TRPC:

APIs Full-Stack Fuertemente Tipadas

Alejandro Vargas A00404840
Juan Jose Muñoz A00450054
David Chicué A00405613
Victoria Restrepo A00405025

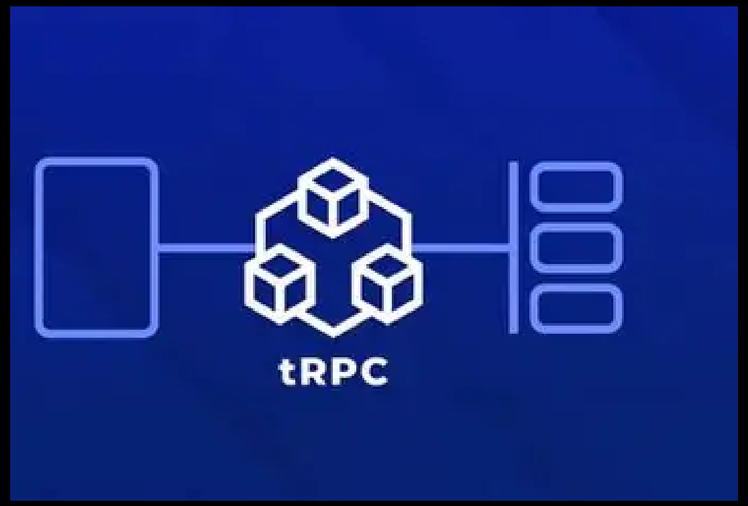
la problematica que aborda

La desconexión Frontend-Backend

- **El mundo REST / GraphQL tradicional:** El backend y el frontend viven en "mundos separados".
- **Duplicación de esfuerzos:** Escribir interfaces en el backend (ej. en Node/Prisma) y volverlas a escribir o generar en el frontend (TypeScript interfaces/types).
- **El infierno del refactor:** Renombrar una propiedad (ej. de userId a authorId) requiere buscar por todo el proyecto.

¿Qué es tRPC?

tRPC es un framework de comunicación que permite construir APIs en TypeScript aplicando el modelo Remote Procedure Call (RPC) mediante Zero Code Generation, exporta el tipo del router del server al cliente HTTP.



¿Que hace tRPC?

logra que una llamada como **trpc.user.get.useQuery(id)** mantenga inferencia de tipos de extremo a extremo sin necesidad de compilar esquemas intermedios ni usar archivos de interfaces manuales.

¿como funciona?

1. Se defines la función en el servidor
2. tRPC exporta el "Tipo" del Router
3. El cliente importa ese tipo como un mapa genérico
4. Invocas la función con autocompletado nativo
5. Validación dual
6. Refactorización en tiempo real

Mecanismo Interno

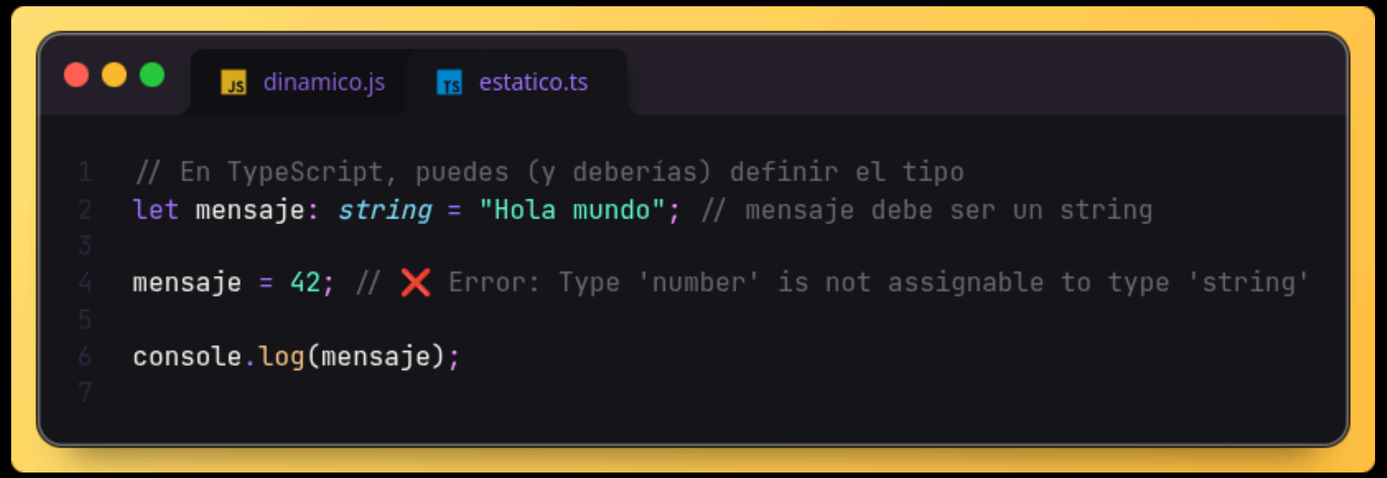
- **1. Arquitectura de Procedimientos:**
  - Organización de la lógica modularizada en AppRouters.
  - Clasificación clara entre operaciones de lectura y modificación.

Mecanismo Interno

- **2. Integración:**
  - Uso de esquemas declarativos en TypeScript para definir la estructura exacta de los datos.
  - Eliminación de validaciones manuales repetitivas en cada endpoint.

