

# INFORME DE RESULTADOS: PRUEBA TÉCNICA

**Candidato:** Desarrollador .NET & Angular

**Presentado por:** Cristian Fidel Rubio Pacheco.

**Fecha:** 27 de diciembre de 2025

**Cargo al que aplica:** Desarrollador Full Stack – MID

---

## Contenido del Documento

1. **Sección 1:** Resolución de Cuestionario Teórico.
2. **Sección 2:** Análisis de Escalamiento y Soporte (Punto 2.3).
3. **Anexo:** Enlaces a Repositorio de Código Fuente y Documentación Técnica.

<https://github.com/Clover2103/Solucion-desarrollador-MID---Cristian-Rubio/tree/main>

# SOLUCIÓN A LAS PREGUNTAS DE LA PRUEBA TEÓRICA

## SECCIÓN 1: CONOCIMIENTO TEÓRICOS.

### 1. El ecosistema .NET: Entendiendo el presente (.NET 10)

Siendo muy transparente, aunque no he profundizado al extremo en todas las versiones de .NET, entiendo perfectamente hacia dónde ha evolucionado el ecosistema, especialmente ahora con **.NET 10**.

- **.NET Framework:** Es la base antigua. Se queda en Windows y hoy en día lo vemos más que todo en mantenimiento de sistemas que ya funcionan. No sería mi primera opción para algo nuevo.
- **.NET 10 (La evolución de Core):** Es el estándar actual. Lo que me atrae de esta versión es su enfoque en el rendimiento extremo y cómo han simplificado el código (con *Minimal APIs*, por ejemplo). Es multiplataforma y es lo que elegiría para cualquier desarrollo moderno que deba ser rápido y escalable.
- **.NET Standard:** Lo veo como el 'pegamento'. Es un contrato que permite que librerías viejas y nuevas hablen el mismo idioma.

**Mi postura:** Mi enfoque es siempre utilizar la versión más estable y moderna, como .NET 10, porque facilita el despliegue en la nube y mejora la productividad del equipo, aunque siempre soy consciente de que en muchas empresas todavía hay mucho valor que mantener en versiones anteriores.

### 2. Angular: Estructura y fluidez

En el frontend, me gusta Angular porque te da un marco de trabajo muy sólido. No tienes que 'inventar la rueda' en cada proyecto.

- **Lo que valoro:** El tipado fuerte de **TypeScript** (que nos evita errores tontos de lógica) y cómo ha evolucionado el manejo del estado con **Signals** en las versiones más recientes. Esto hace que las aplicaciones sean mucho más ligeras.
- **En la UI:** La gran ventaja es que el usuario siente que la aplicación 'vuela'. Al ser una SPA, las transiciones son instantáneas y, gracias a la inyección de dependencias, el código queda muy ordenado y fácil de testear.

### 3. DevOps: Un equipo sincronizado

Para mí, DevOps no es un cargo, es una forma de trabajar. Es dejar atrás esa época donde el desarrollador terminaba su código y 'se olvidaba' de si funcionaba en el servidor o no.

El gran beneficio es la **confianza**. Cuando tienes un flujo de CI/CD bien armado, sabes que tu código será probado y desplegado de forma automática. Esto nos quita el miedo a romper algo y nos permite enfocarnos en lo que realmente importa: resolver problemas de negocio.

#### **4. El flujo Backend y Frontend: Una colaboración**

Lo veo como un equipo donde cada uno tiene una responsabilidad clara pero un objetivo común.

- **Frontend (La cara visible):** Su foco es la interacción. Se encarga de que la experiencia sea intuitiva, de manejar lo que el usuario ve y de comunicarse con el servidor de forma eficiente.
  - *Claves:* Rutas, gestión de la interfaz y consumo de servicios.
- **Backend (El motor):** Es donde ocurre la magia pesada. Valida los datos, asegura que la información esté protegida y gestiona la base de datos.
  - *Claves:* Lógica de negocio, seguridad (como JWT) y persistencia en SQL Server.

## **RESPUES CORRESPONDIENTE AL PUNTO 2.3 EN LA PRUEBA TECNICA.**

### **SECCIÓN 2.3: CONOCIMIENTO PRACTICO.**

Desde mi punto de vista, el problema principal aquí es que no hay una frontera clara de responsabilidades, lo que termina sobrecargando al equipo de desarrollo con temas que no nos corresponden. Mi forma de abordar esto sería la siguiente:

**1. Definir 'quién hace qué':** Me sentaría con el cliente para rayar la cancha. El **Nivel 1** debe ser el filtro: ellos se encargan de lo básico (ayudar al usuario con la contraseña, revisar si tiene internet o si está llenando mal un campo). Mi equipo (**Nivel 2**) entra solo cuando hay un problema real en el código, en la API o en la base de datos que ya construimos.

**2. Armarles una guía rápida:** Para que el Nivel 1 no se sienta perdido, les entregaría un documento sencillo con 'soluciones rápidas'. Si ellos pueden resolver dudas de uso con un manual, a nosotros solo nos llegan los retos técnicos de verdad. Esto nos ahorra tiempo a todos.

**3. Exigir información mínima:** Implementaría una regla simple: para que un caso llegue a mis manos, debe venir con una captura del error y los pasos que hizo el usuario para que fallara. Sin eso, perdemos mucho tiempo adivinando qué pasó.

**Mi valor agregado:** Creo que el éxito de un software no es solo que el código esté limpio, sino que el soporte fluya. Al darle herramientas al Nivel 1, yo puedo concentrarme en mantener la API y la base de datos estables, cumpliendo con los tiempos del contrato y evitando que el cliente se frustre por demoras innecesarias.