

Progettazione e Documentazione API REST

Servizio Backend

- Il servizio messo a disposizione dal backend è disponibile tramite interfaccia di tipo API REST
- La documentazione del progetto deve prevedere anche la parte relativa alla specifica delle API REST: per ogni endpoint il tipo di operazione che si può compiere, formato di eventuali parametri e dati da inviare nel body e formato eventuale body della risposta.
- Uno strumento che potete utilmente impiegare nella fase di progettazione delle API ma anche per documentarle è lo Swagger Editor (che implementa Open API)

Tutorial Open API

<https://support.smartbear.com/swaggerhub/docs/tutorials/openapi-3-tutorial.html>

Il tutorial qui indicato mostra un esempio di definizione di API REST per un semplice servizio, introducendo gli aspetti essenziali della specifica OpenAPI 3.0, un formato che è diventato uno standard de-facto per documentare API REST.

È possibile utilizzare un editor on-line per definire un insieme di API REST, e salvarlo in un file di testo in formato [YAML](#) (che si può ricaricare nell'editor successivamente per prenderne visione o perfezionarlo)

<https://swagger.io/tools/swagger-editor/>

È anche possibile utilizzare JSON per la rappresentazione (anche se è consigliato YAML)

YAML YAML Ain't Markup Language <https://yaml.org/spec/1.2.2>

- It is a data serialization language designed to be human-friendly and work well with modern programming languages for common everyday tasks.
- YAML was specifically created to work well for common use cases such as: configuration files, log files, interprocess messaging, cross-language data sharing, object persistence and debugging of complex data structures.
- It uses Unicode [printable](#) characters, [some](#) of which provide structural information and the rest containing the data itself. YAML achieves a unique cleanness by minimizing the amount of structural characters and allowing the data to show itself in a natural and meaningful way.

L'editor on-line

The screenshot displays the Swagger Editor interface in a web browser. The left pane shows the OpenAPI 3.0 specification in JSON format, and the right pane shows the corresponding visual representation of the API.

OpenAPI 3.0 Specification (JSON):

```
1 openapi: 3.0.0
2 info:
3   version: 1.0.0
4   title: Simple API
5   description: A simple API to illustrate OpenAPI concepts
6
7 servers:
8   - url: https://example.io/v1
9
10 components:
11   securitySchemes:
12     BasicAuth:
13       type: http
14       scheme: basic
15   security:
16     - BasicAuth: []
17
18 paths:
19   /artists:
20     get:
21       description: Returns a list of artists
22       parameters:
23         - name: limit
24           in: query
25           description: Limits the number of items on a page
26           schema:
27             type: integer
28         - name: offset
29           in: query
30           description: Specifies the page number of the artists to be displayed
31           schema:
32             type: integer
33       responses:
34         '200':
35           description: Successfully returned a list of artists
36           content:
```

Visual Representation:

- Title:** Simple API 1.0.0 OAS3
- Description:** A simple API to illustrate OpenAPI concepts
- Servers:** A dropdown menu showing the selected server: `https://example.io/v1`.
- default** section:
- GET /artists** (highlighted in blue)
- POST /artists** (highlighted in green)

Le informazioni contenute in una specifica

- Meta information
- Path items (endpoints):
 - Parameters
 - Request bodies
 - Responses
- Reusable components:
 - Schemas (data models)
 - Parameters
 - Responses
 - Other components

Meta informazione

Versione di OpenAPI,
Versione API, Titolo,
descrizione, base URL e
altre informazioni
generali

```
openapi: 3.0.0
info:
  version: 1.0.0
  title: Simple Artist API
  description: A simple API to illustrate OpenAPI concepts

servers:
  - url: https://example.io/v1

# Basic authentication
components:
  securitySchemes:
    BasicAuth:
      type: http
      scheme: basic
security:
  - BasicAuth: []
```

Sezione dei path

In questa sezione si descrivono gli endpoint e le azioni (verbi HTTP) che si possono compiere sulle risorse corrispondenti.

Viene anche specificata la forma delle possibili risposte

```
paths:
  /artists:
    get:
      description: Returns a list of artists
      # ----- Added lines -----
      responses:
        '200':
          description: Successfully returned a list of artists
          content:
            application/json:
              schema:
                type: array
                items:
                  type: object
                  required:
                    - username
                  properties:
                    artist_name:
                      type: string
                    artist_genre:
                      type: string
                    albums_recorded:
                      type: integer
                    username:
                      type: string
```


Sezione dei path

Viene anche specificata la forma delle possibili risposte, incluse le segnalazioni di errore

```
'400':  
  description: Invalid request  
  content:  
    application/json:  
      schema:  
        type: object  
        properties:  
          message:  
            type: string
```

GET https://example.io/v1/artists?limit=20&offset=3

Si possono specificare parametri per gli endpoint

```
paths:
  /artists:
    get:
      description: Returns a list of artists
      # ---- Added lines -----
      parameters:
        - name: limit
          in: query
          description: Limits the number of items on a page
          schema:
            type: integer
        - name: offset
          in: query
          description: Specifies the page number of the artists to be displayed
          schema:
            type: integer
      # ---- /Added lines -----
      responses:
```

