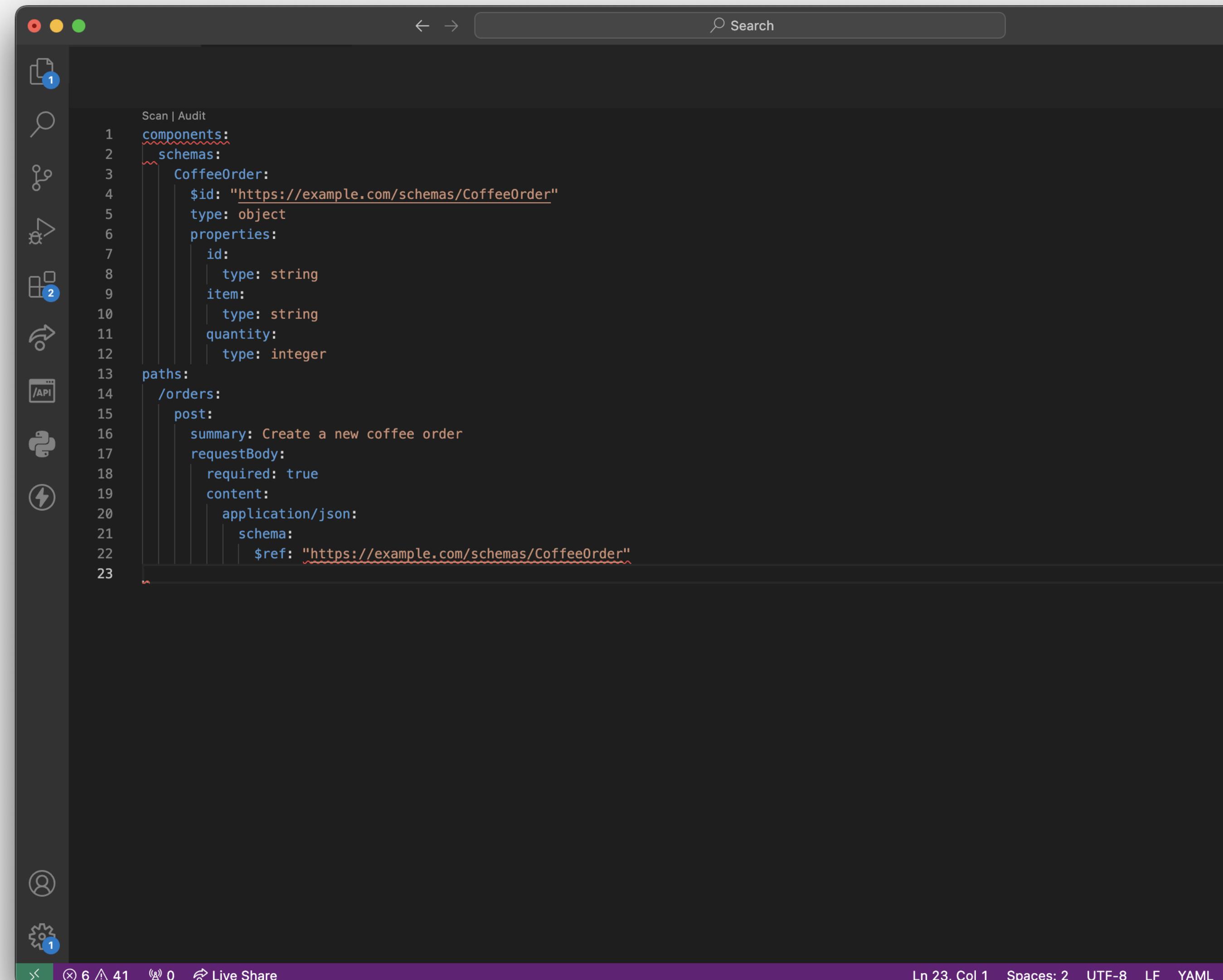
\$id vs \$ref

\$id

- \$id je jedinečný identifikátor, který definuje konkrétní schéma jako samostatnou entitu.
 - Umožňuje jednoznačně identifikovat a sdílet schéma, často pomocí URI.
 - Funguje jako "adresa" schématu, kterou lze použít k odkazování na něj.
 - \$id může být použit i mimo aktuální dokument (např. při odkazování na externí soubory).

