

Programación Avanzada

IIC2233 2023-2

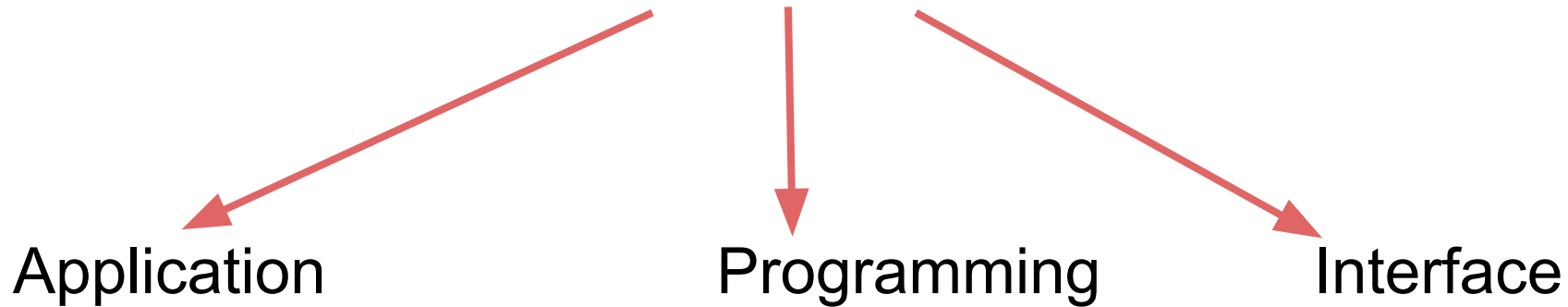
Hernán Valdivieso - Daniela Concha - Francisca Ibarra - Joaquín Tagle - Francisca Cattán



Anuncios

- Hoy tendremos la quinta y penúltima actividad.
- El lunes se publica la última tarea del curso. Esta evalúa programación funcional y webservices.
- El otro jueves tendremos programación/sala de ayuda

API



API

En general, API es un conjunto de funciones que son expuestas por un servicio para ser utilizadas por otros programas.

Nosotros nos enfocaremos específicamente en los servicios web o *web services*.

**Comunicándose a
través de internet**

HTTP (*Hypertext Transfer Protocol*)

Así como en la tarea se definió un protocolo para comunicarse entre cliente y servidor, en el mundo real existe un protocolo altamente utilizado para comunicarse a través de internet: **HTTP**

- Cliente envía una petición, y el servidor le envía una respuesta
- Una vez enviada la respuesta, el servidor “olvida” dicha petición.

¿Qué pasa si queremos enviar una solicitud asegurando que somos nosotros?

URL (*Uniform Resource Locator*)

Cuando estamos navegando en internet, hay cientos de miles de páginas, y cada página puede tener cientos de directorios o recursos. Para poder identificar dentro de este mundo de opciones, existe el concepto de URL. A grandes rasgos, una URL es un texto, que nos ayudará a localizar lo que sea que estemos buscando en internet.

URL (*Uniform Resource Locator*)

