Entregando tu producto como una API

Tomás Garzón

esLibre 2021

¿Quién soy?

- Tomás Garzón - tomasgarzonhervas@gmail.com
- Ingeniero software (Universidad de Granada, especialidad en IA).
- Experiencia en startups (20 proyectos).
- Actualmente: Backend developer en Nucoro (nucoro.com)
- Lenguaje y Framework habituales: Python y Django

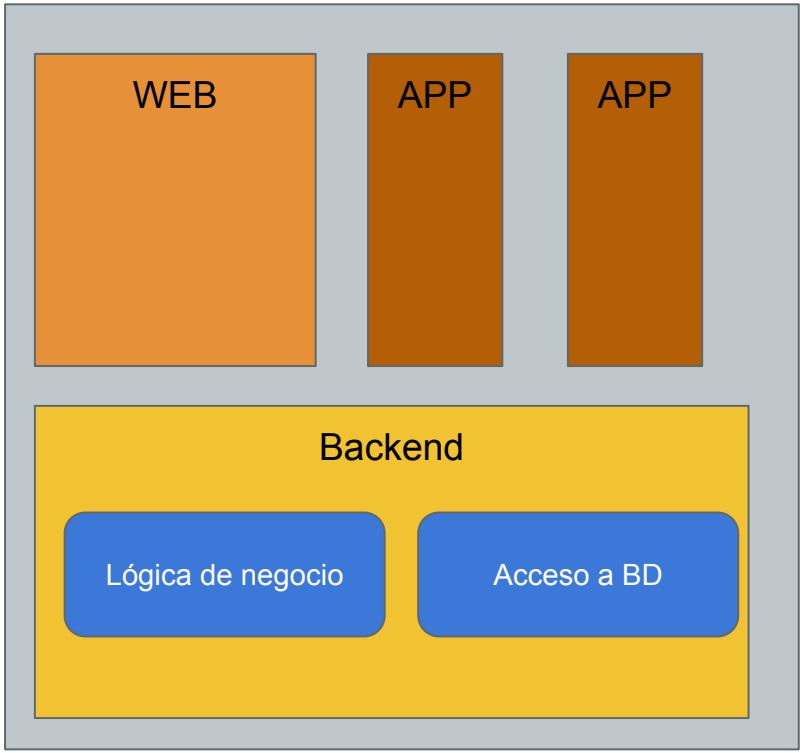
¿Por qué estoy aquí?

- Quiero evolucionar mi producto añadiendo una API?
- Tus clientes o el mercado me demandan APIs con más capacidades?
- Tengo un prototipo/idea y quiero conectarla fácilmente a otros servicios?

¿ Qué es una API?

- https://en.wikipedia.org/wiki/API
- Una interfaz para la comunicación entre sistemas/servicios/programas.
- Esconde los detalles de implementación
- Es un contrato entre dos partes.

