

# UT4-PROGRAMACIÓN DE SERVICIOS EN RED

Programación de Servicios y Procesos

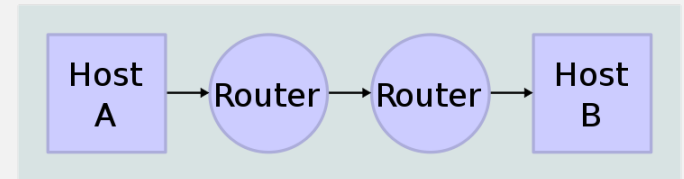
# OBJETIVOS

- Afianzar los conocimientos de TCP/IP
- Aprender a programar aplicaciones siguiendo protocolos estándar como FTP, SMTP o HTTP

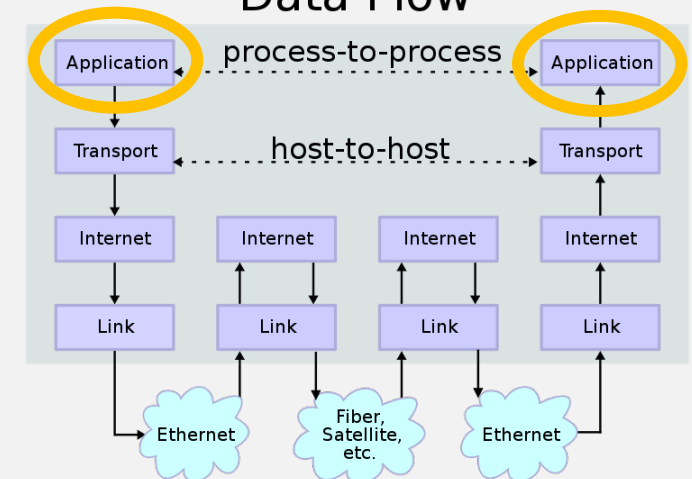
# RECORDAMOS MODELO TCP/IP

- Cuatro capas
- **Aplicación: HTTP, POP, SMTP, etc. Cada cual en su puerto**
- Transporte: comunicación entre equipos
  - TCP
    - Transmission Control Protocol
    - fiable, orientado a conexión, sobrecarga
    - Análogo a una llamada telefónica
  - UDP
    - User Datagram Protocol
    - no fiable, no orientado a conexión, veloz
    - Análogo al sistema postal
- Red: lleva paquetes a su destino
- Enlace: comunicaciones dentro de la propia red

## Network Topology



## Data Flow



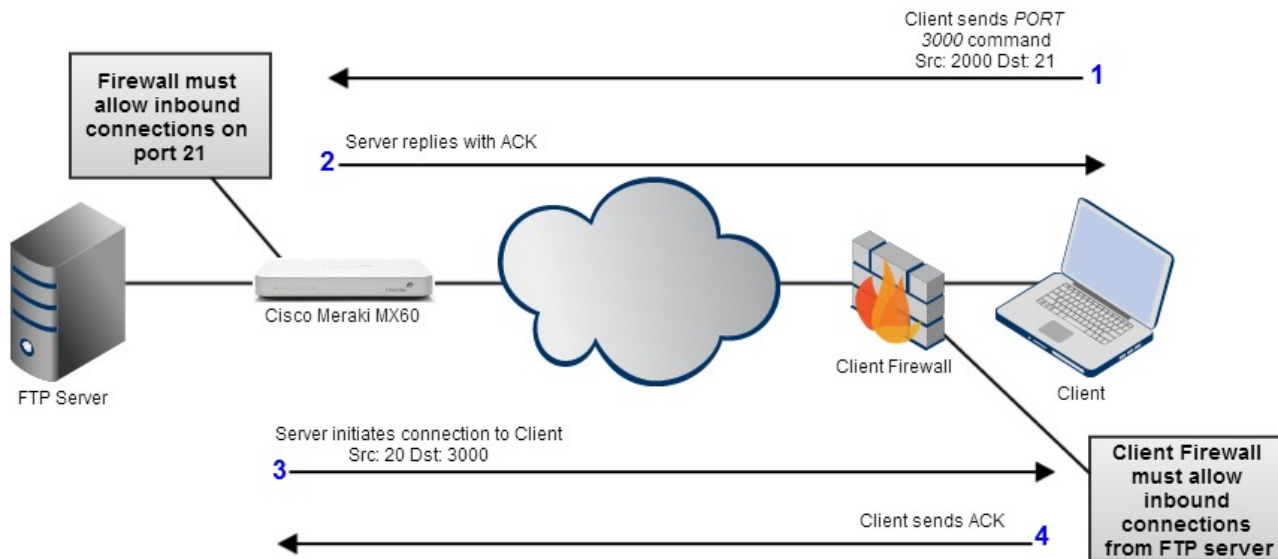
**FTP**

File Transfer Protocol

# FTP

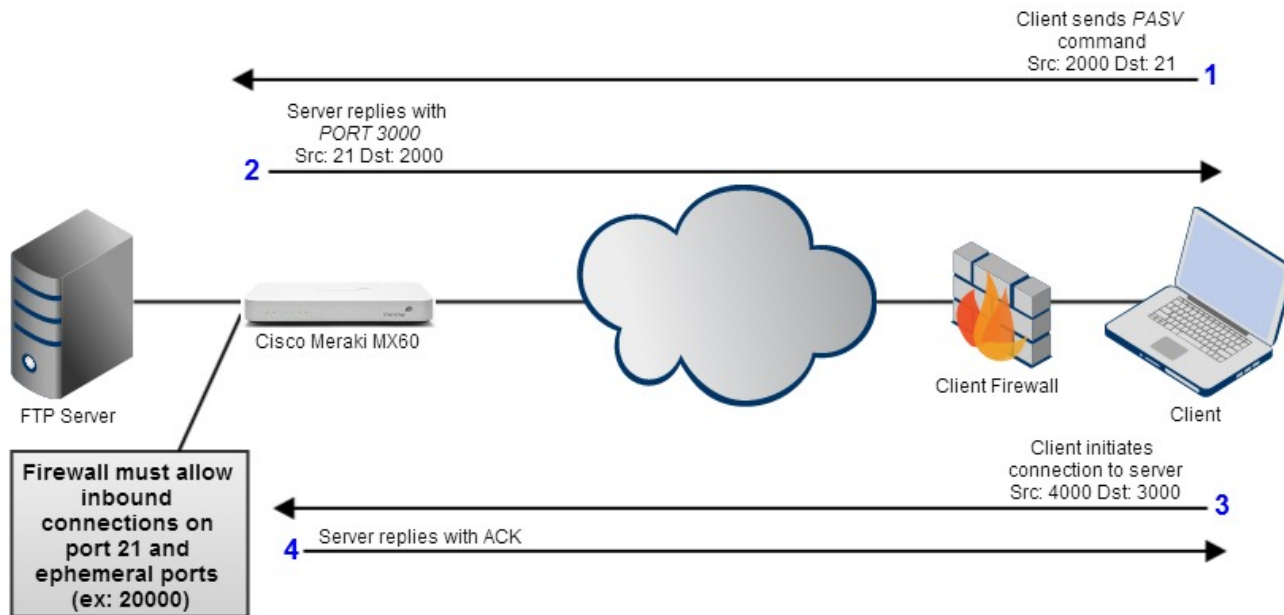
- File Transfer Protocol
- TCP
- Cliente/Servidor
- Acciones. Fichero: subir, bajar, borrar. Directorio: crear, cambiar
- Acceso: anónimo o autorizado
- Puertos
  - Para enviar comandos de control: puerto 21
  - Para el envío de datos... depende.

