UT 4 – API REST

Programación de Servicios y Procesos

Curso 2024-25

Profesor: Agustín González-Quel

API – Application Programming Interface

API (application programming interface) es un intermediario que permite a dos aplicaciones comunicarse de una forma perfectamente definida en cuanto a datos de entrada, códigos de error y datos de respuesta.

- A veces se denomina contrato entre un proveedor y un usuario de información.
- La definición del API establece el contenido que necesita el consumidor (la llamada) y el contenido que necesita el productor (la respuesta).
  - Por ejemplo, el diseño de la API para un servicio meteorológico podría especificar que el usuario facilite un código postal y que el productor responda con una respuesta en dos partes, siendo la primera la temperatura alta y la segunda la baja.

Se puede pensar en una API como

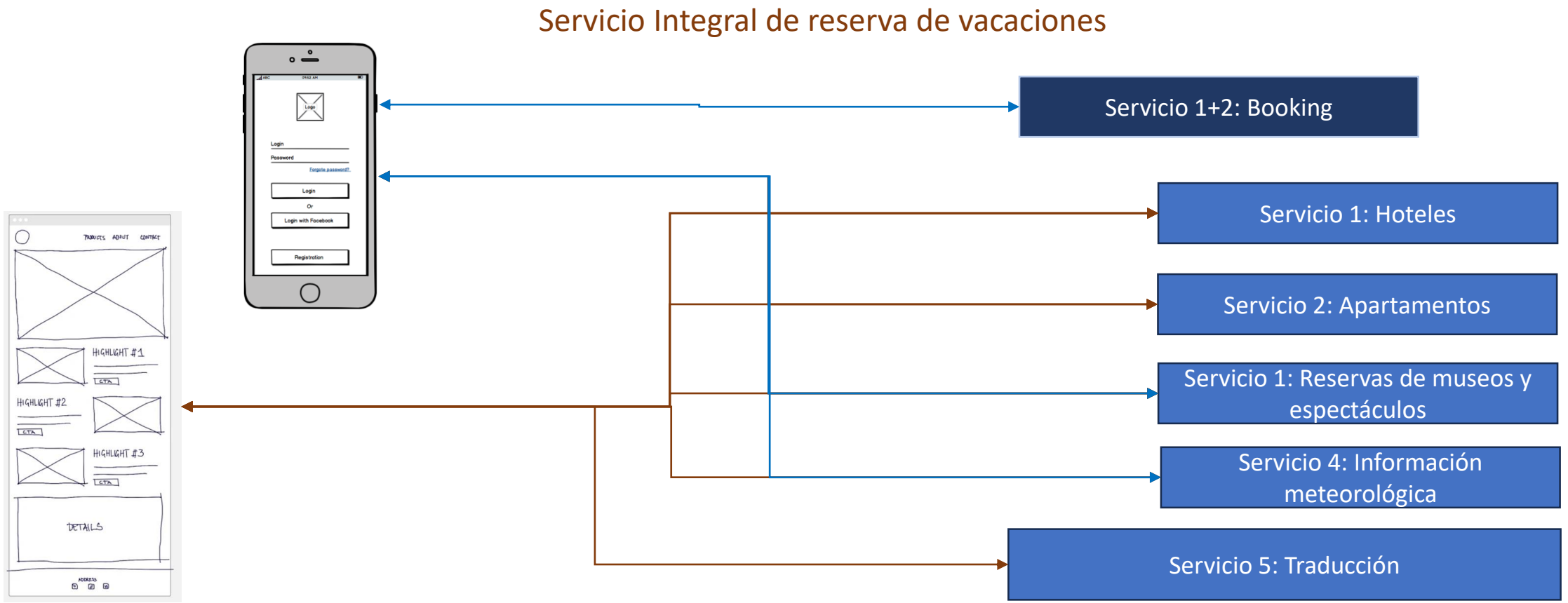
- Mediador entre los usuarios o clientes y los recursos o servicios web que quieren obtener.
- Forma para compartir recursos e información manteniendo la seguridad, el control y la autenticación, es decir, determinando quién tiene acceso a qué.

