

SEGUNDA EDICIÓN

# DISEÑO DE SISTEMAS

UNA GUÍA DE INFORMACIÓN PRIVILEGIADA

Alex Xu



ANAYA  
MULTIMEDIA

# Índice de contenidos

Sobre el autor .....	5
<b>Prólogo</b> .....	<b>12</b>
<b>1. Pasar de cero a millones de usuarios</b> .....	<b>15</b>
Configuración de un solo servidor .....	15
Base de datos .....	17
Qué bases de datos utilizar .....	17
Escalabilidad horizontal y vertical .....	19
Equilibrador de carga .....	19
Replicación de bases de datos .....	21
Caché .....	24
Nivel de caché .....	24
Apreciaciones en el uso de la caché .....	25
Red de entrega de contenidos o CDN ( <i>Content Delivery Network</i> ) .....	27
Apreciaciones en el uso de una CDN .....	29
Capa web sin estado ( <i>stateless</i> ) .....	30
Arquitectura con estado ( <i>stateful</i> ) .....	31
Arquitectura sin estado ( <i>stateless</i> ) .....	32
Centros de datos .....	34
Cola de mensajes .....	36
Registros, métricas, automatización .....	37
Añadir colas de mensajes y distintas herramientas .....	38
Escalabilidad de base de datos .....	38
Escalabilidad vertical .....	38
Escalabilidad horizontal .....	40
Millones de usuarios y más allá .....	44

<b>2. Estimación preliminar</b> .....	<b>47</b>
Potencia de dos .....	47
Números de latencia que todo programador debe conocer .....	48
Números de disponibilidad .....	49
Ejemplo: estimación de CPS y requisitos de almacenamiento de Twitter .....	51
Consejos .....	52
<b>3. Estructura para una entrevista de diseño de sistemas</b> .....	<b>55</b>
Un proceso en cuatro pasos para una entrevista de diseño de sistemas eficaz .....	56
Paso 1 – Comprender el problema y establecer el alcance del diseño .....	56
Paso 2 – Proponer un diseño de alto nivel y obtener aceptación .....	58
Paso 3 – Estudiar el diseño con detenimiento .....	60
Paso 4 – Conclusión .....	62
Asignación de tiempos a cada paso .....	65
<b>4. Diseñar un limitador de frecuencia</b> .....	<b>67</b>
Paso 1 – Comprender el problema y establecer el alcance del diseño .....	68
Requisitos .....	69
Paso 2 – Proponer un diseño de alto nivel y obtener aceptación .....	69
¿Dónde colocar el limitador de frecuencia? .....	69
Algoritmos para limitación de frecuencia .....	72
Arquitectura de alto nivel .....	81
Paso 3 – Estudiar el diseño con detenimiento .....	82
Reglas de limitación de frecuencia .....	83
Exceder el límite de frecuencia .....	83
Diseño detallado .....	84
El limitador de frecuencia en un entorno distribuido .....	85
Optimización del rendimiento .....	87
Monitorización .....	88
Paso 4 – Conclusión .....	88
<b>5. Diseñar hashing consistente</b> .....	<b>91</b>
El problema de la redistribución .....	91
Hashing consistente .....	93
Espacio <i>hash</i> y anillo <i>hash</i> .....	94
Servidores <i>hash</i> .....	94
Claves <i>hash</i> .....	95
Búsqueda del servidor .....	96
Añadir un servidor .....	96
Eliminar un servidor .....	97
Dos problemas con el enfoque básico .....	98
Nodos virtuales .....	99
Encontrar las claves afectadas .....	101
Conclusión .....	102

## 6. Diseñar un almacén de clave-valor 105

Comprender el problema y establecer el alcance del diseño .....	106
Almacén de clave-valor de un solo servidor .....	107
Almacén de clave-valor distribuido .....	107
Teorema CAP .....	107
Componentes del sistema .....	110
Partición de datos .....	111
Replicación de datos .....	112
Consistencia .....	113
Resolución de inconsistencias: control de versiones .....	115
Gestión de fallos .....	118
Diagrama de la arquitectura del sistema .....	124
Ruta de escritura .....	125
Ruta de lectura .....	126
Resumen .....	127

## 7. Diseñar un generador de identificador único en sistemas distribuidos 131

Paso 1 – Comprender el problema y establecer el alcance del diseño .....	132
Paso 2 – Proponer un diseño de alto nivel y obtener aceptación .....	133
Replicación multimaestra .....	133
UUID .....	134
Servidor de tickets .....	135
Método <i>Snowflake</i> (copo de nieve) de Twitter .....	135
Paso 3 – Estudiar el diseño con detenimiento .....	136
Marca de tiempo .....	137
Número secuencial .....	138
Paso 4 – Conclusión .....	138

## 8. Diseñar un reductor de URL 141

Paso 1 – Comprender el problema y establecer el alcance del diseño .....	141
Estimaciones preliminares .....	142
Paso 2 – Proponer un diseño de alto nivel y obtener aceptación .....	142
<i>Endpoints</i> API .....	143
Redirección de URL .....	143
Reducción de URL .....	145
Paso 3 – Estudiar el diseño con detenimiento .....	146
Modelo de datos .....	146
Función <i>hash</i> .....	146
Reducción de URL en profundidad .....	150
Redirección de URL en profundidad .....	151
Paso 4 – Conclusión .....	152

## 9. Diseñar un rastreador web 155

Paso 1 – Comprender el problema y establecer el alcance del diseño .....	157
Estimaciones preliminares .....	158
Paso 2 – Proponer un diseño de alto nivel y obtener aceptación .....	159
URL iniciales .....	160
Frontera URL .....	160
HTML <i>downloader</i> .....	160
Resolutor DNS .....	160
Analizador de contenido .....	161
¿Contenido visto? .....	161
Almacenamiento de contenido .....	161
Extractor de URL .....	161
Filtro URL .....	162
¿URL vista? .....	162
Almacenamiento URL .....	162
Flujo de trabajo del rastreador web .....	163
Paso 3 – Estudiar el diseño con detenimiento .....	164
DFS frente a BFS .....	164
Frontera URL .....	166
HTML <i>downloader</i> .....	170
Robustez .....	172
Extensibilidad .....	173
Detectar y evitar contenido problemático .....	174
Paso 4 – Conclusión .....	174

