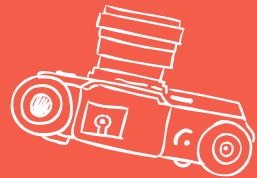


DESARROLLO DE SOFTWARE



AGENDA PARA HOY

- ✓ Arquitecturas un poco de historia.
 - ❖ Arquitectura Monolítica
 - ❖ Arquitectura de Microservicios
- ✓ API
 - ❖ ¿Qué es y qué características tiene?
 - ❖ Rest (Definición, conceptos, diseño, documentación)
- ✓ Implementación Primer API
 - ❖ Express JS – Hola Mundo!

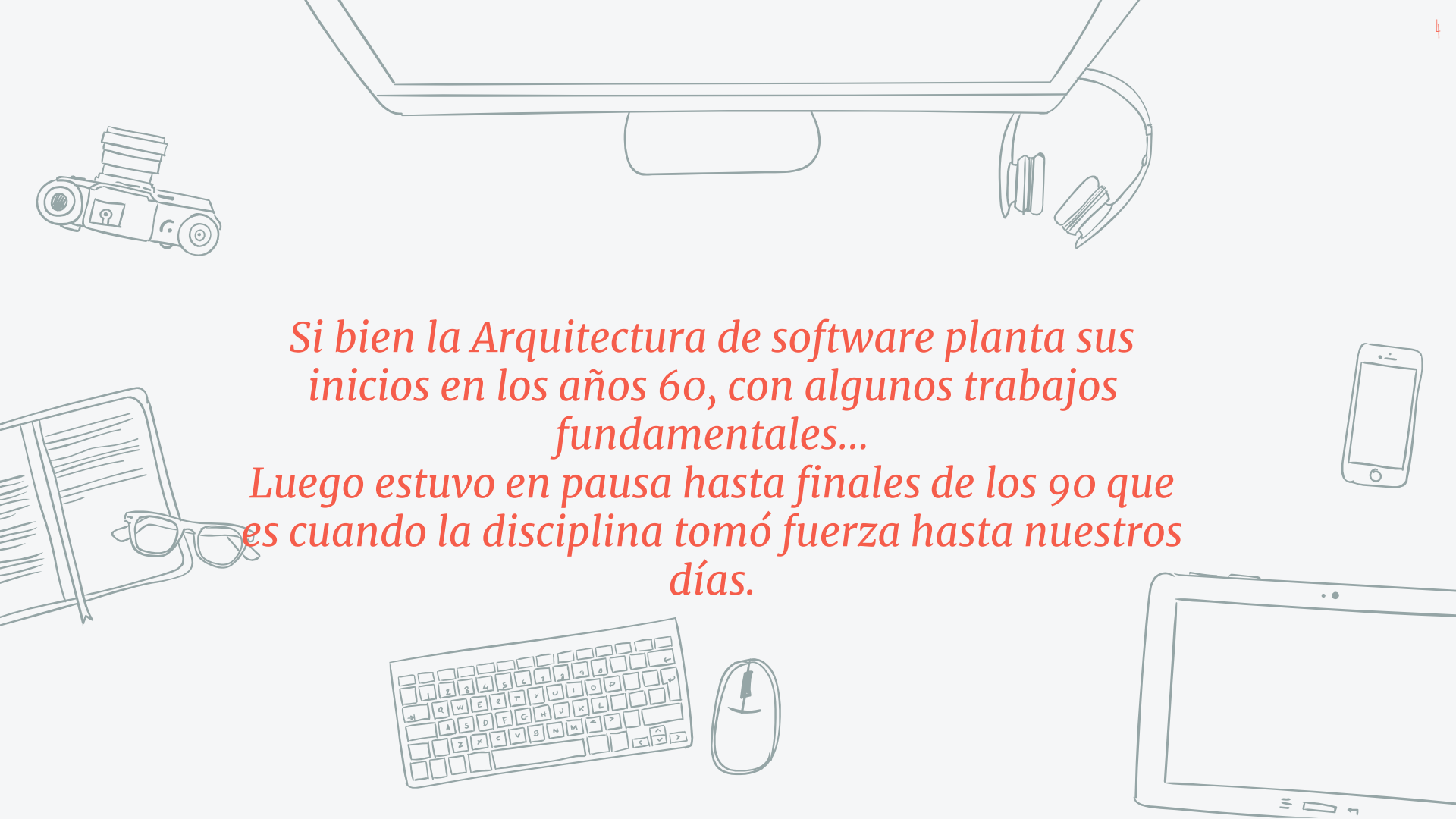


1.