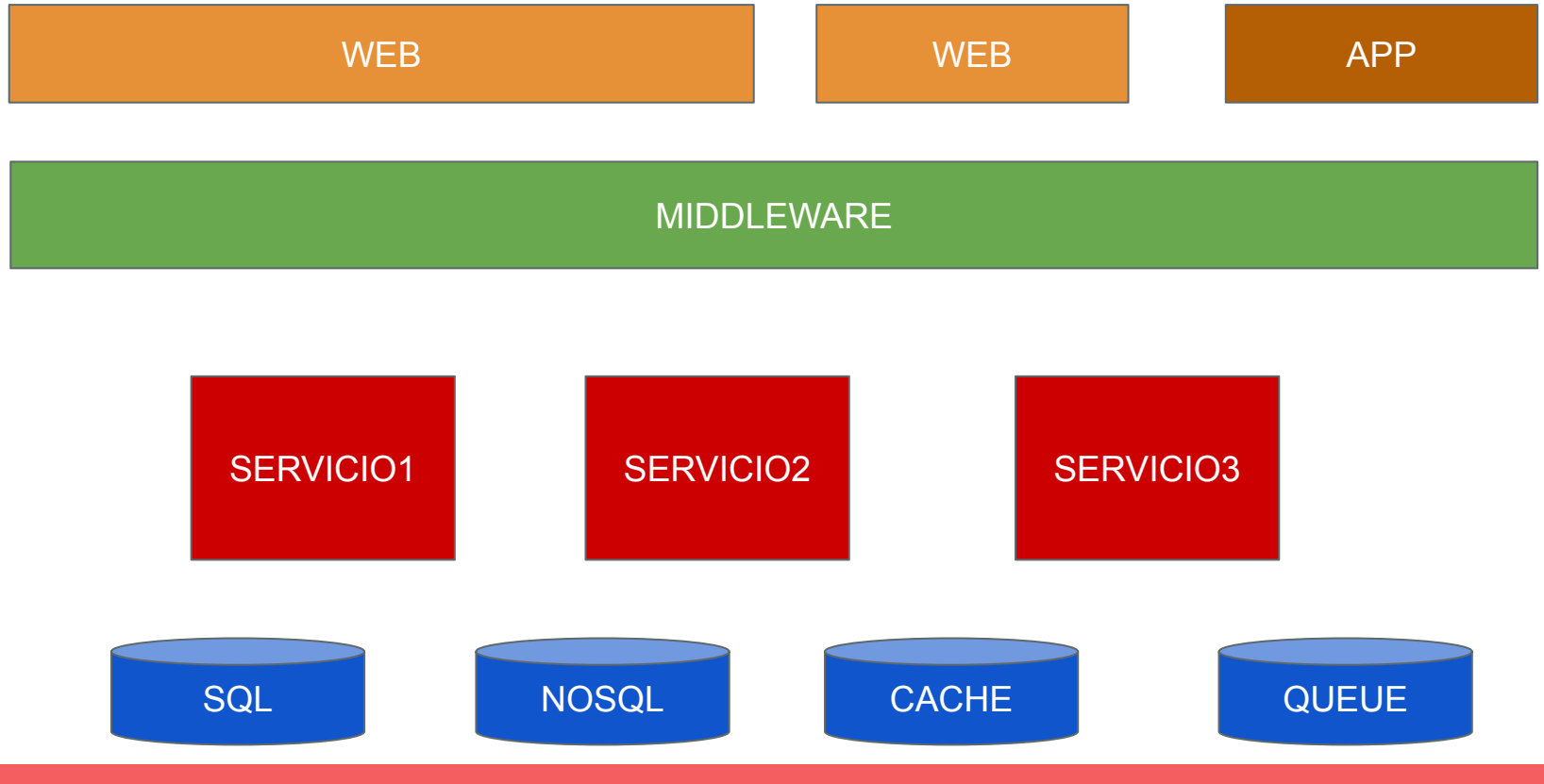
Mi producto actual



Cambio de arquitectura

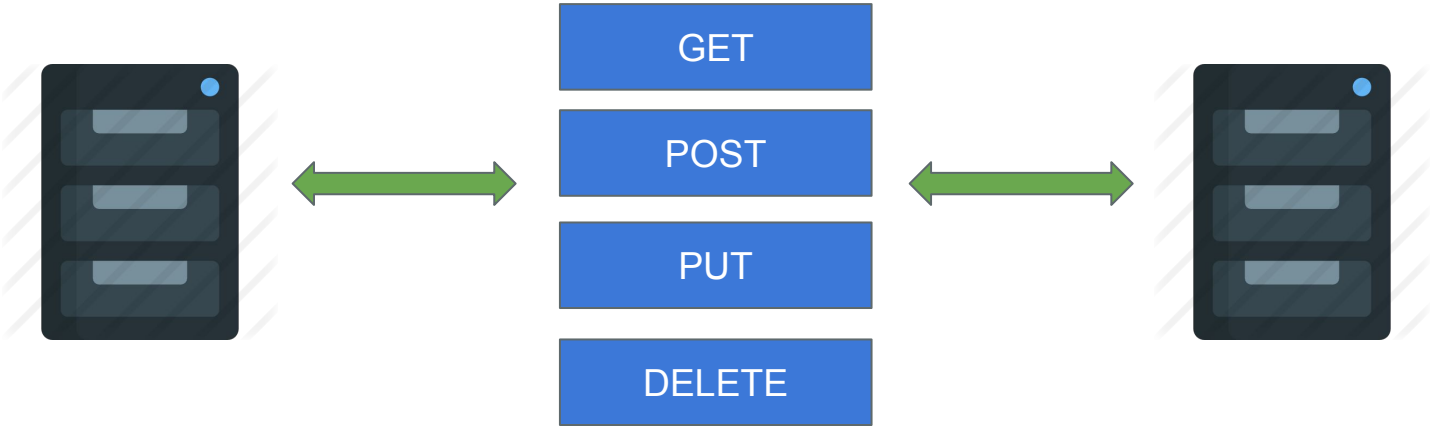
- Inicialmente queríamos cubrirlo todo (frontend, backend, almacenamiento, apps móviles, etc)
- Actualmente, nos orientamos hacia una arquitectura de colaboración:
  - El sistema final está compuesto por muchas piezas/tecnologías → objetivo común.
  - Tu producto puede que sea solo una parte, que resuelve un problema concreto → focalizarse en donde apartas valor diferencial.
  - Mercados/sectores con muchos players → unos pocos grandes y muchos chicos.
  - Soluciones tan complejas que no se pueden orquestar pensando en una sola pieza.
  - **Para poder colaborar hay que comunicarse.**

Mi nuevo producto



Necesito una API

- Distintas aproximaciones para API:
  - Ficheros de intercambio
  - RPC
  - Websocket.
  - REST.
- Consideremos API Rest en profundidad.



