Documento de Arquitectura Draw.io (puede ser ARQUITECTURA.md o PDF)

con al menos:

a. Diagrama sencillo de arquitectura (puede ser C4 Nivel 1 ó 2) mostrando:

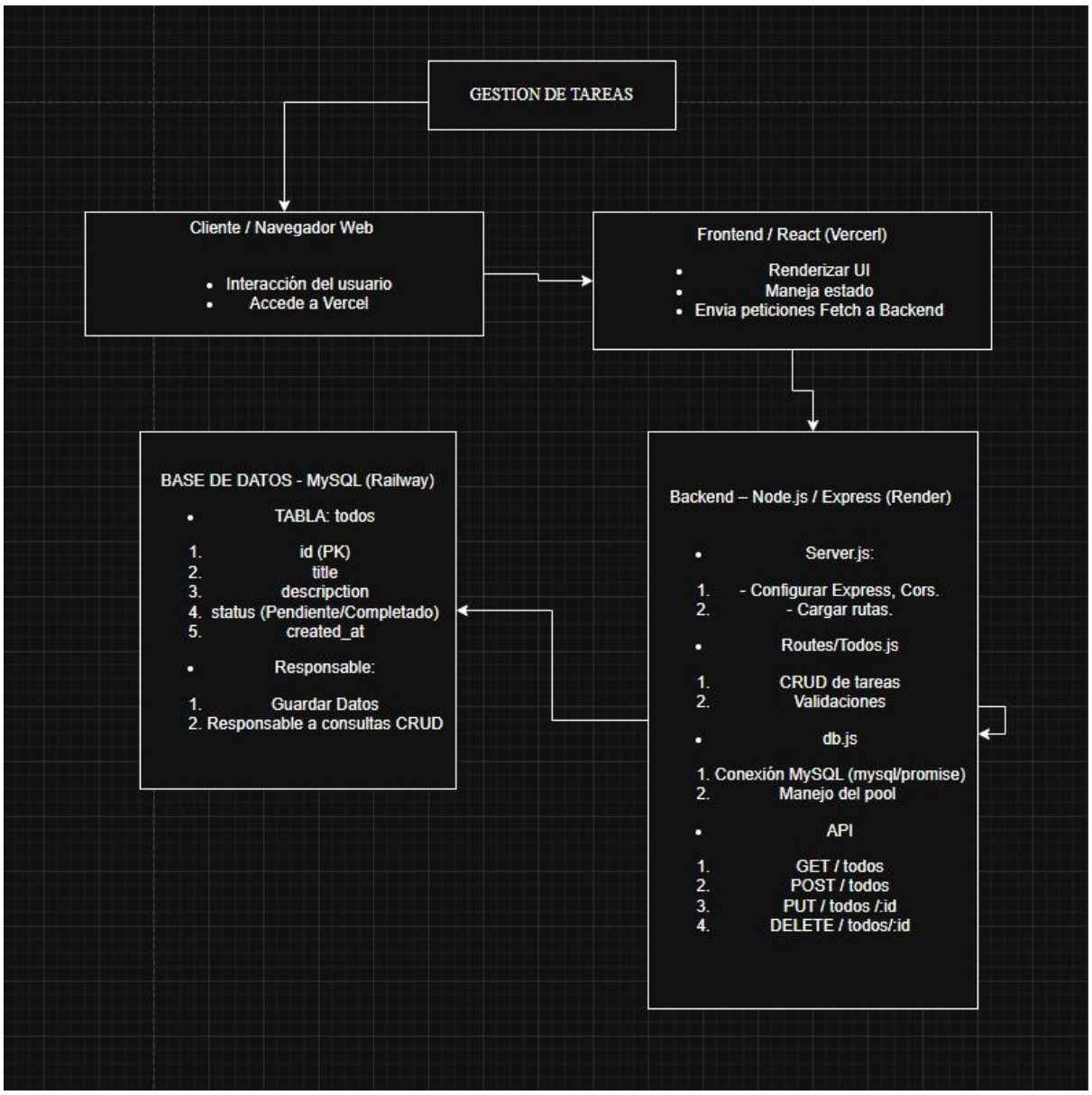
i. Usuario → Frontend (Vercel) → Backend (Render) → DB (Railway).

b. Descripción de los componentes:

i. Frontend (qué hace, principales pantallas).

ii. Backend (capas, rutas, servicios).

iii. Base de datos (modelo de datos, tablas/colecciones).



c. Flujo de una operación típica (por ejemplo: “crear una tarea” paso a

paso).