ARQUITECTURAS

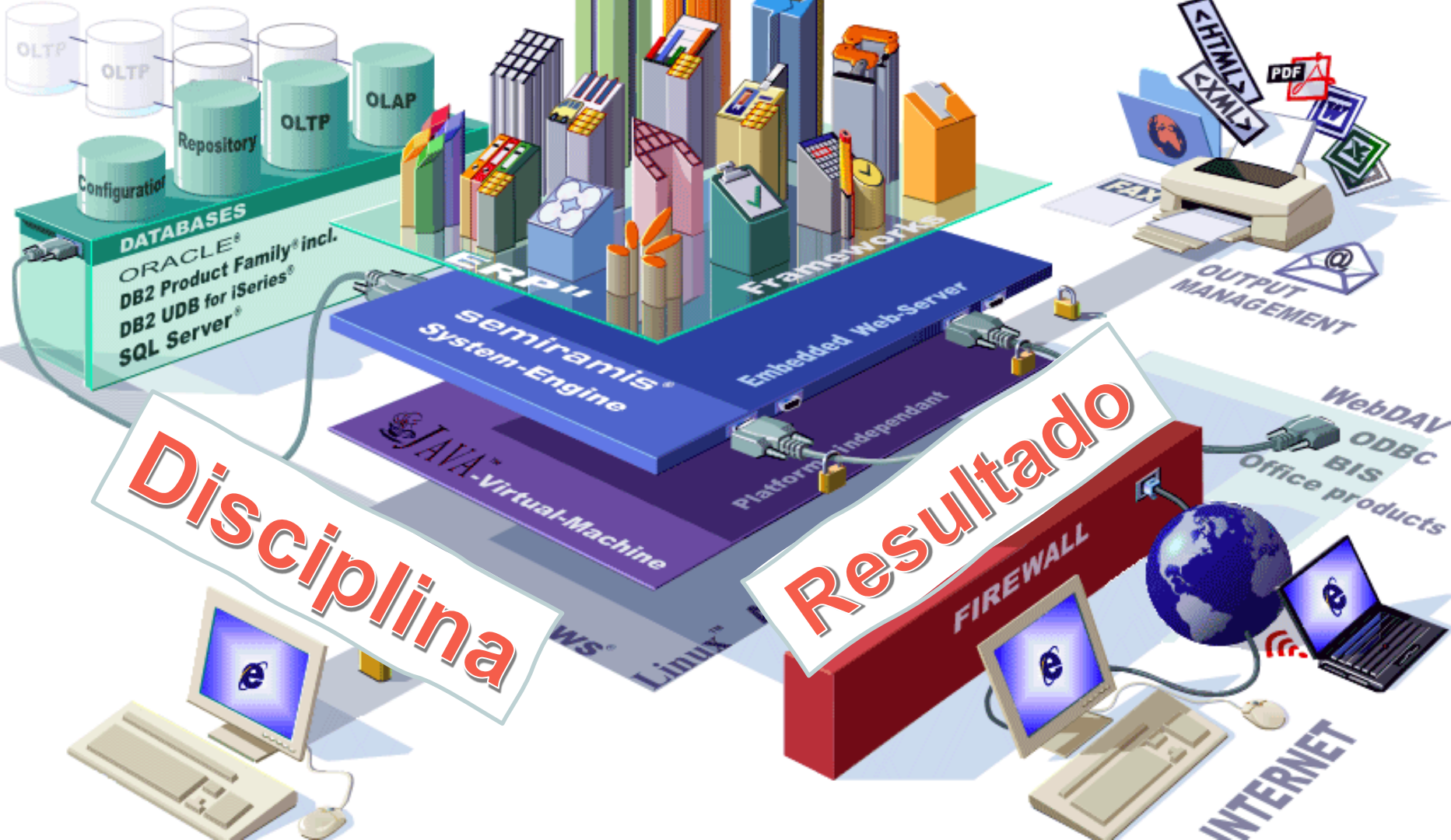
Un Poco de Historia.