## 10. Diseñar un sistema de notificaciones 177

Paso 1 – Comprender el problema y establecer el alcance del diseño .....	177
Paso 2 – Proponer un diseño de alto nivel y obtener aceptación .....	179
Distintos tipos de notificaciones .....	179
Flujo de recogida de información de contacto .....	182
Flujo de envío y recepción de notificaciones .....	182
Paso 3 – Estudiar el diseño con detenimiento .....	187
Fiabilidad .....	187
Componentes adicionales y otras consideraciones .....	188
Diseño actualizado .....	191
Paso 4 – Conclusión .....	192

## 11. Diseñar un sistema de noticias 195

Paso 1 – Comprender el problema y establecer el alcance del diseño .....	195
Paso 2 – Proponer un diseño de alto nivel y obtener aceptación .....	197
API de <i>news feed</i> .....	197
Publicación de <i>feeds</i> .....	198
Creación de flujos de noticias .....	199

Paso 3 – Estudiar el diseño con detenimiento.....	200
Publicación de <i>feeds</i> en detalle .....	200
Recuperación de <i>news feed</i> en detalle.....	204
Arquitectura de la caché .....	205
Paso 4 – Conclusión.....	206
<b>12. Diseñar un sistema de chat</b>	<b>209</b>
Paso 1 – Comprender el problema y establecer el alcance del diseño .....	210
Paso 2 – Proponer un diseño de alto nivel y obtener aceptación.....	211
Sondeo .....	212
Sondeo largo .....	213
WebSocket .....	214
Diseño en el alto nivel.....	216
Modelos de datos .....	220
Paso 3 – Estudiar el diseño con detenimiento.....	222
Descubrimiento de servicio .....	222
Flujos de mensajes .....	223
Presencia en línea.....	227
Paso 4 – Conclusión.....	230
<b>13. Diseñar un sistema para completar palabras de forma automática en un motor de búsqueda</b>	<b>233</b>
Paso 1 – Comprender el problema y establecer el alcance del diseño .....	233
Requisitos.....	235
Estimación preliminar .....	235
Paso 2 – Proponer un diseño de alto nivel y obtener aceptación.....	236
Servicio de recogida de datos .....	236
Servicio de consulta .....	237
Paso 3 – Estudiar el diseño con detenimiento.....	238
Estructura de datos trie .....	239
Servicio de recogida de datos .....	244
Servicio de consultas .....	247
Operaciones del trie .....	249
Dimensionar el almacenamiento.....	250
Paso 4 – Conclusión.....	252
<b>14. Diseñar YouTube</b>	<b>255</b>
Paso 1 – Comprender el problema y establecer el alcance del diseño .....	256
Cálculos preliminares.....	258
Paso 2 – Proponer un diseño de alto nivel y obtener aceptación.....	258
Flujo de subida de vídeos.....	260
Flujo de transmisión de vídeos.....	264

Paso 3 – Estudiar el diseño con detenimiento.....	266
Transcodificación de vídeo .....	266
Modelo DAG .....	267
Arquitectura de transcodificación de vídeo .....	268
Optimizaciones del sistema.....	274
Manejo de errores.....	280
Paso 4 – Conclusión.....	281

## **15. Diseñar Google Drive** **283**

Paso 1 – Comprender el problema y establecer el alcance del diseño .....	283
Cálculos preliminares.....	286
Paso 2 – Proponer un diseño de alto nivel y obtener aceptación.....	286
Interfaces API.....	287
Cuando un solo servidor no es suficiente.....	289
Conflictos de sincronización.....	291
Diseño en el alto nivel.....	292
Paso 3 – Estudiar el diseño con detenimiento.....	294
Servidores de bloques.....	295
Requisito de alta consistencia.....	296
Base de datos de metadatos .....	297
Flujo de subida .....	298
Flujo de descarga.....	299
Servicio de notificaciones.....	301
Ahorrar espacio de almacenamiento.....	302
Manejo de fallos.....	302
Paso 4 – Conclusión.....	304

## **16. El aprendizaje continúa** **307**

Sistemas reales .....	307
Blogs de ingeniería de empresas .....	309

la fecha de caducidad no sea demasiado corta, ya que eso provocará que el sistema vuelva a cargar los datos de la base con demasiada frecuencia. Asimismo, tampoco es aconsejable que la fecha de caducidad sea demasiado larga, puesto que los datos pueden resultar obsoletos.

- **Consistencia:** Esto implica mantener sincronizado el almacén de datos y la caché. Puede producirse inconsistencia porque las operaciones que modifican datos en el almacén y la caché no estén en una única transacción. Si se amplía a varias regiones, mantener la consistencia entre el almacén de datos y la caché se convierte en un desafío. Para más detalles se puede consultar el artículo "Scaling Memcache at Facebook" publicado por Facebook<sup>7</sup>.
- **Mitigar los fallos:** Un solo servidor caché representa un posible SPOF (*Single Point Of Failure*, punto único de fallo), definido en la Wikipedia de la siguiente manera: "Un punto único de fallo o punto de fallo único (en inglés, *single point of failure*, abreviado SPOF) es un componente de un sistema que tras un fallo en su funcionamiento ocasiona un fallo global en el sistema completo, dejándolo inoperante"<sup>8</sup>. Por lo tanto, se recomienda que los distintos servidores caché de distintos centros de datos eviten los SPOF. Otro planteamiento recomendado es excederse en la cantidad prevista de memoria requerida en determinados porcentajes, disponiéndose así de un búfer de almacenamiento a medida que el uso de la memoria aumenta.
- **Política de desalojo:** Cuando la caché está llena, las siguientes peticiones de añadir elementos a la caché pueden dar como resultado la eliminación de los elementos existentes. Esto se denomina desalojo de caché. LRU (*Least-recently-used*), o menos utilizado recientemente, es la política de desalojo de caché más conocida. Se pueden igualmente adoptar otras políticas de desalojo, como LFU (*Least Frequently Used*, utilizado con menos frecuencia) o FIFO (*First in First Out*, primero en entrar, primero en salir), para ajustarse a distintos escenarios de uso.