Otra ventaja de una API es que no es necesario conocer los detalles de implementación del servicio y permite descomponer sistemas complejos en módulos más sencillos.

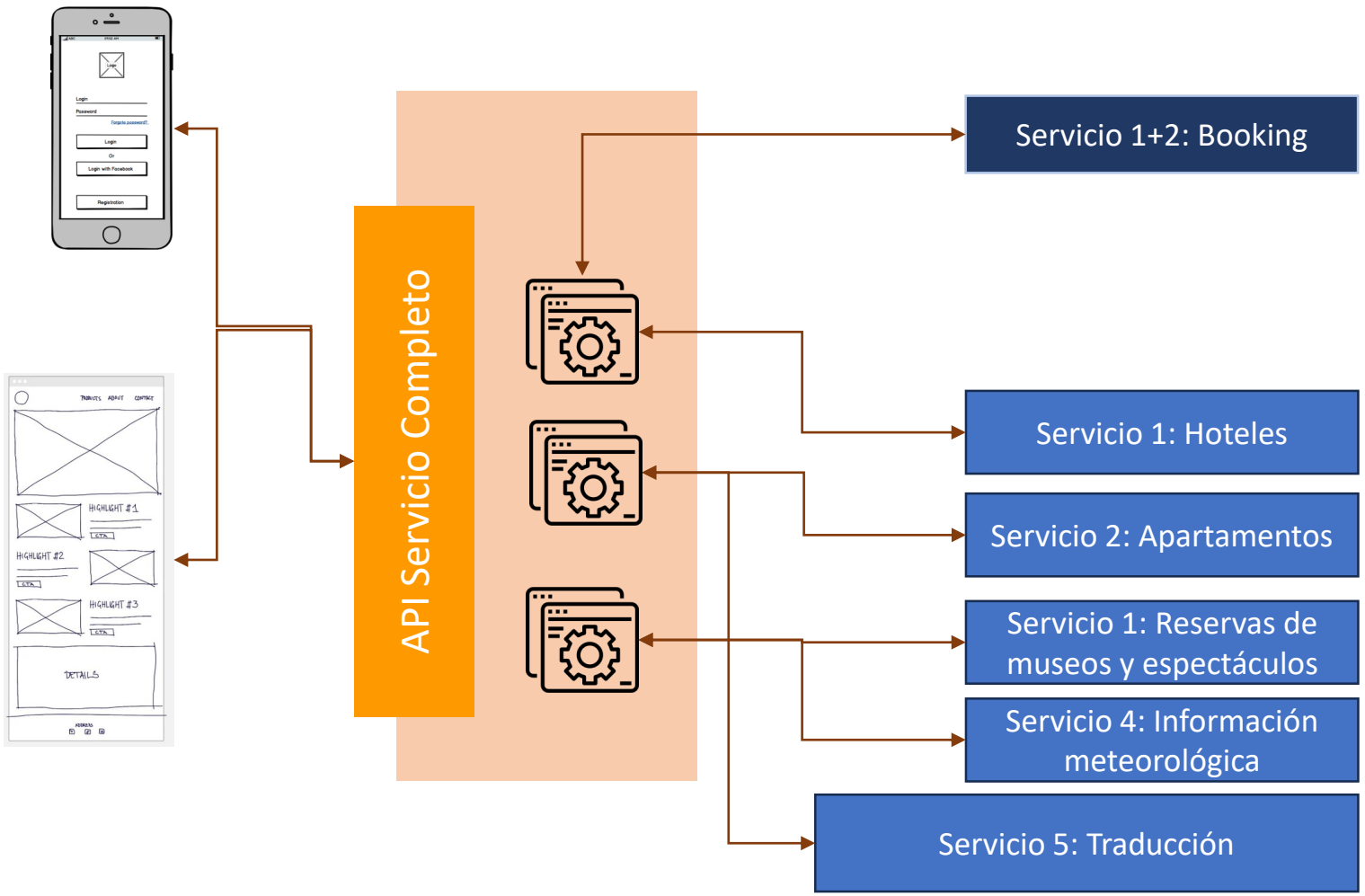
¿Para qué se utilizan APIs?