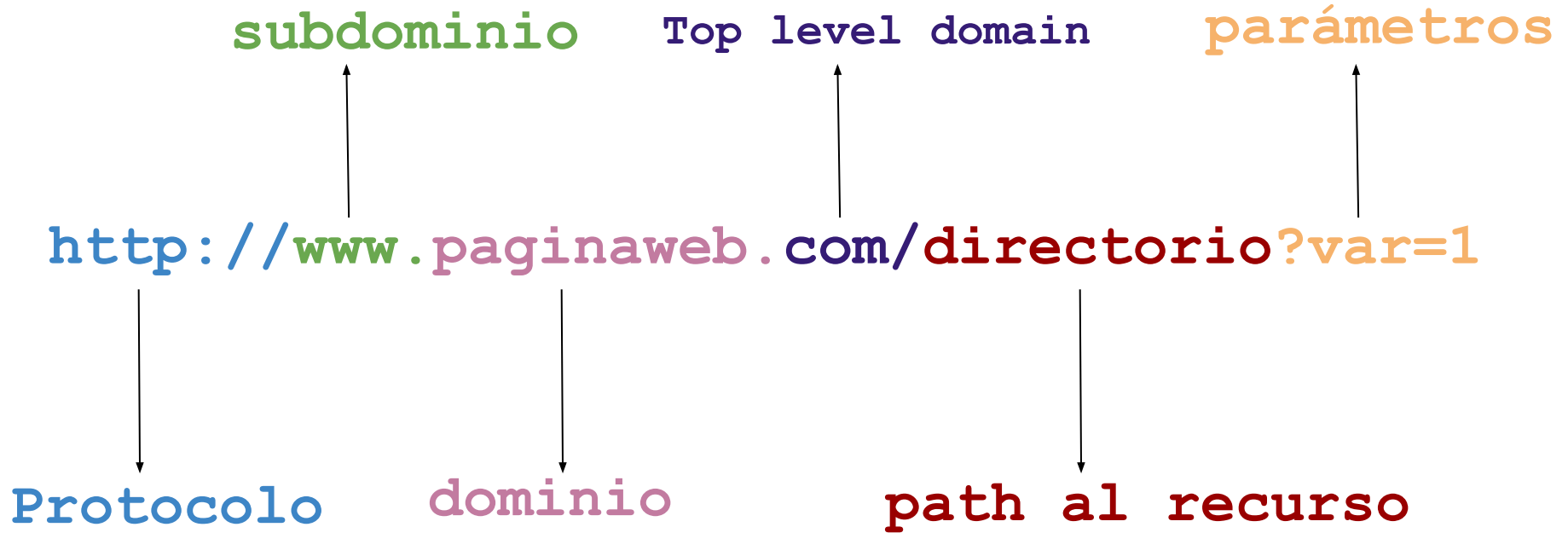
Cuando estamos navegando en internet, hay cientos de miles de páginas, y cada página puede tener cientos de directorios o recursos. Para poder identificar dentro de este mundo de opciones, existe el concepto de URL. A grandes rasgos, una URL es un texto, que nos ayudará a localizar lo que sea que estemos buscando en internet.

Para lograr esto, se define un formato que deben cumplir las URLs para poder apuntar a lo que sea que deseen apuntar.

URL (*Uniform Resource Locator*)

`http://www.paginaweb.com/directorio?var=1`

URL (*Uniform Resource Locator*)



URL (*Uniform Resource Locator*)

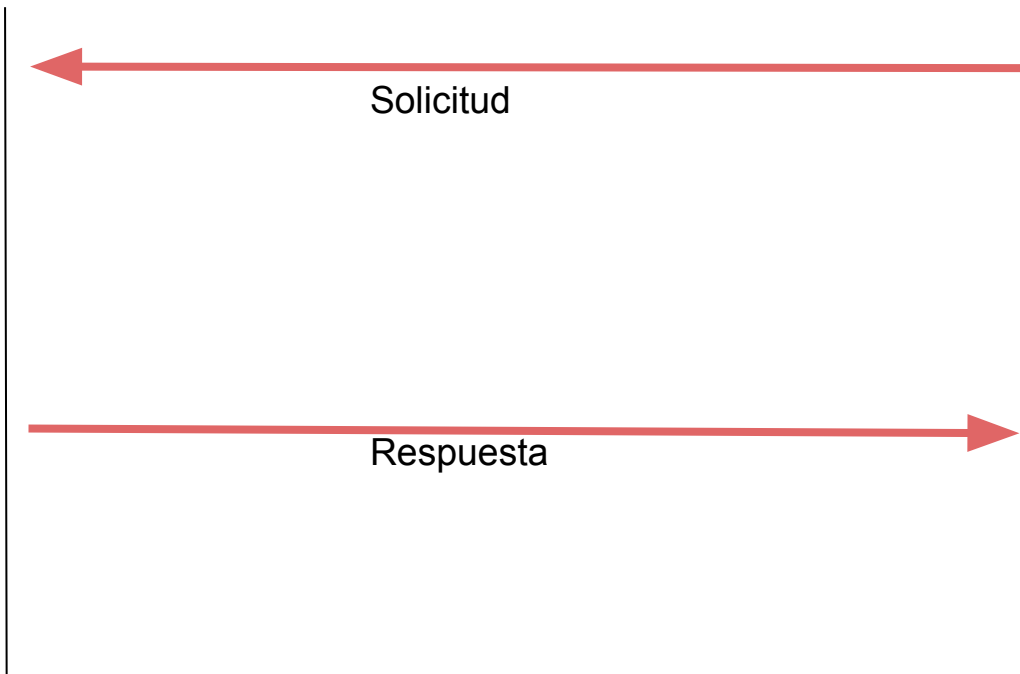
Entonces, ¿cómo funciona la siguiente URL?

