

EE4500 – Design Project

RESTful APIs



Introduction to RESTful APIs

- What is an API?
 - API stands for Application Programming Interface.
 - It allows different software applications to communicate with each other.
- What is REST?
 - REST stands for Representational State Transfer.
 - It is an architectural style for designing networked applications.
 - Roy Fielding's PhD Dissertation UC Irvine in 2000.

HTTP Operations

GET	Retrieve data	HTTP 1.0 (1991)
POST	Submit data	HTTP 1.0 (1991)
PUT	Update or create data	HTTP 1.0 (1991)
DELETE	Remove data	HTTP 1.0 (1991)
PATCH	Modify part of data	HTTP 1.1 (1999)
HEAD	Retrieve headers	HTTP 1.0 (1991)
OPTIONS	Discover communication option	HTTP 1.1 (1999)
TRACE	Perform diagnostic tests	HTTP 1.1 (1999)

Key Principles of RESTful APIs

- **Statelessness:**
 - Each request from client to server must contain all the information needed to understand and process the request.
 - The server does not store any client context between requests.
- **Client-Server Architecture:**
 - The client handles the user interface and user state, while the server handles the data storage and logic.
- **Uniform Interface:**
 - A standard way of interacting with a resource (e.g., using HTTP methods like GET, POST, PUT, DELETE).

Components of a RESTful API

- **Resources:**
 - The primary data elements (e.g., users, orders) that can be accessed via the API.
- **Endpoints:**
 - The URLs where the API can be accessed.
- **HTTP Methods:**
 - GET: Retrieve a resource.
 - POST: Create a new resource.
 - PUT: Update an existing resource.
 - DELETE: Remove a resource.
- **Status Codes:**
 - Indicate the result of the HTTP request (e.g., 200 OK, 404 Not Found).

Advantages of Using RESTful APIs

- Scalability:
 - Statelessness and separation of client and server allow RESTful APIs to handle large numbers of requests efficiently.
- Flexibility:
 - RESTful APIs can handle different types of calls and return various data formats (JSON, XML, etc.).
- Interoperability:
 - RESTful APIs can be used across different platforms and languages.
- Simplicity:
 - RESTful APIs use standard HTTP methods and status codes, making them easy to understand and use.

Example API

```
1  openapi: 3.0.0
2  ▼ info:
3    title: Sample API
4    description: Optional multiline or single-line description in [CommonMark](http://commonmark.org/help/) or HTML.
5    version: 0.1.9
6  ▼ servers:
7    - url: http://api.example.com/v1
8      description: Optional server description, e.g. Main (production) server
9    - url: http://staging-api.example.com
10     description: Optional server description, e.g. Internal staging server for testing
11  ▼ paths:
12    /users:
13      get:
14        summary: Returns a list of users.
15        description: Optional extended description in CommonMark or HTML.
16      ▼ responses:
17        "200": # status code
18          description: A JSON array of user names
19        ▼ content:
20          application/json:
21            schema:
22              type: array
23            ▼ items:
24              type: string
```

Endpoints

- Server URL
- Endpoint path
- Query parameters

```
27  ↘ servers:
28    - url: https://api.example.com/v1
29  ↘ paths:
30  ↘ /users:
31    | get:
32
33    https://api.example.com/v1/users?role=admin&status=active
34  ↘ \____ / \____ / \____ / \____ / \____ / \____ /
35  ↘ | | | | | | | | | | | | | | | | | | | | | |
36  ↘ server URL      endpoint      query parameters
37                                path
```

Path parameters

- Extra information
- Embedded within endpoint
- Value is substituted in

```
38  <(paths:  
39    </users/{userId}:  
40      get:  
41        summary: Get a user by ID  
42        parameters:  
43          - in: path  
44            name: userId  
45            schema:  
46              type: integer  
47              required: true  
48              description: Numeric ID of the user to get
```

GET

- Purpose:
 - Retrieve a resource or a collection of resources.
- Usage:
 - Fetch a list of resources (e.g., GET /users).
 - Fetch a single resource by ID (e.g., GET /users/1).
- Characteristics:
 - Safe (does not alter the state of the server).
 - Idempotent (multiple identical requests have the same effect as a single request).