# MODO ACTIVO



- Activo quiere decir que el **cliente** elige el **puerto** de datos
- Las **conexiones de datos** se **inician** desde el **puerto 20** del **servidor**
- Requiere abrir el puerto en el Firewall del cliente → No se usa





# MODO PASIVO



- Pasivo quiere decir que el **servidor elige puerto de datos**
- Las **conexiones de datos se inician desde el cliente**
- Pasan sin problema a través del Firewall → Sí se usa

# CONFIGURACIÓN SERVIDOR PASIVO

1:1 NAT ⓘ

Name	<input type="text" value="Passive FTP Server"/>			 
Public IP	<input type="text" value="1.1.1.1"/>			
LAN IP	<input type="text" value="192.168.1.1"/>			
Uplink	<input type="text" value="Internet 1"/>			
Allowed inbound connections				
	<b>Protocol</b>	<b>Ports</b>	<b>Remote IPs</b>	<b>Actions</b>
	<input type="text" value="TCP"/>	<input type="text" value="21"/>	<input type="text" value="any"/>	
	<input type="text" value="TCP"/>	<input type="text" value="1024 - 65535"/>	<input type="text" value="any"/>	
	<a href="#">Allow more connections</a>			

[Add a 1:1 NAT mapping](#)



# COMANDOS FTP

- [ABOR](#) - **ab**ort a file transfer
- [CWD](#) - **ch**ange **w**orking **d**irectory
- [DELE](#) - **d**elete a remote file
- [LIST](#) - **l**ist remote files
- [MDTM](#) - return the **m**odification **t**ime of a file
- [MKD](#) - **m**ake a remote **d**irectory
- [NLST](#) - **n**ame **l**ist of remote directory
- [PASS](#) - send **p**assword
- [PASV](#) - enter **p**assive mode
- [PORT](#) - open a data **p**ort
- [PWD](#) - **p**rint **w**orking **d**irectory
- [QUIT](#) - terminate the connection
- [RETR](#) - **r**etrieve a remote file
- [RMD](#) - **r**emove a remote **d**irectory
- [RNFR](#) - **r**ename **f**rom
- [RNTQ](#) - **r**ename **t**o
- [SITE](#) - **s**ite-specific commands
- [SIZE](#) - return the **s**ize of a file
- [STOR](#) - **s**tore a file on the remote host
- [TYPE](#) - set transfer **t**ype
- [USER](#) - send **u**sername
- Fuente [SS64](#)

# PROGRAMEMOS

Programamos un cliente de FTP con [Apache Commons Net](#)

Servidor

[XAMPP](#)

[FileZillaServer](#)



# FTPCLIENT

- **CONECTAR**

- connect(String host)
- getReplyString()
- getReplyCode() (ver [RFC 959](#))
  - FTPReply.isPositiveCompletion(code)
- disconnect()

- **AUTENTICACIÓN**

- login(String user, String pwd)
- logout()

- **MODOS**

- enterLocalPassiveMode()
- enterLocalActiveMode()

- **NAVEGACIÓN**

- printWorkingDirectory()
- listFiles(), listFiles(String path), listNames()
- listDirectories(), listDirectories(String path)
- changeWorkingDirectory(String path),  
changeToParentDirectory()

# FTPCLIENT

- **FICHEROS**

- **setFileType**(int fileType) (ASCII\_FILE\_TYPE o BINARY\_FILE\_TYPE)
- boolean **storeFile**(String nombre, InputStream local)
- boolean **retrieveFile**(String name, OutputStream local)
- boolean **deleteFile**(String pathName)
- rename(old, new)

- **DIRECTORIOS**

- removeDirectory(String path),  
makeDirectory(String path)