7. "Scaling Memcache at Facebook" de R. Nishtala, artículo publicado dentro del 10º simposio USENIX sobre diseño e implementación de sistemas en red (NSDI'13). <https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/nishtala>.

8. Punto único de fallo o SPOF (*Single point of failure*): [https://es.wikipedia.org/wiki/Punto\\_único\\_de\\_fallo](https://es.wikipedia.org/wiki/Punto_único_de_fallo).

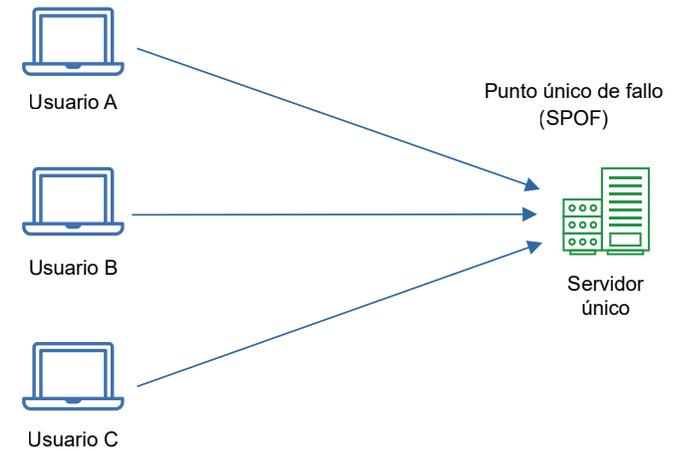


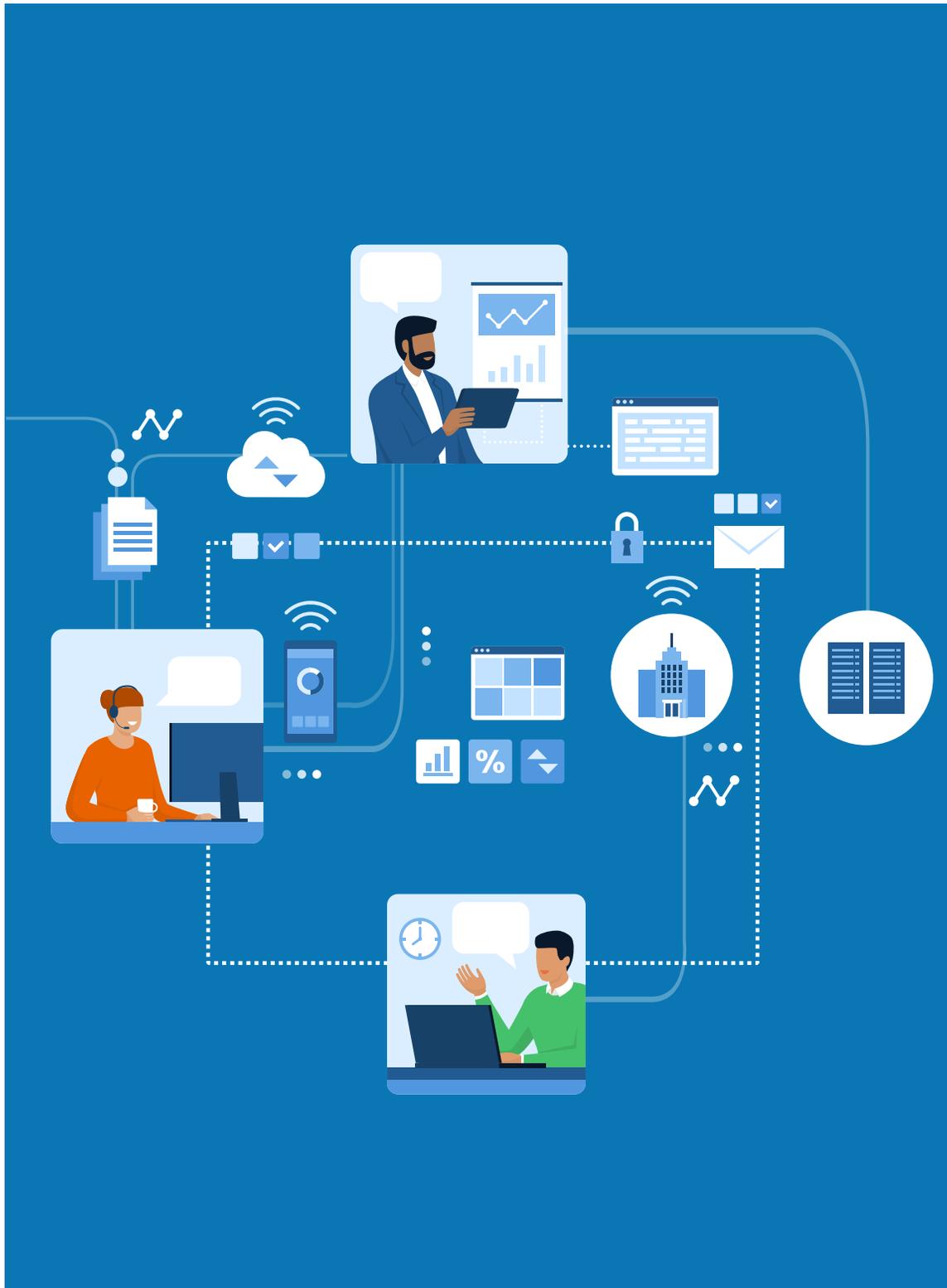
Figura 1.8.

## Red de entrega de contenidos o CDN (*Content Delivery Network*)

Una red de entrega de contenidos o CDN (*Content Delivery Network*) es una red de servidores geográficamente dispersados empleada para entregar contenido estático. Los servidores CDN almacenan en la caché contenido estático como imágenes, vídeos, archivos CSS y JavaScript, etc. El almacenamiento en caché de contenido dinámico es un concepto relativamente nuevo, que está más allá del objetivo de este libro. Permite el almacenamiento en caché de páginas HTML que se basan en *request path* (ruta de petición), *query strings* (cadenas de consulta), *cookies* y cabeceras de petición. Si desea más información al respecto, puede consultar el artículo de Amazon<sup>9</sup> mencionado al pie de esta página. Este libro se centra en cómo utilizar CDN para almacenar en caché contenido estático.

