Fase I. Introspección

Parte 1

1.- Realizar una propuesta comercial y justificar los siguientes puntos:

Alcance: Implementar, documentar y dejar operando un Gestor de Notas Online con ciclo **DevOps** completo (planificación, control de versiones, CI/CD, despliegue y monitoreo básico).
Entregables:

- Repositorio público con código, y tableros.
- Pipeline CI/CD funcionando (build, checks y despliegue automático).
- Sitio productivo.
- Guía de operación y criterios de aceptación.
  Criterios de éxito: cada cambio en main se valida automáticamente y queda publicado sin intervención manual, con tiempo de ciclo < 5 minutos y tasa de fallos en despliegue ≈ 0.
  Modelo de atención: onboarding, setup inicial, handoff documentado.

1.1 Descripción de la solución

- ¿Qué servicio/solución DevOps proponen?

  - ✓ Diseño e implantación de un flujo DevOps para una aplicación web estática (HTML/CSS/JS)
  - ✓ Repositorio Git con ramas protegidas y PRs.
  - ✓ Integración Continua (CI): validaciones automáticas (linting, pruebas unitarias mínimas y verificación de build).
  - ✓ Entrega/Despliegue Continuo (CD): publicación automática en GitHub Pages al aprobar PRs.
  - ✓ Observabilidad ligera: métricas básicas (tiempo de build, estado de jobs) y verificación de disponibilidad.

1.2 Alcance de la solución

- ¿Qué tipo de servicio será?
  Servicio gestionado de DevOps “llave en mano” con modalidad setup + transición:

- ✓ **Fase Setup:** definición del flujo, configuración de herramientas, pipelines y estándares.
- ✓ **Fase Transición:** capacitación breve, documentación operativa y traspaso de la operación a quien mantenga el proyecto.

1.3 ¿Por qué la aplicación de código abierto planteada debe implementar la metodología DevOps?

- **Trazabilidad y calidad:** cambios versionados, revisados y validados automáticamente; esto evita regresiones.
- **Velocidad y seguridad de entrega:** la posibilidad de automatizar build y despliegue reduce riesgo humano y acelera el time-to-production.
- **Escalabilidad del mantenimiento:** reglas claras (lint, pruebas, PR checks) permiten que contribuidores externos colaboren sin romper el producto.
- **Reproducibilidad:** cualquier persona puede clonar, ejecutar y desplegar con el mismo pipeline.
- **Gobernanza técnica:** políticas de ramas, revisiones y requisitos de CI formalizan el proceso incluso en un proyecto sencillo.

1.4 Herramientas propuestas y su función en el proyecto

- **Git + GitHub**: control de versiones, Pull Requests, protección de ramas, Issues y Projects (kanban).
- **GitHub Actions**: motor de **CI/CD** (workflows para lint, pruebas y despliegue automático a Pages).
- **ESLint + Prettier**: estándares de estilo y calidad de código JS.
- **Jest**: pruebas unitarias sobre utilidades (p. ej., filtrado/búsqueda de notas).
- **GitHub Pages**: hosting estático del sitio productivo.
- **Html, Css y Javascript**
- **Framework:** React

(Las siguientes herramientas están en consideración para el proyecto, pero aún no es nada seguro)

- **Lighthouse CI**: verificación automática de performance y accesibilidad.
- **Dependabot**: actualización automática de dependencias si se añaden.

1.5 Beneficios para el usuario final

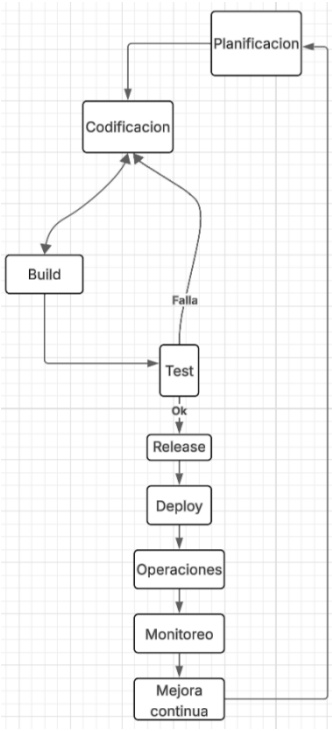
- **Disponibilidad inmediata:** cada mejora aprobada se publica sola; el usuario siempre ve la última versión estable.
- **Mayor calidad:** lint, pruebas y checks reducen errores visibles en UI.
- **Rapidez y simplicidad:** app ligera que carga rápido en cualquier navegador.
- **Transparencia:** historial de cambios público y documentación accesible.
- **Mantenibilidad:** el proyecto sigue mejorando sin interrupciones del servicio.