Parameters

Name	Description
------	-------------

limit	Limits the number of items on a page
-------	--------------------------------------

integer	
---------	--

(query)	
---------	--

offset	Specifies the page number of the artists to be displayed
--------	--

integer	
---------	--

(query)	
---------	--

GET https://example.io/v1/artists?limit=20&offset=3

POST

```
# ----- Added lines -----
post:
  description: Lets a user post a new artist
  requestBody:
    required: true
    content:
      application/json:
        schema:
          type: object
          required:
            - username
          properties:
            artist_name:
              type: string
            artist_genre:
              type: string
            albums_recorded:
              type: integer
            username:
              type: string

  responses:
    '200':
      description: Successfully created a new artist

    '400':
      description: Invalid request
      content:
```

Visualizzazione degli endpoint specificati

GET

/artists

▼

POST

/artists

^

Lets a user post a new artist

Parameters

Try it out

No parameters

Request body required

application/json ▼

Example Value

Schema

```
{
  "artist_name": "string",
  "artist_genre": "string",
  "albums_recorded": 0,
  "username": "string"
}
```

I parametri possono anche essere all'interno del path

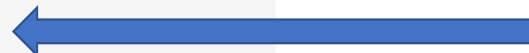
```
/artists/{username}:
  get:
    description: Obtain information about an artist from his or her unique username
    parameters:
      - name: username
        in: path
        required: true
        schema:
          type: string

    responses:
      '200':
        description: Successfully returned an artist
        content:
          application/json:
            schema:
              type: object
              properties:
                artist_name:
                  type: string
                artist_genre:
                  type: string
                albums_recorded:
                  type: integer

      '400':
        description: Invalid request
        content:
          application/json:
```




/artists/{username}

In questo caso la risposta non è un array di item ma è un singolo item.



GET	/artists	✓	🔒
POST	/artists	✓	🔒
GET	/artists/{username}	✓	🔒

Per evitare ridondanza: componenti riutilizzabili

- Schemas (data models)  Esempio gli attributi di ogni «artist»
- Parameters  Esempio limit - page
- Request bodies
- Responses  Esempio risposta errore 400
- Response headers
- Examples
- Links
- Callbacks

Sezione components e riferimenti

```
parameters:
  # ----- Added line -----
  - $ref: '#/components/parameters/PageLimit'
  - $ref: '#/components/parameters/PageOffset'
  # ---- /Added line -----
responses:
  '200':
```

```
  content:
    application/json:
      schema:
        # ----- Added line -----
        $ref: '#/components/schemas/Artist'
        # ---- /Added line -----
responses:
  '200':
    description: Successfully created a new artist
  '400':
    # ----- Added line -----
    $ref: '#/components/responses/400Error'
    # ---- /Added line -----
```

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

```
schemas:
  Artist:
    type: object
    required:
      - username
    properties:
      artist_name:
        type: string
      artist_genre:
        type: string
      albums_recorded:
        type: integer
      username:
        type: string
```

```
# ----- Added lines -----
parameters:
  PageLimit:
    name: limit
    in: query
    description: Limits the number of items on a page
    schema:
      type: integer
```

- Vedere file YAML allegato ... è solo un esempio, dovete ideare voi le API, i parametri, i formati dei dati scambiati, ...

```

1 openapi: 3.0.0
2 info:
3   version: 1.0.0
4   title: Gestione risorse idriche
5   description: API del backend del sistema di gestione risorse idriche
6
7 servers:
8   - url: http://api.greenfarms.org/v1
9
10 paths:
11   /aziende:
12     get:
13       description: Returns a list of companies
14       responses:
15         '200':
16           description: Successfully returned a measure
17           content:
18             application/json:
19               schema:
20                 type: array
21                 items:
22                   $ref: '#/components/schemas/Company'
23         '400':
24           $ref: '#/components/responses/400Error'
25
26
27   /aziende/{idAzienda}/coltivazioni/{idColtivazione}/misure/{idMisura}:
28     get:
29       description: Returns a measure
30       parameters:
31         - name: idAzienda

```

API del backend del sistema di gestione risorse idriche

Servers

http://api.greenfarms.org/v1

default

GET /aziende

GET /aziende/{idAzienda}
GET /coltivazioni/{idColtivazione}
GET /misure/{idMisura}

Schemas

Schemi ripetuti:

```

58 components:
59   schemas:
60     Company:
61       type: object
62       required:
63         - idCompany
64         - name
65         - numFields
66       properties:
67         idCompany:
68           type: integer
69         name:
70           type: string
71         numFields:
72           type: integer
73
74     Misurazione:
75       type: object
76       required:
77         - idColtivazione
78         - data
79         - orario
80         - valore
81       properties:
82         id:
83           type: integer
84         idColtivazione:
85           type: integer
86         data:

```

Schemas

Company ▾ {

idCompany* integer
name* string
numFields* integer

Misurazione ▾ {

id integer
idColtivazione* integer
data* string
pattern: ^\d{4}(0[1-9]|1[012])(0[1-9]|1[12][0-9]|3[01])\$
example: 20210130
Data rilevazione

orario* ▾ {
ora string
pattern: (?:[0-1]\d|[2][0-3])
ora da 00 a 23