Así es como CDN funciona en el nivel alto: cuando un usuario visita una página web, el servidor CDN situado más cerca del usuario entregará contenido estático. Resulta intuitiva la idea de que, cuanto más lejos estén los usuarios de los servidores CDN, más despacio se cargará la web. Por ejemplo, si los servidores CDN están en San Francisco, los usuarios ubicados en Los Ángeles obtendrán contenido más rápido que los usuarios de Europa. La figura 1.9 es un estupendo ejemplo que muestra cómo CDN mejora el tiempo de carga.

9. Entrega de contenido dinámico de Amazon: <https://aws.amazon.com/es/cloudfront/dynamic-content/>.



### 3 Estructura para una entrevista de diseño de sistemas

Acaba usted de conseguir una de esas codiciadas entrevistas presenciales en la empresa de sus sueños. El coordinador de contrataciones le envía el programa del día. Revisando la lista todo le parece de maravilla, hasta que sus ojos se encuentran con esta parte: la entrevista de diseño de sistemas. A menudo las entrevistas de diseño de sistemas intimidan bastante. Podría ser algo tan indefinido como "¿Puede diseñar un producto X muy conocido?". Las preguntas son ambiguas y parecen desproporcionadamente generales, así que es fácil desanimarse. Después de todo, ¿cómo podría alguien diseñar en una hora un producto conocido cuya creación ha necesitado de la participación de cientos, si no miles, de ingenieros?

La buena noticia es que nadie espera que haga esto. El diseño de sistemas en la vida real es complicado en extremo. Por ejemplo, una búsqueda en Google puede parecer engañosamente sencilla; sin embargo, la cantidad de tecnología que subyace bajo esa simplicidad es sorprendente. Si nadie espera que diseñe un sistema de verdad en una hora, ¿cuál es el beneficio de una entrevista de diseño de sistemas?

Una entrevista de este tipo simula la resolución de problemas de la vida real, como por ejemplo cuando dos compañeros de trabajo colaboran en un problema ambiguo y acaban con una solución que cumple sus objetivos. El problema es abierto y no existe la respuesta perfecta. El diseño final es menos importante comparado con el trabajo invertido en el proceso de diseño, con lo que puede demostrar sus habilidades de diseño, defender sus decisiones y responder a los comentarios de una forma constructiva. Demos la vuelta a la tortilla y pensemos en lo que pasa por la cabeza del entrevistador cuando entra en la sala de reuniones para encontrarse con usted. Su principal objetivo es evaluar con la máxima precisión posible las

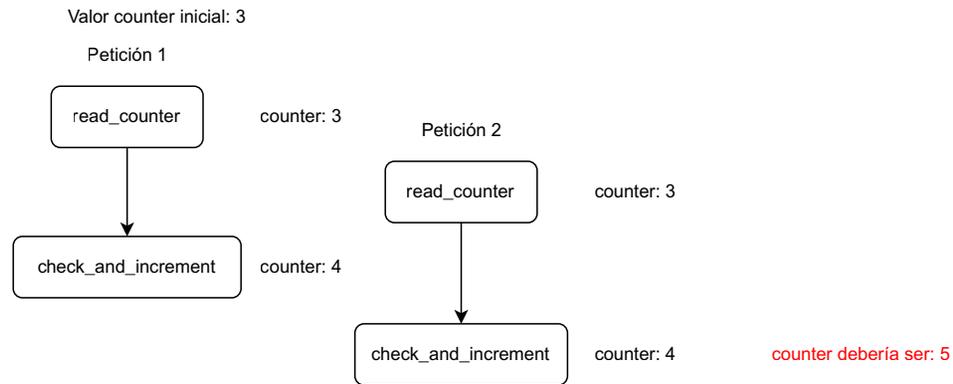


Figura 4.14.

Los bloqueos son la solución más obvia para resolver la condición de carrera. Sin embargo, ralentizan notablemente el sistema. Para resolver el problema se suelen utilizar dos estrategias: código Lua<sup>13</sup> y estructura de datos *sorted sets* (conjuntos ordenados) en Redis<sup>14</sup>.

## El problema de la sincronización

La sincronización es otro factor importante que tener en cuenta en un entorno distribuido. Para soportar millones de usuarios, un solo servidor limitador de frecuencia podría no ser suficiente para gestionar el tráfico. Cuando se utilizan varios servidores, se necesita sincronización. Por ejemplo, en la parte izquierda de la figura 4.15, el cliente 1 envía peticiones al limitador de frecuencia 1 y el cliente 2 las envía al limitador 2. Como la capa web no tiene estado, los clientes pueden enviar peticiones a un limitador de frecuencia distinto, como se muestra en la parte derecha de la figura 4.15. Si no se produce la sincronización, el limitador 1 no contiene datos sobre el cliente 2, de ahí que el limitador no pueda funcionar correctamente.

Una posible solución es utilizar sesiones *sticky* (persistentes), que permiten a un cliente enviar tráfico al mismo limitador de frecuencia. Pero esta solución no es aconsejable, porque no es ni escalable ni flexible. Es mejor método utilizar almacenes de datos centralizados, como Redis. El diseño se muestra en la figura 4.16.

13. "Scaling your API with rate limiters": <https://gist.github.com/ptarjan/e38f45f2dfe601419ca3af937fff574d>.

14. "Better Rate Limiting with Redis Sorted Sets": <https://engineering.classdojo.com/blog/2015/02/06/rolling-rate-limiter/>.