Ejemplo

- Servicio integral de reserva de vacaciones que incorpora hoteles, apartamentos turísticos, entradas a museos y espectáculos, información meteorológica, capacidades multilenguaje, etc.



Arquitecturas con API



Ventajas de esta arquitectura

- Perfecta definición del interfaz
- Independencia de los servicios que hay por debajo
- Capacidad de evolución
- Mejor mantenimiento
- Versionado de APIs
- Registro de llamadas (contabilidad, monetización)
- …

API REST

- API REST (también conocida como API RESTful) es una interfaz de programación de aplicaciones (API o API web) que se ajusta a las restricciones del estilo arquitectónico REST y permite interactuar con servicios web RESTful.
- REST significa transferencia de estado representacional y fue creado por el informático Roy Fielding. (https://en.wikipedia.org/wiki/REST)
- REST es un conjunto de restricciones arquitectónicas, no un protocolo ni un estándar. Los desarrolladores de API pueden implementar REST de diversas maneras, pero suele incluir:
  - Protocolo HTTP
  - Cabeceras y los parámetros también son muy importantes en los métodos HTTP de una solicitud HTTP de API RESTful, ya que contienen información de identificación importante:
    - Códigos de éxito / error.
    - Metadatos de la solicitud: **la autorización**, cookies, etc.
  - Fundamento: toda llamada al API RESTful transfiere una representación del estado del recurso al solicitante o punto final de forma que no hay necesidad de almacenamiento de estados.
  - Contenido se transmite en JSON, HTML o texto plano.
  - Hay cabeceras de solicitud y cabeceras de respuesta, cada una con su propia información de conexión HTTP y códigos de estado.
  - Referencia: https://github.com/argob/estandares/blob/master/estandares-apis.md

API se apoya en el protocolo HTTP y sus métodos

HTTP define un conjunto de métodos de petición para indicar la acción que se desea realizar para un recurso determinado.

- **GET**: El método GET solicita una representación de un recurso específico. Las peticiones que usan el método GET sólo deben recuperar datos.
- **POST**: El método POST se utiliza para enviar una entidad a un recurso en específico, causando a menudo un cambio en el estado o efectos secundarios en el servidor.
- **PUT**: El modo PUT reemplaza todas las representaciones actuales del recurso de destino con la carga útil de la petición.

Menos usados:

- HEAD: El método HEAD pide una respuesta idéntica a la de una petición GET, pero sin el cuerpo de la respuesta.
- DELETE: El método DELETE borra un recurso en específico.
- CONNECT: El método CONNECT establece un túnel hacia el servidor identificado por el recurso.
- OPTIONS: El método OPTIONS es utilizado para describir las opciones de comunicación para el recurso de destino.
- TRACE: El método TRACE realiza una prueba de bucle de retorno de mensaje a lo largo de la ruta al recurso de destino.
- PATCH: El método PATCH es utilizado para aplicar modificaciones parciales a un recurso.

Códigos de respuesta HTTP

100 Series: Situaciones temporales

- 100 Continue
- 101 Switching Protocols
- 102 Processing

200 Series: Éxito

- 200 – OK
- 201 – Created
- 202 – Accepted
- 203 – Non-Authoritative Information
- 204 – No Content
- 205 – Reset Content
- 206 – Partial Content
- 207 – Multi-Status
- 208 – Already Reported
- 226 – IM Used

- **300 Series: Redirección URL**
  - 300 – Multiple Choices
  - 301 – Moved Permanently
  - 302 – Found
  - 303 – Check Other
  - 304 – Not Modified