Oh, vaya,  
Streams 😊

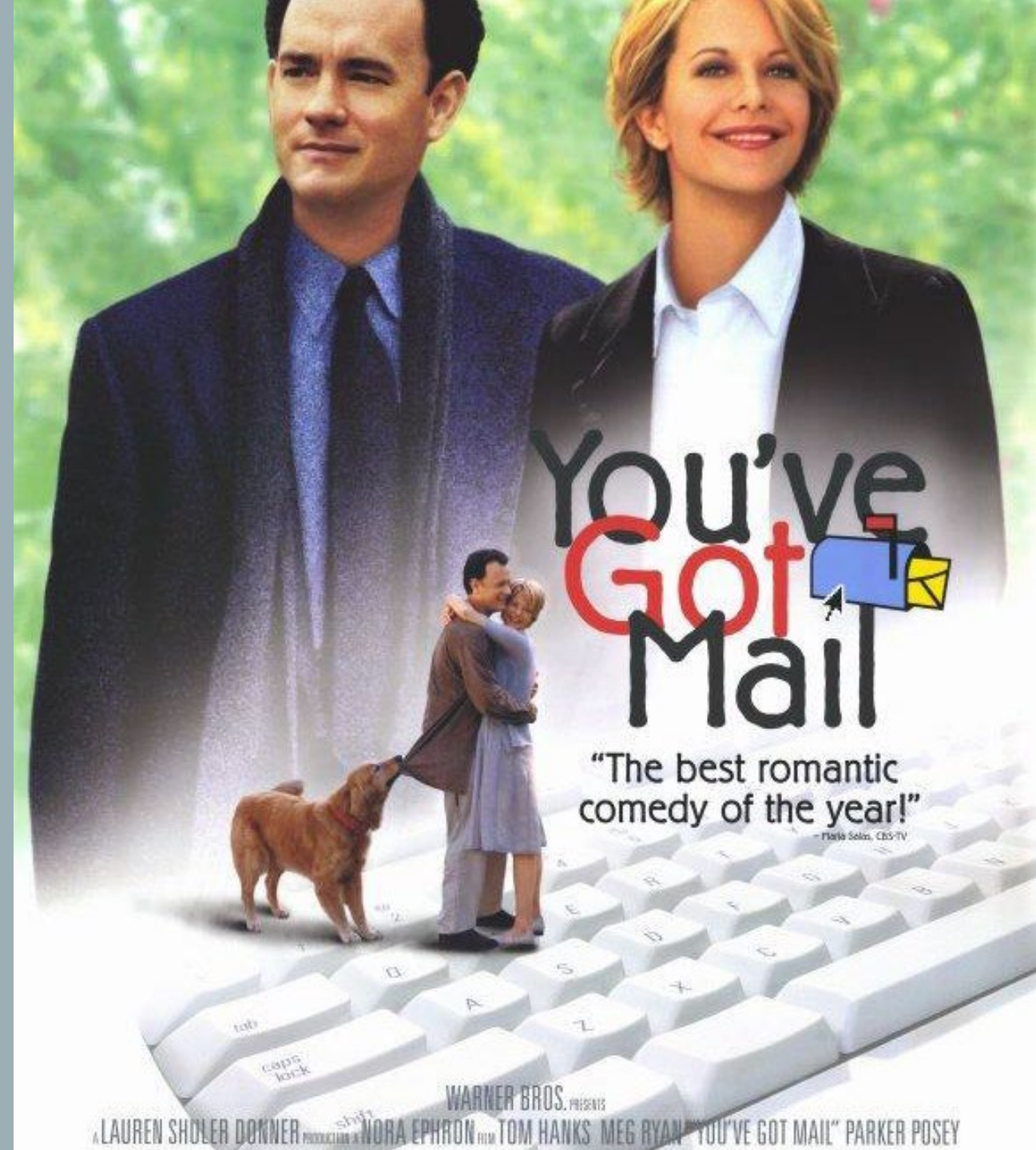
SMTP

Bueno, e IMAP y POP

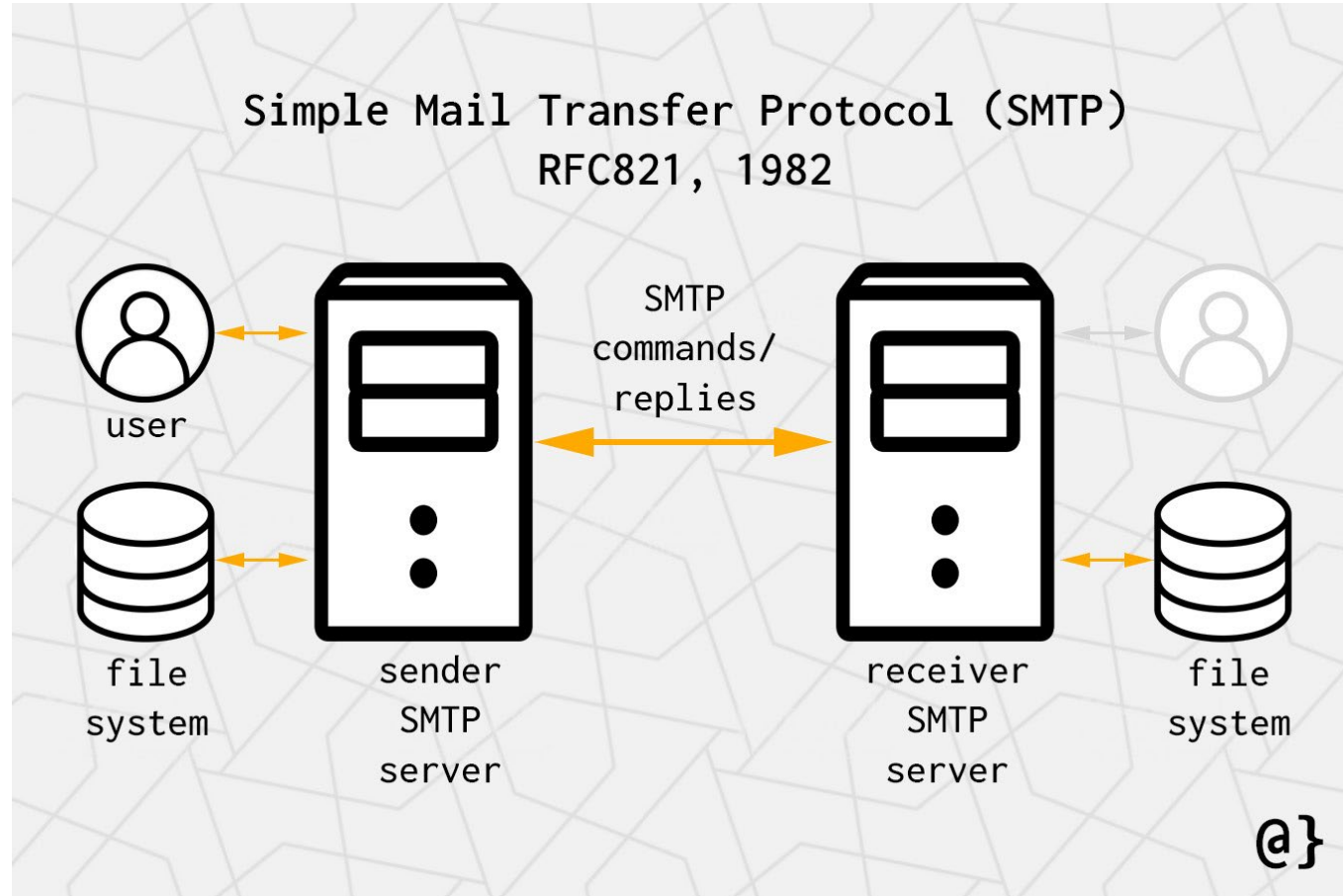
# CORREO ELECTRÓNICO

SU USO SIGUE SIENDO MASIVO EN  
APLICACIONES

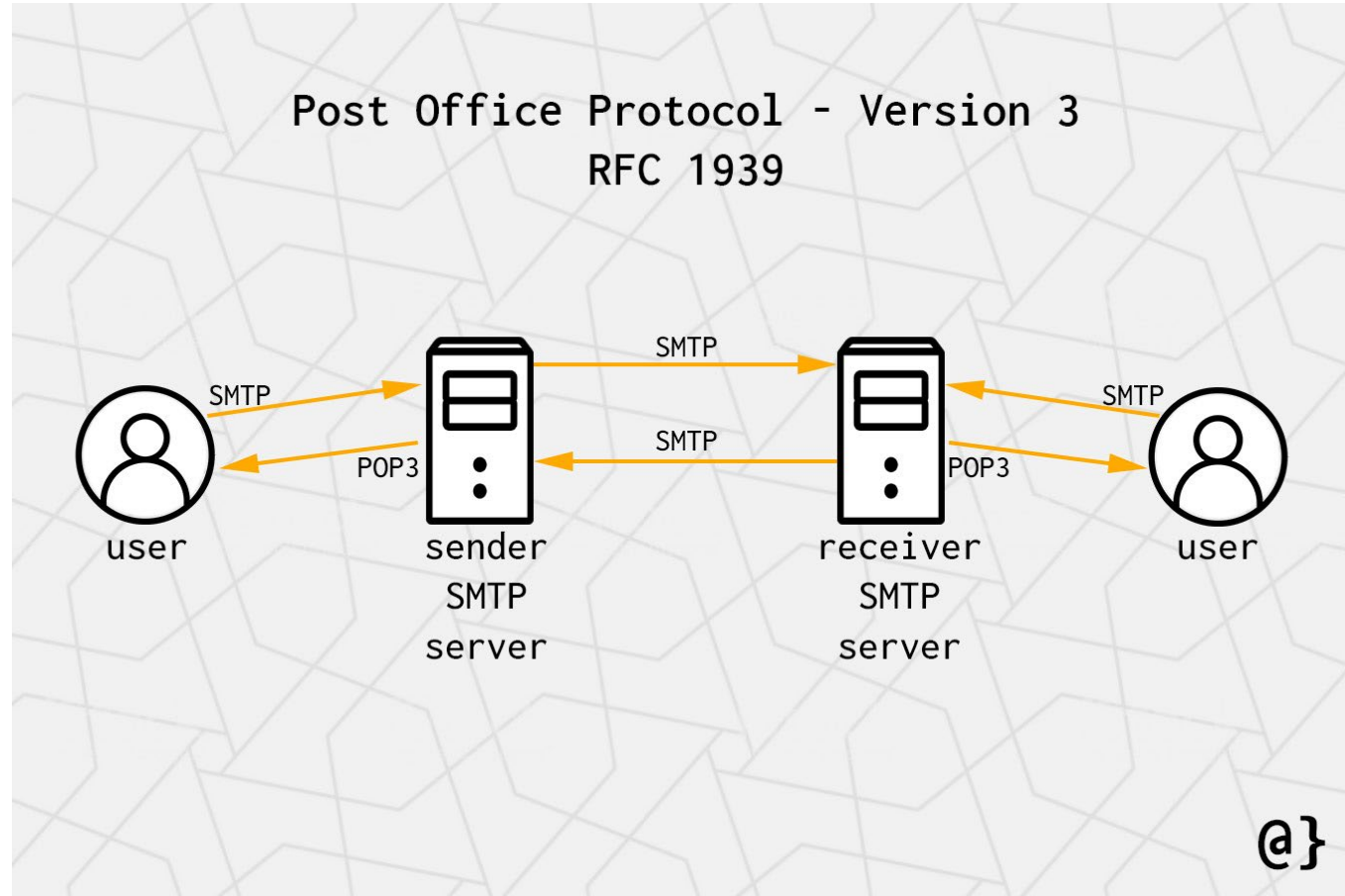
ENVIAR CORREO DESDE UNA  
APLICACIÓN EN SERVIDOR ES CLAVE



# SMTP

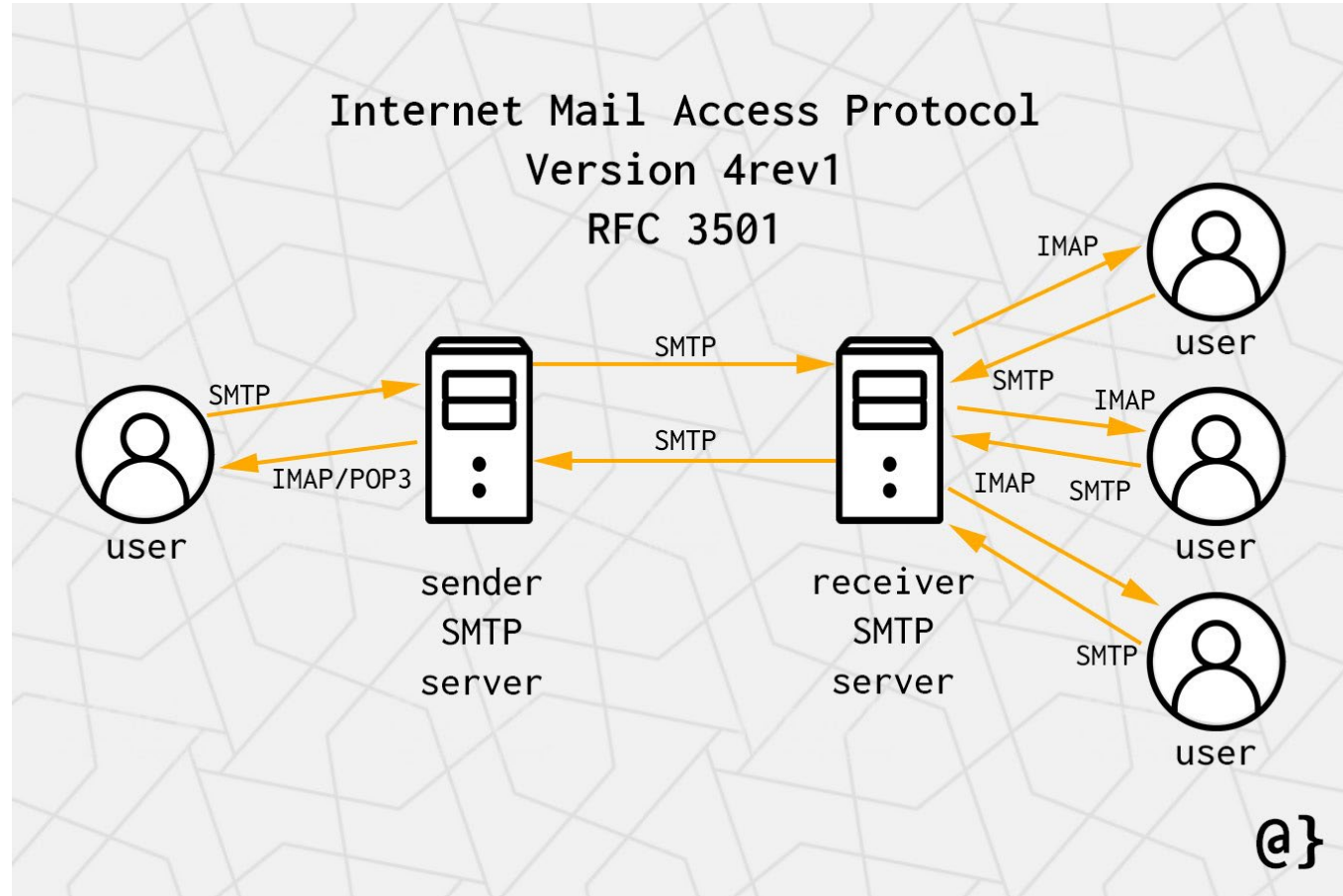


# POP3





# IMAP



# SMTP

- Simple Mail Transfer Protocol
- RFC [821](#)
- Puertos 25 (inseguro) 465 (TLS explícito) 587 (TLS oportunista o STARTTLS)

# COMANDOS SMTP

- **HELO**  
It's the first SMTP command: it starts the conversation identifying the sender server and is generally followed by its domain name.
- **EHLO**  
An alternative command to start the conversation, underlying that the server is using the Extended SMTP protocol.
- **MAIL FROM**  
With this SMTP command the operations begin: the sender states the source email address in the "From" field and actually starts the email transfer.
- **RCPT TO**  
It identifies the recipient of the email; if there are more than one, the command is simply repeated address by address.
- **SIZE**  
This SMTP command informs the remote server about the estimated size (in terms of bytes) of the attached email. It can also be used to report the maximum size of a message to be accepted by the server.
- **DATA**  
With the DATA command the email content begins to be transferred; it's generally followed by a 354 reply code given by the server, giving the permission to start the actual transmission.
