OpenAPI

Angelo Di Iorio

Università di Bologna

ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI

REpresentational State Transfer

L'architettura REST si basa su quattro punti :

1. Definire **risorsa** ogni concetto rilevante dell'applicazione Web
2. Associargli un **URI** come l'**identificatore** e selettore primario
3. Usare i verbi HTTP per esprimere ogni **operazione** dell'applicazione secondo il modello CRUD:
   - creazione di un nuovo oggetto (metodo PUT)
   - visualizzazione dello stato della risorsa (metodo GET)
   - cambio di stato della risorsa (metodo POST)
   - cancellazione di una risorsa (metodo DELETE)
4. Esprimere in maniera parametrica ogni **rappresentazione dello stato interno della risorsa**, personalizzabile dal richiedente attraverso un **Content Type** preciso

IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI

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

IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI

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
- **In queste slide usiamo la versione "2.0" dell'API.** Ci sono alcune differenze con le versioni 3.x ma le funzionalità di base sono rimaste invariate

IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI

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
```


IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI

YAML e JSON

```yaml
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
```

```json
{
"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"
   }
]
}
```


IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI

Struttura base in OpenAPI (versione 2.0 in YAML)

Info generali

Base (utile per le versioni)

Schemi supportati

End-point (operazioni)

Definizioni risorse

```yaml
swagger: "2.0"
info:
  description: "This is a ..."
  version: "1.0.6"
  title: "Swagger Petstore"
  termsOfService: "http://swagger.io/terms/"
  license:
    name: "Apache 2.0"
    url: "http://www.apache.org/licenses/LICEN
host: "petstore.swagger.io"
basePath: "/v2"
tags:
    ....
schemes:
- "https"
- "http"
paths:
  ...

securityDefinitions:
  ...

definitions:
  ...
```


IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI

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

IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI

Struttura di un `path`

```yaml
/pet/{petId}:
  get:
    summary: "Find pet by ID"
    description: "Returns a single pet"
    operationId: "getPetById"
    produces:
    - "application/xml"
    - "application/json"
    parameters:
      ...
    responses:
      ...
  post:
    summary: "Updates a pet in the store with fo
    description: ""
    operationId: "updatePetWithForm"
    consumes:
    - "application/x-www-form-urlencoded"
    produces:
    - "application/xml"
    - "application/json"
    parameters:
      ...
    responses:
      ...
```

Risorsa

Operazioni (metodi HTTP)

Informazioni descrittive

Formati in Input e Output

Parametri in Input

Risposte

IL PRESENTE MATERIALE È R
TÀ DI BOLOGNA
PER FINI NON ISTITUZIONALI

Parametri in input

- I parametri in input sono descritti nella sezione **`parameters`** che definisce una lista di parametri (attenzione al simbolo "-") e per ognuno:
  - tipo del parametro: keyword **`in`** che può assumere valori **`path`**, **`query`** o **`body`**
  - nome (keyword **`name`**) e descrizione (**`description`**)
  - se è opzionale o obbligatorio (**`required`**)
  - formato del/i valore/i che il dato può assumere: keyword **`type`** o **`schema`**, più altre proprietà dipendenti dal tipo di dato

IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI

Tipi di dato

- I dati in input e output possono essere di vario tipo:
  - Primitivo: interi, stringhe, date, booleani
    - **type** indica il dato mentre **format** specifica i dettagli del formato
    - utile la keyword **enum** seguita da una lista per indicare un insieme di possibili valori nei parametri di tipo stringa
  - Oggetto (nel body):
    - si usa la keyword **schema** seguita dalla definizione delle proprietà o molto più frequente dal riferimento alla definizione tramite **$ref**; dettagli nelle prossime slide
  - Array di oggetti o dati primitivi
    - Quando si dichiara il tipo di dato si usa **type:array** seguito da **items** per indicare il tipo degli elementi nel vettore
    - Solitamente nel body ma anche nella parte query

IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI

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`

Array di stringhe

```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
```

IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI

Esempi di parametri nel `body`

Oggetto `<User>` nel body

Parametro nel *path*

```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'
```


IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI

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

IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI

Esempi di modelli e tipi di dato

```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
```

ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA
IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI

Output

- I possibili output (dati, codici HTTP e messaggi di errore) sono definiti attraverso la keyword `responses`
- Si specifica il tipo di output atteso nel body delle risposte, se presenti
- 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

IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI

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
```

