Alumno: Diego Gatica

Materia Programación 1

Curso: Segundo año

Profesor Gerardo Herrera

Fecha de entrega:

Investigación sobre APIs

introducción:

En el contexto actual de constante avance tecnológico y conectividad global, las APIs(interfaces de programación de aplicaciones) se han convertido en un pilar fundamental del desarrollo de software moderno. Estas interfaces permiten la comunicación entre diferentes sistemas, plataformas o aplicaciones, facilitando el intercambio de datos y funciones de manera eficiente y segura.

Este trabajo informa sobre el concepto de APIs y su evolución, los métodos y protocolos más utilizados, los formatos de intercambio y muchísima más información útil para poder estar muy bien informado.

¿Qué es una API?

API significa “interfaz de programación de aplicaciones”, las API son mecanismos que permiten a dos componentes de software comunicarse entre sí mediante un conjunto de definiciones y protocolos. La arquitectura de las API suele explicarse en términos de cliente y servidor. La aplicación que envía la solicitud se llama cliente, y la que envía la respuesta se llama servidor.

Las API web comenzaron poniendo el “comercial” en “.com”, impulsando la visión de nuevas empresas comerciales emergentes que buscan cambiar la forma en que hacemos negocios en la web. Aprovecharon este nuevo medio para poner productos y servicios a disposición de los clientes a través de un único sitio web y garantizar que los socios y revendedores externos pudieran ampliar el alcance de sus plataformas.

**Tipos de APIs**

**API abiertas** **(API públicas):** Estos están disponibles para que cualquiera los use. Por ejemplo, los desarrolladores pueden usar la API de Twitter para mostrar tweets o permitir a los usuarios publicar directamente desde su aplicación.

**API de socios:** Estos se comparten con socios o empresas seleccionadas y generalmente requieren permiso o contratos. Por ejemplo, una plataforma de reserva de hoteles puede utilizar la API asociada de una aerolínea para mostrar opciones de vuelos junto con listados de hoteles.

**API internas (API privadas):** Estos se utilizan únicamente dentro de una empresa para conectar sus propias herramientas y sistemas. Por ejemplo, una tienda en línea podría utilizar una API interna para vincular su sitio web con su sistema de gestión de inventario.

**API compuestas:** Estos combinan múltiples servicios en una sola llamada, lo que ayuda a las aplicaciones a realizar varias acciones a la vez. Por ejemplo, una aplicación de entrega de alimentos podría utilizar una API compuesta para realizar un pedido, actualizar el estado de la entrega y enviar un mensaje de confirmación en una sola solicitud.

**Las API pueden funcionar de cuatro maneras diferentes, según el momento y el motivo de su creación.**

### **API de SOAP**

Estas API utilizan el protocolo simple de acceso a objetos. El cliente y el servidor intercambian mensajes mediante XML. Se trata de una API menos flexible que era más popular en el pasado.

### **API de RPC**

Estas API se denominan llamadas a procedimientos remotos. El cliente completa una función (o procedimiento) en el servidor, y el servidor devuelve el resultado al cliente.

### **API de WebSocket**

La API de WebSocket utiliza objetos JSON para transmitir datos. La API de WebSocket admite la comunicación bidireccional entre las aplicaciones cliente y el servidor. El servidor puede enviar mensajes de devolución de llamadas a los clientes conectados, por lo que es más eficiente que la API de REST.

### **API de REST**

Estas son las API más populares y flexibles que se encuentran en la web actualmente. El cliente envía las solicitudes al servidor como datos. El servidor utiliza esta entrada del cliente para iniciar funciones internas y devuelve los datos de salida al cliente.

Algunos formatos populares de intercambio son:

**JSON:** Es un formato de intercambio de datos liviano, basado en texto e independiente del lenguaje, derivado de JavaScript pero utilizado en muchos lenguajes de programación. JSON es ideal para el intercambio de datos entre un servidor y una aplicación web debido a su simplicidad y facilidad de uso, y es el formato de intercambio de datos moderno más popular. JSON se basa en dos estructuras.

**XML:** Es un lenguaje de marcado que define un conjunto de reglas para codificar documentos en un formato legible tanto por humanos como por máquinas. Se utiliza principalmente para el almacenamiento y transporte de datos. Los datos XML están organizados en una estructura similar a un árbol. Le permite definir sus propias etiquetas y la estructura del documento.