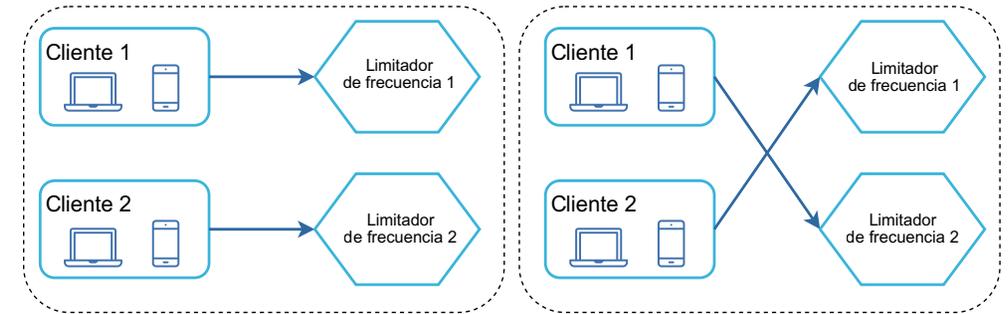


Figura 4.15.

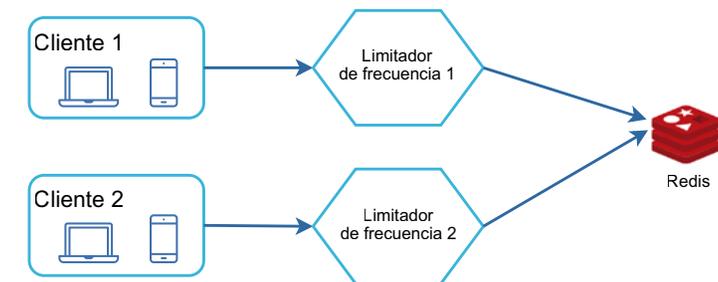


Figura 4.16.

## Optimización del rendimiento

La optimización del rendimiento es un tema habitual en las entrevistas de diseño de sistemas. Hablaremos de dos ámbitos que conviene mejorar.

En primer lugar, la configuración de centros multidados es crucial para un limitador de frecuencia porque la latencia es elevada para usuarios ubicados lejos del centro de datos. La mayoría de los proveedores de servicios de la nube crean muchas ubicaciones de servidor perimetrales por todo el mundo. Por ejemplo, desde el 20 de mayo de 2020, Cloudflare tiene 194 servidores perimetrales geográficamente distribuidos<sup>15</sup>. El tráfico se encamina de forma automática al servidor perimetral más próximo para reducir la latencia.

En segundo lugar, sincronice los datos con un modelo de consistencia provisional. Si no tiene claro esto del modelo de consistencia provisional, consulte la sección "Consistencia" del capítulo 6.

15. Qué es *Edge Computing* o informática de perímetro: <https://www.cloudflare.com/es-es/learning/serverless/glossary/what-is-edge-computing/>.

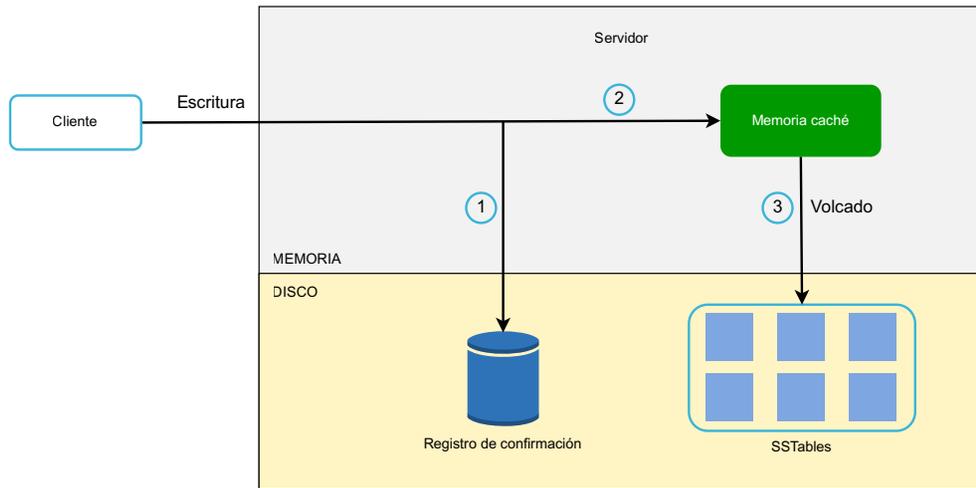


Figura 6.19.

## Ruta de lectura

Una vez que una petición de lectura se dirige a un determinado nodo, este comprueba primero que los datos estén en la memoria caché. Si es así, los datos se devuelven al cliente, como muestra la figura 6.20.

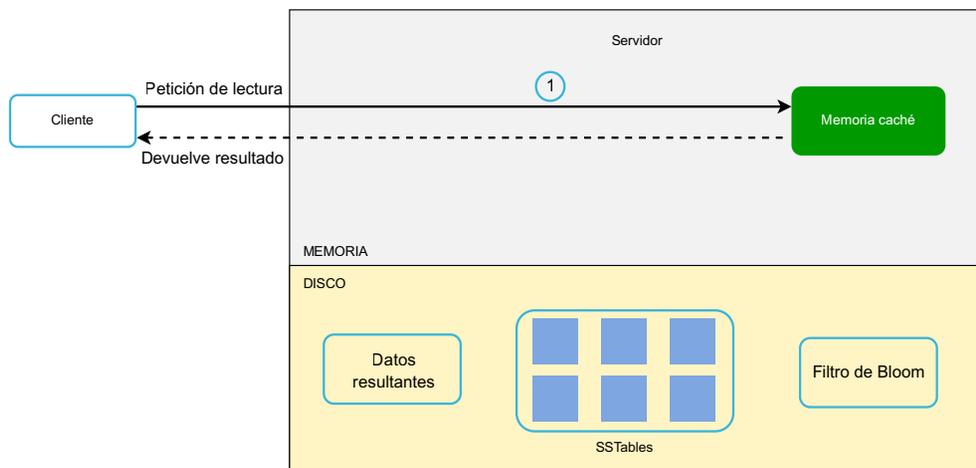


Figura 6.20.

Si los datos no están en la memoria, lo que ocurrirá es que se recuperarán del disco. Necesitamos una manera eficaz de averiguar qué *SSTable* contiene la clave. Para resolver este problema se utiliza el filtro de Bloom<sup>12</sup>.

La ruta de lectura se muestra en la figura 6.21 cuando los datos no están en la memoria.