Parte 2

2.- Planificar y realizar de manera correcta el flujo a seguir de la metodología según sea el proyecto a realizar, considerar: Diagramas, arquitecturas, metodología y explicación de cada proceso a realizar en cada proyecto. (entradas, salidas y procesos).

Diagrama: Flujo Devops

https://lucid.app/lucidchart/c1c6ba49-d479-486b-a872-ee683b1870cd/edit?invitationId=inv_0b90abaf-34ca-4dbb-bb21-17f10e9d68d2



Explicación:

- **Entradas:** requisitos (historia de usuario), backlog, diseño UI/UX, criterios de aceptación.
- **Procesos:** descomposición en tareas, implementar cambios en ramas, ejecutar pipeline.
- **Salidas:** artefacto publicable (sitio estático), release tag, métricas de build/test.
- **Herramientas:** GitHub (issues/PR), GitHub Actions (CI), GitHub Pages (hosting).

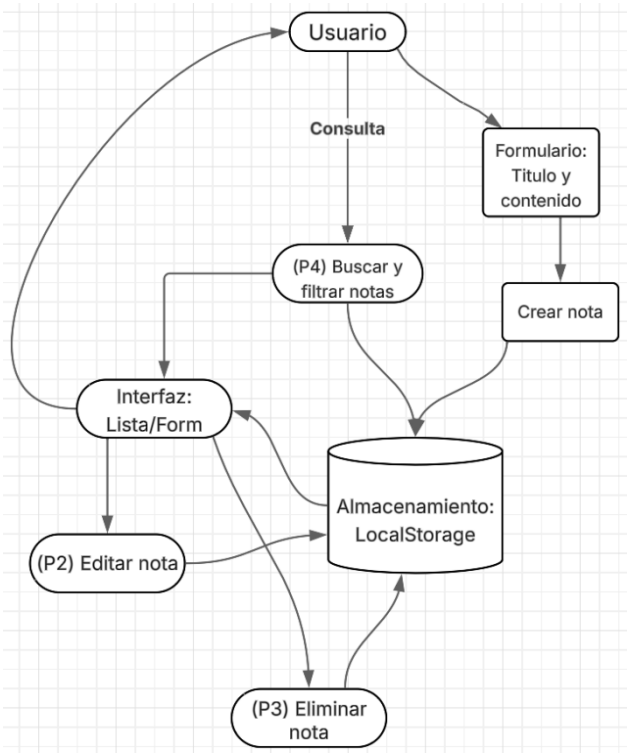
Diagrama de flujo para la gestión de notas:

https://lucid.app/lucidchart/845ec6ee-2f86-4fbd-ac97-0f1fb36f78f7/edit?invitationId=inv_faafb3f4-13e2-4e43-b24f-6a965de675a4

Explicación:

Este diagrama representa cómo funciona una aplicación sencilla de **notas** en donde el usuario puede **crear, editar, eliminar y buscar notas**.

Elementos:



1. **Usuario (User)**

- Es quien interactúa con el sistema.
- Puede ingresar información, consultar notas o realizar acciones sobre ellas.

2. **Formulario (Input: título + contenido)**

- Aquí el usuario escribe el **título** y el **contenido** de la nota.
- Este formulario es el punto de entrada de los datos.

3. **Procesos principales**

- **(P1) Crear nota**: se encarga de tomar los datos del formulario y guardarlos en el almacenamiento.
- **(P2) Editar nota**: permite modificar notas ya creadas y actualizar su información.
- **(P3) Eliminar nota**: borra una nota seleccionada.
- **(P4) Buscar/Filtrar notas**: sirve para consultar las notas almacenadas usando filtros o palabras clave.

4. **Almacenamiento (LocalStorage)**

- Es donde se guardan las notas creadas.
- Permite que los datos se conserven incluso si se recarga la página.
- Como su nombre lo dice el almacenamiento será interno.