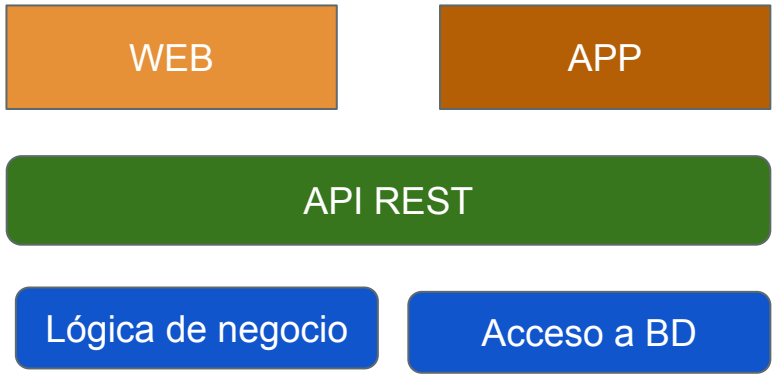
Consideraciones

- Quién va a consumir mi API Rest
- Autenticación y Autorización.
- Documentación.
- Mantenimiento.
- Rendimiento.
- Gestión de Errores.
- Testing

Distintas APIs

- Encuentra el caso de uso → encuentras quién y cómo va a consumir tu API
- Ejemplo:
  - Application API donde es un usuario usando una web o móvil quien interacciona.
  - Integration API donde es un tercero, un sistema externo, quien interacciona para hacer operaciones.
- Cada API puede tener distinta documentación, gestión de errores

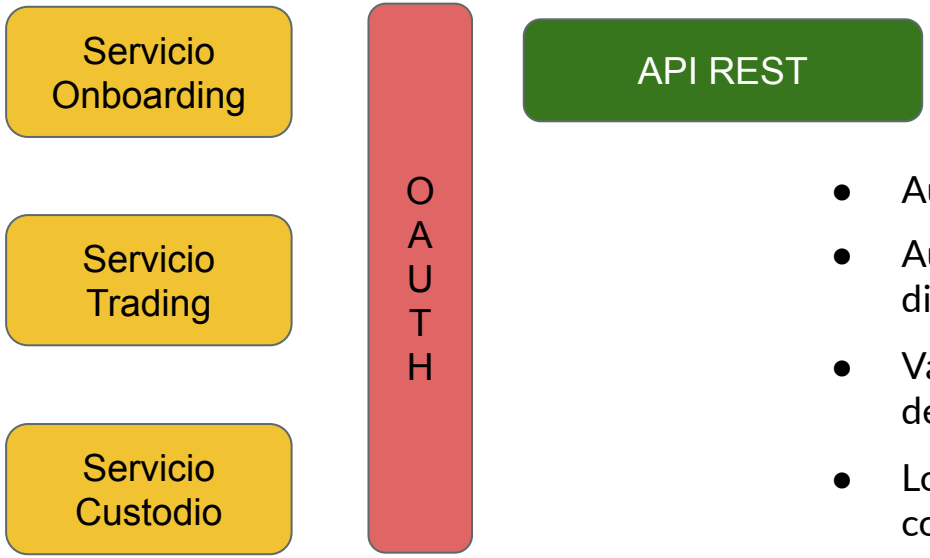
Application API



- Autenticación: usuario/contraseña → JWT
- Autorización: cada usuario puede acceder solo a su información
- Validación y lógica de negocio ligeramente acoplada

- Los errores deben de ser descriptivos para humanos
- Mantenimiento de múltiples versiones compatibles.
- La escalabilidad/performance afectan directamente al usuario

Integration API



- Autenticación mediante Oauth (client ID, Client Secret)
- Autorización: cada servicio puede tener una app con distintos scopes
- Validación y lógica de negocio totalmente desacoplada y extensible
- Los errores deben de ser códigos de un catálogo, comprensibles para otros servicios
- Mantenimiento es más sencillo llevar el control de versiones.
- La escalabilidad/performance afectan a los servicios

Autenticación & Autorización

- El usuario usa su username/password almacenados para obtener el token JWT
- El token en cada llamada identifica al usuario, y le da acceso solo a los recursos asociados a él.
- Con OAuth, definimos distintas Aplicaciones, cada una puede tener distintos scopes.
- Aseguramos que cada servicio (Application) puede acceder solo a los scopes permitidos.
- Una vez tiene acceso a un scope, puede actuar contra cualquier instancia de ese recurso, modificando cualquier dato.
- Herramientas:
  - https://django-oauth-toolkit.readthedocs.io/en/latest/
  - https://pyjwt.readthedocs.io/en/stable/