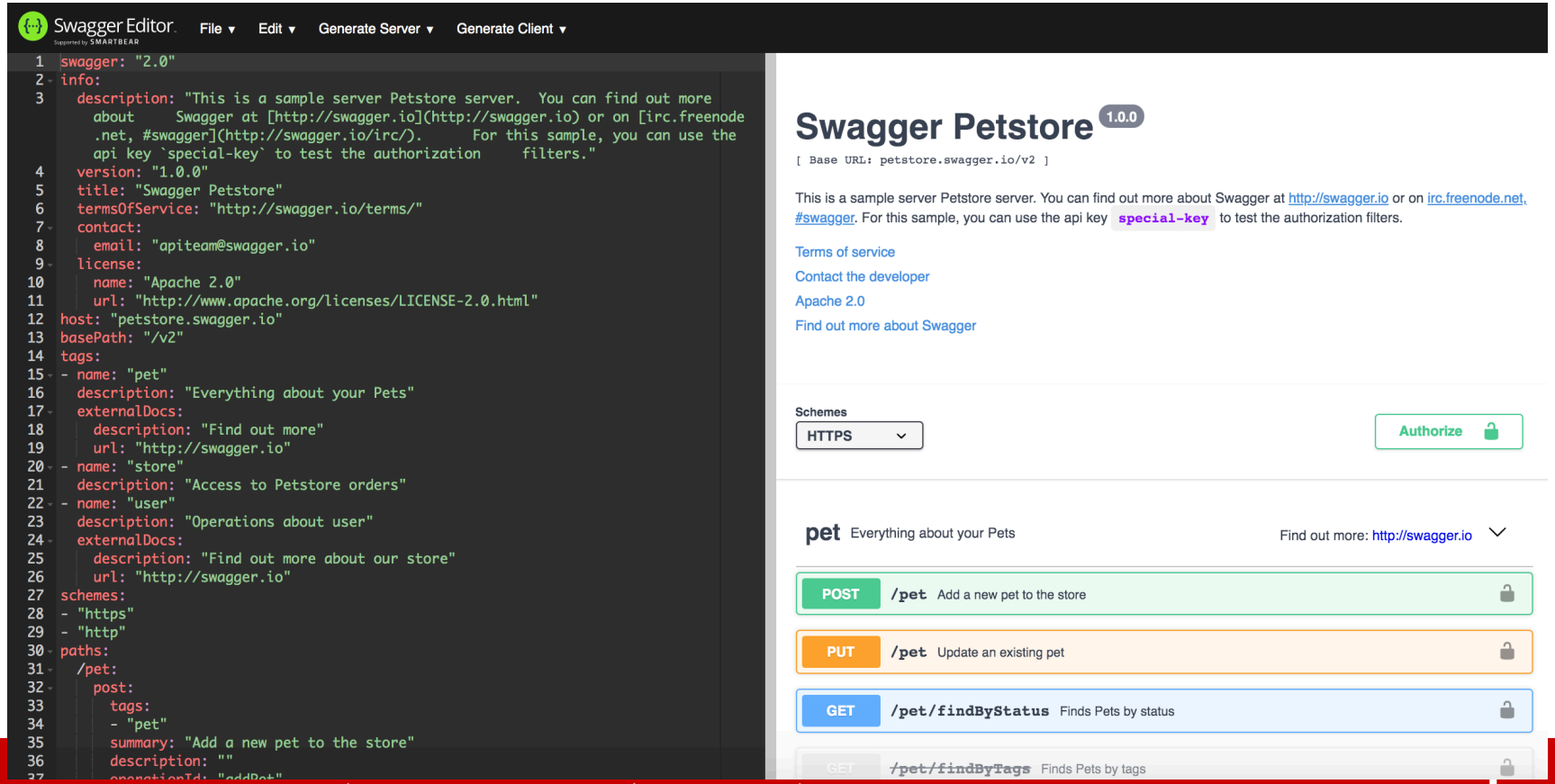
Codici HTTP

Tipo della risposta. Vettore di oggetti `<Pet>`

IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI

Swagger Editor

https://editor.swagger.io/



IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI

Esercizio 1

- 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:
  1. ottenere l'elenco di tutti i menù
  2. ottenere le informazioni di uno specifico menù: ID e descrizione, senza elenco piatti
  3. aggiungere un nuovo piatto ad un menù
- Specificare: URL di accesso, metodi HTTP, parametri e risposte con esempi.
  Non è richiesto includere sezioni *servers*, *tags* e gestire autenticazione

IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI

Esercizio 2

- Progettare un API REST (parziale) per gestire una piattaforma di giochi e descriverla inSwagger/ OpenAPI. Ogni gioco è caratterizzato da un ID (di tipo *intero* per semplicità), un nome (string) e una categoria, che può assumere valori *Shooter*, *Adventure*, *Puzzle*, *Sport*, e un numero di giocatori minimo e massimo, entrambi valori interi. Scrivere un file in formato JSON o YAML.
- L'API permette di:
  1. ottenere l'elenco di tutti i giochi di una data categoria
  2. modificare il numero minimo e massimo di giocatori in un gioco
  3. aggiungere un insieme di giochi e le relative informazioni; è possibile quindi aggiungere anche più di un gioco con un'unica richiesta
- Specificare: URL di accesso, metodi HTTP, parametri e risposte con esempi.
  Non è richiesto includere sezioni *servers*, *tags* e gestire autenticazione

IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI