Caso de Estudio

Actualmente somos una startup en crecimiento y queremos crear una aplicación que compita con Obsidian, una plataforma popular para tomar notas. El objetivo es desarrollar un MVP que permita a los usuarios gestionar notas de forma eficiente, añadiendo etiquetas, organizándolas por categorías y utilizando funciones avanzadas como búsqueda y archivo.

Presentación.

Todo el proyecto tiene que estar en un repositorio de git, asegurar de que sea público para visualizar y clonarlo. Ademas del repositorio, adjunta un video de una demo/explicación del código no superior a 15 minutos, como defensa de tu proyecto e implementación.

Enviar el Link del repositorio (o dar acceso) por correo a las siguientes personas

josue.jimenez@dharbor.com

jose.iriarte@dharbor.com

juan.miranda@dharbor.com

francisco.camacho@dharbor.com

Surimana.ponce@dharbor.com

Preference TechStack: Angular 12+ (si usa angular), Java 17+, Spring boot 3 (si usa Java), Python 3.11 (si usa python)

hasta el viernes a las 23:59 pm

PRUEBA Frontend

Requerimientos Funcionales

1. Los usuarios pueden crear cuenta (SignUp) para registrarse en la aplicación.
2. Los usuarios pueden iniciar sesión (Login) en la aplicación para acceder a sus notas.
3. Los usuarios pueden crear, ver, editar y eliminar sus notas.
4. Los usuarios pueden navegar por todas sus notas en un solo lugar.
5. Los usuarios pueden filtrar notas por etiquetas específicas.
6. Los usuarios pueden buscar notas por título.

7. **(Plus)** Los usuarios pueden buscar notas por título, etiquetas o contenido.
8. **(Plus)** Los usuarios pueden archivar notas para mantenerlas separadas de las activas.
9. **(Plus)** Los usuarios pueden desarchivar notas archivadas.
10. **(Plus)** La aplicación debe recordar el estado de los filtros y búsquedas entre sesiones.

Link Figma: https://www.figma.com/design/JNZctFeGJh5U5bR3YB5EAd/DH-MidPath?node-id=0-1&t=wCXoBN7LdNRCyf8s-1

Criterios Para Considerar al Momento de Desarrollar

Los desarrolladores deberán tomar en cuenta los siguientes puntos al momento de implementar el challenge, ya que serán evaluados durante la revisión:

1. **Calidad del Código:**
    - Uso de **nombres descriptivos** para variables, funciones y componentes.
    - **Funciones concisas** que cumplan una única responsabilidad.
    - Uso adecuado de **comentarios** para explicar partes del código complejas o no evidentes.
2. **Arquitectura:**
    - Implementación de una **arquitectura limpia** con separación de responsabilidades (por ejemplo, dividir lógica de negocio, UI y manejo de datos en capas).
    - Uso de **patrones de diseño** adecuados como MVC, MVVM o Clean Architecture, etc.
3. **Manejo del Estado:**
    - Uso eficiente de herramientas para manejar el estado (por ejemplo, NgRx, NGXS).
    - Evitar redundancias en el estado y garantizar la sincronización correcta entre UI y lógica.
4. **Detección de Cambios:**
    - Optimización en el manejo de cambios (ejemplo: ChangeDetectionStrategy.OnPush en Angular).
    - Minimización de renders innecesarios y mejora en el rendimiento.

5. **Accesibilidad:**
    - Adherencia a estándares de accesibilidad como **WCAG**, con navegación por teclado, atributos aria, y buen contraste de colores.
6. **Diseño y UX (Figma LInk):**
    - Diseño atractivo e intuitivo.
    - Flujos de usuario claros y coherentes.
7. **Extras (Nice to Have):**
    - Creatividad en la implementación de funciones adicionales, como notas favoritas, recordatorios de filtros, o configuraciones personalizables.
8. **Optimización y Rendimiento:**
    - Minimización del tamaño de los archivos entregados (ejemplo: uso de Lazy Loading).
    - Gestión eficiente de las solicitudes de red y almacenamiento local si se usa.