*Si bien la Arquitectura de software planta sus
inicios en los años 60, con algunos trabajos
fundamentales...*

*Luego estuvo en pausa hasta finales de los 90 que
es cuando la disciplina tomó fuerza hasta nuestros
días.*



HISTORIA ANTIGUA

Sistemas centralizados: La ventanilla con EL de Sistemas



1

Mejoran las comunicaciones y las redes se vuelven indispensables

3

A finales de los 90 y principios de los 2000 se desarrolla el concepto actual de la Arquitectura de software

5

Terminales bobas conectadas a los mainframes



2

Aplicaciones en Capas Cliente – Servidor

4

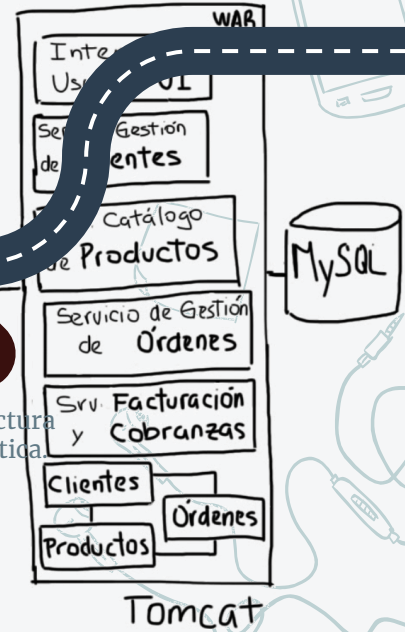


Browser - Apache

6

Arquitectura Monolítica

Arquitectura monolítica



MIREMOS UN POCO LOS PATRONES...

Arquitectura Monolítica

Despliegue centralizado.

Mayor acoplamiento y cohesión.

Un cambio implica un redespliegue de todo el sistema.

Patrones orientados a capas con algunas particularidades.



Arquitectura de Microservicios

Despliegue distribuido.

Menor acoplamiento y mayor cohesión.

Un cambio no implica desplegar nuevamente todo el sistema.

Patrones orientados a microservicios con dependencias.



ARQUITECTURA DE REFERENCIA

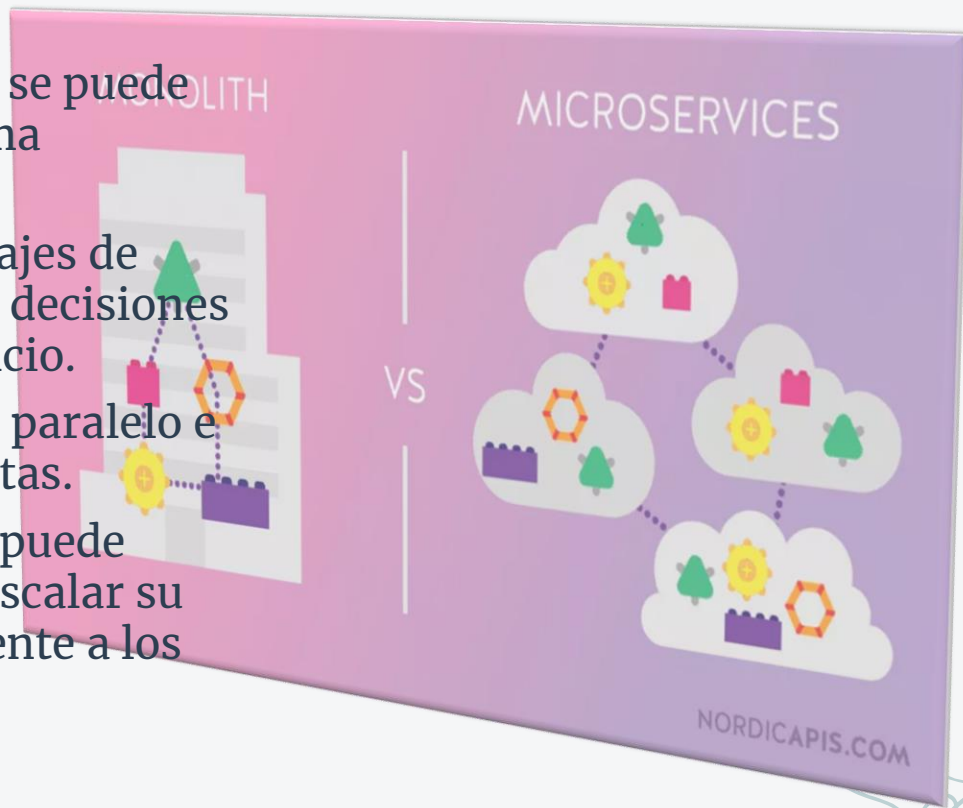


Una estructura de componentes a construir
para implementar los requerimientos.