- **VRFY**  
The server is asked to verify whether a particular email address or username actually exists.
- **TURN**  
This command is used to invert roles between the client and the server, without the need to run a new connection.
- **AUTH**  
With the AUTH command, the client authenticates itself to the server, giving its username and password. It's another layer of security to guarantee a proper transmission.
- **RSET**  
It communicates the server that the ongoing email transmission is going to be terminated, though the SMTP conversation won't be closed (like in the case of QUIT).
- **EXPN**  
This SMTP command asks for a confirmation about the identification of a mailing list.
- **HELP**  
It's a client's request for some information that can be useful for the a successful transfer of the email.
- **QUIT**  
It terminates the SMTP conversation.

Source	Destination	Protocol	Length	Info
193.146.123.99	192.168.1.35	SMTP	88	S: 220 smtp.educa.madrid.org ESMTP
192.168.1.35	193.146.123.99	SMTP	75	C: EHLO [192.168.1.35]
193.146.123.99	192.168.1.35	SMTP	153	S: 250-smtp.educa.madrid.org   PIPELINING   SIZE 15728640   AUTH LOGIN PLAIN   8BITMIME
192.168.1.35	193.146.123.99	SMTP	95	C: AUTH PLAIN [REDACTED]
193.146.123.99	192.168.1.35	SMTP	76	S: 235 nice to meet you
192.168.1.35	193.146.123.99	SMTP	105	C: MAIL FROM:<juan.agui2@educamadrid.org> SIZE=2045
193.146.123.99	192.168.1.35	SMTP	62	S: 250 ok