Criterio de Evaluación	Descripción	Puntuación Máxima
Funcionalidad	La implementación cumple con los requerimientos funcionales básicos (crear, ver, editar, eliminar, filtrar, buscar).	10
Arquitectura	Uso de una arquitectura limpia, separando responsabilidades de manera adecuada y siguiendo principios como SOLID.	10
Clean Code	Código limpio, legible y mantenible, siguiendo buenas prácticas como nombres descriptivos, funciones concisas y comentarios útiles.	10
Manejo del Estado	Implementación eficiente del manejo del estado, minimizando redundancias y asegurando sincronización correcta entre UI y datos.	10
Detección de Cambios	Optimización en la detección de cambios y renderizado (ej. uso de estrategias como ChangeDetectionStrategy.OnPush)	10
Rendimiento	Gestión eficiente de recursos, minimización del tamaño de los archivos	10

	(Lazy Loading, Tree Shaking) y rendimiento en la carga de datos.	
Extras (Nice to Have)	Creatividad e implementación de funcionalidades adicionales, como temas, modo offline	10
Experiencia de Usuario (UX)	Flujo de usuario intuitivo, interfaz atractiva y funcionalidades fáciles de usar, con una navegación clara y accesible.	10
Uso de Librerías y Dependencias	Uso correcto y justificado de librerías externas, evitando el sobreuso y garantizando que sean compatibles con los requerimientos.	10
Consistencia en el Diseño Visual (Colores y System UI)	Uso de una paleta de colores coherente, un sistema de diseño reutilizable y transiciones que mejoren la experiencia visual.	10

PRUEBA Backend

1. Gestión de Usuarios

- Registro de nuevos usuarios.
- Autenticación de usuario.
- Obtener información del usuario autenticado.

2. Gestión de Notas

- Crear una nueva nota.
- Listar todas las notas del usuario autenticado (con filtros opcionales).
- Ver detalles de una nota específica.
- Editar una nota existente.
- Eliminar una nota.

3. Gestión de Etiquetas

- Crear una nueva etiqueta.
- Obtener todas las etiquetas del usuario.
- Eliminar una etiqueta (opcionalmente con la opción de reasignar notas a otra etiqueta).

4. Funciones Avanzadas (Extra)

- Archivar una nota.
- Desarchivar una nota.
- Búsqueda avanzada por título, etiquetas o contenido.
- Guardar el estado de filtros y búsquedas del usuario.
- Recuperar el estado de filtros y búsquedas guardado.

Criterios Para Considerar en la evaluación

1. Calidad del Código

- Uso de nombres descriptivos para clases, métodos, variables y endpoints.
- Implementación de métodos concisos que cumplan con los principios SOLID.
- Aplicación de estándares de codificación, siguiendo guías de estilo para BE.

2. Arquitectura

- Uso adecuado de patrones de diseño como MVC o repository pattern, según el contexto.
- Configuración y modularización del proyecto para facilitar escalabilidad y mantenimiento.

3. Gestión de Datos

- Diseño eficiente del esquema de la base de datos, asegurando normalización, relaciones bien definidas (en bases SQL) o uso eficiente de colecciones (en bases NoSQL).

Criterio de Evaluación	Descripción	Puntuación Máxima
Funcionalidad	La implementación cumple con los requerimientos funcionales básicos (CRUD, autenticación, roles, validaciones).	20
Arquitectura	Uso de una arquitectura modular y limpia, siguiendo principios como SOLID y patrones como MVC	15
Clean Code	Código limpio, legible y mantenible, con nombres descriptivos, funciones concisas y comentarios útiles.	15
Manejo del Estado	Manejo adecuado de errores y excepciones, con respuestas claras y consistentes (códigos de estado HTTP, mensajes descriptivos).	15
Manejo de Base de Datos	Diseño eficiente de esquemas, consultas optimizadas y uso correcto de transacciones y relaciones entre tablas.	15
Rendimiento	Optimización del rendimiento, tiempos de respuesta rápidos, uso eficiente de recursos y diseño preparado para escalar.	10
Documentacion	Documentación clara de la API (ej. Swagger/OpenAPI), guías de instalación y uso, comentarios útiles en el código.	10

Retos Adicionales Frontend

- Sincronización en tiempo real: Implementar un mecanismo (WebSockets o WebRTC) para que las notas se sincronicen en tiempo real entre múltiples pestañas/dispositivos del mismo usuario.

- Modo offline con sincronización posterior: Permitir que las notas se creen/editen sin conexión y se sincronicen automáticamente cuando vuelva el internet.

- Internacionalización (i18n): Incluir soporte multi-idioma (por ejemplo: español/inglés) con cambio dinámico de idioma.

Retos Adicionales Backend

- Control de versiones de notas: Guardar historial de cambios y permitir revertir a una versión anterior.

- Roles y permisos: Implementar al menos dos roles (ej. usuario y admin), con diferencias en capacidades (admin puede ver y gestionar todas las notas).

- Exportación e importación: API para exportar notas (por ejemplo a JSON o Markdown) y reimportarlas.