**YAML:** Es un formato de serialización de datos legible por humanos. Es especialmente adecuado para archivos de configuración y datos que están siendo editados directamente por humanos. YAML utiliza una sintaxis de espacios en blanco no estricta con pares clave-valor. Puede representar escalares (cadenas, números), listas y matrices asociativas.

**Métodos**

Los métodos HTTP son la columna vertebral de API permitiendo una comunicación fluida entre clientes y servidores. Ya sea que esté recuperando datos, creando recursos o eliminando registros, comprender estos métodos es crucial para crear API eficientes y escalables.

Estos métodos forman la base de las interacciones API, garantizando que los desarrolladores puedan realizar acciones específicas de manera predecible y eficiente. Por ejemplo, cuando navega por un sitio web, su navegador envía una solicitud GET para recuperar el contenido de la página. De manera similar, cuando envía un formulario, se envía una solicitud POST para procesar los datos.

### **Métodos HTTP clave:**

* GET: Recuperar datos del servidor.
* POST: Crear un nuevo recurso o enviar datos.
* PUT: Actualizar o reemplazar un recurso existente.
* ELIMINAR: Eliminar un recurso.

|  | **SOAP** | **REST** | **GraphQL** |
| --- | --- | --- | --- |
| Enfoque primario | Intercambio de mensajes | Trabajando con recursos | Servicio de consultas de datos anidados complejos |
| Protocolo | SOAP | HTTP | Agnóstico (funciona a través de HTTP) |
| Formato de datos | XML | JSON, XML | JSON/XML (agnóstico) |
| Contrato/Esquema | Requerido (WSDL) | Opcional (OpenAPI/Swagger) | Se requiere un esquema fuerte |
| Estilo | Más rígido | Estilo arquitectónico | Tiempo de ejecución + lenguaje de consulta |
| Almacenamiento en caché | Desafiante | Sencillo a través de encabezados | Complejo |
| Manejo de errores | Incorporado | No hay forma estándar | Soporte parcial |
| Seguridad | Varios estándares disponibles | Autenticación básica, OAuth 2.0 | Se necesita lógica personalizada |
| Rendimiento | SOAP tiene más gastos generales | Más rápido | Rápido para consultas específicas |
| Curva de aprendizaje | Alto | Bajo | Moderado |
| Cuándo usarlo | Aplicaciones empresariales internas que comparten datos confidenciales y estructurados | API públicas, aplicaciones web y móviles | Aplicaciones web/móviles con requisitos de datos jerárquicos complejos |

Los lenguajes más populares para el desarrollo de API incluyen a JavaScript, Python, Java, C#. Y cada uno ofrece ventajas únicas para las diferentes necesidades que tenga un proyecto.

* Node.js con JavaScript se destaca en el manejo de muchas conexiones simultáneas y ofrece un vasto ecosistema de paquetes a través de npm. La arquitectura basada en eventos lo hace ideal para aplicaciones en tiempo real.
* [Python con Django](https://www.netguru.com/services/python-development) proporciona sólidas funciones de seguridad y una excelente integración de bases de datos. Su sintaxis simple acelera el tiempo de desarrollo.
* [Spring Boot](https://www.netguru.com/blog/backend-frameworks) ofrece aplicaciones Java de nivel empresarial con una configuración mínima.
* ASP.NET Core aporta un sólido rendimiento y capacidades multiplataforma al desarrollo de C#.

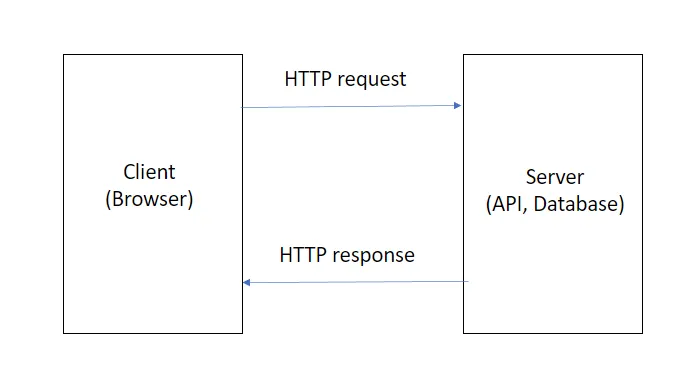
Herramientas de testeo

* Postman es una herramienta de prueba de API popular y adaptable diseñada para facilitar el desarrollo y las pruebas de API. Tanto los desarrolladores como los no desarrolladores pueden usarlo debido a su interfaz gráfica de usuario intuitiva, que no requiere un conocimiento profundo del código.