- **300 Series: Redirección URL (cont)**
  - 305 – Use Proxy
  - 306 – Switch Proxy
  - 307 – Temporary Redirect
  - 308 – Permanent Redirect

- **400 Series: Error en la llamada.**
  - 400 – Bad Request
  - 401 – Unauthorised
  - 402 – Payment Required
  - 403 – Forbidden
  - 404 – Not Found
  - 405 – Method Not Allowed
  - 406 – Not Acceptable
  - 407 – Proxy Authentic
  - 408 – Request Timeout
  - 409 – Conflict
  - 410 – Gone
  - 411 – Length Required
  - 412 – Precondition Failed
  - 413 – Payload Too Large
  - 414 – URI Too Long
  - 415 – Unsupported Media Type

400 Series (cont)

- 416 – Range Not Satisfiable
- 417 – Expectation Failed
- 418 – I'm a teapot
- 421 – Misdirected Request
- 422 – Unprocessable Entity
- 423 – Locked
- 424 – Failed Dependency
- 426 – Upgrade Required
- 428 – Precondition Required
- 429 – Too Many Requests
- 431 – Request Header Fields Too Large
- 451 – Unavailable For Legal Reasons

- **500 Series: Error de servidor**
  - 500 – Internal Server Error
  - 501 – Not Implemented
  - 502 – Bad Gateway
  - 503 – Service Unavailable
  - 504 – Gateway Timeout
  - 505 – HTTP Version Not Supported
  - 506 – Variant Also Negotiates
  - 507 – Insufficient Storage
  - 508 – Loop Detected
  - 510 – Not Extended
  - 511 – Network Authentication Required

Tema específico: Autenticación

Autenticación es la validación de identidades en el acceso a una aplicación.

API REST ofrece varios mecanismos

- Autenticación básica: Basada en pareja usuario-contraseña
  - El par usuario-contraseña se envía en cada llamada incluida en el header.
  - Evita el uso de cookies, tokens u otros artificios para mantener la sesión.

- API Key: Código que identifica de forma unívoca a cada usuario
  - La seguridad se basa en la longitud de la cadena y en el proceso de gestión de identidades.

- OAuth 2.0
  - La autenticación del usuario no la realiza el API sino un servidor OAuth que conoce al usuario.
  - El servidor OAuth devuelve al usuario un token de acceso que es el que valida en el API
  - El API identifica al usuario por el token.

https://konghq.com/blog/engineering/common-api-authentication-methods

Operaciones con APIs

Consumo de API

- Desde herramienta: Postman
- Desde programa, con librerías específicas que realizan llamadas HTTP
  - requests (Python).

Creación de API

Aplicaciones que ofrecen un API

- Basadas en librerías como FastAPI, Flask, etc. (Python)

  Nosotros usaremos FastAPI
- Usando herramientas específicas: API Management tools o API Gateway
  - MuleSoft, WebMethods, Kong, AWS, IBM, etc.

Uso de API REST

Consumo de API REST

- Consultar la documentación de la API REST para comprender cómo funciona:
  - URL base
  - Requisitos de seguridad/autenticación
  - Definición de métodos (método HTTP, ruta URL, formato de respuesta)

Usar un cliente API REST para pruebas

Postman - https://www.postman.com/

Atacar el API Rest desde el sistema que estemos usando.

