Reto #5: Acortador de URLs Inteligente y Resistente

1. **Contexto del problema:**

Tu empresa necesita un servicio interno para generar URLs cortas que sean fáciles de compartir y recordar, especialmente para campañas de marketing o enlaces de documentación interna. El servicio debe ser robusto, persistente (al menos de forma simulada), y capaz de manejar un volumen moderado de solicitudes. Además, se requiere un mecanismo inteligente para generar los códigos cortos que minimice colisiones.

2. **Requerimientos técnicos:**

- **API HTTP para Acortar y Redirigir:**
    - Implementa un servidor HTTP en Go (net/http o chi).
    - **Endpoint para acortar:** POST /shorten
        - Acepta un cuerpo JSON con la URL larga (ej. {"long_url": "https://muy-larga-url-de-ejemplo.com/path/to/resource?param=value"}).
        - Genera un **código corto único** (ej. "abc12d").
        - Almacena la relación código_corto -> URL_larga.
        - Retorna un JSON con la URL corta generada (ej. {"short_url": "http://localhost:8080/abc12d"}).
    - **Endpoint para redirigir:** GET /{short_code}
        - Toma el short_code de la ruta.
        - Busca la URL larga asociada en el almacenamiento.
        - Si la encuentra, redirige al cliente a la URL larga usando un **redireccionamiento HTTP 301 (Moved Permanently)** o **307 (Temporary Redirect)**, justifica tu elección en el README.
        - Si no la encuentra, devuelve un error 404 Not Found.
- **Generación de Códigos Cortos:**
    - Implementa una función para generar códigos cortos. Este código debe ser:
        - **Aleatorio pero predecible:** Utiliza una combinación de tiempo actual, un generador de números aleatorios y algún hashing básico (ej. MD5 o SHA-1) de la URL larga para generar un identificador base, del cual se extraiga el código corto. Esto ayuda a evitar colisiones y hace que la generación sea "única" para una URL dada.
        - **De longitud fija:** Por ejemplo, 6 a 8 caracteres alfanuméricos.

        - **Colisión-resistente:** Implementa una lógica de reintento si el código generado ya existe en el almacenamiento. Define un límite de reintentos para evitar bucles infinitos.

- **Almacenamiento de Datos (Simulado):**
    - Utiliza un mapa en memoria (map[string]string para short_code -> long_url) para simular la base de datos.
    - El acceso a este mapa debe ser **completamente seguro para la concurrencia** utilizando sync.RWMutex.
- **Manejo de Errores y Validaciones:**
    - Valida la URL de entrada en el POST /shorten (ej. debe ser una URL válida, no vacía).
    - Devuelve errores HTTP apropiados (400 Bad Request, 404 Not Found, 500 Internal Server Error) cuando sea necesario.
    - Maneja el caso de colisiones en la generación de códigos cortos.
- **Estructura del Proyecto:**
    - Separa la lógica de la API (handlers) de la lógica de negocio (generación de códigos, almacenamiento) en paquetes o módulos distintos para mantener la modularidad y la cohesión.

3. **Restricciones:**

- **No se permite el uso de bases de datos reales** (como PostgreSQL, MongoDB, Redis, etc.). El almacenamiento debe ser puramente en memoria, simulando un almacenamiento persistente con un mapa concurrente.
- **No se permiten librerías externas para la generación de códigos cortos.** Debes implementar tu propia lógica utilizando las funciones estándar de Go (crypto, math/rand, time).
- El manejo de rutas y peticiones HTTP debe hacerse directamente con net/http o, como máximo, con chi para un routing más limpio, pero sin abstraer la lógica central del servicio.

4. **Criterios de evaluación:**

- **Dominio de net/http / chi:** Correcta configuración del servidor, manejo de rutas con parámetros, y procesamiento de solicitudes/respuestas HTTP.
- **Diseño de Almacenamiento Concurrente:** Aplicación eficiente y segura de sync.RWMutex (o sync.Mutex) para el almacenamiento en memoria.

- **Lógica de Generación de Códigos Cortos:** Implementación robusta del algoritmo de generación de códigos, incluyendo la resistencia a colisiones y los reintentos.
- **Manejo de Errores y Validaciones:** Exhaustividad en el manejo de errores del cliente y del servidor, retornando códigos de estado HTTP semánticos.
- **Estructura del Código:** Modularidad, separación de responsabilidades y uso de paquetes para organizar la lógica.
- **Uso de Redirecciones HTTP:** Correcta aplicación y justificación del código de estado de redirección (301 vs 307).

5. Preguntas de reflexión previas a codificar:

1. **Generación de Códigos Únicos:** ¿Qué combinación de time, rand y alguna función hash usarías para generar un código que sea suficientemente único y minimice colisiones, sin ser excesivamente complejo? ¿Cómo asegurarías que el código generado contenga solo caracteres alfanuméricos válidos?
2. **Manejo de Colisiones:** Si el código corto generado ya existe, ¿cuál sería la estrategia para generar uno nuevo? ¿Cuántos reintentos serían razonables antes de fallar la solicitud? ¿Cómo evitarías un "live-lock" bajo alta concurrencia?
3. **Concurrencia en el Mapa:** ¿Por qué un map[string]string simple no es seguro para la concurrencia en Go? ¿Cómo sync.RWMutex resuelve este problema para operaciones de lectura y escritura simultáneas?
4. **Elección de Redirección (301 vs 307):** ¿Cuál es la diferencia entre un HTTP 301 Moved Permanently y un HTTP 307 Temporary Redirect? ¿Cuál sería más apropiado para un acortador de URLs y por qué?
5. **Modularidad:** ¿Qué responsabilidades deberían estar en el paquete main, cuáles en un paquete shortener (para la lógica de negocio) y cuáles en un paquete handler (para la API HTTP)?

6. Entrega esperada:

- Un archivo main.go que inicie el servidor HTTP.
- Archivos Go adicionales organizados en paquetes (ej. shortener/service.go, shortener/store.go, handlers/http.go) que contengan la lógica de negocio y los manejadores de HTTP.
- Un archivo README.md explicando:
    - Tu algoritmo de generación de códigos cortos y cómo manejas las colisiones.
    - La justificación de tu elección entre HTTP 301 y 307 para la redirección.

    - Cómo aseguraste la concurrencia en el almacenamiento de URLs.
    - Tu estructura de proyecto y por qué la elegiste.
- Un archivo shortener_test.go (o archivos de prueba por paquete) con pruebas unitarias para la lógica de generación de códigos, el almacenamiento concurrente y las pruebas de integración para los endpoints de la API (POST /shorten, GET /{short_code}). **Asegúrate de usar go test -race** para validar la seguridad concurrente de tu almacenamiento.