ARQUITECTURA DE MICROSERVICIOS

- **Agilidad:** cada microservicio se puede diseñar y desarrollar de forma independiente.
- **Diversidad:** diferentes lenguajes de programación, tecnologías y decisiones de diseño en cada microservicio.
- **Productividad:** desarrollo en paralelo e independencia de herramientas.
- **Independencia:** cada equipo puede desarrollar, implementar y escalar su servicio de forma independiente a los demás.



2. APIs

Una breve discusión.



¿QUÉ ES UN API? – CONCEPTOS

- Un API (Application Programming Interface) define un conjunto reglas y protocolos para comunicar componentes de software.
- Un API define:
 - ❑ Estructuras de los datos que se intercambian entre los programas
 - ❑ Acciones que los programas están preparados para realizar.
 - ❑ Estándar de comunicación a utilizar.

IMPLEMENTANDO MICROSERVICIOS MEDIANTE APIS

Comunicación

Los servicios de una Arquitectura de Microservicios se comunican entre sí a través de APIs.

Esto permite a los servicios interactuar de forma segura y estandarizada, lo que facilita el intercambio de datos y la coordinación de tareas

Escalabilidad y Flexibilidad

Al dividir cada servicio en una aplicación diferente cada servicio puede escalar de forma independiente, lo que permite aplicaciones más flexibles y escalables.

Los APIs permiten que los servicios se comuniquen de manera eficiente aun cuando escalan horizontalmente.

Separación de Responsabilidad

Cada servicio en una arquitectura de microservicios tiene una responsabilidad específica.

Los APIs definen las interfaces de cada servicio de forma que cada equipo puede trabajar de manera independiente en cada servicio con la seguridad que la integración será exitosa.

En resumen: los APIs son fundamentales para la implementación de una arquitectura de microservicios.

Proporcionan una forma estandarizada de comunicación entre servicios, permiten mayor escalabilidad y flexibilidad y promueven la separación de responsabilidades.



REST

Representational State Transportation.

¿QUÉ ES REST (REPRESENTATIONAL STATE TRANSFER)?

- REST es una de las muchas formas en las que podemos implementar un API.
 - ☐ Quizás la forma más popular y ampliamente difundida.
 - ☐ Propone la implementación de APIs como servicios consumibles a través de HTTP (o servicios WEB)
- Conlleva la creación natural de servicios web escalables, flexibles, que permiten la interoperabilidad e integración de los sistemas.

¿CÓMO FUNCIONA?



Nota: REST API define el protocolo, pero entonces... ¿Qué programamos?.

Esencialmente vamos a programar la funcionalidad que recibe el Request, lo procesa y genera el Response que es retornado al cliente.

PRINCIPIOS REST

- **Cliente / Servidor.**
 - Comunicados a través del protocolo HTTP.
- **Stateless (Sin estado).**
 - Fundamental para asegurar la escalabilidad.
- **Cache**
 - No ejecutar cada vez lo mismo, ante una petición idéntica corresponderá la misma respuesta.
- **Arquitectura de capas en cada servicio.**
- **Interfaz uniforme.**
 - Aparece el concepto de URI y su importancia en REST