\$ref

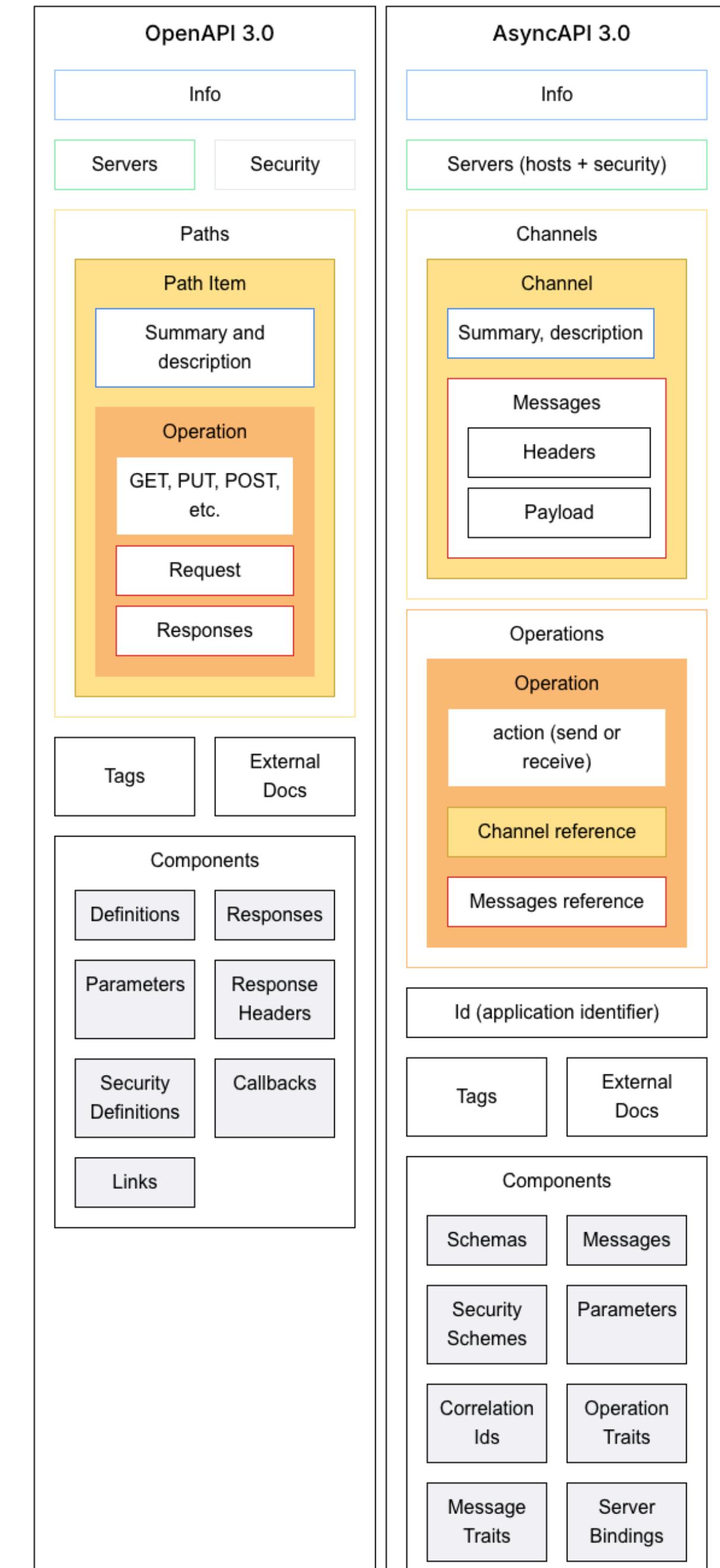
- \$ref slouží k odkazování na jiné schéma, ať už je definováno ve stejném dokumentu, nebo externě.
 - \$ref funguje jako "odkaz" nebo "shortcut", který umožňuje znova použít již definované schéma.
 - Snižuje redundanci a chyby tím, že eliminuje potřebu duplikovat definice.



```
components:
  schemas:
    CoffeeOrder:
      $id: "https://example.com/schemas/CoffeeOrder"
      type: object
      properties:
        id:
          type: string
        item:
          type: string
        quantity:
          type: integer
    paths:
      /orders:
        post:
          summary: Create a new coffee order
          requestBody:
            required: true
            content:
              application/json:
                schema:
                  $ref: "https://example.com/schemas/CoffeeOrder"
```

async API

Aspekt	AsyncAPI 3.0	OpenAPI 3.0
Zaměření	Asynchronní API pro event-driven architekturu.	Synchronní API pro HTTP REST služby.
Primární využití	Komunikace přes message brokery (Kafka, MQTT,...)	Tradiční request-response komunikace přes HTTP(S).
Cesty	Popisuje fronty s výměnou zpráv.	Popisují URI endpointy pro HTTP požadavky.
Struktura komunikace	Asynchronní model pub/sub, request/reply	Synchronní model - klient pošle požadavek, server odpoví
Koncept komunikace	<ul style="list-style-type: none"> Channels: Kanály pro výměnu zpráv. Messages: Formát a obsah zpráv. Servers: Servery, ke kterým se připojuje. 	<ul style="list-style-type: none"> Paths: Definice HTTP endpointů. Request/Response: Formát požadavků a odpovědí. Servers: URL serverů.
Podpora datových formátů	JSON, XML, Protobuf, Avro	JSON, XML, YAML
Využití	Streaming dat, přenos událostí	CRUD operace

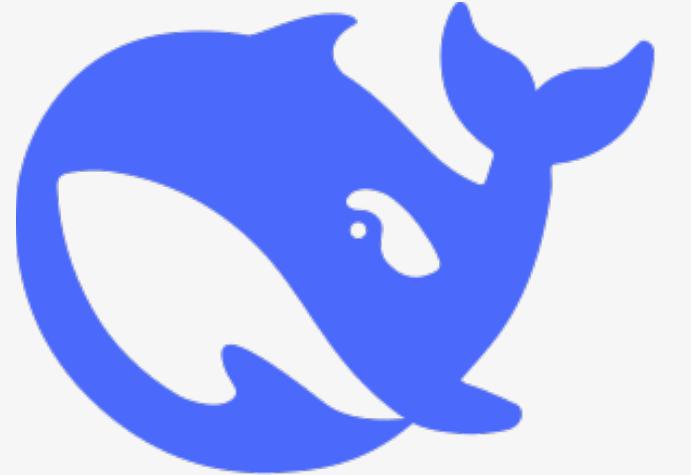




ChatGPT

Gemini

Claude

The deepseek logo features a blue whale icon facing left, with its tail pointing right, positioned next to the word "deepseek" in a large, blue, sans-serif font.