`https://www.github.com/IIC2233/Syllabus/issues?page=2`

**¿Cómo funciona la
comunicación una
vez que accedo a
una URL?**

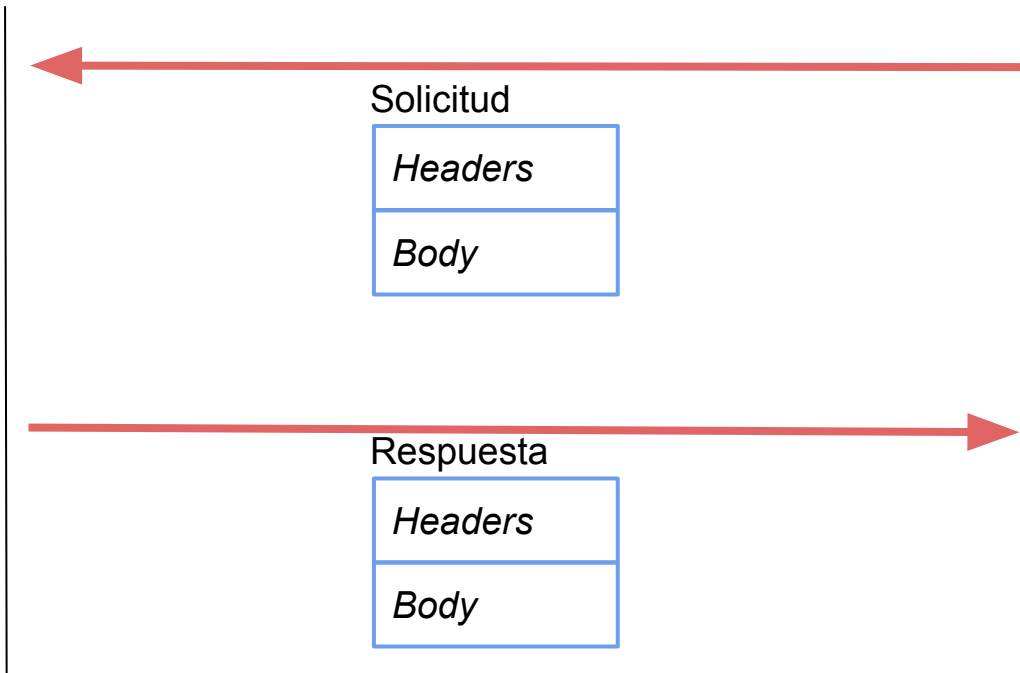
Servidor
HTTP

Cliente
HTTP



Servidor
HTTP

Cliente
HTTP



Tenemos el cómo comunicarnos. Ahora...

**¿Qué es lo que
comunicamos?**

Tipos de solicitudes: Informando al servidor lo que queremos hacer.

- **GET:** Pedimos la representación de un recurso sin cambiar nada en el servidor.
- **POST:** Creamos un recurso.
- **PATCH:** Aplica modificaciones parciales a un recurso.
- **PUT:** Reemplaza completamente un recurso existente.
- **DELETE:** Elimina un recurso.

Códigos de respuesta: Informando al cliente lo que ocurrió.

- **200:** Ok, solicitud exitosa.
- **403:** La solicitud es correcta, pero se rechaza dar una respuesta.
- **404:** El recurso solicitado no se encuentra en el servidor.
- **500:** Error interno del servidor

Y muchos más...

**Una URL puede
(o no) tener
varias acciones
asociadas**

Ejemplo





Supongamos que tenemos la página web <http://www.paginaiiic2233.com/>, que tiene un recurso estudiantes. Para poder interactuar con los datos de la página, se habilitaron las siguientes URLs

<http://www.paginaiiic2233.com/estudiantes>

<http://www.paginaiiic2233.com/estudiante/<id>>

Donde el id es un número. Considerando esto, podríamos pensar que dicha página tiene las siguientes acciones:

Ejemplo

URL	GET	POST	PUT	PATCH	DELETE
<u>/estudiantes</u>	Obtiene una lista de todos los estudiantes	Crea un nuevo estudiante			
<u>/estudiante/1/</u>	Obtiene los datos del estudiante con id 1		Reemplaza algunos atributos del estudiante con id 1	Reemplaza todos los datos del estudiante con id 1	Elimina al estudiante con id 1

*Esto es solo un ejemplo, las funcionalidades dependerán de lo que decida quien implementó el *webservice*

**¿Cómo sabe una
API quién soy?**

Autenticación

Problema

- Hay acciones, como editar la base de datos, que no todo usuario puede hacer.
- Generalmente las APIs se componen de funciones que “no memorizan entre una solicitud y otra”. Por lo tanto, no podemos hacer una solicitud de “login” y luego una de “modificar base de datos” en donde recuerde el login previo.
- Surge la necesidad de un mecanismo que permita, en cada solicitud, poder identificar al usuario.

