Descrivere API con OpenAPI

Angelo Di Iorio

Università di Bologna

Descrivere una RESTful API

- Una API è RESTful se utilizza i principi REST nel fornire accesso ai servizi che offre
- Per documentare un API è necessario definire:
  - **end-point** (*URI / route*) che supporta
    - separando collezioni e elementi singoli
  - **metodi** HTTP di accesso
    - Cosa succede con un GET, un PUT, un POST, un DELETE, ecc.
  - **rappresentazioni** in **Input** e **Output**
    - Di solito non si usa un linguaggio di schema, ma un esempio fittizio e sufficientemente complesso
  - **condizioni di errore** e i **messaggi** che restituisce in questi casi

Swagger e Open API

- Swagger è un ecosistema di tool per la creazione, costruzione, documentazione e accesso ad API soprattutto in ambito REST.
- In particolare ha creato un linguaggio per la documentazione di API REST e strumenti per l'editazione e la documentazione e il test di queste API.
- Nel 2016, il linguaggio è stato reso di pubblico dominio ed è diventato Open API
- Open API può essere serializzato sia in JSON che in YAML
- Standard industriale per API REST
- Generazione automatica di documentazione, modelli e codice

Apriamo una parentesi: YAML

- YAML (Ain't a Markup Language) è una linearizzazione di strutture dati con sintassi ispirata a Python:
  - simile a JSON (in realtà un superset)
  - indentazione come modello di annidamento
  - supporto di tipi scalari (stringhe, interi, float), liste (array) e array associativi (coppie <chiave>:<valore>)

```
nome: Angelo

cognome: Di Iorio

ufficio:
  città: Bologna
  civico: 14
  via: Ranzani

corsi:
  - Programmazione
  - "Tecnologie Web"
```

```
name: Sagre
news:
  - id: 1
    titolo: Sagra del ...
    articolo: Lo stand ...
    immagine: sagra.jpg
  - id: 2
    titolo: Tortellini per tutti
    articolo: Bologna la patria...
    immagine: tortelli.jpeg
```

OpenAPI (2.0) in YAML*

```yaml
host: "petstore.swagger.io"
basePath: "/v2"
tags:
- name: "pet"
  description: "Everything about your Pets"
  externalDocs:
    description: "Find out more"
    url: "http://swagger.io"
schemes:
- "https"
- "http"
paths:
  /pet:
    post:
      tags:
      - "pet"
      summary: "Add a new pet to the store"
      description: ""
      operationId: "addPet"
      consumes:
      - "application/json"
      - "application/xml"
      produces:
      - "application/xml"
      - "application/json"
      parameters:
      - in: "body"
        name: "body"
```

* Gli esempi mostrati in queste slide usano la versione 2.0 di OpenAPI

Sezione `paths`

- La parte centrale di un'API descrive i percorsi (URL) corrispondenti alle operazioni possibili sull'API
- Seguono la struttura: **`<host>/<basePath>/<path>`**
- Per ogni percorso (path o endpoint) si definiscono tutte le possibili operazioni che, secondo i princìpi REST, sono identificate dal metodo HTTP corrispondente
- Per ogni **`path`** quindi ci sono tante sottosezioni quante sono le operazioni e per ognuna:
  - Informazioni generali
  - Parametri di input e di output

Struttura di un `path`

Risorsa

Operazioni (metodi HTTP)

```
/pet/{petId}:
  get:
    summary: "Find pet by ID"
    description: "Returns a single pet"
    operationId: "getPetById"
    produces:
    - "application/xml"
    - "application/json"

    parameters:
    ...
    ...

  post:
    summary: "Updates a pet in the store with form data"
    description: ""
    operationId: "updatePetWithForm"

    parameters:
    ...
    ...
```

Formati in Output

Parametri in Input

Parametri in input

- I parametri in input sono descritti nella sezione **`parameters`** e per ogni parametro è possibile definire:
  - tipo del parametro: keyword **`in`** che può assumere valori **`path`**, **`query`** o **`body`**
  - nome (**`name`**) e descrizione (**`description`**)
  - se è opzionale o obbligatorio (**`required`**)
  - formato del/i valore/i che il dato può assumere (**`schema`**)
    - Il tipo può essere scalare (interi, stringhe, date, ecc.), o un oggetto o un vettore di valori scalari o oggetti