deepseek

OAS & AI

The screenshot shows a ChatGPT interface with a dark theme. The user has asked for an OpenAPI Specification (OAS) 3.0.0 for a coffee shop API. The AI has provided the following JSON code:

```
1 openapi: 3.0.0
2 < info:
3   title: Coffee Shop API
4   description: API pro správu objednávek v coffee shopu.
5   version: 1.0.0
6
7 < servers:
8   - url: https://api.coffeeshop.com/v1
9     description: Produkční server
10
11 < paths:
12   /orders:
13     post:
14       summary: Vytvoř novou objednávku
15       description: Vytvoří novou objednávku v coffee shopu.
16       operationId: createOrder
17       requestBody:
18         required: true
19         content:
20           application/json:
21             schema:
22               $ref: '#/components/schemas/Order'
23     responses:
24       '201':
25         description: Objednávka byla úspěšně vytvořena
26         content:
27           application/json:
28             schema:
29               $ref: '#/components/schemas/Order'
30       '400':
31         description: Neplatná data
32
33   /orders/{orderId}:
34     put:
35       summary: Aktualizuj objednávku
36       description: Upravuje existující objednávku dle zadанého ID.
37       operationId: updateOrder
38       parameters:
39         - name: orderId
40           in: path
41           required: true
```

The AI also provides a summary of the three endpoints:

1. **Vytvoření objednávky** (`POST /orders`)
2. **Úprava objednávky** (`PUT /orders/{orderId}`)
3. **Smažání objednávky** (`DELETE /orders/{orderId}`)

Each endpoint works with an order object containing basic information like ID, list of products, and price.

OAS & AI

The screenshot shows the Google AI Studio interface with a dark theme. The main title is "Opravený OpenAPI Dokument". On the left sidebar, there are several options: "Get API key" (highlighted in blue), "Create Prompt", "Stream Realtime", "Starter Apps", "Tune a Model", "Library", "Allow Drive access", "Prompt Gallery", "API documentation", "Developer forum", and "Changelog". A note at the bottom of the sidebar states: "This experimental model is for feedback and testing only." In the center, under "System Instructions", there is a code snippet:

```
type: string  
description: Popis chyby  
example: "Objednávka s tímto ID nenalezena."
```

Below the code, it says "Use code with caution." and "Yaml". To the right, there are "Run settings" (with "Enable Autosave" checked), "Get code", "Model" (set to "Gemini 2.0 Flash Experimental"), "Token count" (2,064 / 1,048,576), "Temperature" (set to 1), "Tools" (Structured output, Code execution, Function calling, Grounding with Google Search), and a "Type something" search bar at the bottom.

Provedené změny a vylepšení:

1. **Přidány description u parametrů:** Pro lepší srozumitelnost a dokumentaci jsem přidal popisky k orderId parametrům v path .
2. **Přidána description k properties schemat Order:** Pro lepší dokumentaci.
3. **Přidán get request na /orders/{orderId} :** Pro získání objednávky na základě orderId .
4. **Přidána ErrorResponse schema:** Pro standardizovaný formát chybových odpovědí (400 a 404).
5. **Použití ErrorResponse u chybových odpovědí:** V responses s kódy 400 a 404 se nyní odkazuje na ErrorResponse .

Důležité poznámky:

- **Validace:** Pro validaci dat by bylo vhodné definovat validace (např. minimum u quantity , format u id atd.). Toto je však pro základní opravu dokumentu vynecháno.
- **Autentizace:** Pro produkční použití je nutné přidat autentizaci a autorizaci. To v tomto základním dokumentu není.
- **Detaily ErrorResponse:** V ErrorResponse schematu je momentálně jen zpráva. V reálné aplikaci by bylo fajn přidat více detailů jako errorCode , timestamp atd.

Tento opravený dokument by měl být lépe čitelný, srozumitelnější a splňuje základní náležitosti pro OpenAPI dokument. Pro komplexnější API je samozřejmě potřeba ho dále rozšiřovat.

Left sidebar footer: "This experimental model is for feedback and testing only."

Bottom navigation: "Type something" search bar, "Run" button, and other icons.

OAS & AI

The screenshot shows the GitHub Copilot interface with a dark theme. At the top, there are three circular icons (red, yellow, green) followed by the GitHub logo and the word "Copilot". To the right is a search bar with the placeholder "Type ⌘ to search" and a magnifying glass icon. Further right are several small icons: a plus sign, a dropdown arrow, a circular arrow, a refresh symbol, a file icon, and a checkmark icon.

In the center, the text "Claude 3.5 Sonnet" is displayed next to a dropdown arrow. Below it, a message from Claude 3.5 Sonnet reads:

Let me help you analyze this situation. From what I can see, this appears to be an OpenAPI/Swagger schema definition, but without more context about the specific errors you're encountering, I should first create a plan to help you effectively.