Autenticación

Solución

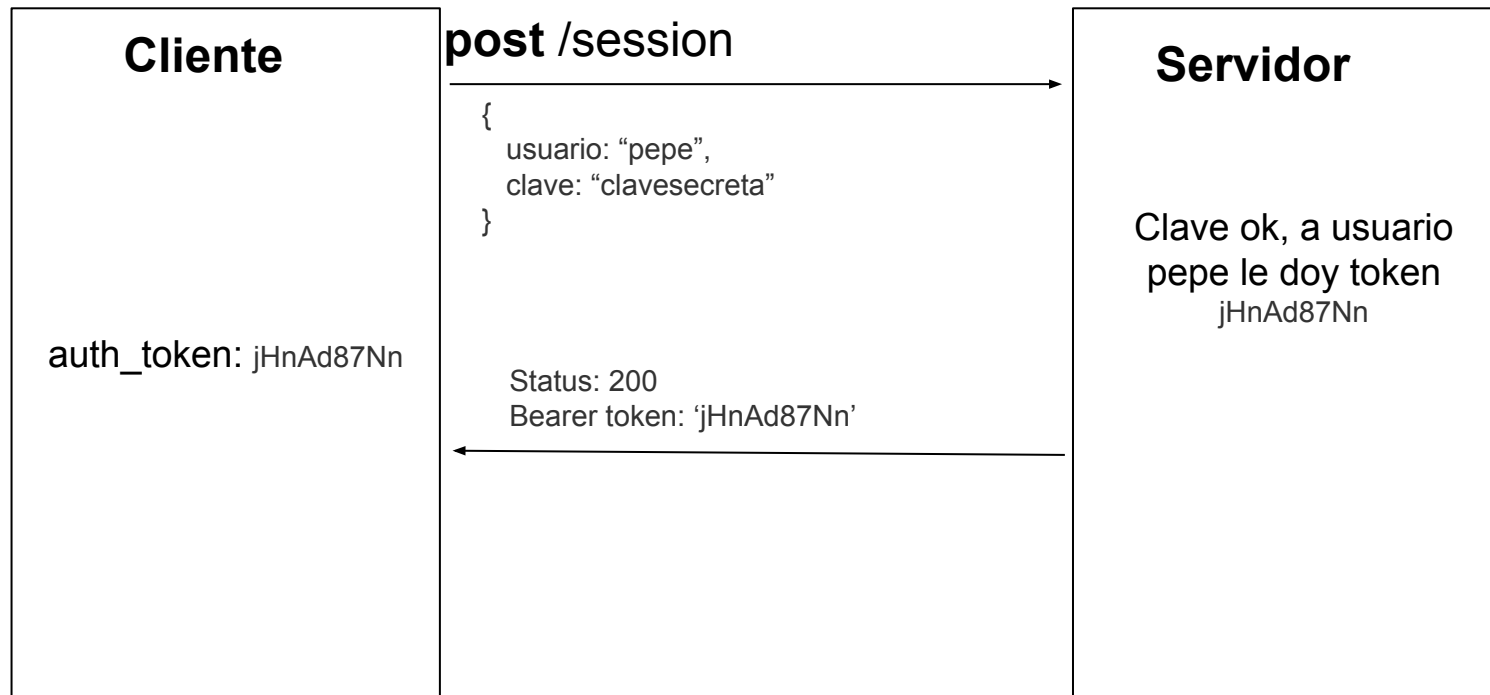
- En el *headers* podemos incluir un **token especial** de acceso que a la API le sirve para identificar y verificar si los permisos.