Esempi di parametri `path` e `query`

```yaml
/pet/{petId}:
  get:
    summary: Find pet by ID
    description: Returns a single pet
    operationId: getPetById
    parameters:
      - name: petId
        in: path
        description: ID of pet to return
        required: true
        type: integer
        format: int64
```

Parametro `<petId>` nell'URI

Parametro `<status>` nella parte query dell'URI `/pet/?status=ready`

```yaml
/pet/:
  get:
    summary: Finds Pets by status
    operationId: findPetsByStatus
    parameters:
      - name: status
        in: query
        description: Status values that need to be considered for filter
        required: true
        type: array
        items:
          type: string
```

Esempi di parametri nel `body`

Oggetto `<User>` nel body

```yaml
/user/{username}:
  put:
    tags:
      - user
    summary: Updated user
    description: This can only be done by the logged in user.
    operationId: updateUser
    parameters:
      - name: username
        in: path
        description: name that need to be updated
        required: true
        type: string
      - in: body
        name: body
        description: Updated user object
        required: true
        schema:
          $ref: '#/definitions/User'
```

Oggetti e definizioni

- Nell'esempio precedente il body contiene un oggetto di tipo **`User`**; viene infatti passata un'intera risorsa (o meglio la sua rappresentazione) come parametro
- La sezione **`definitions`** permette di definire i tipi degli oggetti, le loro proprietà e possibili valori
- Questi tipi possono essere referenziati (tramite **`schema`** -> **`$ref`**) sia nelle richieste che nelle risposte

Esempi di schemi

```yaml
User:
  type: object
  properties:
    id:
      type: integer
      format: int64
    username:
      type: string
    firstName:
      type: string
    lastName:
      type: string
    email:
      type: string
    password:
      type: string
    phone:
      type: string
    userStatus:
      type: integer
      format: int32
      description: User Status
```

```yaml
Order:
  type: object
  properties:
    id:
      type: integer
      format: int64
    petId:
      type: integer
      format: int64
    quantity:
      type: integer
      format: int32
    shipDate:
      type: string
      format: date-time
    status:
      type: string
      description: Order Status
      enum:
        - placed
        - approved
        - delivered
    complete:
      type: boolean
```

Output

- L'output (dati e codici e messaggi di errore) sono definiti attraverso la keyword **`responses`**
- Si specifica il tipo di output atteso nel body della risposta
- Inoltre ogni risposta ha un id numerico univoco, associato al codice HTTP corrispondente
  - **`200`** viene usato per indicare che non c'è stato alcun errore
  - da **`400`** in su vengono in genere usati per indicare messaggi di errore