Paso 1 — Usuario ingresa datos

En el frontend, el usuario escribe:

Título

Descripción (opcional)

Y hace clic en Crear tarea.

Paso 2 — Frontend manda solicitud al backend

El frontend ejecuta:

```
fetch(${VITE_API_URL}/todos, {
  method: "POST",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({ title, description })
})
```

Paso 3 — Backend recibe la petición

Express ejecuta:

Valida que title no esté vacío

Inserta la tarea en MySQL:

INSERT INTO todos (title, description)

VALUES (?, ?)

Paso 4 — Base de datos guarda la tarea

La DB genera:

id

created_at

status = "pendiente"

Paso 5 — Backend responde

{

"id": 15,

"title": "Estudiar",

"description": "Apuntes",

"status": "pendiente"

}

Paso 6 — Frontend actualiza la interfaz

El frontend refresca la lista llamando a:

GET /todos

Y muestra la tarea recién creada.

d. Descripción breve del pipeline de CI (qué se ejecuta y cuándo).

Aunque EL proyecto es sencillo, el objetivo del CI es demostrar:

CI se ejecuta cuando:

Se hace push a main

Se crea un Pull Request

¿Qué incluye el pipeline?

Un workflow simple como este:

```
name: CI

on:
  push:
    branches: ["main"]
  pull_request:
    branches: ["main"]


jobs:
  build:
    runs-on: ubuntu-latest


    steps:
      - uses: actions/checkout@v3
      - name: Install deps
        run: npm install
      - name: Build frontend
        run: cd frontend && npm install && npm run build
```

3. Documentación de API (API.md collecciones o OpenAPI/Swagger):

Descripción:

Esta API permite gestionar tareas (todos) almacenadas en una base de datos MySQL. Incluye endpoints para crear, listar, actualizar y eliminar tareas.

Base URL

Si usas Railway o Vercel:

https://Tabajo-Final.vercel.app/api/todos

a. Endpoints disponibles.

API.md — Documentación de la API de Tareas

Función	Método	Endpoint
Obtener todas las tareas	GET	/api/tasks
Crear una nueva tarea	POST	/api/tasks
Actualizar una tarea	PUT	/api/tasks/:id
Marcar como completada/pendiente	PATCH	/api/tasks/:id/status
Eliminar una tarea	DELETE	/api/tasks/:id

b. Método HTTP.

Cada endpoint utiliza los siguientes métodos:

GET → Consultar información

POST → Crear un nuevo registro

PUT → Actualizar un registro existente

DELETE → Eliminar un registro

c. Body esperado.

POST → Crear tarea

```
{
  "titulo": "Comprar comida",
  "descripcion": "Comprar leche, huevos y pan",
  "estado": "pendiente"
}
```

PUT → Actualizar tarea

```
{
  "titulo": "Comprar comida",
  "descripcion": "Comprar leche, huevos, pan y frutas",
  "estado": "completada"
}
```

Campos de la tabla tareas:

id (auto-incremental)

titulo (varchar)

descripcion (text)

estado (varchar: pendiente / completada)

d. Respuestas de ejemplo.

✓ GET /api/tareas


```
[
 {
  "id": 1,
  "titulo": "Comprar comida",
  "descripcion": "Comprar leche, huevos y pan",
  "estado": "pendiente"
 },
 {
  "id": 2,
  "titulo": "Hacer ejercicio",
  "descripcion": "30 minutos de cardio",
  "estado": "completada"
 }
]
```


✓ GET /api/tareas/1


```
{
 "id": 1,
 "titulo": "Comprar comida",
 "descripcion": "Comprar leche, huevos y pan",
 "estado": "pendiente"
}
```


✓ POST /api/tareas

Respuesta:


```
{
  "message": "Tarea creada correctamente",
  "tarea": {
    "id": 5,
    "titulo": "Estudiar programación",
    "descripcion": "Repasar API REST",
    "estado": "pendiente"
  }
}
```


✓ PUT /api/tareas/1


```
{
  "message": "Tarea actualizada correctamente"
}
```


✓ DELETE /api/tareas/1


```
{
  "message": "Tarea eliminada correctamente"

}
```


e. Códigos de estado HTTP.

Código	**Descripción**
200 OK	Petición exitosa
201 Created	Recurso creado correctamente
400 Bad Request	Datos incompletos o inválidos
404 Not Found	No se encontró la tarea solicitada
500 Internal Server Error	Error del servidor