Open Opened 3 hours ago by **Rubén Montero**

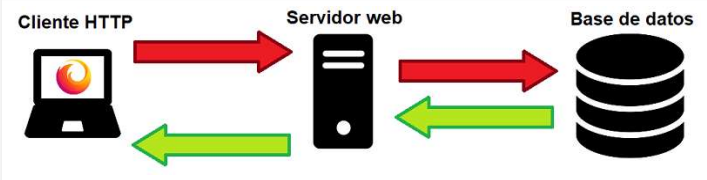
Nuestro API REST dashboards (sobre el papel)

Resumen

- Veremos nociones básicas de un API REST: Verbos HTTP, peticiones y respuestas
- Introduciremos el API REST *dashboards* que usaremos en el proyecto
- Señalaremos el fichero OpenAPI (YAML) usado para describir dicho API REST

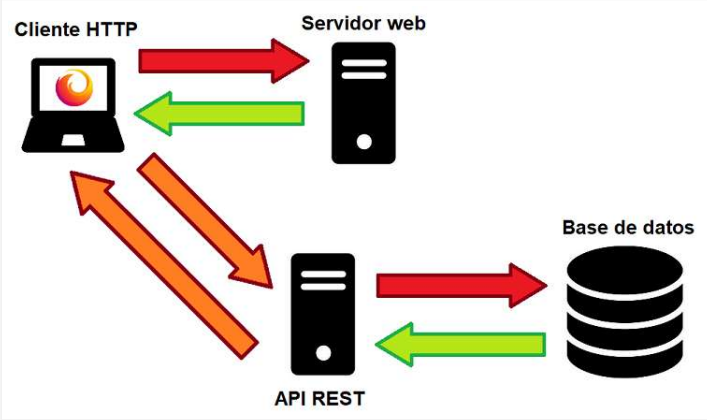
Descripción

Cuando se programa una página web lado servidor (PHP, Django,...) es el servidor web quien se encarga establecer una conexión con la base de datos, usando la tecnología que sea, y, luego, responder al cliente con un documento HTML bien formado y completo:



Esta es una arquitectura muy *cómoda* y *segura* para el cliente, pero *todo el peso* se lo lleva el servidor. La arquitectura lado servidor, generalmente, produce páginas web más lentas y menos escalables.

¿No funcionaría mejor si le entregamos al cliente una página web con mucho JavaScript que se encargue de pedir los datos ella misma?

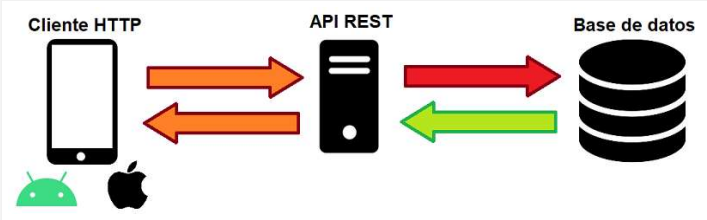


Una arquitectura cliente *distribuye* de forma más eficiente la computación necesaria para *renderizar* las páginas web.

APIs REST

Una API REST se trata de una *fachada HTTP* empleada para exponer los datos de una base de datos.

Será útil para que nuestras páginas web consuman o publiquen datos ejecutando código JavaScript. ¡Ojo! También será útil para otras plataformas que necesitan de APIs REST para consumir datos (e.g.: *apps* móviles).



Peticiones HTTP

Las peticiones HTTP tienen 3 partes:

- **Línea de petición**. En la primera línea se especifica el verbo (ó método) HTTP y el *recurso* solicitado. Algo así:

```
GET /publicaciones
```

Los verbos más usados son GET, POST, PUT, DELETE

- **Cabeceras**. Son pares clave-valor que especifican información adicional. Hay unas cuantas cabeceras predefinidas, pero también podemos incluir nuestras cabeceras *no estándar*.

```
User-agent: Mozilla Firefox
Accept: application/json
```

- **Cuerpo**. Viene después de una línea en blanco e indica información adicional que se manda al servidor. Por ejemplo, si estamos usando una petición POST para crear una nueva *publicación*, seguramente viaja en el cuerpo. El formato del cuerpo es libre, aunque trabajaremos con JSON, como se muestra a continuación:

```
{
  'texto_publicación': '¡Estoy trasteando con APIs REST!'
}
```

Respuestas HTTP

Las respuestas HTTP también constan de 3 partes:

- **Línea de respuesta**: Contiene un código de respuesta. ¿Te suena el típico *404 Not Found*? Es un código de respuesta HTTP. Se dividen en 5 categorías: 1xx Información, 2xx Éxito, 3xx Redirección, 4xx Error cliente, 5xx Error servidor.
- **Cabeceras**: Pares clave-valor enviados en la respuesta. Ls famosas *cookies* viajan en las cabeceras de petición/respuesta.
- **Cuerpo**: Donde el API REST responde información relevante. Suele venir en formato JSON. En caso de una petición web, normalmente obtenemos HTML del servidor.

Nuestro caso: El API REST *Dashboards*

Nosotros vamos a crear una página de *dashboards* que son *categorías* donde los usuarios hacen preguntas y respuestas anónimamente.

Ya está disponible el API REST para el proyecto. Puedes abrir un navegador y consultar cualquiera de los 3 *endpoints* GET disponibles:

- http://raspi:8081/api/v1/dashboards
- http://raspi:8081/api/v1/dashboards/3
- http://raspi:8081/api/v1/dashboards/3/questions/1

La definición formal de este API REST está disponible en el fichero `react-dashboards/django-backend/doc/swagger/swagger.yaml` de tu repositorio. Está en formato OpenAPI.

La tarea

Abre Visual Studio Code, y en la pestaña `Complementos`, **instala** `Swagger Viewer`:

Luego, también desde Visual Studio Code, **abre** el fichero `react-dashboards/django-backend/doc/swagger/swagger.yaml`.

Pulsa F1 y ejecuta *Preview Swagger* (esta opción sólo está disponible si has instalado el complemento `Swagger Viewer`):

Deberías ver una *previsualización* del API REST *dashboards*:

Ahí está la información de *cómo es* el API REST que atacaremos en las siguientes tareas. Curiosea un poco.

🏆 Por último

No hagas nada más. ¡Esta tarea no es de código! No hace falta subir ningún *commit*.

Rubén Montero @ruben.montero changed milestone to %Sprint 3 3 hours ago