POST

- Purpose:
 - Create a new resource.
- Usage:
 - Submit data to be processed to a specified resource (e.g., POST /users to create a new user).
- Characteristics:
 - Not idempotent (multiple identical requests can create multiple resources).

PUT

- Purpose:
 - Update an existing resource or create a resource if it does not exist.
- Usage:
 - Update a specific resource by ID (e.g., PUT /users/1 to update user with ID 1).
- Characteristics:
 - Idempotent (multiple identical requests have the same effect as a single request).

DELETE

- Purpose:
 - Remove a resource.
- Usage:
 - Delete a specific resource by ID (e.g., DELETE /users/1 to delete user with ID 1).
- Characteristics:
 - Idempotent (multiple identical requests have the same effect as a single request).

PATCH

- Purpose:
 - Apply partial modifications to a resource.
- Usage:
 - Update part of a specific resource by ID (e.g., PATCH /users/1 to update part of the user with ID 1).
- Characteristics:
 - Not necessarily idempotent (depends on implementation).

Summary

- GET: Retrieve data without changing it.
- POST: Create new data.
- PUT: Update or create data.
- DELETE: Remove data.
- PATCH: Modify part of data.

Example – Pet Store

Example: Pet Store

Servers <https://petstore3.swagger.io/api/v3> ▾

Authorize 

pet Everything about your Pets Find out more ^

PUT [`/pet`](#) Update an existing pet  ▾

POST [`/pet`](#) Add a new pet to the store  ▾

GET [`/pet/findByStatus`](#) Finds Pets by status  ▾

GET [`/pet/findByTags`](#) Finds Pets by tags  ▾

GET [`/pet/{petId}`](#) Find pet by ID  ▾

Demo: Postman & Curl

The screenshot shows the Postman application interface. On the left, the 'My Workspace' sidebar is visible with options for Collections, Environments, and History. In the center, a POST request is being made to <https://petstore3.swagger.io/api/v3/pet>. The 'Body' tab is selected, showing a JSON payload:

```
1 {  
2   "id": 11,  
3   "name": "Benji",  
4   "category": {}
```

Below the request, the response details are shown: 200 OK, 345 ms, 508 B. The 'Pretty' tab of the response body shows the same JSON structure.

On the far left, a terminal window displays the execution of a curl command:

```
(base) bruce@Satie:~ $ curl -X 'POST'  'https://petstore3.swagger.io/api/v3/pet'  -H 'accept: application/json' -H 'Content-Type: application/json' -d '{  
  "id": 10,  
  "name": "Curly",  
  "category": {}  
  "id": 1,  
  "name": "Dogs"  
},  
"photoUrls": [  
  "string"  
],  
"tags": [  
  {  
    "id": 0,  
    "name": "string"  
  }  
],  
"status": "available"  
}'  
{"id":10,"category": {"id":1,"name": "Dogs"}, "name": "Curly", "photoUrls": ["string"], "tags": [{"id":0,"name": "string"}]}
```