1. El sistema comprueba antes que nada si los datos están en la memoria. Si no lo están, continúa al paso 2.
2. Si los datos no están en la memoria, el sistema comprueba el filtro de Bloom.
3. El filtro de Bloom se utiliza para intentar averiguar qué *SSTables* pueden contener la clave.
4. Las *SSTables* devuelven el resultado del conjunto de datos.
5. El resultado del conjunto de datos se devuelve al cliente.

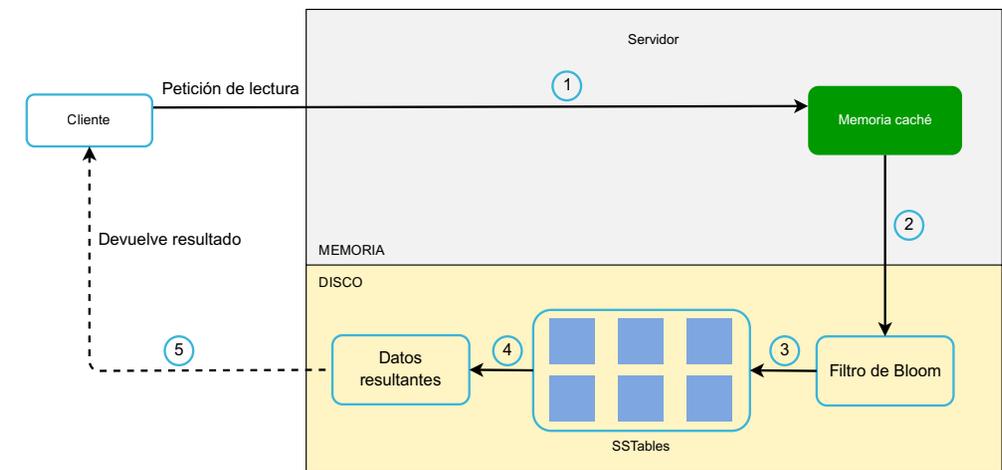


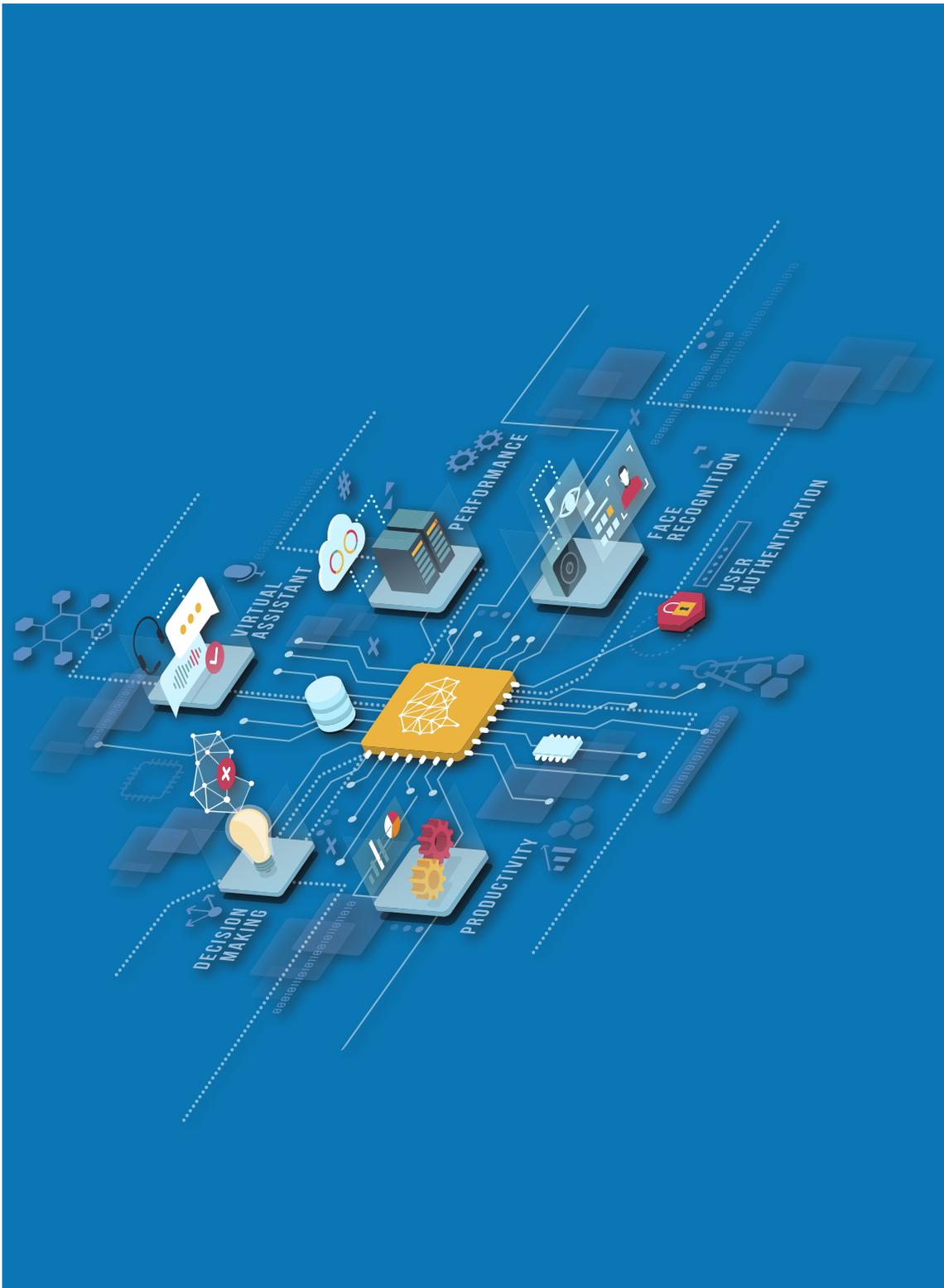
Figura 6.21.

## Resumen

Este capítulo abarca muchos conceptos y técnicas. Para refrescarle la memoria, la siguiente tabla resume los aspectos tratados y las correspondientes técnicas empleadas para diseñar un almacén de clave-valor distribuido.