## REST Assured es una biblioteca basada en Java diseñada para escenarios de prueba complejos. Permite pruebas de API expertos para utilizar scripts prescritos y aprovechar funciones avanzadas. Ideal para pruebas de integración y pruebas continuas en un entorno de desarrollo fluido. Proporciona soporte sólido para marcos de prueba y una rica biblioteca de pruebas para trabajar con API REST.

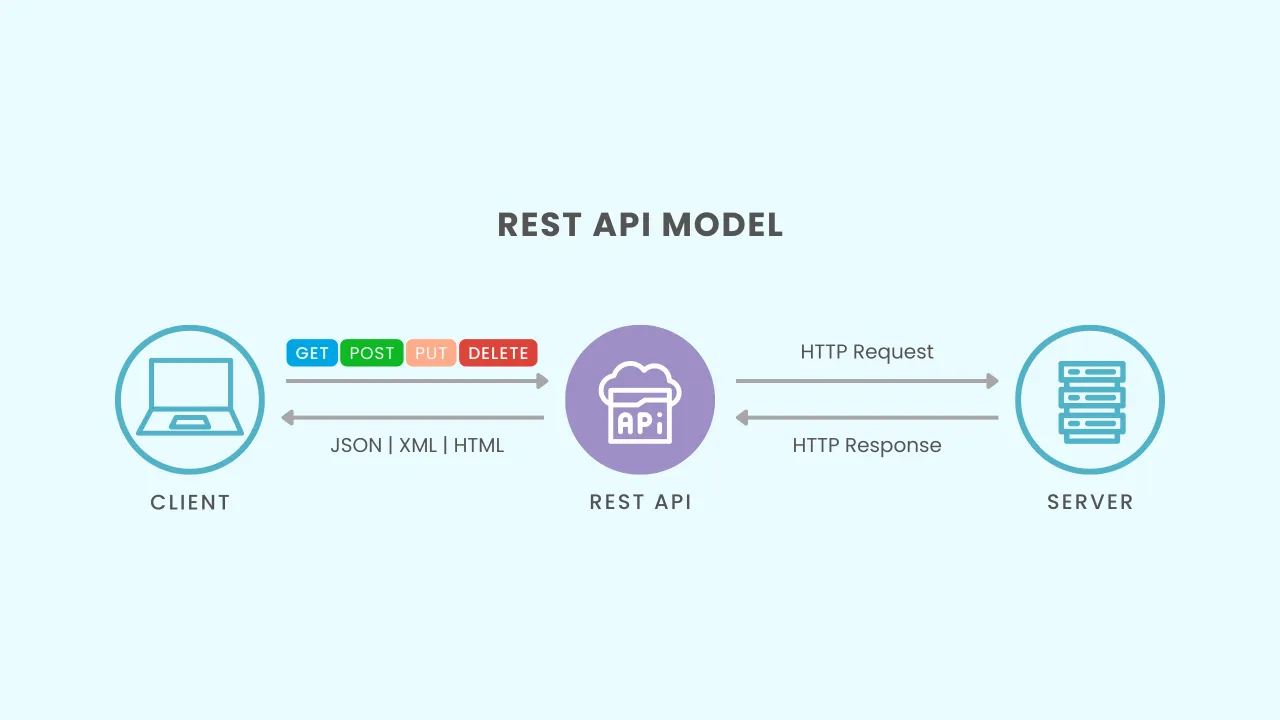
* Insomnia es una API de código abierto herramienta Se utiliza para diseñar, depurar y probar API RESTful y GraphQL, simplificando los flujos de trabajo tanto para desarrolladores como para evaluadores. Al admitir esquemas API, ayuda a validar todos los puntos finales con sus estructuras definidas, fomentando un proceso de desarrollo de API más eficiente.

**¿Cómo funciona un modelo de solicitud/respuesta?**



1. El cliente envía una solicitud
2. El servidor analiza la solicitud
3. El servidor procesa la solicitud
4. El servidor envía la respuesta
5. El cliente analiza la respuesta y la utiliza

**Esquema de cliente-servidor al consumir una API**



# **Claves API: simples pero limitadas**

Las claves API son la forma más básica de autenticación. Son fáciles de implementar pero tienen limitaciones importantes:

Si bien son simples, las claves API son esencialmente contraseñas que no caducan y no pueden revocarse fácilmente por usuario. Son adecuados para servicios internos pero insuficientes para aplicaciones orientadas al usuario.