Ejemplo de cómo obtener el token



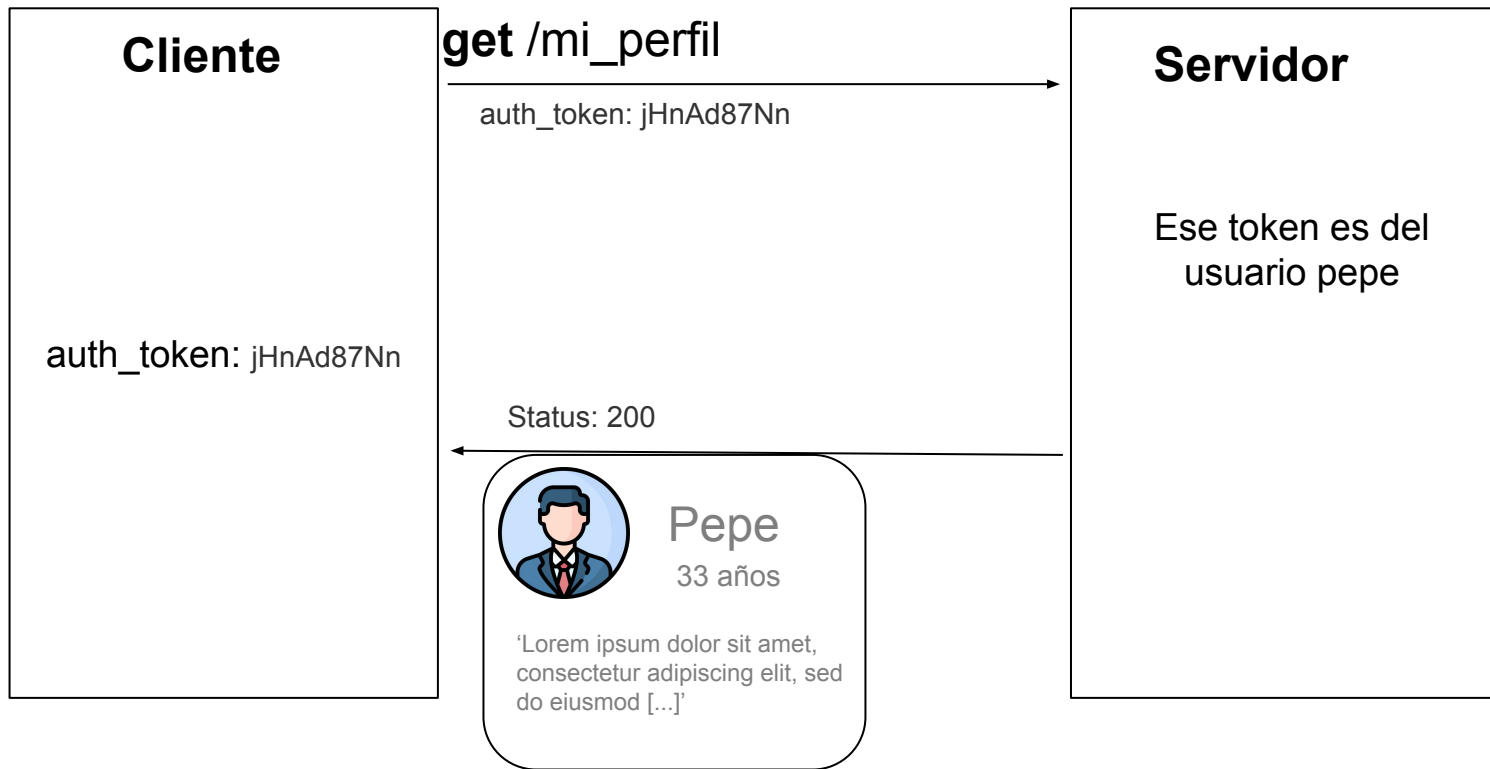
Ejemplo de cómo obtener el token



Ejemplo de cómo obtener el token



Ejemplo de cómo obtener el token



**¿Y esto en python
cómo se ve?**

requests

```
import requests
```

```
import json
```

```
response = requests.get(url)
```

```
print(response.status_code)
```

```
print(response.json())
```

```
datos_a_subir = { "nombre": "Pedro", "edad": 25 }
```

```
headers = { "Authorization": "token GjsdxYkdLSDandsGH" }
```

```
response = request.post(url, data=json.dumps(datos_a_subir), headers=headers)
```

Temas adicionales de interés

Temas adicionales de interés

- Hay tokens que incluso pueden almacenar información: [JWT.io](https://jwt.io)
- Existen muchos más códigos HTTP
 - [Códigos de estado de respuesta HTTP](#)
 - Existe su explicación gatuna: [HTTP Cats](#)
 - El [código 418](#) es cuando el servidor se rehúsa a preparar café porque es una tetera.
- Existen librerías para programar rápidamente una API, como [Flask](#) o [FastApi](#).

Programación Avanzada

IIC2233 2023-2

Hernán Valdivieso - Daniela Concha - Francisca Ibarra - Joaquín Tagle - Francisca Cattán