192.168.1.35	193.146.123.99	SMTP	90	C: RCPT TO:<da3d1a@iesclaradelrey.es>
193.146.123.99	192.168.1.35	SMTP	62	S: 250 ok
192.168.1.35	193.146.123.99	SMTP	60	C: DATA
193.146.123.99	192.168.1.35	SMTP	86	S: 354 go ahead punk, make my day
192.168.1.35	193.146.123.99	SMTP	1511	C: DATA fragment, 1503 bytes
192.168.1.35	193.146.123.99	SMTP/IMF	852	from: =?utf-8?q?Juan=20Ag=c3=bc=c3=ad?= <juan.agui2@educamadrid.org>, subject: Hola DA2D1A
193.146.123.99	192.168.1.35	SMTP	107	S: 250 ok 1670697179 qp 16412 by smtp.educa.madrid.org
192.168.1.35	193.146.123.99	SMTP	60	C: QUIT
193.146.123.99	192.168.1.35	SMTP	90	S: 221 smtp.educa.madrid.org Goodbye.

GO AHEAD PUNK, MAKE MY DAY

Más [detalles](#)

# NEGOCIACIÓN TLS

- 192.168.1.35 IP de cliente
- 193.146.123.99 IP de smtp.educa.madrid.org
- Puerto destino 465
- TLS forzado (implicit)

Source	Destination	Protocol	Length	Info
192.168.1.35	193.146.123.99	TLSv1.2	239	Client Hello
193.146.123.99	192.168.1.35	TLSv1.2	1506	Server Hello
193.146.123.99	192.168.1.35	TLSv1.2	1506	Certificate
193.146.123.99	192.168.1.35	TLSv1.2	326	Server Key Exchange, Server Hello Done
192.168.1.35	193.146.123.99	TLSv1.2	180	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
193.146.123.99	192.168.1.35	TLSv1.2	312	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
193.146.123.99	192.168.1.35	TLSv1.2	117	Application Data
192.168.1.35	193.146.123.99	TLSv1.2	104	Application Data
193.146.123.99	192.168.1.35	TLSv1.2	193	Application Data