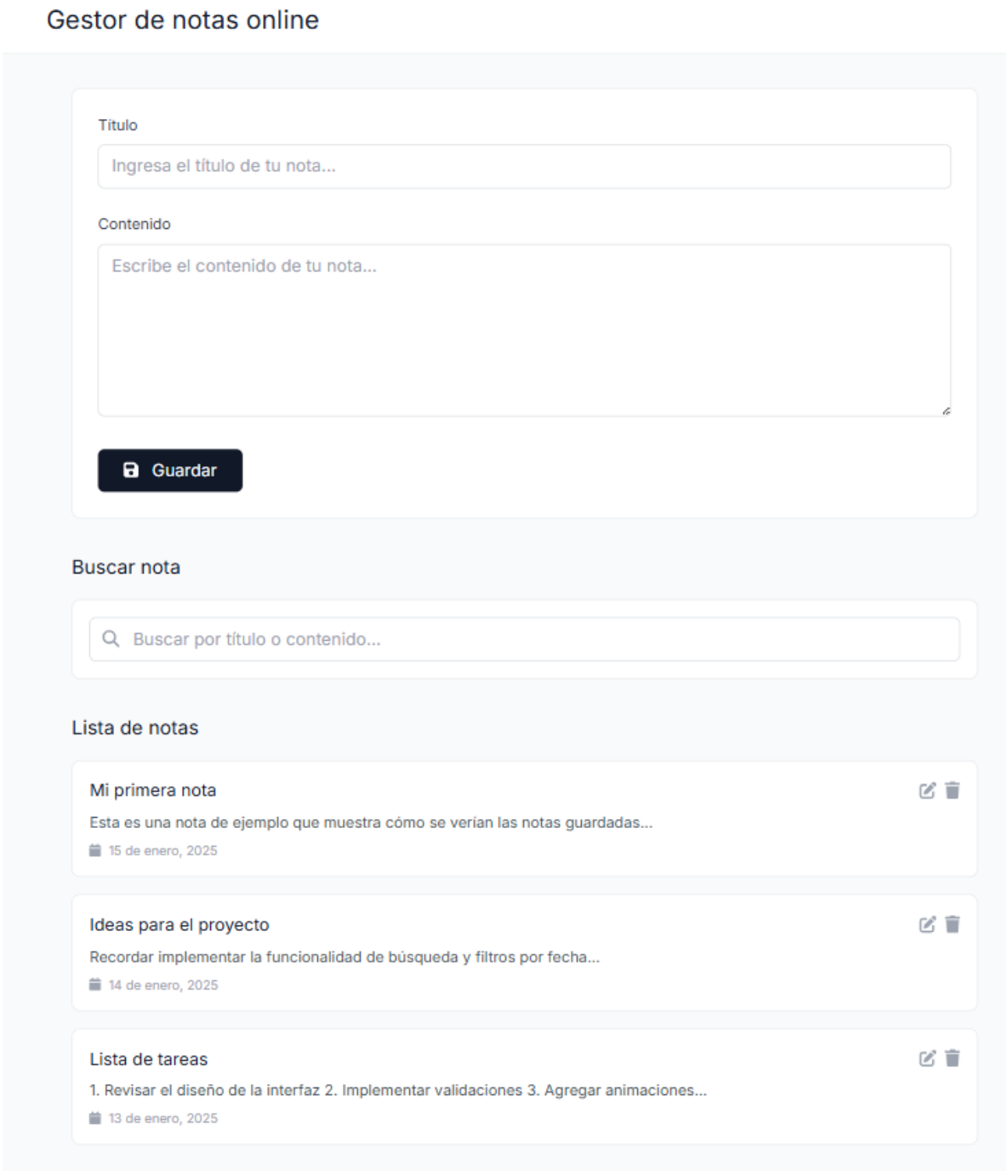
5. **Interfaz (Lista / Form)**

- Es la parte visual que muestra al usuario la **lista de notas** y el **formulario**.
- A través de ella el usuario puede ver sus notas y elegir acciones como editar o eliminar.

Flujo de información

1. El **usuario** introduce una nota en el formulario.
2. Esa información pasa al **proceso de crear nota (P1)**, que la guarda en **LocalStorage**.
3. El **almacenamiento** actualiza la **interfaz**, mostrando la nueva nota.
4. Desde la **interfaz**, el usuario puede:
    - **Editar una nota (P2)** → la actualización se guarda en el almacenamiento.
    - **Eliminar una nota (P3)** → se borra del almacenamiento.
5. El **usuario también puede buscar/filtrar (P4)**, lo que consulta en el almacenamiento y muestra solo los resultados en la interfaz.

3.- Diseño de interfaz representativa (En función de toda la estructura planteada).



(***Este diseño me ayudo una IA a hacerlo, lo edite yo, así logrando tener la idea mas clara de como quiero que quede la primera interfaz)***