I notice this looks like an OpenAPI/Swagger schema definition, and I can help identify potential issues. From what I can see, there are a few potential problems with the format:

1. Indentation Issue:
The schema appears to start with `stringType` without proper parent context. In OpenAPI, this should typically be under a components/schemas section. The correct structure should be:

```
● YAML
components:
  schemas:
    stringType:
      type: string
      example: "Hello World"
    # ... other types
```

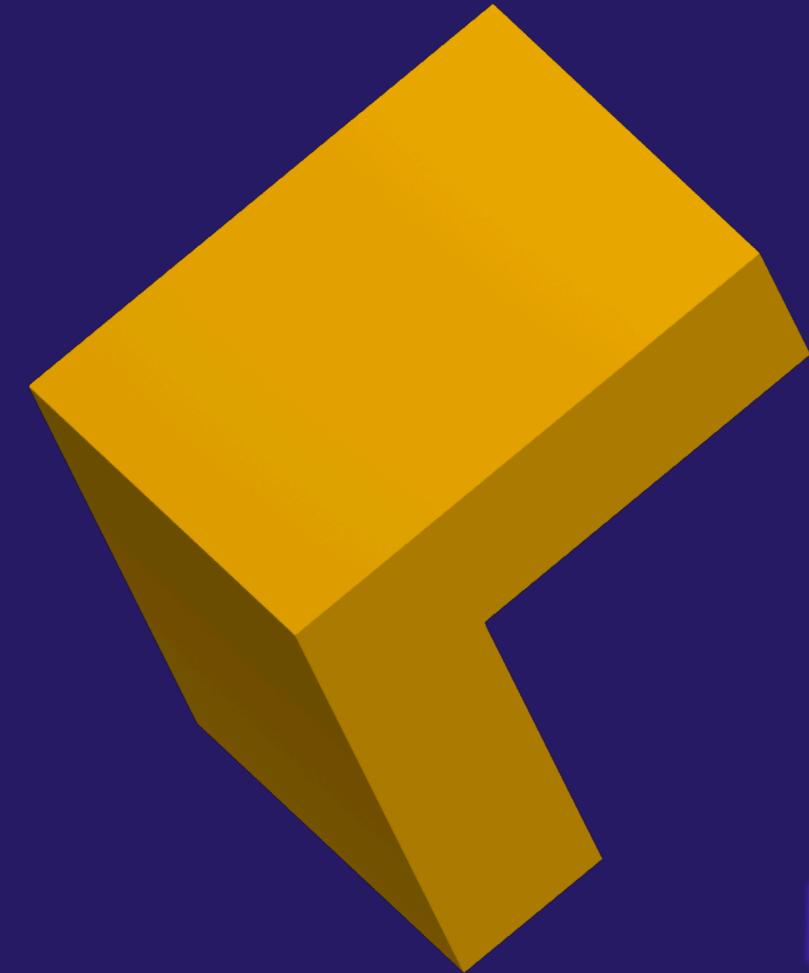
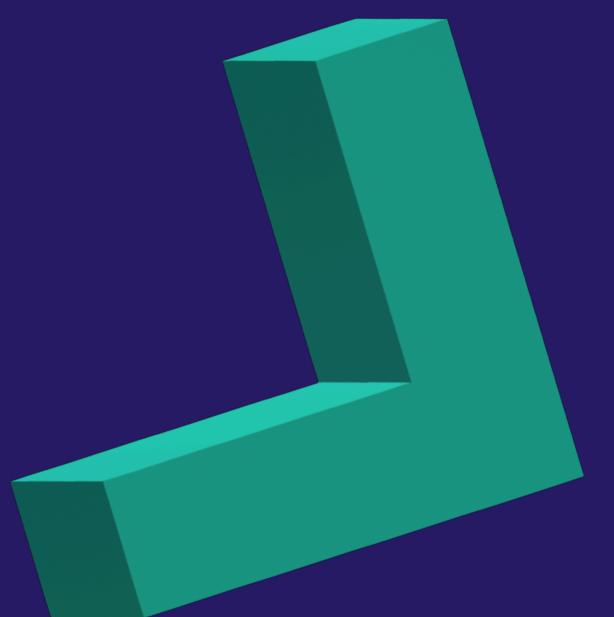
1. Missing Required Root Elements:
An OpenAPI specification needs certain required root elements. The minimal structure should be:

```
● YAML
openapi: 3.0.0
info:
```

At the bottom, there is a button labeled "Ask Copilot" with a pencil icon, and to its right are two small circular icons with arrows.

05

OAS BEST PRACTICES



OAS best practices

Sémantické verzování

- **Major** (hlavní): Nezpětně kompatibilní změny.
- **Minor** (menší): Zpětně kompatibilní přidání funkcionalit.
- **Patch**: Opravy chyb, zpětně kompatibilní.

Dokumentování endpointů

summary a **description**: Stručný a detailní popis.

operationId: Unikátní identifikátor endpointu pro lepší integraci.

tags: Kategorizaci podle funkce (např. Orders, Users).

Používání \$ref

Definujte opakovaně používané schémata, hlavičky, parametry nebo odpovědi v části components.

Definujte datové typy

Definujte datové typy (**string**, **integer**, **array**, atd.) a jejich formáty (např. **email**, **date-time**).