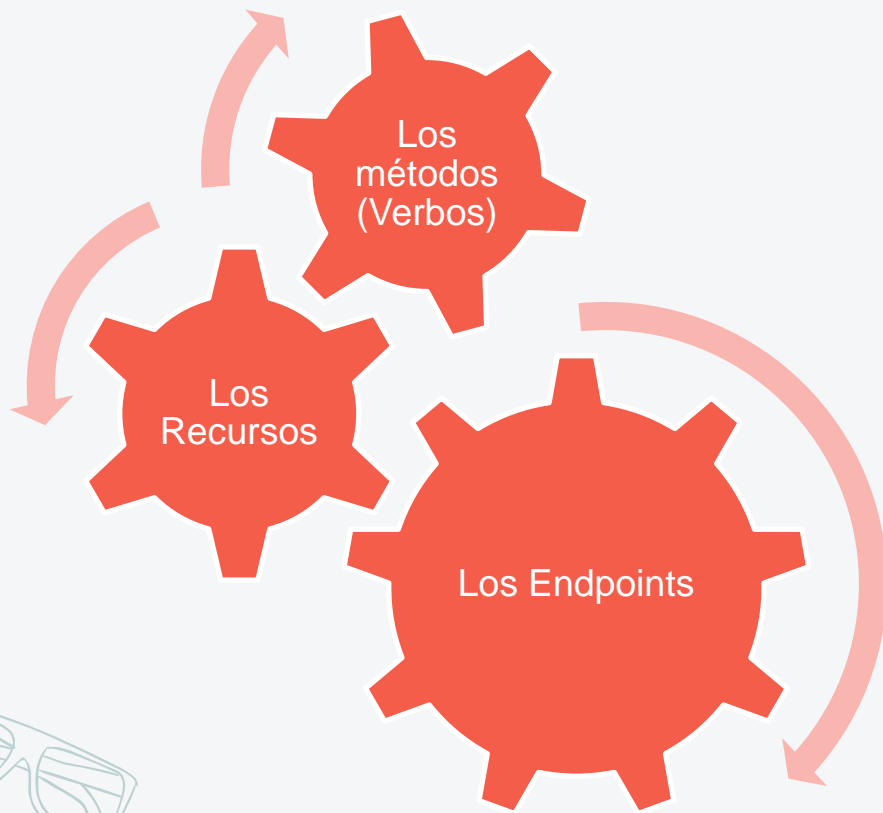
DISEÑO DE LA API

Cuando entramos en el terreno del Diseño de la API
entran en juego dos aspectos fundamentales.

La identificación de los recursos y La documentación.



ANATOMÍA DE UNA REST API



IDENTIFICACIÓN DE RECURSOS (URI)

- Un recurso es cualquier porción de datos que pueda ser identificada de manera unívoca.
 - Es decir, un objeto, una imagen, un video o incluso una API misma.
- Aparecen los conceptos de URI, URL y URN.
 - URI(Identificador de Recurso Uniforme): cadena de caracteres que identifica de manera única un recurso.
 - URL(Localizador de Recurso Uniforme): es un tipo de uri que se utiliza para especificar la ubicación de un recurso en la web, e incluye el protocolo utilizado.
 - URN(Nombre de Recurso Uniforme): es otro tipo de URI que se utiliza para identificar unívocamente un recurso pero que depende de un servicio de resolución de nombres para su ubicación real.

ARMADO DE URI – ESTRUCTURAR LA API

➤ Cómo armar una URI o mejor la estructura de URIs

- Usar sustantivos en plural
- Cuidar la granularidad

➤ Los Métodos o Verbos HTTP

- GET obtener lista de recursos o un recurso específico
- PUT crear o actualizar recursos
- PATCH específicamente actualizar un recurso
- POST crear un recurso donde el id se genera automáticamente
- DELETE eliminar un recurso
- HEAD verificar si un recurso existe o no

FINALMENTE, REST VS REST FULL

REST

Arquitectura basada en HTTP.

Determina el acceso a los recursos a través de URIs.

Es cliente/servidor, stateless, etc.



RESTful

Implementación de Servicios basada en REST.

Uso de los verbos de HTTP estándar como acciones.

Uso de códigos de estado HTTP como respuestas.

DISEÑO DE NUESTRA API

➤ Categoría de Recursos

- Colección

GET <http://mi-dominio.edu/api/categorias>

- Instancia / Documento

GET <http://mi-dominio.edu/api/categorias/{cat-id}>

- Controlador

GET <http://mi-dominio.edu/api/categorias/organizar>

➤ Organización jerárquica – Granularidad

GET <http://mi-dominio.edu/api/registros/>

GET <http://mi-dominio.edu/api/registros/{id}>

GET <http://mi-dominio.edu/api/registros/{id}/categoria>

DOCUMENTACIÓN DE LA API swagger.io



SWAGGER

- Swagger
 - Herramienta open source concebida para diseñar, documentar y probar APIs.
 - Utiliza el estándar OpenAPI Specification
- ¿Por qué es importante el diseño y documentación de las APIs?
 - Al igual que los diagramas de flujo en primer año, ir directamente al código en general va a producir resultados mediocres.