Demo: Open API

Swagger Editor. Supported by SMARTBEAR

File ▾ Edit ▾ Insert ▾ Generate Server ▾ Generate Client ▾ About ▾

Try our new Editor ↗

```
1 openapi: 3.0.3
2 ...
3   info:
4     title: Swagger Petstore - OpenAPI 3.0
5     description: |-  
6       This is a sample Pet Store Server based on the OpenAPI 3.0 specification. You can find out  
7         more about  
8         Swagger at [https://swagger.io](https://swagger.io). In the third iteration of the pet store,  
9           we've switched to the design first approach!  
10      You can now help us improve the API whether it's by making changes to the definition itself  
11        or to the code.  
12      That way, with time, we can improve the API in general, and expose some of the new features  
13        in OAS3.  
14
15      _If you're looking for the Swagger 2.0/OAS 2.0 version of Petstore, then click [here](https://editor.swagger.io/?url=https://petstore.swagger.io/v2/swagger.yaml). Alternatively,  
16        you can load via the `Edit > Load Petstore OAS 2.0` menu option!_
17
18      Some useful links:  
19      - [The Pet Store repository](https://github.com/swagger-api/swagger-petstore)  
20      - [The source API definition for the Pet Store](https://github.com/swagger-api/swagger-petstore/blob/master/src/main/resources/openapi.yaml)
21
22      termsOfService: http://swagger.io/terms/
23
24      contact:  
25        email: apiteam@swagger.io
26
27      license:  
28        name: Apache 2.0
29        url: http://www.apache.org/licenses/LICENSE-2.0.html
30
31      version: 1.0.11
32
33      externalDocs:  
34        description: Find out more about Swagger
35        url: http://swagger.io
36
37      servers:  
38        - url: https://petstore3.swagger.io/api/v3
39
40      tags:  
41        - name: pet
42          description: Everything about your Pets
43          externalDocs:  
44            description: Find out more
45            url: http://swagger.io
46
47        - name: store
48          description: Access to Petstore orders
49          externalDocs:  
50            description: Find out more about our store
51            url: http://swagger.io
52
53        - name: user
54          description: Operations about user
55
56      paths:  
57        /pet:  
58          put:  
59            tags:  
60              pet:
```

Servers Authorize

pet Everything about your Pets Find out more ^

PUT `/pet` Update an existing pet

POST `/pet` Add a new pet to the store

GET `/pet/findByStatus` Finds Pets by status

GET `/pet/findByTags` Finds Pets by tags

GET `/pet/{petId}` Find pet by ID

POST `/pet/{petId}` Updates a pet in the store with form data

DELETE `/pet/{petId}` Deletes a pet

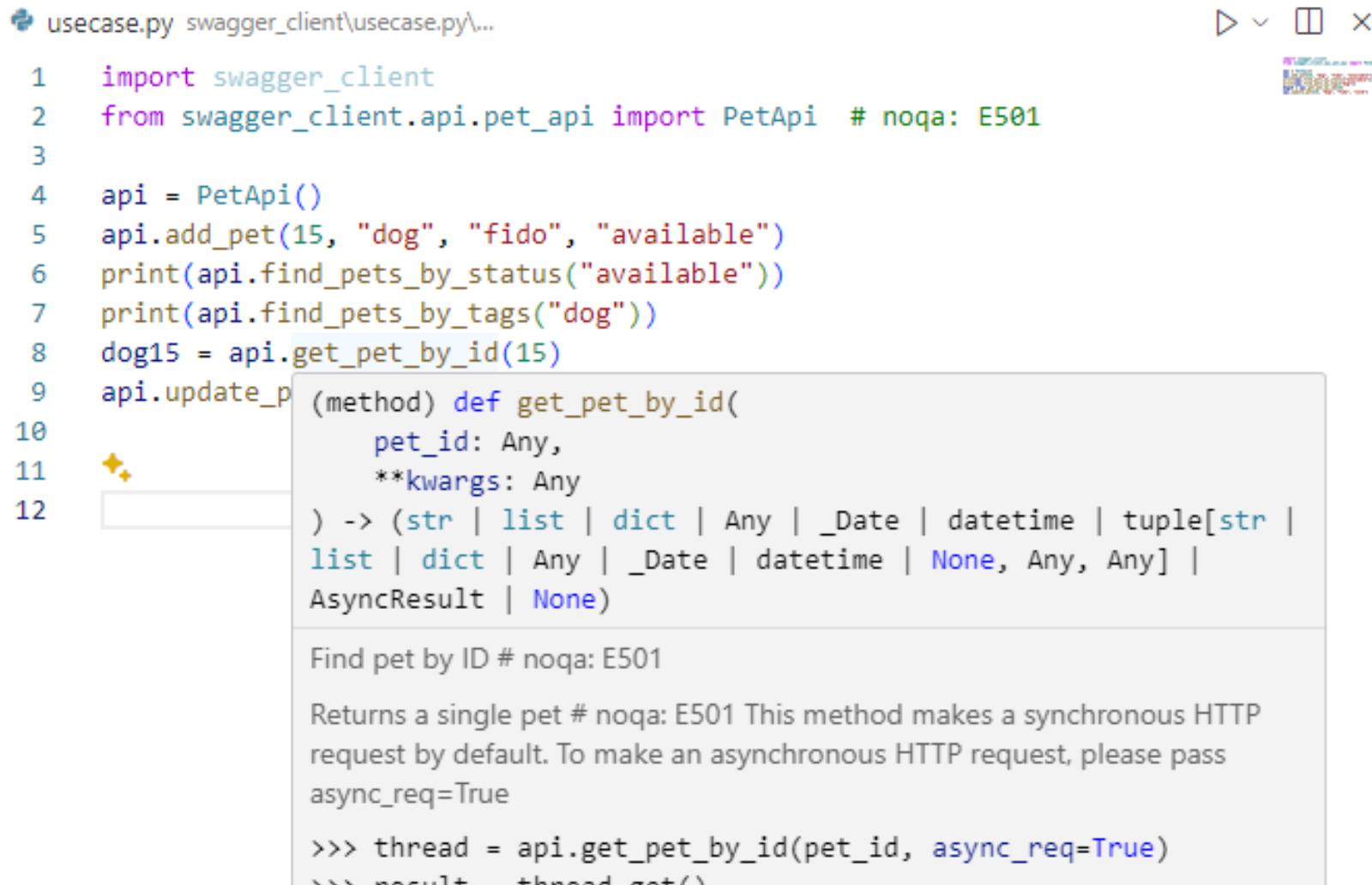
POST `/pet/{petId}/uploadImage` uploads an image