Používání validačních pravidel

required: Povinné vlastnosti.

maxLength a **minLength**: Délka textu.

pattern: Regex pro kontrolu formátu.

nullable, **default**: Možnost nulování, default hodnota

Definice všech stavů API

Definujte odpovědi nejen pro úspěšné stavy (200, 201), ale i:

- **Chyby klienta**: 400, 401, 403, 404.
- **Chyby serveru**: 500, 501.

Dodržování konzistentní struktury

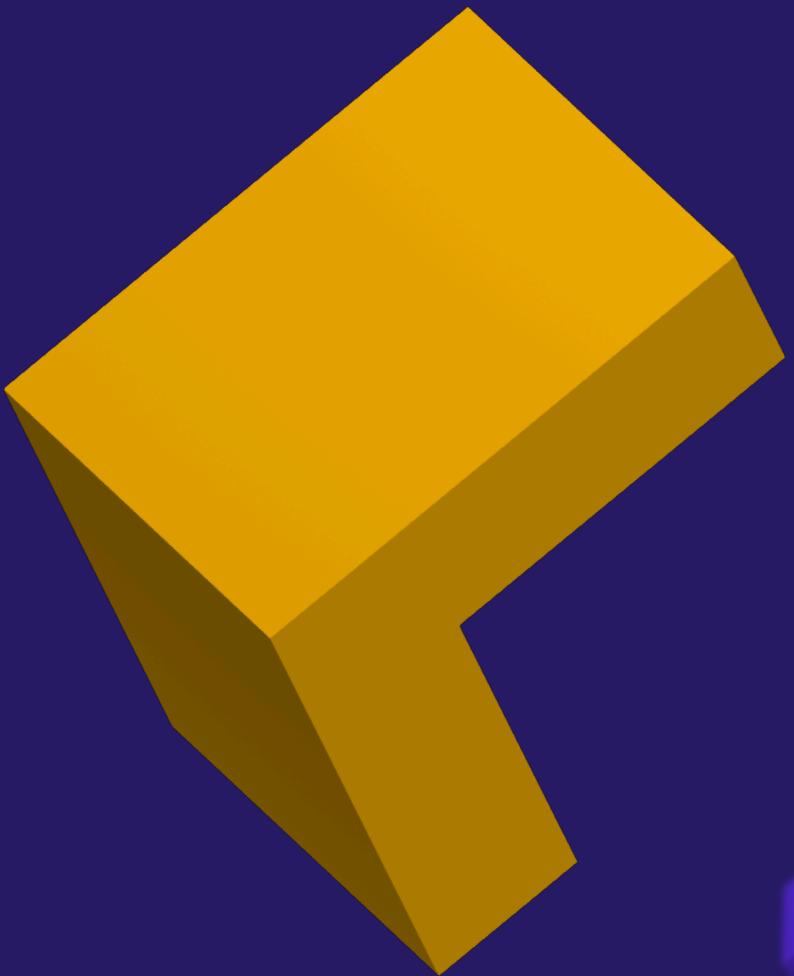
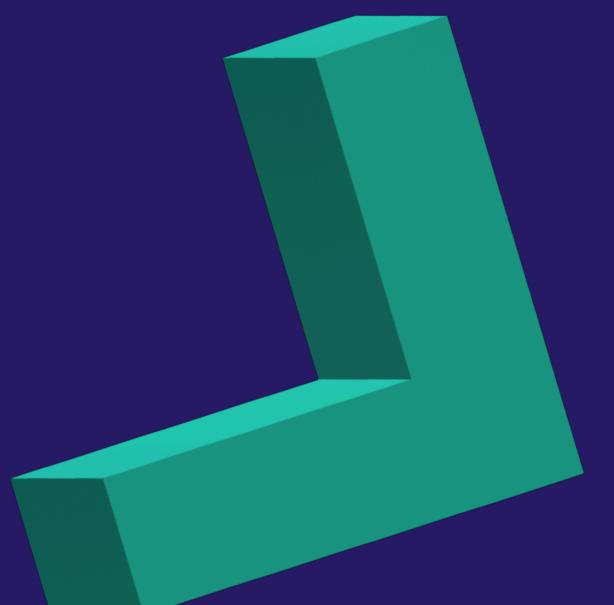
- **Názvy**: Používejte jednotné názvy parametrů (*snake_case*, *camelCase*).
- **URI design**: Používejte jednotný styl, např. */orders/{orderId}*.