OPEN API 3 – ARCHIVO YAML

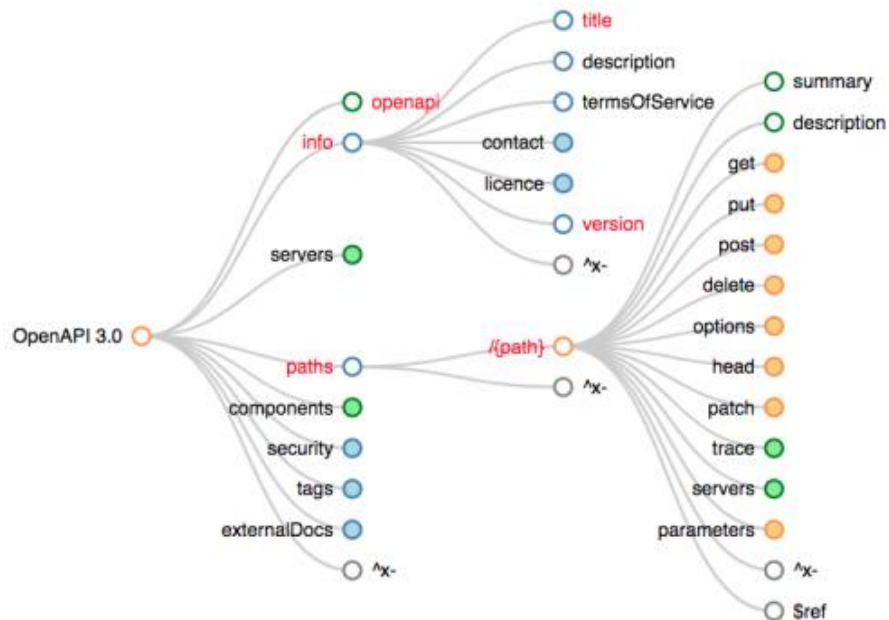
- La documentación en Swagger se basa en un archivo .yaml

Sintaxis Mezcla de JSON y Python

La indentación determina la jerarquía

Los guiones representan listas

```
1 openapi: 3.0.3
2 info:
3   title: API de Registro de Comias
4   description: Esta API se encarga de mantener la información del registro diario de
5     comidas para una persona.
6   version: 1.0.1
7 servers:
8   # Added by API Auto Mocking Plugin
9   - description: SwaggerHub API Auto Mocking
10    url: https://virtserver.swaggerhub.com/utn-frc/registro-comidas/1.0.1
11    - url: http://api.registrocomidas/v1
12 tags:
13   - name: categoria
14     description: Este endpoint maneja toda la información relacionada con las categorías de
15
```



OpenAPI Object



Modified object!



Allows extension with x- properties



OpenAPI Specification

Description

OpenAPI 3.0 top level object. This is the root document object for the OpenAPI Specification document.