12. Filtro de Bloom: [https://es.wikipedia.org/wiki/Filtro\\_de\\_Bloom](https://es.wikipedia.org/wiki/Filtro_de_Bloom).

# 9 Diseñar un rastreador web



En este capítulo vamos a centrarnos en el diseño de un rastreador web, una pregunta interesante y bastante habitual en las entrevistas de diseño de sistemas.

Los rastreadores web, también llamados arañas web, son *bots* o robots. Los utilizan los motores de búsqueda para descubrir en la web contenido nuevo o actualizado, por ejemplo una página web, una imagen, un vídeo, un archivo PDF, etc. El rastreador web empieza recopilando páginas web y sigue después los enlaces a dichas páginas para recuperar el contenido nuevo. La figura 9.1 muestra un ejemplo visual del proceso de rastreo.

Una araña web tiene muchos usos diferentes:

- **Indexado de motores de búsqueda:** Es el uso más frecuente. El rastreador recoge páginas web con las que crea un índice local para los motores de búsqueda. Por ejemplo, Googlebot es el rastreador web que está detrás del motor de búsqueda de Google.
- **Archivado web:** Es el proceso de recopilar información de la web con el objetivo de conservar datos para el futuro. Por ejemplo, muchas bibliotecas nacionales manejan rastreadores para archivar sitios web, como la Librería del Congreso de los Estados Unidos<sup>1</sup> y el archivo web de la Unión Europea<sup>2</sup>.
- **Minería web:** El explosivo crecimiento de la web presenta una oportunidad sin precedentes para la minería de datos. Esta metodología ayuda a descubrir conocimiento útil dentro de Internet. Por ejemplo, las empresas

1. Librería del Congreso de los Estados Unidos: <https://www.loc.gov/web-archives/>.

2. Archivo web de la Unión Europea: <https://op.europa.eu/es/web/web-tools/euwebarchive>.

## Flujo de recogida de información de contacto

Para enviar notificaciones es necesario recopilar *tokens* de dispositivo móvil, números de teléfono o direcciones de correo electrónico. Como se muestra en la figura 10.7, cuando un usuario instala nuestra aplicación o se registra por primera vez, los servidores API recogen información de contacto del usuario y la almacenan en la base de datos.

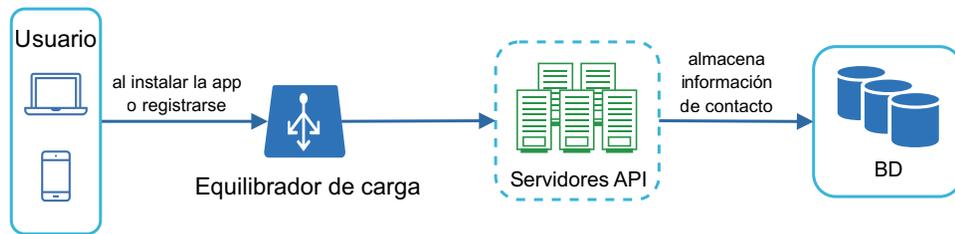


Figura 10.7.

La figura 10.8 muestra tablas de base de datos simplificadas para almacenar información de contacto. Las direcciones de email y los números de teléfono se almacenan en la tabla *user* (usuario), mientras que los *tokens* de dispositivo se guardan en la tabla *device* (dispositivo). Un usuario puede tener varios dispositivos, con lo cual se puede enviar una notificación *push* a todos sus dispositivos.

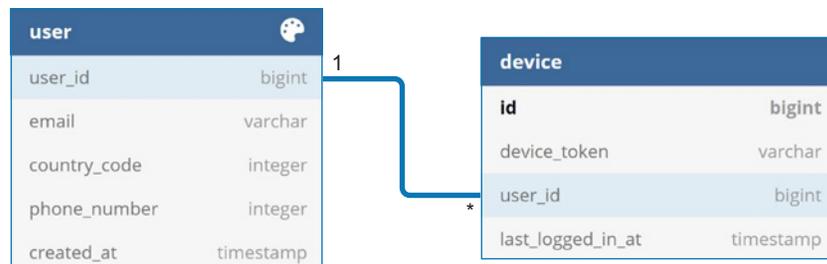


Figura 10.8.

## Flujo de envío y recepción de notificaciones

Presentamos a continuación el diseño inicial y después propondremos algunas optimizaciones.

## Diseño en el alto nivel

La figura 10.9 muestra el diseño, explicándose cada componente del sistema a continuación.

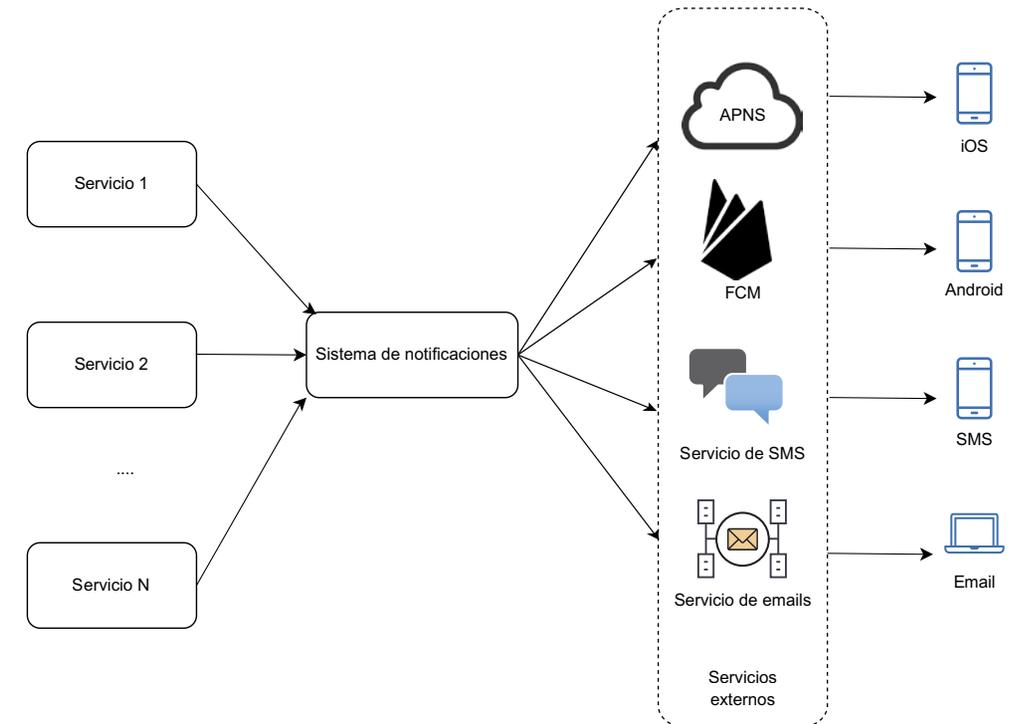


Figura 10.9.