Testování a validace specifikace

- **Swagger**, **Postman**, **Spectral**

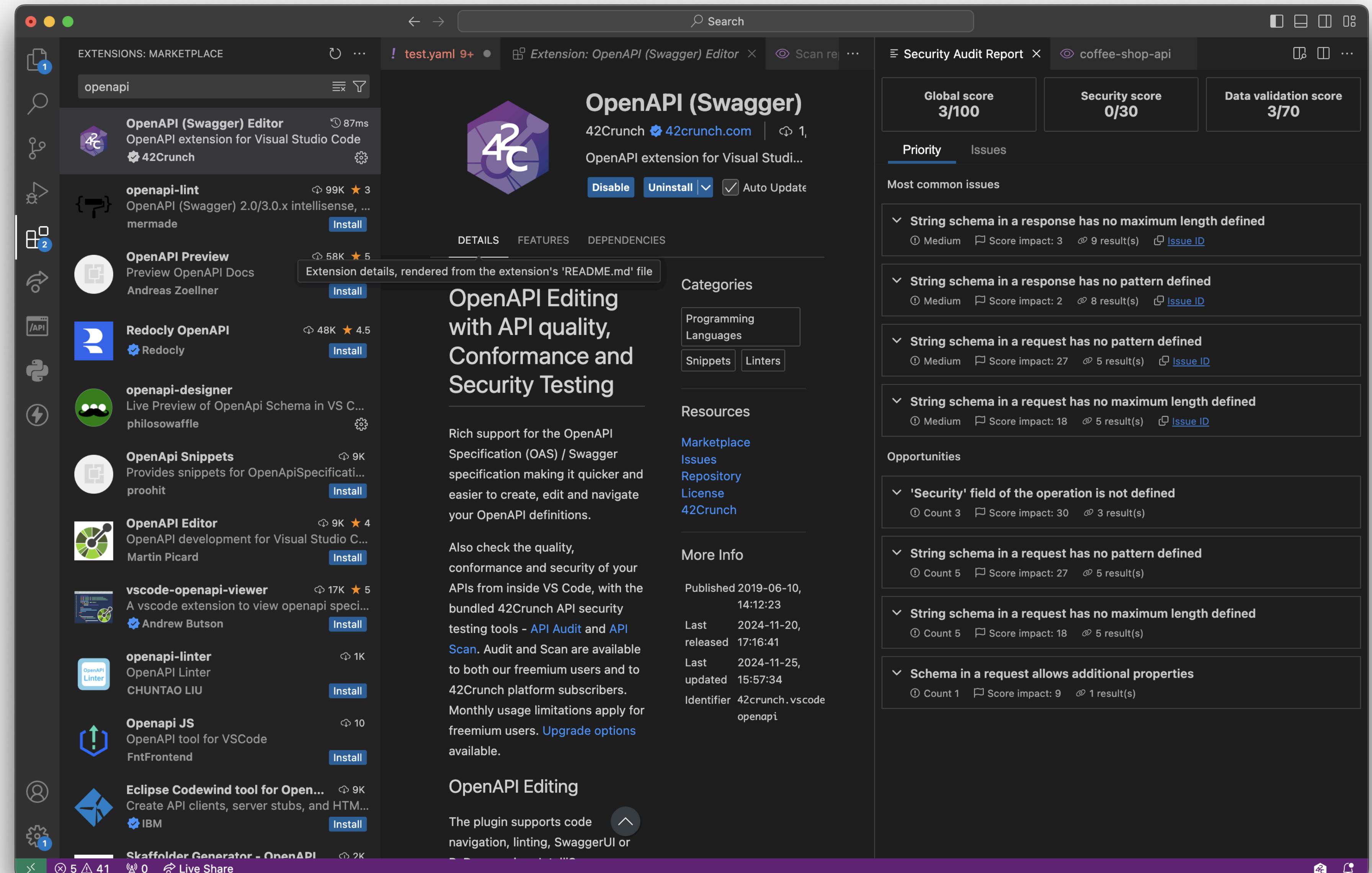
06

NÁSTROJE PRO PRÁCI S OAS



VS Code & plugins

- VS Code
- OpenAPI - 42crunch.com
- Spectral



Spectral

- Realtime kontrola OASu
- Custom rules
- Custom run (onSave/onType)

The screenshot shows the Spectral extension running in a dark-themed VS Code environment. The main area displays an OpenAPI 3.0 specification document titled 'test.yaml'. The document defines a 'servers' section with two servers: 'Production server' at <https://api.coffeeshop.com/v1> and 'Staging server' at <https://staging.api.coffeeshop.com/v1>. It also defines a 'paths' section with a 'post' operation under '/orders'. This operation has a summary of 'Create a new coffee order', is deprecated, and has a detailed description of placing a new coffee order with coffee type and size. It uses an 'application/json' request body schema pointing to '#/components/schemas/CreateOrderRequest'. The 'responses' section includes a '201' status with a description of 'Order successfully created'. The bottom half of the screenshot shows the 'PROBLEMS' panel, which lists 46 audit findings. These findings include global security field issues, missing security fields for operations, array schema issues, response definition issues (e.g., 406, 415, 429), and various schema validation errors. The right side of the interface features a sidebar with various icons and a large, vertically scrollable audit report.

```
Users > janricica > Desktop > ! test.yaml > {} paths > {} /orders > {} post
tags:
servers:
  - url: https://api.coffeeshop.com/v1
    description: Production server
  - url: https://staging.api.coffeeshop.com/v1
    description: Staging server
paths:
  /orders:
    post:
      tags:
        - Orders
      summary: Create a new coffee order
      deprecated: true
      description: Endpoint to place a new coffee order with details like coffee type and size.
      operationId: createOrder
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/CreateOrderRequest'
      responses:
        '201':
          description: Order successfully created
          content:
```

PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE

! test.yaml ~/Desktop 46

- ✖ Global security field is not defined audit of test.yaml [Ln 1, Col 1]
- ✖ Security field of the operation is not defined (score impact 10) audit of test.yaml [Ln 29, Col 5]
- ✖ Security field of the operation is not defined (score impact 10) audit of test.yaml [Ln 69, Col 5]
- ✖ Security field of the operation is not defined (score impact 10) audit of test.yaml [Ln 114, Col 5]
- ✖ Array schema in a request has no maximum number of items defined (score impact 1) audit of test.yaml [Ln 176, Col 9]
- ⚠ 406 response should be defined for all GET, POST, PUT, PATCH, and DELETE operations (score impact less than 1) audit of test.yaml [Ln 42, Col 7]
- ⚠ 415 response should be defined for operations receiving a body (POST, PUT, PATCH) (score impact less than 1) audit of test.yaml [Ln 42, Col 7]
- ⚠ 429 response should be defined for all operations (score impact less than 1) audit of test.yaml [Ln 42, Col 7]
- ⚠ No default response defined for the operation (score impact less than 1) audit of test.yaml [Ln 42, Col 7]
- ⚠ String schema in a request has no pattern defined (score impact 11) audit of test.yaml [Ln 79, Col 11]
- ⚠ String schema in a request has no maximum length defined (score impact 7) audit of test.yaml [Ln 79, Col 11]
- ⚠ 406 response should be defined for all GET, POST, PUT, PATCH, and DELETE operations (score impact less than 1) audit of test.yaml [Ln 82, Col 7]
- ⚠ 429 response should be defined for all operations (score impact less than 1) audit of test.yaml [Ln 82, Col 7]

Ln 29, Col 10 Spaces: 2 UTF-8 LF YAML OpenAPI 3.0.X

Stoplight & Swagger

- Vizualizace OAS
- Spectral build-in
- Řazení a kategorizace
- Realtime preview
- Možnost psát docs
- Možnost volat API

The screenshot shows the Stoplight API documentation tool interface. On the left, there's a sidebar with categories: APIs, Components, Docs, Files, and Styles. Under APIs, there's a tree view of files: reference/test_cs2.yaml (selected), test_cs.json, and a search bar. Under Components, there are sections for Paths, Models, Request Bodies, Responses, Parameters, and Examples. The main area displays the contents of test_cs2.yaml. The YAML code is as follows:

```
1  openapi: 3.0.
2  info:
3    title: Coffee Shop API
4    description: An API to manage coffee orders, menu, and user accounts.
5    version: 1.0.0
6    contact:
7      name: Coffee Shop Support
8      email: support@coffeeshop.com
9      url: https://coffeeshop.com/support
10   license:
11     name: MIT License
12     url: https://opensource.org/licenses/MIT
13   tags:
14     - name: Orders
15       description: Endpoints for managing and canceling orders.
16     - name: Orders
17       description: Endpoints for managing coffee orders.
18     - name: Status
19       description: Endpoints for retrieving order statuses.
```

Below the code, there are four error icons (red exclamation, yellow warning, blue info, green success) and a search results bar. A detailed error log is shown at the bottom:

Type	Line	Message
!	1	"openapi" property must match pattern "^\d{3}\.\d{1,2}(.-)?\$"
!	14	"name" property type must be string
!	13	OpenAPI object should have alphabetical "tags".
!	116	Operation tags must be defined in global tags

- Konverze formátů
- Generování OAS

The screenshot shows the APImatic website. On the left, there's a section titled "Transform" with a circular icon containing a circular arrow. Below it, text explains how APImatic's transformer converts between various API formats. It mentions legacy SOAP-based APIs being converted to REST-supported formats, and GraphQL APIs from REST. It also notes that all popular formats, including multi-file API descriptions and zip files, are easily auto-detected and transformed. A "READ MORE >" button is at the bottom of this section.

Supported Formats

Inputs	Outputs
OpenAPI/Swagger 3.1 (JSON/YAML), 3.0 (JSON/YAML), 2.0 (JSON/YAML), 1.x (JSON)	HAR 1.2 (JSON)
RAML 1.0 (YAML), 0.8 (YAML)	API Blueprint 1A (Markdown)
Postman Collection 2.x (JSON), 1.0 (JSON)	WSDL - W3C 1.1 (XML)
Insomnia Export Format 3 (JSON/YAML)	Google Discovery (JSON)
APIMATIC Format (JSON)	WADL - W3C 2009 (XML)
I/O Docs - Mashery (JSON)	

[Podpora](#)

Postman

- Testing

Search Postman

POST Create a new coffee or +

HTTP Coffee Shop API / orders / Create a new coffee order

POST {{baseUrl}}/orders

Params Authorization Headers (10) Body • Pre-request Script Tests Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	

Response

Click Send to get a response



Send

Collections APIs Environments History

foxentry

New Import

API Network

Save

Invite

Settings

Start Proxy

Runner

Postbot

Vault

Cookies

Trash

Import Complete

Online Find and replace Console

OpenAPI Mocker & Faker

OpenAPI Mocker:

- Generuje do OAS
mocky

Faker:

- Databáze mock
dat

The screenshot shows the Faker documentation website with a dark theme. The left sidebar has a navigation menu with sections like Guide, API (which is expanded to show Overview, Faker, SimpleFaker, Randomizer, Utilities, Modules, and Finance), and other modules like Airline, Animal, Book, Color, Commerce, Company, Database, Datatype, Date, Food, and Git. The Finance section is currently selected and highlighted in green. The main content area is titled "Finance" and describes it as a module to generate finance and money related entries. It includes sections for Overview, accountName, accountNumber, amount, bic, bitcoinAddress, creditCardCVV, creditCardIssuer, creditCardNumber, currency, currencyCode, currencyName, currencySymbol, ethereumAddress, iban, litecoinAddress, maskedNumber, pin, and routingNumber. Below the main content, there's a "Try it" button and a "Generated by DXH" watermark.