OpenAPI Object Change log



Deleted properties



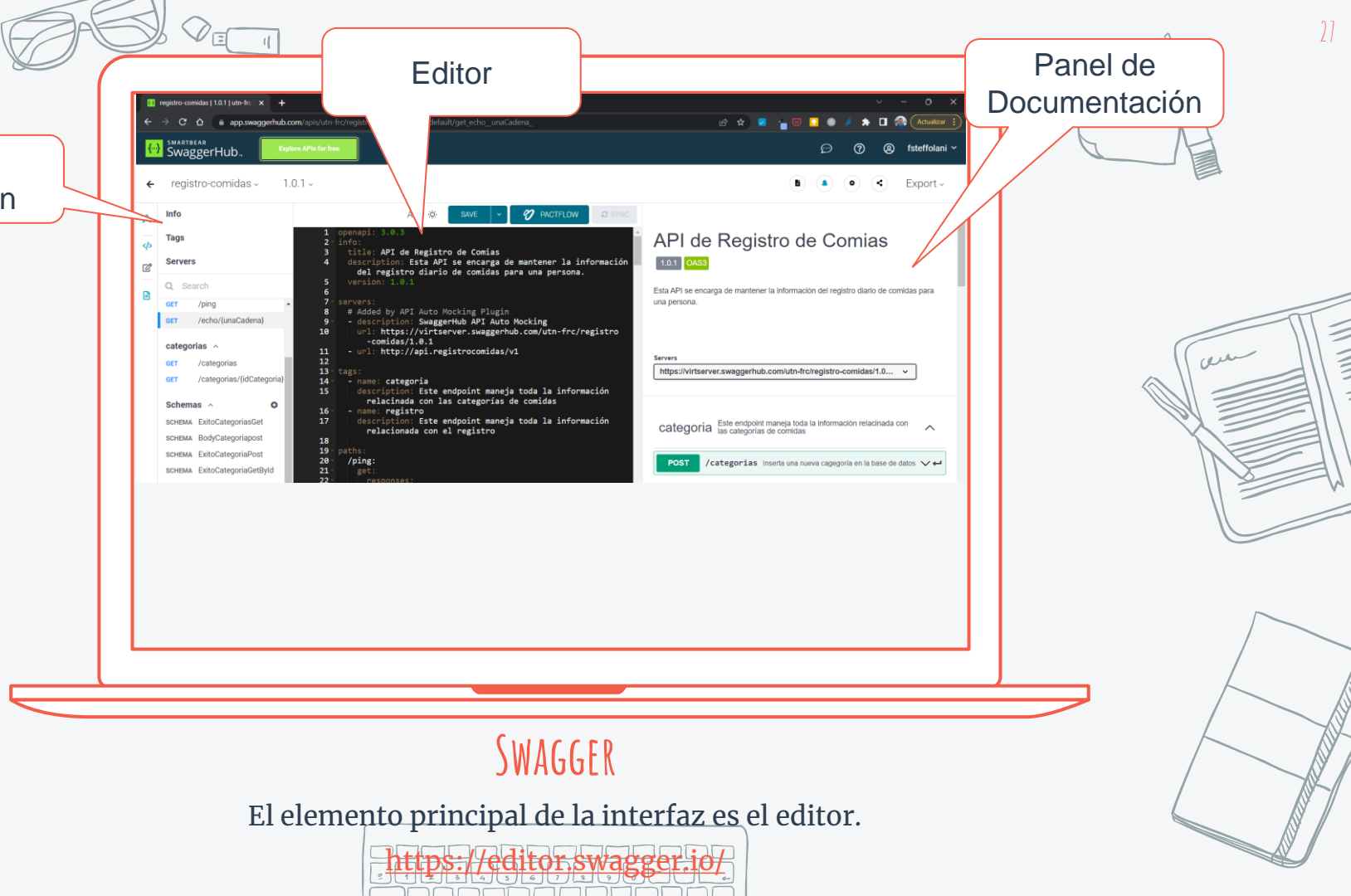
New properties

What's new

The new OpenAPI Specification version 3.0 offers many welcomed improvements and new features (see [OpenAPI blog post series](#) about this).

Here are the noticeable changes on top level (navigate through the tree to see what happened on other levels):

- Bye bye **swagger** and hello **openapi**.
- Reusable definitions are centralized in **components** making the document more consistent (the previous version mixed reusable and



2.

A IMPLEMENTAR NUESTRA API

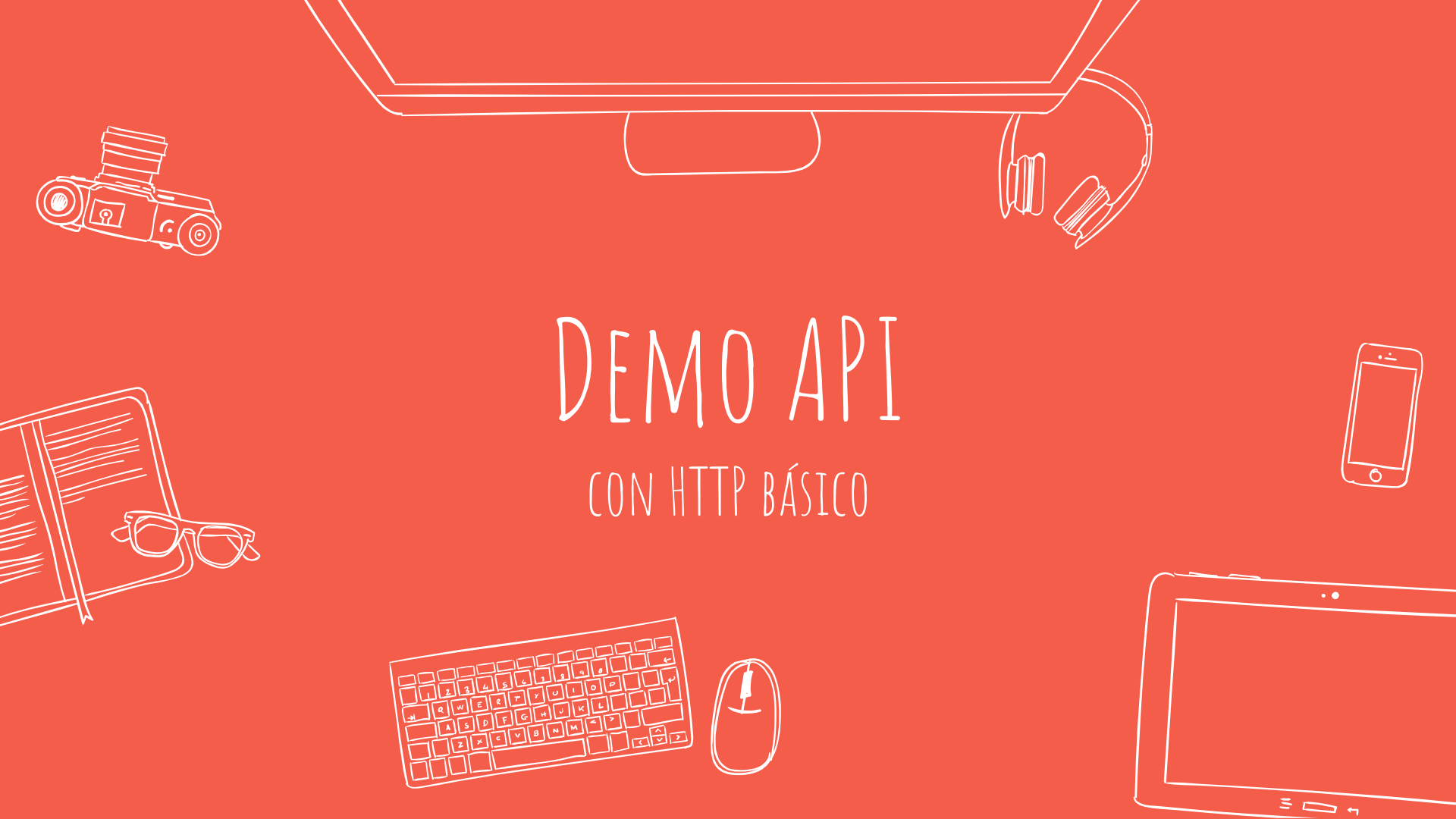
Pero y entonces qué programamos...



