

Entregando tu producto como una API

Tomás Garzón
PyConES 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.

“Nuevas” arquitecturas

- 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 aportas 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.**
- Nuevas arquitecturas ⇒ nuevos modelos de negocio

Necesito una API

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



Design Principles

1. Haz una cosa y hazla bien.
2. Ten un “get started” fácil y rápido.
3. Mantén una coherencia intuitiva.
4. Gestiona errores con significado.
5. Diseña pensando en la escala y el performance.
6. Evita los breaking changes.

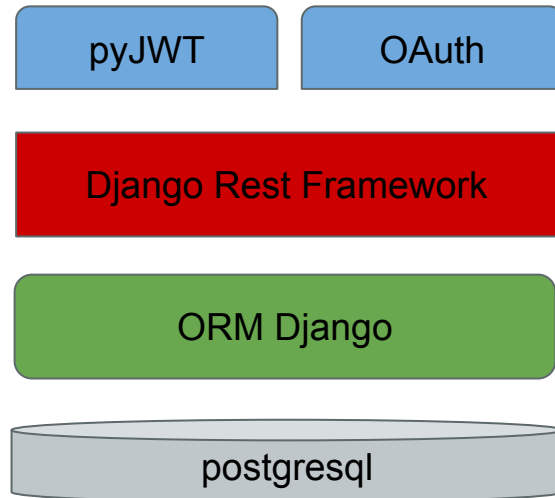
Design process

1. Escribe las especificaciones.
2. Inicia un proceso de revisión interna.
3. Comparte la API con partners y obtén feedback primero.
4. Abre la API en beta.

Implementation process

- Autenticación y Autorización
- Diseño de test basados en los specs.
- ¿Qué acciones/métodos necesita? -
 - Views.
 - Serializadores.
 - Queries.
 - Errores.
 - Logs
- Realizar test de rendimiento y performance.
- Escribir API Spec en el código.
- Escribir release notes asociadas (breaking changes?)

Stack



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 (Aplicación) 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/>

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/behavereastful/behavereastful> (BDD)
 - <https://github.com/schemathesis/schemathesis>

Implementation: Views, serializers, errors and logs

1. Don't repeat yourself: rest_framework implementa mucha funcionalidad.
2. Paginación por defecto activada.
3. Usuario autenticado por defecto activada.
4. Throttling por defecto activado.
5. Extras:
 - a. Uso de routers anidados, pero sin abusar para evitar repetir APIs o complicarlas.
 - b. Uso de serializadores anidados, sin abusar para evitar repetir views o serializadores
 - c. Filtros y ordenaciones: coherencia en los nombres y criterios
 - d. Campos expandibles, pueden extender la api para hacerse similar a GraphQL (ojo con el rendimiento)

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 estos 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/>
- Generar SDKs y uso interno para hacer test de integración.

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.
- Puedes definir distintas configuraciones para cada endpoint, para evitar ataques a tu API o un uso inapropiado (uso externo de tu API).
- Otros usos:
 - Evitar llamadas a endpoints duplicadas (múltiples pagos, órdenes de compra repetidos, etc)
- Herramientas:
 - <https://www.django-rest-framework.org/api-guide/throttling/>

Cambios y breaking changes

- Llevar un log de qué cambios que incluye cada modificación en la API.
- Filosofía: “lo que funcionaba ayer debe de funcionar mañana”
- Cambios que puedan romper una integración de un cliente deben de evitarse.
- Planificar y comunicar los breaking changes.
 - Deprecar APIs.
 - Estudiar el uso de estas APIs.
- Uso de versionado de APIs.
- Quizás sea el mayor reto al que se enfrentará tu startup

Gracias

Tomás Garzón Hervás

tomasgarzonhervas@gmail.com