# CLIENT HELLO

Cliente manda HELLO  
indicando versión de TLS y  
tipos de cifrado que soporta

```
▼ Transport Layer Security
  ▼ TLSv1.2 Record Layer: Handshake Protocol: Client Hello
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 180
    ▼ Handshake Protocol: Client Hello
      Handshake Type: Client Hello (1)
      Length: 176
      Version: TLS 1.2 (0x0303)
      > Random: 6394de40556aacc5aead7f4331ae3e7d58874d9f329c28d31ae9753e42beef86
      Session ID Length: 0
      Cipher Suites Length: 42
      ▼ Cipher Suites (21 suites)
        Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xc02c)
        Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
        Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
        Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
        Cipher Suite: TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 (0xc009f)
        Cipher Suite: TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 (0xc009e)
        Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 (0xc024)
        Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 (0xc023)
        Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (0xc028)
        Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (0xc027)
        Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA (0xc00a)
        Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA (0xc009)
        Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
        Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
        Cipher Suite: TLS_RSA_WITH_AES_256_GCM_SHA384 (0xc009d)
        Cipher Suite: TLS_RSA_WITH_AES_128_GCM_SHA256 (0xc009c)
        Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA256 (0xc003d)
        Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA256 (0xc003c)
        Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0xc0035)
        Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0xc002f)
        Cipher Suite: TLS_RSA_WITH_3DES_EDE_CBC_SHA (0xc000a)
      Compression Methods Length: 1
      > Compression Methods (1 method)
```

# SERVER HELLO

Servidor responde con su propio HELLO indicando el cifrado elegido

```
▼ Transport Layer Security
  ▼ TLSv1.2 Record Layer: Handshake Protocol: Client Hello
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 180
  ▼ Handshake Protocol: Client Hello
    Handshake Type: Client Hello (1)
    Length: 176
    Version: TLS 1.2 (0x0303)
    > Random: 6394de40556aacc5ae7f4331ae3e7d58874d9f329c28d31ae9753e42beef86
    Session ID Length: 0
    Cipher Suites Length: 42
  ▼ Cipher Suites (21 suites)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xc02c)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
    Cipher Suite: TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 (0xc009f)
    Cipher Suite: TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 (0xc009e)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 (0xc024)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 (0xc023)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (0xc028)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (0xc027)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA (0xc000a)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA (0xc0009)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc0014)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc0013)
    Cipher Suite: TLS_RSA_WITH_AES_256_GCM_SHA384 (0xc009d)
    Cipher Suite: TLS_RSA_WITH_AES_128_GCM_SHA256 (0xc009c)
    Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA256 (0xc003d)
    Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA256 (0xc003c)
    Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0xc0035)
    Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0xc002f)
    Cipher Suite: TLS_RSA_WITH_3DES_EDE_CBC_SHA (0xc000a)
    Compression Methods Length: 1
    > Compression Methods (1 method)
```

# SERVER CERTIFICATE

Servidor responde con su  
propio HELLO indicando el  
cifrado elegido

```
✓ Certificate: 308206a33082058ba003020102020c627c191079a84feed6ce2c0d300d06092a864886f7... (id-at
  ✓ signedCertificate
    version: v3 (2)
    serialNumber: 0x627c191079a84feed6ce2c0d
    > signature (sha256WithRSAEncryption)
    > issuer: rdnSequence (0)
    > validity
    ✓ subject: rdnSequence (0)
      ✓ rdnSequence: 6 items (id-at-commonName=*.educa.madrid.org,id-at-organizationName=Agenc
        > RDNSequen item: 1 item (id-at-countryName=ES)
        > RDNSequen item: 1 item (id-at-stateOrProvinceName=Madrid)
        > RDNSequen item: 1 item (id-at-localityName=Madrid)
        > RDNSequen item: 1 item (id-at-organizationalUnitName=Madrid Digital)
        > RDNSequen item: 1 item (id-at-organizationName=Agencia para la Administración Digi
        > RDNSequen item: 1 item (id-at-commonName=*.educa.madrid.org)
      ✓ subjectPublicKeyInfo
        > algorithm (rsaEncryption)
        > subjectPublicKey: 3082010a0282010100b1b509aa08c3fbb77a9fae69476e6707d7fa8c8a1aba737d0e
        > extensions: 9 items
    > algorithmIdentifier (sha256WithRSAEncryption)
    Padding: 0
```



# TLS OPORTUNISTA

- 192.168.1.35 IP de cliente
- 193.146.123.99 IP de smtp.educa.madrid.org
- Puerto destino **587**
- TLS oportunista (explicit)

Source	Destination	Protocol	Length	Info
193.146.123.99	192.168.1.35	SMTP	98	S: 220 smtp.educa.madrid.org ESMTP + stunnel
192.168.1.35	193.146.123.99	SMTP	75	C: EHLO [192.168.1.35]
193.146.123.99	192.168.1.35	SMTP	87	S: 250-EHLO [192.168.1.35] Welcome
193.146.123.99	192.168.1.35	SMTP	68	S: 250 STARTTLS
192.168.1.35	193.146.123.99	SMTP	64	C: STARTTLS
193.146.123.99	192.168.1.35	SMTP	68	S: 220 Go ahead
192.168.1.35	193.146.123.99	TLSv1.2	239	Client Hello
193.146.123.99	192.168.1.35	TLSv1.2	1506	Server Hello
193.146.123.99	192.168.1.35	TLSv1.2	1506	Certificate
193.146.123.99	192.168.1.35	TLSv1.2	326	Server Key Exchange, Server Hello Done
192.168.1.35	193.146.123.99	TLSv1.2	180	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
193.146.123.99	192.168.1.35	TLSv1.2	312	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
192.168.1.35	193.146.123.99	TLSv1.2	104	Application Data
193.146.123.99	192.168.1.35	TLSv1.2	193	Application Data

## SERVIDORES EDUCA MADRID

PROTOCOLO	SERVIDOR CORREO ENTRANTE	SERVIDOR CORREO SALIENTE
<b>POP3</b>	pop.educa.madrid.org	smtp01.educa.madrid.org (requiere autenticación)
<b>IMAP</b>	imap.educa.madrid.org	smtp01.educa.madrid.org (requiere autenticación)