Esempio di risposta

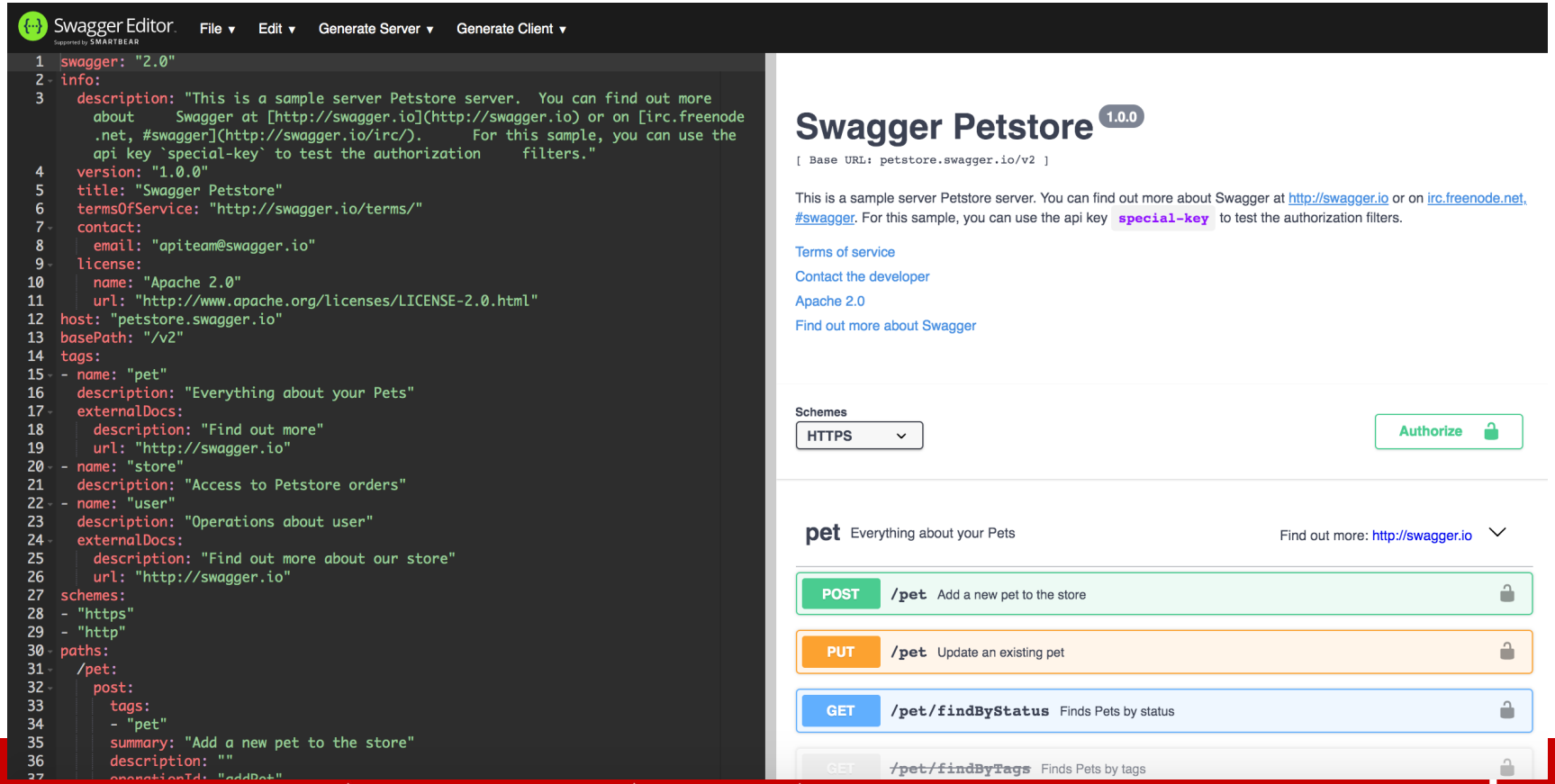
```yaml
/pet/:
  get:
    summary: Finds Pets by status
    operationId: findPetsByStatus
    parameters:
      - name: status
        in: query
        description: Status values that need to
        required: false
        type: array
        items:
          type: string
    responses:
      '200':
        description: successful operation
        schema:
          type: array
          items:
            $ref: '#/definitions/Pet'
      '400':
        description: Invalid status value
```

Codici HTTP

Vettore di oggetti `<Pet>`

Swagger Editor

https://editor.swagger.io/



Esercizio

- Progettare un API REST (parziale) per la gestione di un ristorante e descriverla in OpenAPI (JSON o YAML). Il ristorante offre menù diversi, ognuno caratterizzato da un ID e una descrizione testuale; ogni menù include diversi piatti, ognuno caratterizzato da un ID, una descrizione testuale e un prezzo. Tutti gli attributi sono obbligatori.
- L'API permette di:
  - ottenere l'elenco di tutti i menù
  - ottenere le informazioni di uno specifico menù (ID e descrizione, senza elenco piatti)
  - aggiungere un nuovo piatto ad un menù
- Specificare: URL di accesso, metodi HTTP, parametri e risposte con esempi.