Mecanismo Interno

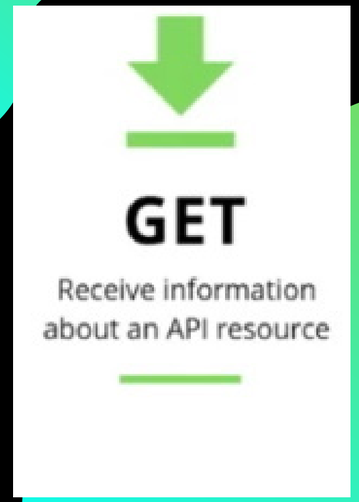
- **3. Doble Capa de Seguridad y Tipado:**
  - **Validación en Runtime (Servidor):** Intercepta y rechaza peticiones con datos malformados o faltantes antes de ejecutar la lógica de negocio.
  - **Inferencia en Compile-time (Cliente):** Extrae automáticamente los tipos de TypeScript desde el esquema, garantizando autocompletado nativo y detección temprana de errores en el editor de código.



Estructura de Código: Queries

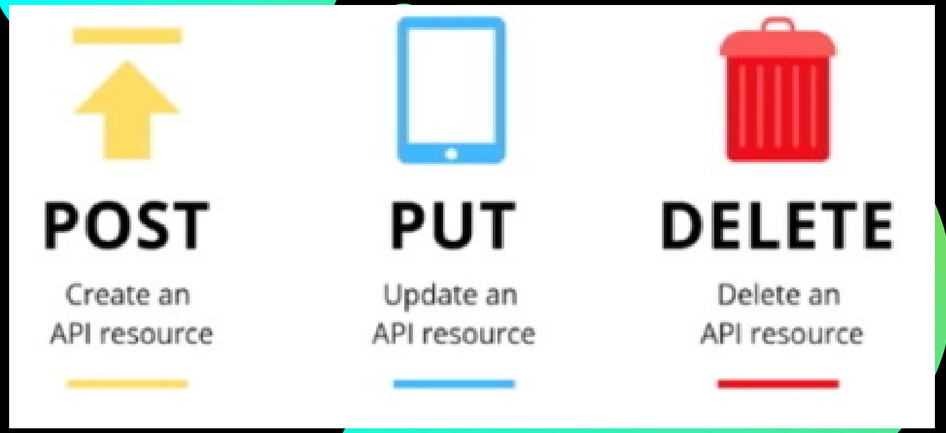
Queries (Procedimientos de Lectura):

- Diseñadas para consultar información sin alterar el estado del sistema (análogo conceptual a métodos GET).
- Integración nativa con librerías de caché y sincronización de estado en el cliente (ej. TanStack React Query).
- **Ejemplo estructural:** Obtener el perfil de un usuario pasando un ID validado mediante un esquema.



Estructura de Código: Mutations

- **Diseñadas para realizar operaciones de escritura,** actualización o eliminación (análogo a métodos POST, PUT, DELETE).
- Ejecutan efectos secundarios en la base de datos tras validar rigurosamente el payload de entrada.
- **Ejemplo estructural:** Registrar un nuevo usuario o actualizar campos específicos de una cuenta existente.



tRPC vs. REST vs. GraphQL

	tRPC:	REST:	GraphQL:
Características:	Protocolo de llamadas a procedimientos remotos tipados para TypeScript.	Estándar universal basado en recursos HTTP.	Lenguaje de consulta y manipulación de datos con un único endpoint.
Ventajas:	Sincronización automática de tipos sin generacion de código	Alta interoperabilidad y compatibilidad con cualquier lenguaje o cliente.	El cliente solicita estrictamente los datos que necesita, evitando el exceso o la escasez de información.
Desventajas:	Acoplamiento estricto al ecosistema TypeScript y no apto para APIs públicas abiertas a clientes heterogéneos.	Requiere mantenimiento manual de documentación y contratos de tipos desacoplados entre el cliente y el servidor.	Curva de aprendizaje elevada y mayor complejidad arquitectónica inicial (sobreingeniería para proyectos pequeños o medianos).

Casos de Uso y Escenarios de Aplicación

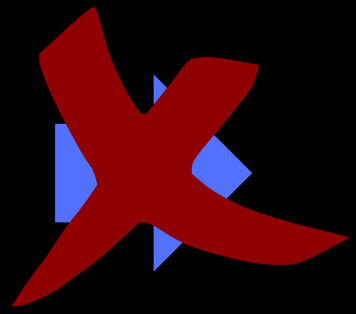
Entornos recomendados

- **Aplicaciones Full-Stack unificadas:** Proyectos que emplean TypeScript de extremo a extremo (ej. Next.js, Remix, T3 Stack) o arquitecturas en Monorepo.
- **Prototipado rápido y MVPs:** Equipos pequeños o medianos donde la velocidad de iteración, el tipado estricto y la refactorización segura reducen el tiempo de entrega.

Casos de Uso y Escenarios de Aplicación

Limitaciones arquitectónicas

- **APIs públicas de libre consumo:** Sistemas que deban ser consumidos por clientes externos desarrollados en lenguajes heterogéneos (Python, Go, Swift, etc.).
- **Microservicios desacoplados:** Entornos corporativos grandes donde el backend y el frontend están separados por equipos independientes y tecnologías heterogéneas.



¿Preguntas?

Referencias

tRPC - Move Fast and Break Nothing. End-to-end typesafe APIs made easy. | tRPC

GRACIAS