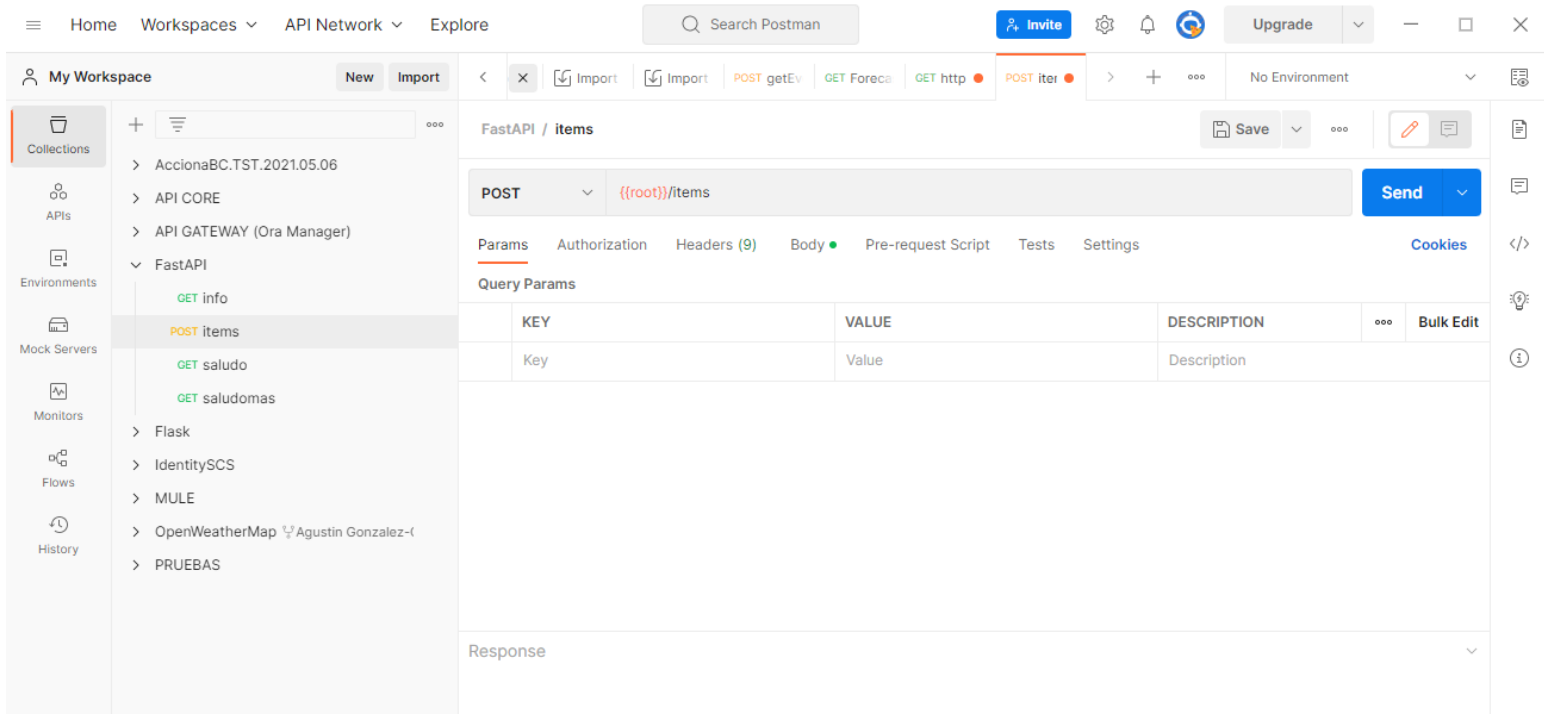
- Practicaremos con requests-Python

Algunos sitios para practicar

https://www.thecocktaildb.com/api.php
https://swapi.dev/
https://opendata.aemet.es/
https://www.deepl.com/pro-api

Qué necesitamos saber

- URL base
- Path del endpoint
- Método
- Formatos de envío de datos
  - Parámetros Path
  - Parámetros Query String
  - Parámetros Body (data)
- Mecanismo de autenticación
  - Token
  - API Key



Postman es un cliente HTTP pensado para trabajar con API REST y nos facilita la explotación de APIs.