# **OAuth 2.0: el patrón oro**

OAuth 2.0 proporciona autenticación segura basada en tokens con permisos detallados. Separa la autenticación de la autorización y admite la expiración y actualización del token.

OAuth es más complejo de implementar, pero ofrece seguridad superior a través de permisos con alcance, vencimiento de tokens y la capacidad de revocar el acceso sin cambiar las credenciales.

**Validaciones:**

* **Validación de entrada: su primera línea de defensa**

Cada dato que ingresa a su API es un vector de ataque potencial. La validación integral de entradas evita ataques de inyección, corrupción de datos y explotación del sistema.

* **Validar todo**

Nunca confíe en las aportaciones del cliente. Validar:

* Tipos de datos (cadena, número, booleano)
* Formatos (correo electrónico, URL, fecha)
* Rangos (valores mínimos/máximos)
* Campos requeridos y parámetros opcionales
* Longitudes de matriz y estructuras de objetos
* **Validación de esquemas en la práctica**

Utilice bibliotecas de validación de esquemas para verificar automáticamente las solicitudes entrantes

# **Formatos de error consistentes**

Utilice formatos de error estandarizados en toda su API para evitar fugas de información a través de respuestas inconsistentes.

**Monitoreo y registro: su inteligencia de seguridad**

No puedes proteger lo que no puedes ver. El monitoreo y el registro integrales son esenciales para detectar ataques, depurar problemas y mantener el cumplimiento.

**Técnicas de seguridad avanzadas:**

* Puertas de enlace API
* Autenticación y autorización
* Políticas de seguridad del contenido
* Seguridad de versiones de API
* Pruebas automatizadas
* Pruebas manuales
* Seguridad continua

Tabla comparativa entre metodologías ágiles (Scrum, Kanban) y el enfoque tradicional Waterfall, aplicando al desarrollo de software en general (incluyendo proyectos de APIs)

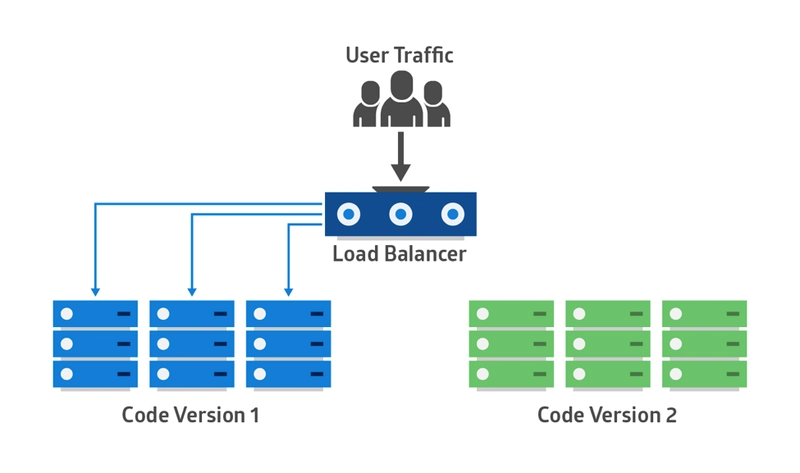
| **Aspecto** | **Waterfall (Tradicional)** | **Agile (Scrum, Kanban, Agile)** |
| --- | --- | --- |
| Estructura del proceso | Secuencial, fases rígidas | Iterativa e incremental (sprints o flujo continuo) |
| Adaptación a cambios | Muy limitada | Alta adaptabilidad y capacidad de reevaluación |
| Involucramiento de stakeholders | Solo al inicio y al final | Participación constante del cliente/usuario |
| Planificación | Extensa al comienzo, fija | Flexible, plan mínimo viable ajustable |
| Testing | Solo al final de desarrollo | Integrado en cada iteración (Scrum) o flujo continuo |
| Entregas | Una sola al final | Entregas frecuentes e incrementales |
| Documentación | Detallada y extensa | Mínima necesaria para avanzar |
| Equipo | Funcional y jerárquico | Multifuncional, autoorganizado |
| Riesgos | Detectados tarde, costoso corregir | Identificados temprano, más económicos de corregir |
| Ideal para | Requisitos bien definidos y estables | Entornos con cambios frecuentes o necesidades dinámicas |

