

## EXAMEN PRÁCTICO FULL STACK DEVELOPER

Nombre: Bryan Manuel Pineda Orózco

DPI: 3001 38695 0101

### Instrucciones:

El objetivo de esta prueba es evaluar tus habilidades como desarrollador Full Stack para diseñar e implementar una aplicación completa utilizando Laravel / NodeJS (backend), React (frontend web) y React Native (aplicación móvil), aplicando buenas prácticas.

Desarrollaras una aplicación llamada “Gestor de tareas”, donde los usuarios puedan:

- Registrarse e iniciar sesión
- Crear, editar, listar y eliminar tareas personales
- Visualizar las tareas tanto en web como en una aplicación móvil

### A. ¿Qué tomamos en cuenta al revisar tu prueba?

- El código entregado tiene que funcionar y cumplir con todo lo solicitado.
- Las instrucciones y la documentación deben ser claras.
- Debe cumplir con los requisitos técnicos solicitados (incluir las librerías / frameworks especificados, versiones pedidas, tests indicados, instrucciones para correrlo, documentación, etc)

### B. ¿Cómo envío la prueba?

Debes dejar tu código en un repo git público (o uno al cual podamos acceder) y luego enviar la información a las personas que te enviaron esta prueba con el URL y datos de acceso.

La estructura debe ser la siguiente:

```
/backend → Laravel / NodeJS + Dockerfile + docker-compose
├── Dockerfile
└── ...

/frontend → React + Dockerfile
├── Dockerfile
└── ...

/mobile-app → React Native CLI o Expo
├── Dockerfile
└── ...

docker-compose.yml
.dockerignore
README.md
```

## **SERIE A - Backend (Laravel / NodeJS + Express):**

### Requerimientos funcionales:

- API RESTful con endpoints:
  - Registro (POST /api/register)
  - Login (POST /api/login)
  - CRUD de tareas (/api/tasks)
- Cada tarea debe tener: title, description, status (pendiente, en progreso, completada), user\_id, created\_at, updated\_at
- Autenticación con Laravel Sanctum o JWT.
- Validaciones mediante Form Request o express-validator.
- Uso de migraciones, seeders y ORM (utilizando como motor de base de datos MySQL).
- Dockerizar con Docker Compose y documentación para ejecutar el servicio en el archivo README.md

### Buenas prácticas esperadas:

- Aplicar el patrón Repository + Service Layer, separando la lógica de negocio del controlador.
- Cumplimiento de principios SOLID.
- Uso de DTOs o Resources para formatear las respuestas del API.
- Nombres consistentes y arquitectura organizada
- Tests unitarios básicos
- Dockerizar con Docker Compose y documentación para ejecutar el servicio en el archivo README.md

## **SERIE B - Frontend Web (ReactJS + TailwindCSS):**

### Requerimientos funcionales:

- Login y registro.
- Pantalla de listado de tareas.
- Crear, editar y eliminar tareas
- Filtro por estado (pendiente, en progreso, completada)
- Interfaz limpia, responsiva y funcional
- Test unitarios básicos
- Dockerizar con Docker Compose y documentación para ejecutar el servicio en el archivo README.md

### Buenas prácticas esperadas:

- Aplicar arquitectura component-based + hooks
- Estructura sugerida:  
*src/*
  - /— api/* → *Axios config y servicios REST*
  - /— components/* → *UI components reutilizables*
  - /— context/* → *Context o store global*
  - /— hooks/* → *Hooks personalizados*
  - /— pages/* → *Páginas principales (Login, Tasks)*
  - /— services/* → *Lógica de negocio (abstracción de API)*
  - utils/* → *Helpers y constantes*
- Separar la lógica de presentación y de negocio
- Implementar manejo de errores y loading states
- Reutilizar componentes (inputs, modales, botones)

## **SERIE C – Frontend Móvil (React Native CLI o Expo):**

### Requerimientos funcionales:

- Login y registro.
- Listado de tareas del usuario autenticado.
- Crear, editar y eliminar tareas
- Manejo de estados y comunicación con el API

### Buenas prácticas esperadas:

- Aplicar patrón Container-Presenter
- Estructura sugerida:  
*src/*
  - /— api/* → *Configuración de endpoints*
  - /— components/* → *UI*
  - /— screens/* → *Vistas principales*
  - /— hooks/* → *Reutilización de lógica*
  - /— navigation/* → *Stack y Tab navigators*
  - /— services/* → *Lógica de negocio*
  - store/* → *Manejo de estado global (Context/Zustand/Redux)*
- Uso de React Navigation para estructura de pantallas.
- Guardar token y sesión con AsyncStorage (CLI) o SecureStore (Expo)
- Manejar loading states y errores de API.
- Código legible, modular y consistente.

- Documentar por medio del README.md los pasos claros de instalación y ejecución.

**OPCIONAL – Bonus:**

- Integrar pruebas unitarias y de integración.
- Propuesta de un pipeline simple de CI/CD con Github Actions.
- Despliegue en alguna nube de su preferencia.