Envío de contenido (parámetros) en la llamada al API (1)

- **Path parameters: En la URL del endpoint,** La URL del endpoint se extiende con el valor o valores que se envían:

https://opendata.aemet.es/opendata/api/prediccion/especifica/municipio/horaria/{Código_Municipio}

https://opendata.aemet.es/opendata/api/prediccion/especifica/municipio/horaria/23055

http://192.168.1.40:8090/saludamemas/Maite/58

Envío de contenido (parámetros) en la llamada al API (2)

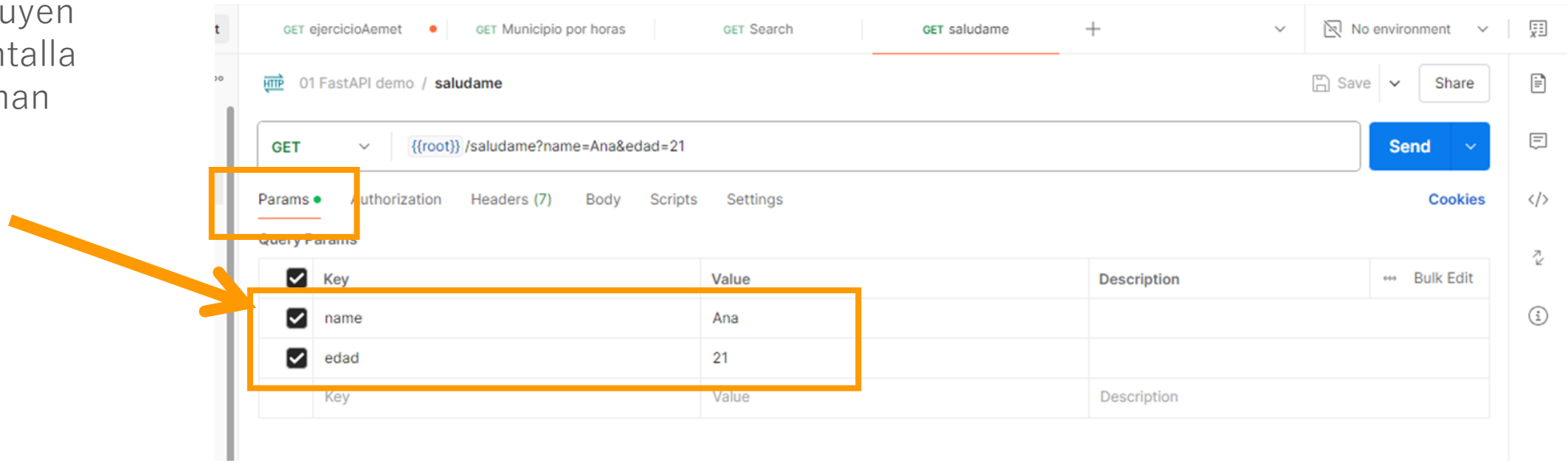
- **Parámetros Query**:

Se envían parámetros en la URL después de una interrogación (?) que separa el endpoint de la lista de parámetros.

La lista de parámetros son pares nombreparámetro=valor separados por el carácter &