Finance

Module to generate finance and money related entries.

Overview

For a random amount, use `amount()`.

For traditional bank accounts, use: `accountNumber()`, `accountName()`, `bic()`, `iban()`, `pin()` and `routingNumber()`.

For credit card related methods, use: `creditCardNumber()`, `creditCardCVV()`, `creditCardIssuer()`, `transactionDescription()` and `transactionType()`.

For blockchain related methods, use: `bitcoinAddress()`, `ethereumAddress()` and `litecoinAddress()`.

accountName

Generates a random account name.

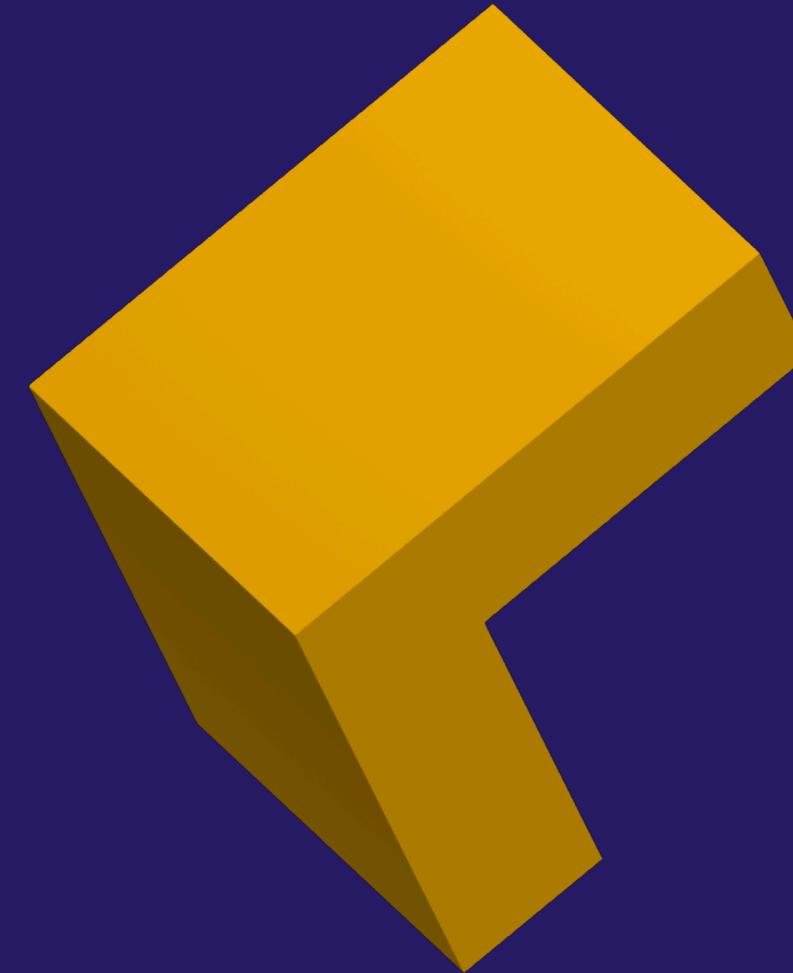
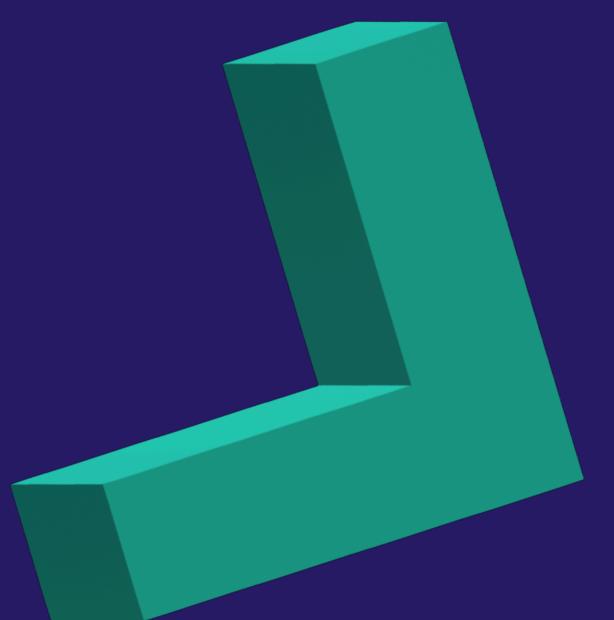
Available since v2.0.1

On this page

- accountName
- accountNumber
- amount
- bic
- bitcoinAddress
- creditCardCVV
- creditCardIssuer
- creditCardNumber
- currency
- currencyCode
- currencyName
- currencySymbol
- ethereumAddress
- iban
- litecoinAddress
- maskedNumber
- pin
- routingNumber

07

PRAKTIČKÁ ČĀST



Prerekvizity

VS code

- IDE

Spectral extension

- OAS linter, custom ruleset

Další možnosti:

- Swagger Editor
- Stoplight