- **Servicio 1 a N:** Puede ser un microservicio, un *cron job* o un sistema distribuido que activa eventos de envío de notificaciones. Por ejemplo, un servicio de facturación envía emails para recordar a los clientes el vencimiento de sus pagos, o un sitio web de compras les dice a sus clientes mediante mensajes SMS que sus paquetes serán entregados mañana.
- **Sistema de notificaciones:** Es la pieza central del envío y recepción de notificaciones. Por empezar por algo sencillo, solo se utiliza un servidor de notificaciones, que proporciona API para los servicios 1 a N y crea cargas útiles de notificación para servicios externos.



# 13 Diseñar un sistema para completar palabras de forma automática en un motor de búsqueda

Cuando se busca o consulta algo en Google o se va de compras en Amazon, al escribir en el cuadro de búsqueda suelen aparecer una o varias alternativas que pueden coincidir con el término buscado. A esta función se le denomina autocompletar, búsqueda anticipada o búsqueda incremental. La figura 13.1 presenta un ejemplo de una búsqueda en Google que muestra una lista de posibles resultados coincidentes cuando se introduce la palabra "dinero" en el cuadro de búsqueda. Estamos frente a una función importante de muchos productos, lo cual nos lleva a la pregunta de la entrevista: diseñe un sistema para completar palabras de forma automática, también formulada como "diseñe un *top k*" o "diseñe las *k* consultas más buscadas".

## Paso 1 – Comprender el problema y establecer el alcance del diseño

El primer paso para enfrentarse a cualquier pregunta en una entrevista de diseño de sistemas es plantearle al entrevistador las cuestiones suficientes que le permitan aclarar los requisitos. Este es un ejemplo de interacción entre candidato y entrevistador:

**Candidato:** ¿Las coincidencias se soportan solamente al principio de la consulta o también a la mitad?

**Entrevistador:** Solo al principio de la búsqueda.

## Paso 4 – Conclusión

En este capítulo hemos propuesto un diseño para crear Google Drive. La combinación de una elevada consistencia, un bajo ancho de banda de red y una rápida sincronización lo hacen interesante. Nuestro diseño contiene dos flujos: gestión de metadatos de archivo y sincronización de archivos. El servicio de notificaciones es otro componente del sistema de gran importancia. Utiliza sondeo largo para mantener a los clientes actualizados con los cambios de archivos.

Como ocurre con cualquier otra pregunta de una entrevista de diseño de sistemas, no existe la respuesta perfecta. Cada empresa tiene sus limitaciones particulares, por lo tanto el sistema que se diseñe se debe adaptar a ellas. Conocer las concesiones y las opciones tecnológicas ofrecidas por el diseño ideado es también primordial. Si queda tiempo al final de la entrevista, puede hablar de las distintas opciones de diseño.

Por ejemplo, se pueden subir archivos directamente a la nube desde el cliente en lugar de pasar por los servidores de bloques. La ventaja de este método es que acelera la subida de archivos, ya que basta con transferir los archivos una sola vez al almacenamiento en la nube. En nuestro diseño el archivo se transfiere primero a los servidores de bloques y después a la nube. No obstante, esta nueva visión tiene varios inconvenientes:

- Primero, hay que implementar la misma lógica de fragmentación, compresión y codificación en distintas plataformas (iOS, Android, Web). Es propensa a los errores y requiere un gran esfuerzo de ingeniería. En nuestro diseño, todas estas lógicas se llevan a cabo en un único lugar: los servidores de bloques.
- Segundo, como es muy fácil manipular un cliente o acceder a él ilegalmente, implementar lógica de codificación en el lado del cliente no es lo ideal.

Otra evolución del sistema que se puede considerar es llevar la lógica de conexión y desconexión a un servicio distinto. Llamémoslo servicio de presencia. Sacando el servicio de presencia de los servidores de notificaciones, otros servicios pueden integrar fácilmente la funcionalidad de conexión y desconexión.

¡Felicidades por haber llegado tan lejos! Dese una palmadita en la espalda. ¡Buen trabajo!

Esta obra es una fuente de información privilegiada, basada en las preguntas que se realizan en las más difíciles entrevistas técnicas. Los entrevistadores piden a los entrevistados que diseñen una arquitectura para un sistema de software, que puede ser un *news feed* (noticias), una búsqueda en Google, un sistema de chat, etc.

Las empresas adoptan las entrevistas de diseño de sistemas porque las habilidades de comunicación y resolución de problemas que se ponen a prueba en estas entrevistas son similares a las requeridas en el trabajo cotidiano de un ingeniero de software. Son evaluados basándose en el modo en el que cada uno analiza un problema impreciso y en cómo resuelve el problema paso a paso. Las habilidades sometidas a prueba implican además el modo en el que explica la idea, discute con otros y evalúa y optimiza el sistema.

El objetivo de este libro es ofrecer una estrategia fiable y una base sólida de conocimiento para abordar no solo las preguntas de diseño de sistemas en una entrevista laboral, sino para que el ingeniero de software pueda contar con la estrategia correcta para crear un sistema escalable o solucionar problemas en la arquitectura de los sistemas.

Esta obra cuenta con:

- Una estructura en cuatro pasos para resolver cualquier pregunta en una entrevista de diseño de sistemas.
- 16 preguntas realizadas en la vida real en entrevistas de diseño de sistemas con soluciones detalladas.
- 188 diagramas que explican de manera visual el funcionamiento de los distintos sistemas.
- Ejemplos que ilustran un acercamiento sistemático y práctico con pasos detallados.