*El problema reside en escuchar peticiones
que lleguen a través de la red y ser capaz de
contestarlas.*

DEMO API

CON HTTP BÁSICO



ENTENDIENDO EL PROBLEMA



Qué hicimos para nuestra primera API



Utilizamos el módulo nativo de Nodejs HTTP



Definimos una función para manejar las peticiones con los atributos **request** y **response**

```
import http from "http";|
```



Creamos el servidor

```
37 (async function start() {  
38   // Crear un servidor HTTP  
39   const server = http.createServer(handleRoot);  
40 }
```



Iniciamos el servidor

```
52 // Escuchar en el puerto 3000  
53 const PORT = 3000;  
54 server.listen(PORT, () => {  
55   console.log(`Servidor REST escuchando en el puerto ${PORT}`);  
56 });  
57 }());|
```

3.

EXPRESS JS

Lo mismo mucho más simple





Como siempre, cuando un trabajo se repite a través de diferentes proyectos con distintas particularidades pero idénticos fundamentos aparecen los Frameworks...



Express es un Framework para la construcción de APIs en Javascript con Nodejs.



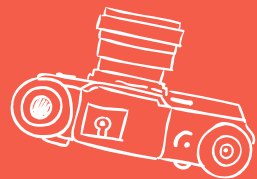


EXPRESS

Express es un Framework.

DEMO API

CON EXPRESS



SIMPLIFICANDO CON EL FRAMEWORK EXPRESSJS

37

✓ Qué hicimos para nuestra primera API

- ✓ Instalamos express y lo importamos
- ✓ Instanciamos el servidor
- ✓ Agregamos respuesta a rutas específicas

```
JS index.js > ...  
1   import express from "express";  
2  
3   const app = express();  
4
```

```
4  
5   // Handler function para la ruta raíz  
6   app.get("/", (req, res) => {  
7     res.send("¡Bienvenido al servidor REST!");  
8   });  
9
```

✓ Iniciamos el servidor

```
27  
28   (async function start() {  
29     // Escuchar en el puerto 3000  
30     const PORT = 3000;  
31     app.listen(PORT, () => {  
32       console.log(`Servidor REST escuchando en el puerto ${PORT}`);  
33     });  
34   })();  
35
```

PRÓXIMOS PASOS...

- Mejorar el Ruteo
- Agregar validaciones
- Reorganizar la API de acuerdo con una anatomía estándar.
- Conectar a la base de datos
- Definir una capa de acceso a datos
- Agregar funcionalidad

GRACIAS!
Preguntas?