http://registerStudents.com/additem?name=Eva&age=25

O se incluyen en la pantalla de Postman



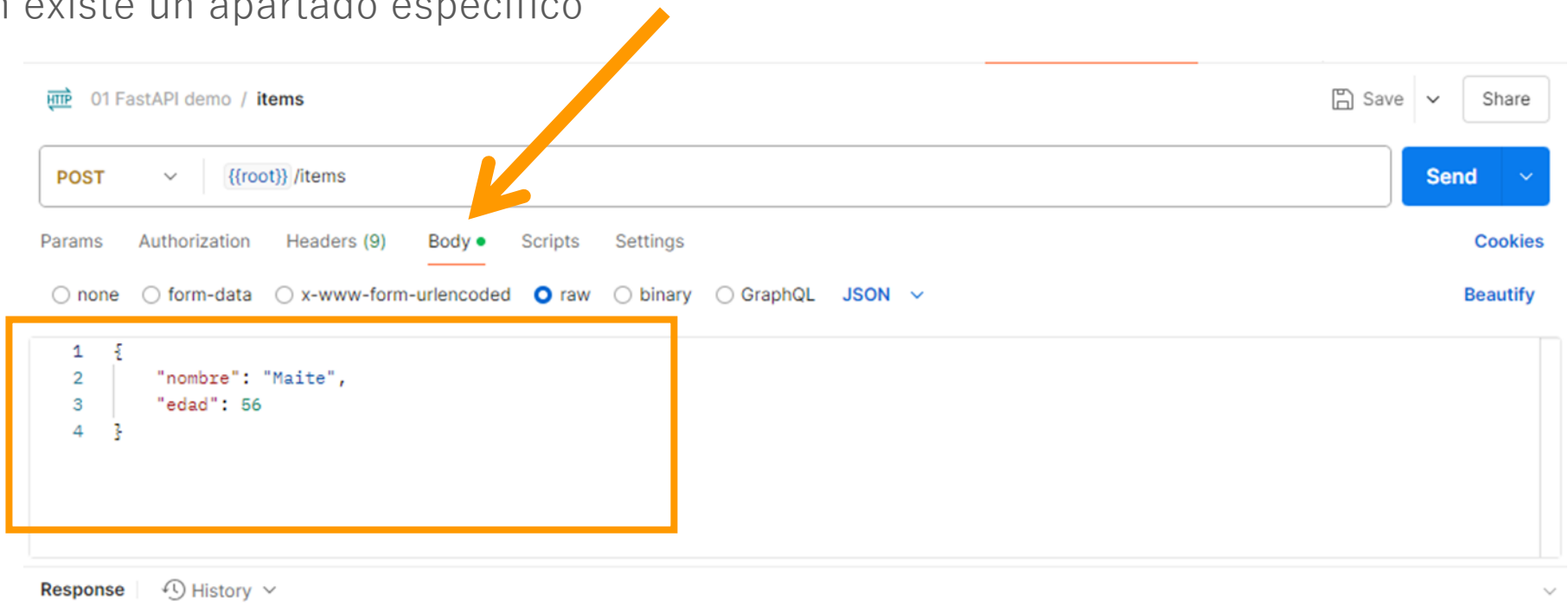
Envío de contenido (parámetros) en la llamada al API (y 3)

- **Parámetros en el body de la llamada**: Se incluye un JSON en la estructura de la llamada HTTP

```
wget --method POST --header 'Content-Type:text/html’ --body-data '{"nombre":"Maite", "edad":56}’ 'http://192.168.1.40:8090/items’

curl --location 'http://192.168.1.40:8090/items’ --header 'Content-Type: text/html' --data '{"nombre":"Maite", "edad":56}'
```

Si usamos Postman existe un apartado específico



Trabajando con HTTP desde programa

Librería **requests**

- Permite hacer llamadas HTTP: páginas web, API REST, …
- https://requests.readthedocs.io/en/latest/

Peticiones básicas

```
r = requests.get('https://api.github.com/events’)
r = requests.get('https://restapi.org/’, auth=(‘user’, ‘pwd’))
r = requests.post('https://httpbin.org/post')
r = requests.put('https://httpbin.org/put’)
```

Paso de parámetros en requests

- Parámetros URL: Se construye la cadena de la llamada y es la que se usa en la petición requests

```
req1 = requests.get(URLBase+"/saludamemas"+"/"+nombre+"/"+str(edad))
```

- Parámetros tipo query

Se construye un JSON que se pasa como parámetro con la opción **params** de la llamada

```
parametros = {"name":'Maite', 'edad':17}
req1 = requests.get(URLBase+"/saludame", params=parametros)
```

- Parámetros en el Body

Se construye un JSON que se pasa como parámetro con la opción **data** de la llamada

```
datos = json.dumps({"nombre":"Maite", "edad":58})
req1 = requests.put(URLBase+"/tomajson", data=datos)
```

Autenticación con requests

Autenticación básica

- Necesitamos la clase HTTPBasicAuth
- Pasaremos una tupla con los datos de autenticación y se incorporan a la llamada en el parámetro auth.

```
basic = HTTPBasicAuth('ciudad', 'escolar')
url = URLBase + endpoint
response = requests.get(url, auth=basic)
```