**Estrategias de despliegue y mantenimiento:**

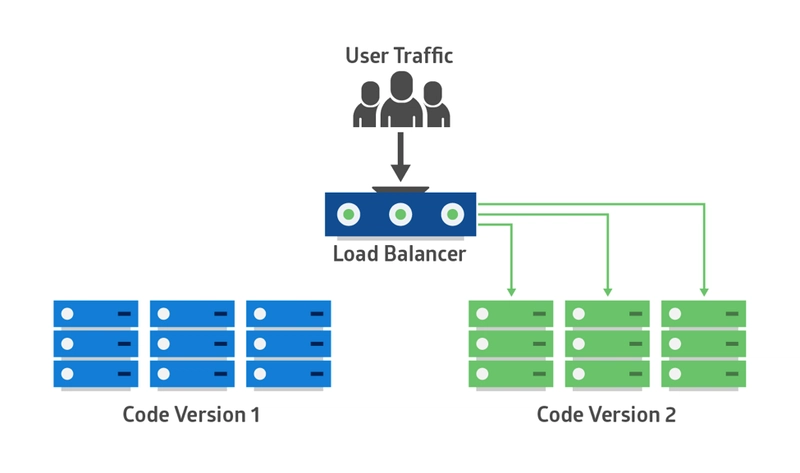
## **Despliegue azul-verde, rojo-negro o A/B**

Este es otro proceso a prueba de fallos. En este método, dos entornos de producción idénticos funcionan en paralelo.

Uno es el entorno de producción que se ejecuta actualmente y que recibe todo el tráfico de usuarios (representado como Azul). El otro es un clon del mismo, pero inactivo (Verde). Ambos utilizan el mismo back-end de base de datos y configuración de aplicación:



La nueva versión de la aplicación se implementa en un entorno ecológico y se prueba su funcionalidad y rendimiento. Una vez que los resultados de las pruebas son exitosos, el tráfico de la aplicación se enruta de azul a verde. Green se convierte entonces en la nueva producción.

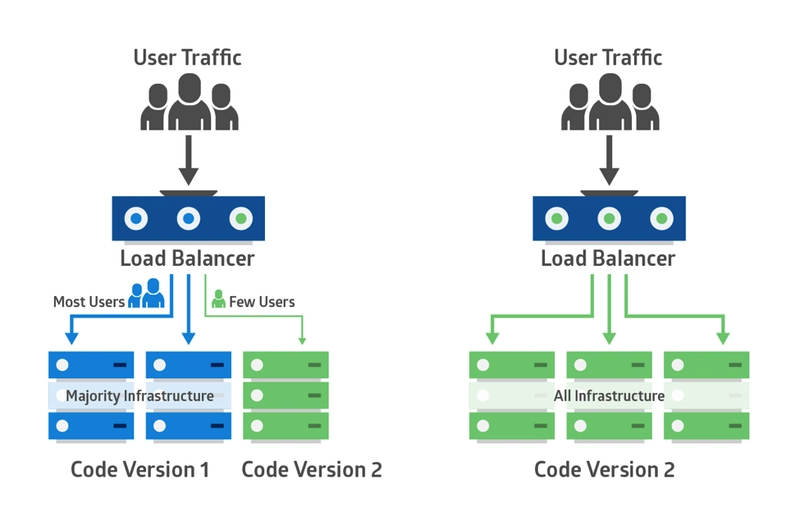


## **Despliegue canario**

La implementación de Canary es como azul-verde, excepto que es más reacia al riesgo. En lugar de cambiar de azul a verde en un solo paso, se utiliza un enfoque por fases.

Con la implementación de Canary, se implementa un nuevo código de aplicación en una pequeña parte de la infraestructura de producción. Una vez que se cierra la sesión de lanzamiento de la aplicación, solo unos pocos usuarios son enrutados a ella. Esto minimiza cualquier impacto.

Sin errores reportados, la nueva versión puede implementarse gradualmente en el resto de la infraestructura. La siguiente imagen demuestra el despliegue canario:



El principal desafío de la implementación de Canary es idear una forma de enrutar a algunos usuarios a la nueva aplicación. Además, es posible que algunas aplicaciones siempre necesiten el mismo grupo de usuarios para realizar pruebas, mientras que otras pueden requerir un grupo diferente cada vez.

