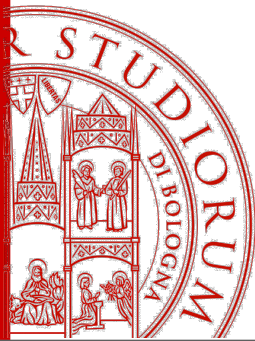




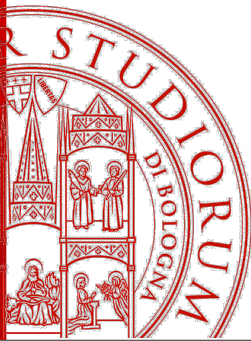
Descrivere API con OpenAPI

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

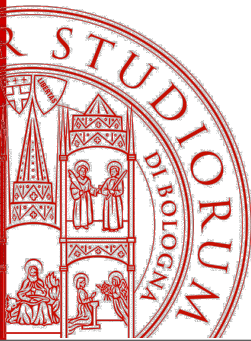
corsi:
  - Programmazione
  - "Tecnologie Web"
```

```
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*

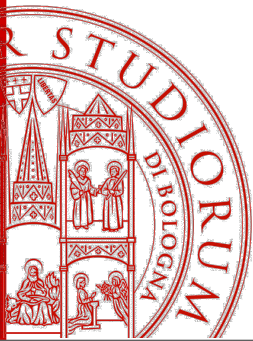
```
12 host: "petstore.swagger.io"
13 basePath: "/v2"
14 tags:
15 - name: "pet"
16   description: "Everything about your Pets"
17   externalDocs:
18     description: "Find out more"
19     url: "http://swagger.io"
20 schemes:
21 - "https"
22 - "http"
23 paths:
24   /pet:
25     post:
26       tags:
27       - "pet"
28       summary: "Add a new pet to the store"
29       description: ""
30       operationId: "addPet"
31       consumes:
32       - "application/json"
33       - "application/xml"
34       produces:
35       - "application/xml"
36       - "application/json"
37       parameters:
38       - in: "body"
39       name: "body"
40       description: "Pet object to add to the store"
```

* 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 principi 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

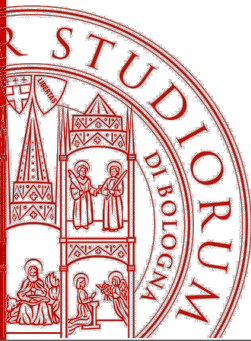
Risorsa

```
/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

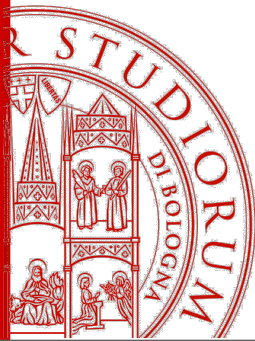
Parametri in Input

Operazioni
(metodi HTTP)



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

```
/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

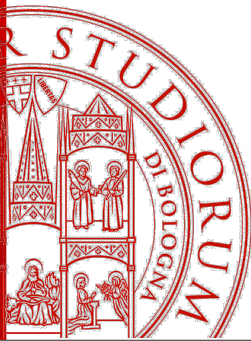
```
/pet/:  
get:  
  summary: Finds Pets by status  
  operationId: findPetsByStatus  
  parameters:  
    - name: status  
      in: query  
      description: Status values that need to be considered for filter  
      required: true  
      type: array  
      items:  
        type: string
```