## PUERTOS EDUCA MADRID

SEGURIDAD	TIPO DE CUENTA / PROTOCOLO		CORREO SALIENTE
	POP	IMAP	
Seguro SSL	995	993	465
Seguro TLS	////////	////////	587

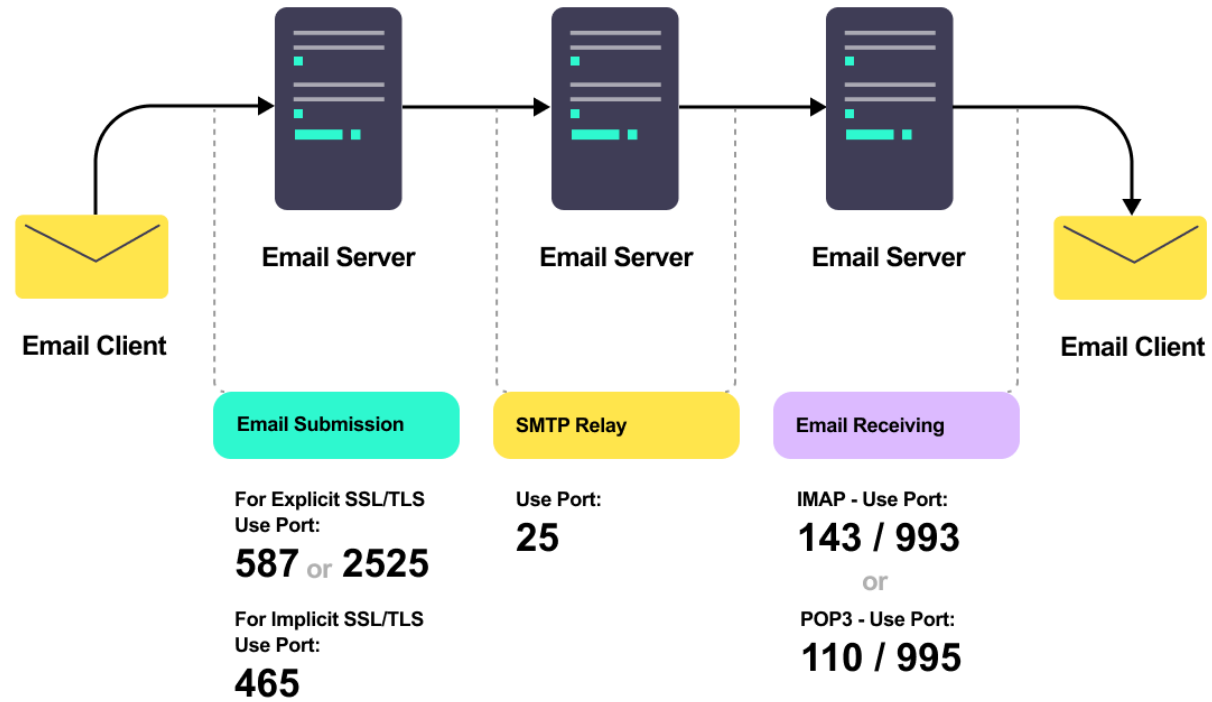
# Y GMAIL

Incoming connections to the IMAP server at `imap.gmail.com:993` and the POP server at `pop.gmail.com:995` require SSL.

The outgoing SMTP server, `smtp.gmail.com`, requires TLS.

Use port 465, or port 587 if your client begins with plain text before issuing the STARTTLS command. ([Gmail for developers](#))

# PUERTOS



# ¡PROGRAMEMOS!

- Utilizamos [Apache Commons Email](#)
- Maven:  
org.apache.commons:commons-email:1.5
- Escribe un programa que, utilizando la clase SimpleEmail pregunte interactivamente por
  - Usuario
  - Contraseña
  - Dirección de destinatario
  - Dirección de remitente
  - Asunto
  - Cuerpo del mensaje
- Y lo envíe utilizando la cuenta de EducaMadrid con TLS Explícito

## Y UN POQUITO MÁS

- Utiliza la clase `EmailAttachment` y `MultiPartEmail` para enviar un correo con una imagen adjunta desde tu disco local
- Utiliza la clase `HtmlEmail` para enviar un correo formateado en HTML. Recuerda que algunos clientes no soportan HTML por lo que tienes que añadir el texto en plano con `setTextMsg`
- Utiliza la clase `ImageHtmlEmail` para enviar un mensaje con una imagen embebida desde tu disco local
- Pssst, mira [aquí](#)

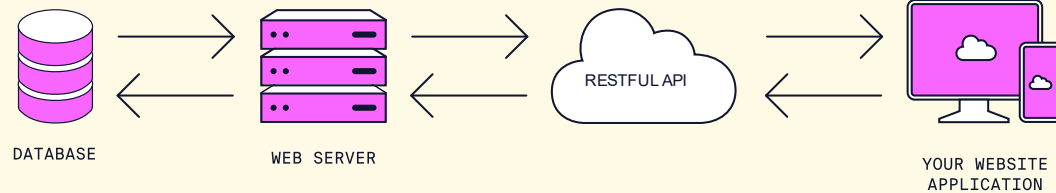
# CONSUMIR REST API

¡Por fin!



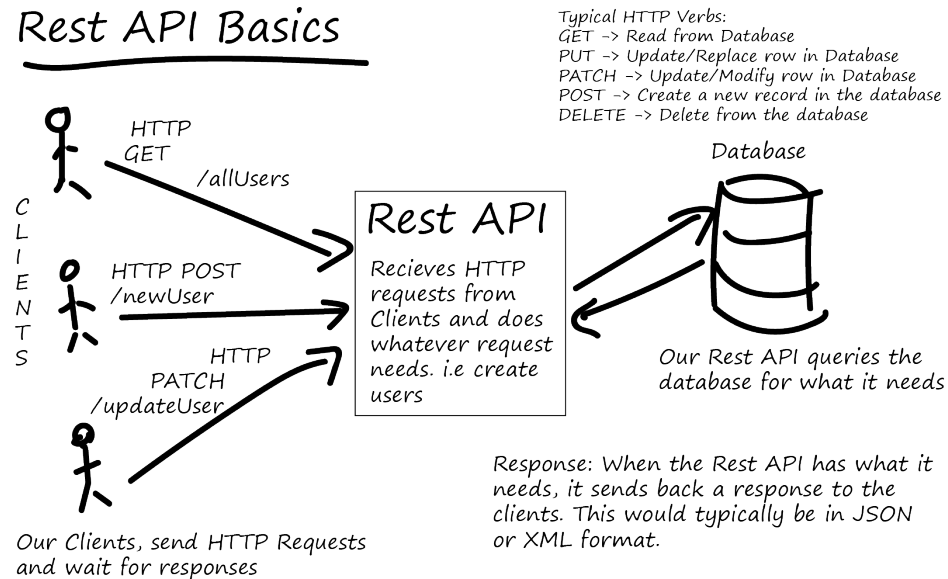
# REST

## What is Rest API?



- REpresentational State Transfer
- Es un estilo de arquitectura para exponer servicios en la red
- No es un protocolo
- Pero utiliza el protocolo HTTP
- Cuando un cliente realiza una petición a una *API RESTful*, recibe una representación del estado de un recurso, generalmente en formato JSON

## Rest API Basics



# RESTFUL API

- Una arquitectura **cliente-servidor** compuesta por clientes, servidores y recursos, con solicitudes gestionadas a través de **HTTP**.
- Comunicación cliente-servidor **sin estado**, lo que significa que no se almacena información del cliente entre las solicitudes de obtención y cada solicitud es independiente y no está conectada.
- Datos almacenables en **caché** que agilizan las interacciones cliente-servidor.
- Una interfaz uniforme entre los componentes para que la información se transfiera en una forma estándar

# PETICIONES REST

REST requiere que un cliente realice una solicitud al servidor para recuperar o modificar datos en el servidor.

Una solicitud generalmente consiste en:

1. un verbo HTTP, que define qué tipo de operación realizar
2. un encabezado, que permite al cliente pasar información sobre la solicitud
3. una ruta a un recurso
4. un cuerpo de mensaje opcional que contiene datos



# VERBOS HTTP

Hay 4 verbos HTTP básicos que usamos en solicitudes para interactuar con recursos en un sistema REST:

- GET: recupera un recurso específico (por id) o una colección de recursos
- POST: crea un nuevo recurso
- PUT: actualiza un recurso específico (por id)
- DELETE: eliminar un recurso específico por id.



## ENCABEZADOS

- En el encabezado de la solicitud, el cliente envía el tipo de contenido que puede recibir del servidor.
- Los tipos MIME, que se utilizan para especificar los tipos de contenido en el campo *Accept*, constan de un tipo y un subtipo. Están separados por una barra inclinada (/).
- En APIS REST el tipo más común es `application/json` aunque también puede haber `application/xml`
- Si nuestro cliente acepta esos tipos de respuesta lo especifica con la cabecera *Accept*:  
`application/json, application/xml`



# RUTAS

- Las solicitudes deben contener una ruta a un recurso en el que se debe realizar la operación.
- En las API RESTful, las rutas deben diseñarse para ayudar al cliente a saber qué está pasando. Convencionalmente, la primera parte de la ruta debería ser la forma plural del recurso. Esto hace que las rutas anidadas sean fáciles de leer y comprender.
- Una ruta como `fashionboutique.com/customers/223/orders/12` es clara en lo que apunta, incluso si nunca antes ha visto esta ruta específica, porque es jerárquica y descriptiva.
- Podemos ver que estamos accediendo al pedido con id 12 para el cliente con id 223



# RESPUESTAS

- En los casos en que el servidor envía una carga útil de datos al cliente, el servidor debe incluir un tipo de contenido en el encabezado de la respuesta.
- Este campo de encabezado Content-Type alerta al cliente sobre el tipo de datos que está enviando en el cuerpo de la respuesta.
- Estos tipos de contenido son tipos MIME, tal como están en el campo de Accept del encabezado de la solicitud.
- El Content-Type que el servidor devuelve en la respuesta debe ser una de las opciones que el cliente especificó en el campo de Accept de la solicitud.

HTTP/1.1 200 (OK)

Content-Type: application/json

# CÓDIGOS DE RESPUESTA

Status code	Meaning
200 (OK)	This is the standard response for successful HTTP requests.
201 (CREATED)	This is the standard response for an HTTP request that resulted in an item being successfully created.
204 (NO CONTENT)	This is the standard response for successful HTTP requests, where nothing is being returned in the response body.
400 (BAD REQUEST)	The request cannot be processed because of bad request syntax, excessive size, or another client error.
403 (FORBIDDEN)	The client does not have permission to access this resource.
404 (NOT FOUND)	The resource could not be found at this time. It is possible it was deleted, or does not exist yet.
500 (INTERNAL SERVER ERROR)	The generic answer for an unexpected failure if there is no more specific information available.



PROBEMOS



## FUENTES



[What is a REST API?](#)



[What is a RESTful API?](#)



[What is REST?](#)

# IMPLEMENTAR REST API

Con Spring Boot

# CONTROLADOR REST

- Las clases que contienen los métodos de la API REST
- Anotados con `@RestController`
- El valor de retorno de los métodos de estas clases se convierte automáticamente a JSON con la biblioteca Jackson

# RECURSOS

- Las clases que representan los recursos que se consultan, crean, modifican o borran a través de la API REST
- Podemos utilizar la anotación `@Data` de Lombok para que genere automáticamente getters/setters, constructor, equals, hashCode y toString.

```
import lombok.Data;
```

```
@Data
```

```
public class Country {
```

```
    private String code;  
    private String name;  
    private String currency;  
    private String capital;  
    private int population;
```

```
}
```

# RUTADO

- Cómo asociamos un método en un controlador a una URI en la API REST
- @RequestMapping de tipo general
- @GetMapping
- @PostMapping
- @PutMapping
- @DeleteMapping

## PASAR DATOS A LA API

- Identificadores: se indican entre llaves en la anotación de Mapping
  - Se recuperan mediante `@PathVariable` en el método del controlador
- En métodos que reciben datos
  - Se pasan anotados con `@RequestBody` en el método del controlador
  - Recuerda poner el Content-type en la petición

# RESPUESTAS

- `ResponseEntity<T>` representa una respuesta REST, en la que se puede especificar el código de error
- `ResponseEntity.of(Optional<T>)` devuelve
  - 200 y la entidad o
  - 404 si recibe un `Optional.Empty`
- `ResponseEntity<T>().status(HttpStatus.code).body(entidad)` permite especificar el código y la entidad.
  - Para POST, código 201 Created
  - Para DELETE, código 204 No content



# PROGRAMEMOS

1. Ejemplo sencillo con [Countries](#)
2. Ejemplo con base de datos Scott