**Laboratorio Backend-Frontend**

**1. Enunciado**

En equipos de **máximo 2 estudiantes**, desarrollen una **aplicación web Gestor de Tareas** con frontend y backend separados. La app debe permitir: registro e inicio de sesión de usuarios, CRUD completo de tareas (crear, leer, actualizar, eliminar), y un **dashboard** que muestre las tareas agrupadas por estado: *Sin iniciar*, *En progreso*, *Completadas*. Cada grupo elige su *stack* (ej.: React/Vue/Angular + Node/Flask/Django + base de datos relacional/noSQL).

**2. Objetivos de la actividad**

* Aplicar separación frontend/backend y diseño de API REST.
* Implementar autenticación de usuarios (registro/login) y autorización básica.
* Diseñar UI para gestionar tareas y dashboard por estado.
* Persistir datos en base de datos y exponer API segura.
* Documentar y desplegar la aplicación localmente (o en un servicio gratuito).

**3. Requisitos funcionales mínimos**

1. Registro de usuario con validación (nombre, email, contraseña).
2. Inicio de sesión con sesión persistente (JWT o cookies con sesión).
3. Formulario para crear tareas: título, descripción, fecha límite (opcional), prioridad (baja/media/alta), estado (sin iniciar, en progreso, completa).
4. Mostrar un **dashboard** con 3 columnas o secciones: *Sin iniciar*, *En progreso*, *Completadas* con listados y contadores.
5. CRUD de tareas (crear, leer lista, actualizar estado/detalle, eliminar).
6. Acceso protegido: cada usuario solo ve y manipula sus tareas.
7. Validaciones de entrada y manejo de errores (frontend y backend).
8. Documentación mínima: README con instrucciones de instalación/ejecución y lista de endpoints.

**4. Requisitos no funcionales / recomendaciones**

* Código versionado en Git (repo por grupo).
* Tests básicos (al menos pruebas unitarias para 2–3 endpoints o funciones críticas).
* Linter y formateo (ESLint/Prettier para JS; o herramientas equivalentes).
* UI responsiva (móvil y desktop).
* Buenas prácticas REST (status codes, mensajes claros).

**7. Sugerencias tecnológicas**

* Frontend: React (Vite), Vue (Vite), Angular.
* Backend: Node.js + Express, Python Flask/FastAPI, Django REST Framework.
* DB: PostgreSQL, MySQL, SQLite (local), MongoDB.
* Autenticación: JWT (tokens) o sesiones con cookies seguras.

**8. Modelo de datos**

Usuario

* id (UUID / integer)
* name (string)
* email (string, único)
* password\_hash (string)
* created\_at, updated\_at

Tarea (Task)

* id
* user\_id (FK)
* title (string)
* description (text)
* status (enum: "todo"|"in\_progress"|"done")
* priority (enum: "low"|"medium"|"high")
* due\_date (datetime, opcional)
* created\_at, updated\_at

**9. Endpoints API recomendados (REST)**

Autenticación

* POST /api/auth/register — { name, email, password } → 201 / 400
* POST /api/auth/login — { email, password } → 200 + token / 401
* POST /api/auth/logout — (si usa cookies) → 200

Usuarios

* GET /api/users/me — devuelve info del usuario autenticado

Tareas

* GET /api/tasks — lista de tareas del usuario (opciones: ?status=todo)
* POST /api/tasks — crea tarea { title, description, due\_date, priority }
* GET /api/tasks/:id — ver detalle (solo si owner)
* PUT /api/tasks/:id — actualizar tarea (title, description, status, priority, due\_date)
* PATCH /api/tasks/:id/status — actualizar solo status
* DELETE /api/tasks/:id — eliminar tarea

Notas:

* Usar códigos HTTP adecuados (201 creado, 200 OK, 400 bad request, 401/403 auth/authorization, 404 not found).
* Autorización: validar que task.user\_id === auth.user\_id.

**10. Estructura de carpetas recomendada**

A continuación, dos árboles: uno para **frontend** y otro para **backend**. Usa nombres claros, scripts npm, y separa concerns.

**Estructura general del repositorio (monorepo simple)**

/mi-gestor-tareas/ # repo raíz

/frontend/ # app cliente (React/Vue/Angular)

/backend/ # API (Express/Flask/Django)

README.md

.gitignore  
  
**11. Guía rápida de seguridad y buenas prácticas**

* **Nunca** guardar contraseñas en texto; usar bcrypt/argon2 (hash + salt).
* Usar HTTPS en producción.
* Guardar secrets (JWT secret, DB credentials) en variables de entorno (.env) y no en repo.
* Configurar CORS permitiendo solo orígenes necesarios.
* Limitar intentos de login (rate limiting) — al menos mencionar su importancia.
* Validar y sanitizar inputs (evitar inyección).
* Implementar políticas de autorización (cada usuario solo sus recursos).

**12. Instrucciones de entrega (formato)**

Cada grupo entrega:

* Repo Git (GitHub/GitLab) con frontend/ y backend/.
* README principal con:
  + Cómo instalar dependencias y ejecutar (frontend y backend).
  + Variables de entorno requeridas.
  + Endpoints principales y ejemplos de request.
  + Capturas de pantalla del dashboard.
* Scripts npm para iniciar (e.g., npm run dev) y para test (npm test).
* Archivo DEMO.md con credenciales de prueba (user/test) o link a demo.