EE4500 – Design Project

RESTful APIs

JAMES COOK UNIVERSITY
AUSTRALIA

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

```yaml
openapi: 3.0.0
info:
  title: Sample API
  description: Optional multiline or single-line description in [CommonMark](http://commonmark.org/help/) or HTML.
  version: 0.1.9
servers:
  - url: http://api.example.com/v1
    description: Optional server description, e.g. Main (production) server
  - url: http://staging-api.example.com
    description: Optional server description, e.g. Internal staging server for testing
paths:
  /users:
    get:
      summary: Returns a list of users.
      description: Optional extended description in CommonMark or HTML.
      responses:
        "200": # status code
          description: A JSON array of user names
          content:
            application/json:
              schema:
                type: array
                items:
                  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       server URL              endpoint    query parameters
36                                path
```

Path parameters

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

```yaml
paths:
  /users/{userId}:
    get:
      summary: Get a user by ID
      parameters:
        - in: path
          name: userId
          schema:
            type: integer
          required: true
          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

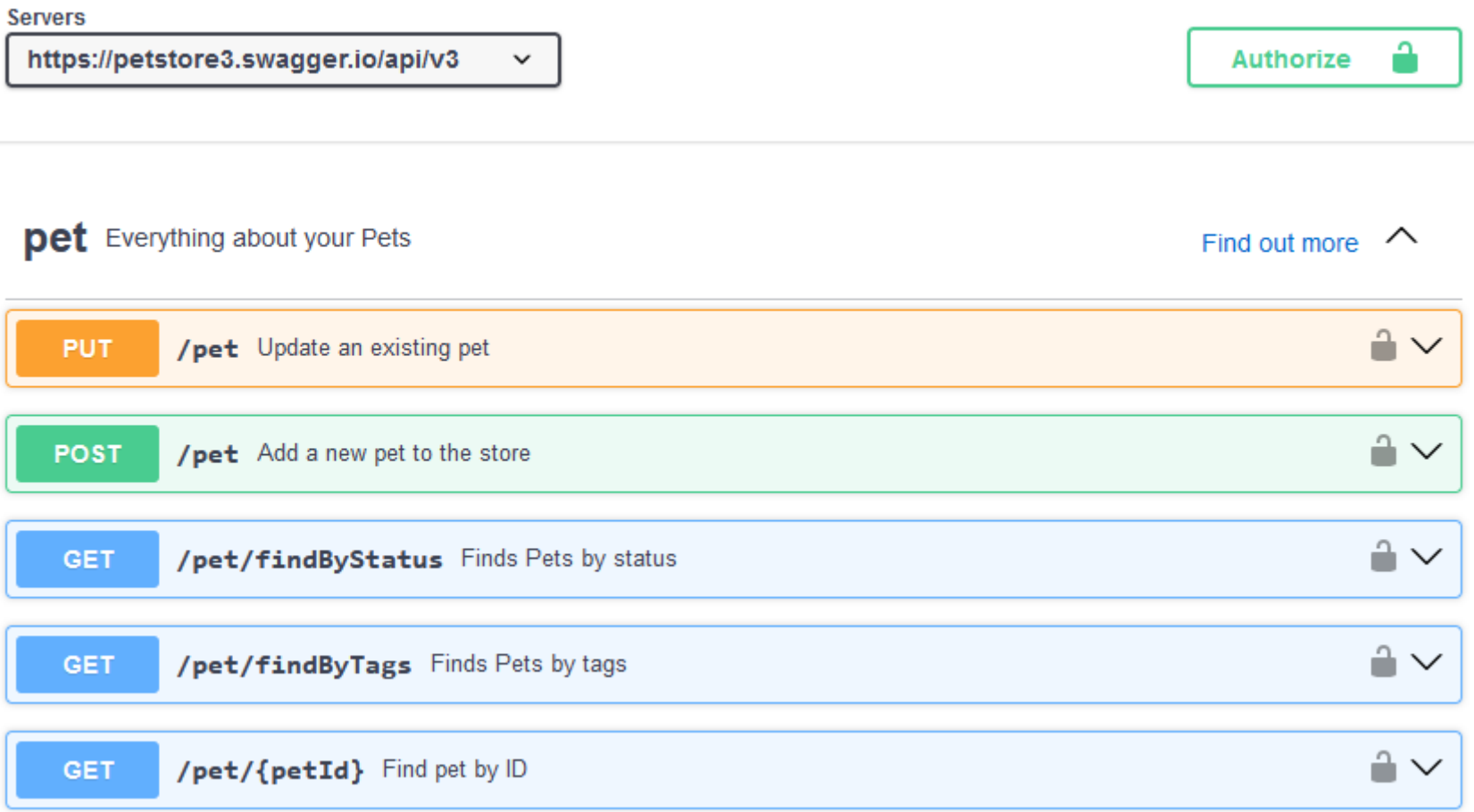
- Purpose:
  - Apply partial modifications to a resource.