Esempi di parametri nel body

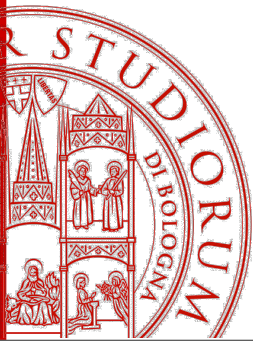
Oggetto <User> nel body

```
/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

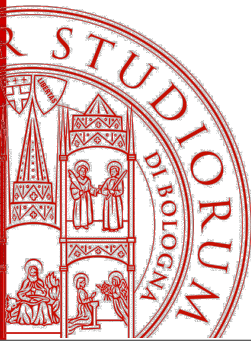
- Nell'esempio precedente il body contiene un oggetto di tipo **User**; viene infatti passata un'intera risorsa (o meglio la sua rappresentazione) come parametro
- La sezione **definitions** permette di definire i tipi degli oggetti, le loro proprietà e possibili valori
- Questi tipi possono essere referenziati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



Esempi di schemi

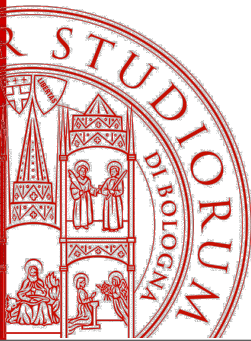
```
User:
  type: object
  properties:
    id:
      type: integer
      format: int64
    username:
      type: string
    firstName:
      type: string
    lastName:
      type: string
    email:
      type: string
    password:
      type: string
    phone:
      type: string
    userStatus:
    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

```
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

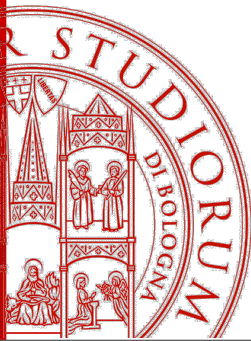


Esempio di risposta

```
/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/>

The screenshot displays the Swagger Editor interface. On the left, a code editor shows the Swagger JSON definition for the Petstore API. On the right, the rendered UI for the Swagger Petstore API is shown, including the title 'Swagger Petstore 1.0.0', a description, and a list of endpoints.

```
1 swagger: "2.0"
2 info:
3   description: "This is a sample server Petstore server. You can find out more
4     about Swagger at [http://swagger.io](http://swagger.io) or on [irc.freenode
5     .net, #swagger](http://swagger.io/irc/). For this sample, you can use the
6     api key `special-key` to test the authorization filters."
7   version: "1.0.0"
8   title: "Swagger Petstore"
9   termsOfService: "http://swagger.io/terms/"
10  contact:
11    email: "apiteam@swagger.io"
12  license:
13    name: "Apache 2.0"
14    url: "http://www.apache.org/licenses/LICENSE-2.0.html"
15  host: "petstore.swagger.io"
16  basePath: "/v2"
17  tags:
18    - name: "pet"
19      description: "Everything about your Pets"
20      externalDocs:
21        description: "Find out more"
22        url: "http://swagger.io"
23    - name: "store"
24      description: "Access to Petstore orders"
25    - name: "user"
26      description: "Operations about user"
27      externalDocs:
28        description: "Find out more about our store"
29        url: "http://swagger.io"
30  schemes:
31    - "https"
32    - "http"
33  paths:
34    /pet:
35      post:
36        tags:
37          - "pet"
38        summary: "Add a new pet to the store"
39        description: ""
40        operationId: "addPet"
```

Swagger Petstore ^{1.0.0}
[Base URL: petstore.swagger.io/v2]

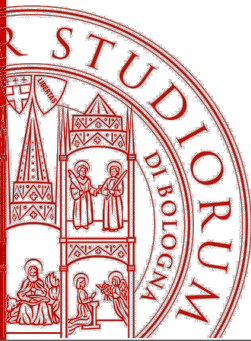
This is a sample server Petstore server. You can find out more about Swagger at <http://swagger.io> or on irc.freenode.net, #swagger. For this sample, you can use the api key `special-key` to test the authorization filters.

[Terms of service](#)
[Contact the developer](#)
[Apache 2.0](#)
[Find out more about Swagger](#)

Schemes: **HTTPS** [Authorize](#)

pet Everything about your Pets Find out more: <http://swagger.io>

- POST** /pet Add a new pet to the store
- PUT** /pet Update an existing pet
- GET** /pet/findByStatus Finds Pets by status
- GET** /pet/findByTags Finds Pets by tags



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.