store Access to Petstore orders Find out more about our store ^

GET `/store/inventory` Returns pet inventories by status

POST `/store/order` Place an order for a pet

Using an API



```
usecase.py swagger_client\usecase.py\...
1 import swagger_client
2 from swagger_client.api.pet_api import PetApi # noqa: E501
3
4 api = PetApi()
5 api.add_pet(15, "dog", "fido", "available")
6 print(api.find_pets_by_status("available"))
7 print(api.find_pets_by_tags("dog"))
8 dog15 = api.get_pet_by_id(15)
9 api.update_p (method) def get_pet_by_id(
10     pet_id: Any,
11     **kwargs: Any
12 ) -> (str | list | dict | Any | _Date | datetime | tuple[str |
list | dict | Any | _Date | datetime | None, Any, Any] |
AsyncResult | None)
Find pet by ID # noqa: E501
Returns a single pet # noqa: E501 This method makes a synchronous HTTP
request by default. To make an asynchronous HTTP request, please pass
async_req=True
>>> thread = api.get_pet_by_id(pet_id, async_req=True)
>>> result = thread.get()
```

The Bezos Mandate



Bezos Mandate

1. All teams will henceforth expose their data and functionality through service interfaces.
2. Teams must communicate with each other through these interfaces.
3. There will be no other form of interprocess communication allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network.
4. It doesn't matter what technology they use. HTTP, Corba, Pubsub, custom protocols — doesn't matter.
5. All service interfaces, without exception, must be designed from the ground up to be externalizable. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions.
6. Anyone who doesn't do this will be fired.
7. Thank you; have a nice day!

Data and Functionality Exposure

1. All teams will henceforth expose their data and functionality through service interfaces.

Enforcing Standardized Communication

2. Teams must communicate with each other through these interfaces.
3. There will be no other form of interprocess communication allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network.

Technology Agnosticism

4. It doesn't matter what technology they use. HTTP, CORBA, PubSub, custom protocols — it doesn't matter.

Externalization

5. All service interfaces, without exception, must be designed from the ground up to be externalizable. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions.



Next Business models