- Usage:
  - Update part of a specific resource by ID (e.g., PATCH /users/1 to update part of the user with ID 1).
- Characteristics:
  - Not necessarily idempotent (depends on implementation).

Summary

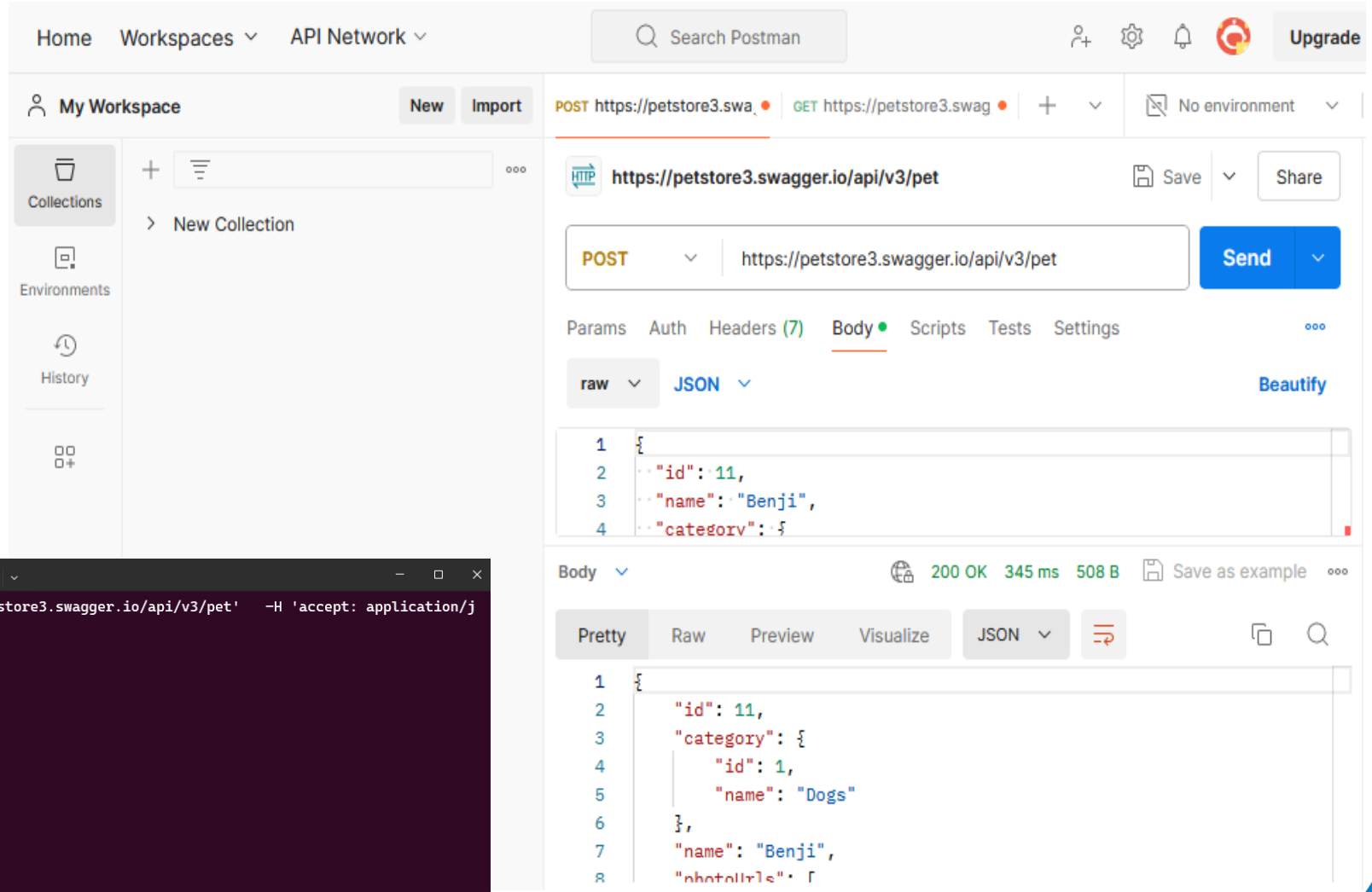
- GET: Retrieve data without changing it.
- POST: Create new data.
- PUT: Update or create data.
- DELETE: Remove data.
- PATCH: Modify part of data.

Example – Pet Store

Example: Pet Store

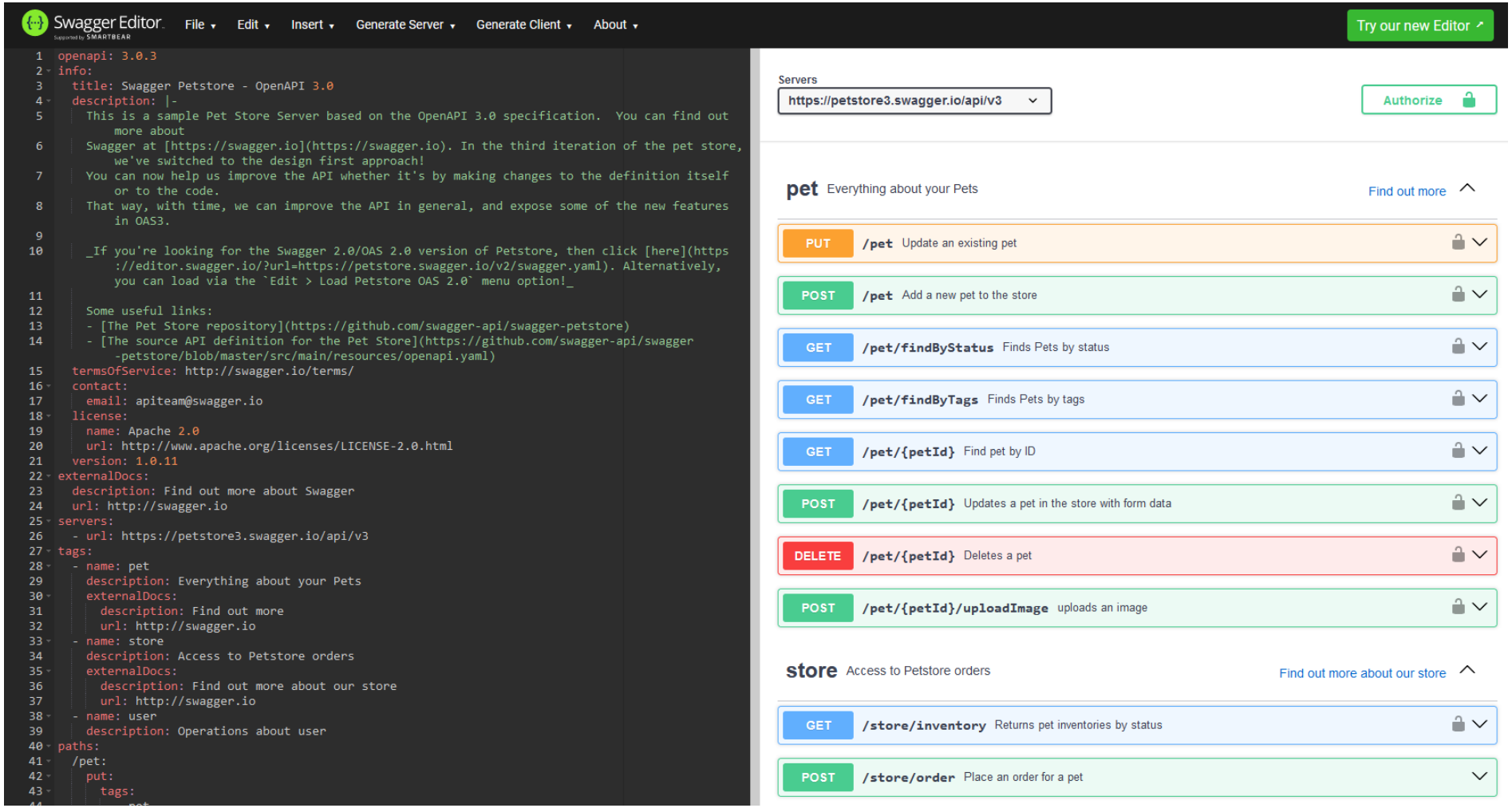


Demo: Postman & Curl



JAMES COOK UNIVERSITY AUSTRALIA

Demo: Open API



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
10
11
12
```

```
(method) def get_pet_by_id(
    pet_id: Any,
    **kwargs: Any
) -> (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