Auditoría y Logs

- Mantener logs de acceso te dará información muy valiosa sobre:
  - cómo se usa tu API.
  - cuánto tarda en responder tu API.
  - obtener trazas de error o secuencias de funcionamiento incorrectas.
- Sistema de auditoría: almacenamiento de todas las llamadas que se hacen en tu API.
  - Qué accesos se hacen a tu API.
  - Quién realiza esos accesos y con qué objetivo (GET, POST, PUT, DELETE).
- Herramientas:
  - rsyslog integrado con Django.
  - https://pypi.org/project/django-easy-audit/

Documentación y Soporte

- La documentación es clave por que supone la especificación del “contrato”.
- Mantener la documentación de forma manual es tedioso.
- Para startups, sin un equipo específico de soporte, podemos optar por usar OpenAPI (https://swagger.io/specification/)
- Documentación autogenerada a partir del código fuente.
- Herramientas:
    - https://drf-spectacular.readthedocs.io/en/latest/

Rendimiento

- Startups → velocidad para encontrar Market-Fit.
- Cuando enfrentamos nuestra API a casos reales → Desastre!!!
- Fase de refactorizar y Optimizar.
  - Activar la creación en bulk de objetos (bulk_create/bulk_update en Django)
  - Mide tiempos de respuesta (end to end).
  - Mide consultas a la base de datos y/o otros sistemas externos.
  - Optimiza las consultas:
    - Cacheando consultas repetitivas.
    - Realizando joins y evitando campos innecesarios
- Enfrenta a tu sistema contra un framework de test de carga.
- Herramientas:
  - https://locust.io/

API throttling

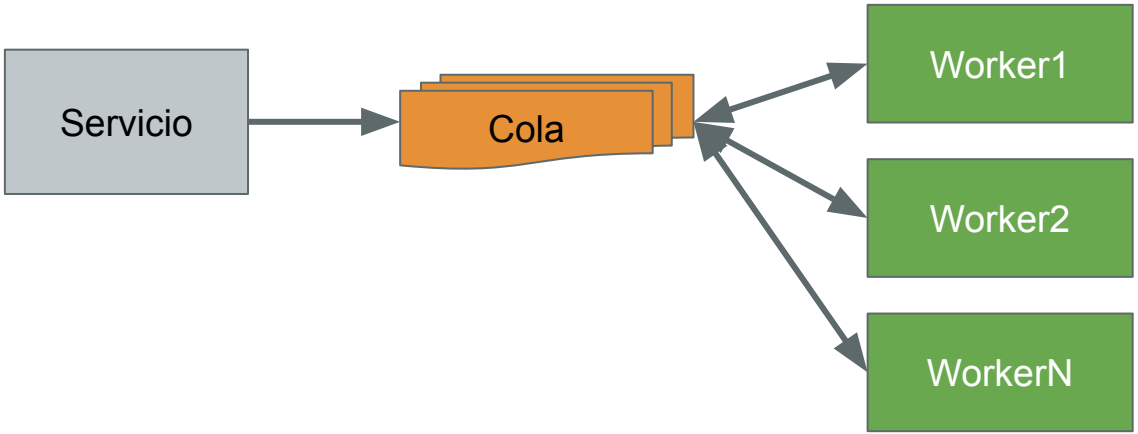
- Proceso para limitar el número de API request que un usuario hace en un intervalo de tiempo.
- Application API: puedes definir distintas configuraciones para cada endpoint, para evitar ataques a tu API o un uso inapropiado (uso externo de tu API).
- Integration API: no te hace falta ser muy estrictos por que tu API va a ser consumida por servicios que se conectarán por IP internas
- Otros usos:
  - Evitar llamadas a endpoints duplicadas (múltiples pagos, órdenes de compra repetidos, etc)
- Herramientas:
  - https://www.django-rest-framework.org/api-guide/throttling/

Testing

- La metodología de TDD encaja muy bien cuando trabajamos con API Rest.
- Necesitamos tests de integración más complejos para workflows (secuencia de llamadas a la API)
- Herramientas:
  - https://www.django-rest-framework.org/api-guide/testing/
  - https://github.com/behave-restful/behave-restful (BDD)

Camino por recorrer

- Esto no vale para todos los problemas/dominios.
- Grandes volúmenes de datos/llamadas:
  - Websocket: implementar restful usando mensajes en un canal de websocket.
  - Patrón cola de mensajes



Gracias

Tomás Garzón Hervás

tomasgarzonhervas@gmail.com