## **Seguimiento posterior al despliegue**

Incluso después de adoptar todas esas mejores prácticas, las cosas aún pueden fracasar. Por eso, monitorear los problemas que ocurren inmediatamente después de una implementación es tan importante como planificar y ejecutar una implementación perfecta.

Una herramienta de monitoreo del rendimiento de aplicaciones (APM) puede ayudar a su equipo a monitorear métricas de rendimiento críticas, incluidos los tiempos de respuesta del servidor después de las implementaciones. Los cambios en la arquitectura de la aplicación o del sistema pueden afectar drásticamente el rendimiento de la aplicación.

Una solución de monitoreo de errores como [Rollbar](http://rollbar.com/) es igualmente esencial. Notificará rápidamente a su equipo sobre errores nuevos o reactivados de una implementación que podrían descubrir errores importantes que requieran atención inmediata.

**Analizando un ejemplo práctico para poder guiarse mejor**

## **Ejemplo real 1: API de Spotify**

* **Uso**: Spotify ofrece una API que permite acceder a canciones, álbumes, artistas, listas de reproducción, etc.
* **Quién la usa**: desarrolladores que quieren crear apps de música, dashboards personalizados, visualizaciones de usuarios, etc.

### **Diagrama de flujo básico:**

[App del usuario] --> [Envia solicitud GET a la API de Spotify]

↓

[Spotify válida token de acceso]

↓

[Devuelve JSON con info del artista o canción]

↓

[App muestra los datos al usuario]

## **Modelo de madurez REST (Richardson)**

Este modelo tiene 4 niveles (0 a 3). Spotify está en el **nivel 3**, porque:

* Usa métodos HTTP correctamente (GET, POST, etc.)
* Tiene recursos bien definidos con URLs (/v1/tracks/{id})
* Usa hipermedios (enlaces dentro de las respuestas para navegar entre datos)

Conclusión:

Las APIs se han convertido en una parte esencial del desarrollo de software moderno. Permiten conectar servicios, automatizar procesos e integrar funcionalidades sin necesidad de crearlas desde cero. A lo largo del trabajo se abordaron sus componentes clave, protocolos, métodos, herramientas y buenas prácticas. Comprender su funcionamiento no solo mejora la eficiencia del desarrollo, sino que abre la puerta a soluciones más dinámicas, escalables e innovadoras.

Bibliografía:

Amazon Web Services. (s.f.). *¿Qué es una API?*. AWS.<https://aws.amazon.com/es/what-is/api/>

API7.ai. (s.f.). *HTTP Methods in APIs*. API7 Learning Center.<https://api7.ai/learning-center/api-101/http-methods-in-apis>

Beeceptor. (s.f.). *Data Exchange Formats*. Beeceptor Docs.<https://beeceptor.com/docs/concepts/data-exchange-formats/>

The Linux Code. (s.f.). *Different Types of APIs: SOAP vs REST vs GraphQL*.<http://thelinuxcode.com/different-types-of-apis-soap-vs-rest-vs-graphql/>

Netguru. (s.f.). *How to Create an API: Step-by-Step Guide for Beginners*.<https://www.netguru.com/blog/how-to-create-an-api>

BrowserStack. (s.f.). *Top 8 API Examples (Free & Open Public APIs)*.<https://www.browserstack.com/guide/api-examples>

Frugal Testing. (s.f.). *Rest Assured vs Postman: Which API Testing Tool Is Right For You?*.<https://www.frugaltesting.com/blog/rest-assured-vs-postman-which-api-testing-tool-is-right-for-you>

Abstracta. (2023, marzo 28). *Insomnia vs Postman: ¿Cuál herramienta de testing de APIs te conviene más?*.<https://abstracta.us/blog/testing-tools/insomnia-vs-postman/>

Sharma, A. (2021, junio 18). *Understanding Request and Response Model – A General Overview*. Medium. <https://medium.com/@imakashsharma135/understanding-request-and-response-model-a-general-overview-831c24d2288>

JubaerCoding. (2023, agosto 3). *API Security: A Complete Guide for Developers*. Medium. <https://medium.com/@jubaercoding/api-security-a-complete-guide-for-developers-fbbb8f37fceb>

MostlyJason. (2021, septiembre 29). *Intro to Deployment Strategies: Blue-Green, Canary and More*. DEV Community.<https://dev.to/mostlyjason/intro-to-deployment-strategies-blue-green-canary-and-more-3a3>