API Key

- Añadimos la clave a las cabeceras en formato JSON con la clave “api_key”

```
authString = {"api_key": key}
response = requests.get(url, params=authString)
```

Requests devuelve un objeto response → Contenido

Contenido de response

- .encoding: charset usado
- .content: contenido de la respuesta en binario
- .json(): resultado de la llamada en formato json si aplica, error en caso contrario.
- .status_code: código de error de HTTP
- .text: texto Unicode de la respuesta

```
>>> r = requests.get('https://api.github.com/user', auth=('user', 'pass'))
>>> r.status_code
200
>>> r.headers['content-type']
'application/json; charset=utf8'
>>> r.encoding
'utf-8'
>>> r.text
'{"type":"User"...'
>>> r.json()
{'private_gists': 419, 'total_private_repos': 77, ...}
```

Creamos nuestro propio API

Creando APIs con FastAPI

Podemos crear API para acceder a nuestras aplicaciones o como frontal de aplicaciones de terceros.

- 3 librerías principales en Python: FastAPI, Flask, Django

Trabajaremos con FastAPI

- https://fastapi.tiangolo.com/

Mecanismos:

```
pip install fastapi
pip install uvicorn
```

Cómo funciona:

- Usamos ***decorators*** para definir el API
- Accedermos a documentación URL/docs URL/redoc
- Arrancamos nuestro API con uvicorn

```
if __name__ == "__main__":
    uvicorn.run(app, host="192.168.1.40", port=8090)
```

```
@app.METODO(PATH)

@app.get("/")
@app.put("/inicio/add")
@app.post("/")
```

Paso de parámetros con FastAPI (1)

3 formas de recibir información en el API

- **Parámetros en URL**: Se pone entre llaves un nombre de variable que manejará como parámetro de entrada la función que implementa el endpoint.

```
@app.get("/hola1/{nombre}")
def hola1(nombre: str):
    return {"contenido": "Hola {}".format(nombre)}
```

- **Parámetros Query**: Se define como parámetros de entrada en la función que implementa el endpoint.

```
@app.get("/hola2")
def hola2(name: str):
    return {"contenido": "Hola {}".format(name)}
```

Paso de parámetros con FastAPI (2)

- **Body**: Se puede construir como un único dict o como un JSON compuesto de varios campos.

```
@app.post("/items/")
# formato de la llamada URLBASE/items/ y en un body con el siguiente JSON {"nombre":"Mar", "edad":56}
def create_item(elem: dict = Body(...)):
    value = {"contenido": "Hola {}, tienes {} años".format(elem['nombre'], elem['edad'])}
    return(value)
```

```
@app.put("/tomajson/")
# formato de la llamada URLBASE/tomajson/ y en un body con el siguiente JSON {"nombre":"Mar", "edad":56}
def tomajson(nombre: str = Body(...), edad: int = Body(...)):
    n = nombre
    e = edad
    value = {"contenido": "Hola {}, tienes {} años".format(n, e)}
    return(value)
```

Autenticación básica con FastAPI

Puede incorporarse autenticación básica al API

```
from fastapi import Depends, HTTPException, status
from fastapi.security import HTTPBasic, HTTPBasicCredentials

# Implementación seguridad
security = HTTPBasic()
app = FastAPI(dependencies=[Depends(security)])

# Validación del usuario
def validar(creds: HTTPBasicCredentials = Depends(security)):
    username = creds.username
    password = creds.password
    if username == "ciudad" and password == "escolar":
        print("Usuario validado correctamente")
        return True
    else:
        # From FastAPI
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Usuario no autenticado correctamente",
            headers={"WWW-Authenticate": "Basic"},
        )


@app.get("/info")
def read_root(acceso = Depends(validar)):
    if acceso:
        return(json.loads('{"contenido": "API autenticado ", "autor": "Agustin"}'))
```

Ejercicio del Aula Virtual