Español:

**Ejercicio Técnico: Plataforma de Microblogging Simplificada**

---

# 1. Descripción

Diseñar e implementar, en Go y con arquitectura hexagonal, un servicio backend que reproduzca las funcionalidades básicas de una plataforma de microblogging (tipo Twitter). El objetivo no es recrear todas las características de Twitter, sino entregar un MVP escalable y optimizado para lecturas.

---

# 2. Objetivo

Construir una API que permita a los usuarios:

1. **Publicar tweets** (mensajes de hasta 280 caracteres).
2. **Seguir a otros usuarios**.
3. **Consultar su timeline**, obteniendo los tweets de las cuentas que siguen, ordenados de más recientes a más antiguos.

---

# 3. Requisitos Funcionales

- **Tweets**
    - Endpoint para crear un tweet: máximo 280 caracteres.
    - Endpoint para listar todos los tweets de un usuario.
- **Follow**
    - Endpoint para seguir/dejar de seguir a otro usuario.
    - Validar que no se pueda seguir dos veces al mismo usuario.
- **Timeline**
    - Endpoint para obtener los últimos N tweets de todas las cuentas que el usuario sigue.
    - Paginación por cursor o por página.

---

# 4. Requisitos No Funcionales

- **Escalabilidad**: diseño orientado a soportar millones de usuarios y decenas de millones de tweets.
- **Lecturas optimizadas**: elegir estrategia de almacenamiento (por ejemplo, caché, índices, almacenamiento por colas o materialized views) que minimice latencia en la consulta de timeline.
- **Contenedorización**: todo el servicio debe poder ejecutarse via Docker.
- **Pruebas**: cobertura de unit tests mínimo para los casos de uso (use cases).

---

# 5. Supuestos

- No es necesario implementar autenticación completa ni gestión de sesiones.
- El identificador de usuario (userID) llegará en cada request (por header, parámetro o body).
- Se puede elegir cualquier base de datos (SQL, NoSQL, in-memory) siempre que justifique la elección.

---

## 6. Criterios de Evaluación

1. **Arquitectura**: claridad en el diagrama high-level y en la separación de capas (hexagonal architecture).
2. **Lenguaje y framework**: uso idiomático de Go.
3. **Infraestructura**: Dockerfile(s) y, opcionalmente, docker-compose para levantar todos los componentes.
4. **Persistencia**: modelo de datos adecuado y justificado.
5. **Tests**: unit tests para los casos de uso principales.
6. **Documentación**: README con instrucciones claras para clonar, compilar, testear y ejecutar el servicio.

---

## 7. Entrega

- **Repositorio público** GitHub, con acceso para revisión.
- **README.md** que incluya:
    - Descripción general.
    - Dependencias y pre-requisitos.
    - Pasos para construir y ejecutar con Docker.
    - Endpoints disponibles y ejemplos de uso.

---

Portugues:

**Exercício Técnico: Plataforma de Microblogging Simplificada**

---

## 1. Descrição

Projetar e implementar, em Go com arquitetura hexagonal, um serviço backend que reproduza as funcionalidades básicas de uma plataforma de microblogging (semelhante ao Twitter). O objetivo não é recriar todas as features do Twitter, mas entregar um MVP escalável e otimizado para leitura.

---

## 2. Objetivo

Construir uma API que permita aos usuários:

1. **Publicar tweets** (mensagens de até 280 caracteres).
2. **Seguir outros usuários**.
3. **Consultar sua linha do tempo** (timeline), obtendo os tweets das contas que seguem, ordenados do mais recente ao mais antigo.

---

## 3. Requisitos Funcionais

- **Tweets**
    - Endpoint para criar um tweet: máximo 280 caracteres.
    - Endpoint para listar todos os tweets de um usuário.
- **Follow**
    - Endpoint para seguir/parar de seguir outro usuário.
    - Validar que não seja possível seguir o mesmo usuário duas vezes.
- **Timeline**
    - Endpoint para obter os últimos N tweets de todas as contas que o usuário segue.
    - Paginação por cursor ou por página.

---

## 4. Requisitos Não Funcionais

- **Escalabilidade**: design pensado para suportar milhões de usuários e dezenas de milhões de tweets.
- **Leitura otimizada**: escolher estratégia de armazenamento (por exemplo, cache, índices, filas ou views materializadas) que minimize a latência na consulta da timeline.
- **Conteinerização**: todo o serviço deve rodar via Docker.
- **Testes**: cobertura mínima de unit tests para os casos de uso principais.

---

## 5. Suposições

- Não é necessário implementar autenticação completa nem gerenciamento de sessões.
- O identificador do usuário (userID) será enviado em cada request (por header, parâmetro ou body).
- Qualquer banco de dados pode ser escolhido (SQL, NoSQL, in-memory), desde que a escolha seja justificada.

---

## 6. Critérios de Avaliação

1. **Arquitetura**: clareza no diagrama de alto nível e na separação de camadas (hexagonal architecture).
2. **Linguagem e framework**: uso idiomático de Go.
3. **Infraestrutura**: Dockerfile(s) e, opcionalmente, docker-compose para orquestrar componentes.
4. **Persistência**: modelo de dados adequado e bem justificado.
5. **Testes**: unit tests cobrindo os casos de uso principais.
6. **Documentação**: README com instruções claras para clonar, compilar, testar e executar o serviço.

---

## 7. Entrega

- **Repositório público** no GitHub (ou similar), com acesso para revisão.
- **README.md** contendo:
    - Descrição geral.
    - Dependências e pré-requisitos.

- Passos para build e execução via Docker.
- Endpoints disponíveis e exemplos de uso.