

FASE I: BUSINESS ARCHITECTURE

Aprendizaje-Express — ISS-01

Esqueleto del Proyecto Backend: Node.js + Express 5 + TypeScript 5.9 + CommonJS

⚡ GATE de Salida: `npx tsc --noEmit (0 errores)`

El ISS-01 establece la fundación técnica y el ciclo de vida del servidor HTTP

Contrato & Manifiesto

npm & CommonJS

- Definición explícita de 'type': 'commonjs' en package.json.
- Fijación estricta de dependencias clave (Express v5.2, TS ~5.9.2).
- Scripts de ejecución sin contaminación del disco mediante ts-node.
- Aislamiento total de librerías de producción vs. desarrollo.

Tipado & Compilador

TypeScript Strict

- Configuración de tsconfig.json con 'strict': true activo desde el día 1.
- Separación clara entre código fuente (./src) y compilado (./dist).
- Detección temprana de errores de nulidad y tipos en tiempo de compilación.
- Uso del flag --noEmit para validaciones continuas de calidad.

Patrón de Servidor

Clase Encapsulada App

- Encapsulamiento del ciclo de vida de Express en src/config/index.ts.
- Punto de entrada minimalista en src/server.ts (5 líneas).
- Arquitectura por features reservando directorios por dominio.
- Middleware de resiliencia para captura limpia de SyntaxError en JSON.

Gestión rigurosa de paquetes y scripts para desarrollo y producción

Dependencias de Producción (dependencies)

- `express (^5.0.0)`: Framework web HTTP minimalista y rápido.
- `cors (^2.8.5)`: Middleware para política de acceso Cross-Origin.
- `dotenv (^16.4.5)`: Carga de variables de entorno desde `.env`.
- `morgan (^1.10.0)`: Logger de peticiones HTTP para consola.

Dependencias de Desarrollo (devDependencies)

- `typescript (~5.9.2)`: Compilador estricto de tipos en JS.
- `ts-node (^10.9.2)`: Ejecución directa de `.ts` en `Node.js`.
- `nodemon (^3.1.0)`: Reinicio automático del proceso al guardar.
- `@types/*`: Definiciones de tipos para `express`, `cors`, `morgan`, etc.

SCRIPTS PRINCIPALES EN PACKAGE.JSON

- `"build": "tsc"` —> Compila el código de `src/` a `dist/` para distribución/producción.
- `"dev": "nodemon --watch src --ext ts --exec ts-node -- src/server.ts"` —> Ejecución reactiva en memoria.

TypeScript 5.9 garantiza aislamiento de código y seguridad de tipos

```
"rootDir": "./src"
```

Aislamiento de Código Fuente

- Restringe la entrada del compilador tsc únicamente al directorio src/.
- Evita la contaminación del build compilado con scripts raíz o de prueba.
- Garantiza que la estructura interna de módulos sea predecible.

```
"outDir": "./dist"
```

Destino de Distribución

- Redirige la salida JavaScript compilada al directorio dist/.
- Mantiene limpia la raíz del proyecto sin archivos .js dispersos.
- Define el artefacto ejecutable final para entornos de producción.

```
"strict": true
```

Tipado Riguroso

- Activa todas las comprobaciones estrictas de tipo de TypeScript.
- Obliga a manejar explícitamente valores null y undefined.
- Elimina variables imprecisas de tipo 'any' implícito.

Organización modular por dominios de negocio (Feature-First)

```
src/
├── config/
│   └── index.ts # Clase App
├── database/
│   └── seeders/ # Reserva
├── routes/
│   └── index.ts # Enrutador
├── shared/
│   └── errors/ # Middlewares
└── features/
    ├── business/
    │   └── clients/ # Feature
server.ts # Entrypoint
```

src/config/ — Centralización de App & Servidor

Contiene la clase App en index.ts. Encapsula configuraciones, middlewares y ciclo de vida de Express.

src/shared/ — Capa Transversal Compartida

Almacena utilidades globales, manejadores de error HTTP (SyntaxError) y configuración común.

src/routes/ — Enrutador Raíz Centralizado

Agregador que conecta las rutas de cada feature bajo prefijos unificados de la API.

src/features/business/ — Módulos de Dominio (Features)

Organización por dominio. El subdirectorío clients/ aloja el primer CRUD en capas de la Fase I.

Separación entre el punto de entrada (server.ts) y la configuración (App)

```
import { App } from "../config";  
  
const server = new App();  
server.listen();
```

Rol del archivo:

- Única responsabilidad: Instanciar App e iniciar el listener HTTP.
- Archivo ultra-limpio (5 líneas totales).
- Facilita pruebas automatizadas al desvincular la app del puerto.

Clase Encapsulada: src/config/index.ts (App)

- settings(): Configura el puerto HTTP (process.env.PORT || 4000).
- middlewares(): Registra morgan, cors, express.json() y urlencoded.
- routes(): Placeholder agregador de enrutamiento de features.
- docs(): Placeholder reservado para la documentación Swagger UI.
- errorHandler(): Captura errores sintácticos de JSON (bodyErrorHandler).
- dbConnection(): Placeholder para inicialización del ORM Sequelize.
- listen(): Orquesta dbConnection() y levanta app.listen().

Ejecución ordenada desde el constructor de App hasta el listener HTTP

1. constructor()

Instancia Express

```
settings()  
middlewares()  
routes()  
docs()  
errorHandling()
```

2. listen()

Arranque Async

Invoca el método de inicio público desde server.ts

3. dbConnection()

Infraestructura

Sincroniza BD/ORM antes de recibir tráfico

4. app.listen()

Apertura HTTP

 Servidor en puerto 4000 escuchando

El middleware `bodyErrorHandler` previene fugas de información por JSON malformado

Manejo Seguro de Excepciones Sintácticas

- Escenario: Un cliente envía un body HTTP con sintaxis JSON inválida (ej. comilla faltante).
- Comportamiento por defecto: `express.json()` arroja un `SyntaxError` que, sin capturar, expone un stack trace con rutas del servidor.
- Solución ISS-01: Registro del middleware `bodyErrorHandler` en `errorHandling()` después de las rutas.
- Resultado: Intercepta el `SyntaxError` y responde inmediatamente un HTTP 400 Bad Request estructurado.

```
// src/shared/errors/body-error.middleware.ts
export const bodyErrorHandler = (err, req, res, next) => {
  if (err instanceof SyntaxError && 'body' in err) {
    return res.status(400).json({
      error: 'Malformed JSON body'
    });
  }
  next(err);
};

// Respuesta HTTP devuelta al Cliente:
HTTP/1.1 400 Bad Request
Content-Type: application/json
{ "error": "Malformed JSON body" }
```

MATRIZ DE ACEPTACIÓN — GATE DE SALIDA (ISS-01)

Criterio de Aceptación	Comando / Verificación	Resultado Esperado
1. Compilación de Tipos	<code>npx tsc --noEmit</code>	Exit Code 0 (0 errores de TypeScript)
2. Árbol de Archivos Base	<code>find src -type f sort</code>	src/server.ts y src/config/index.ts creados
3. Ejecución del Servidor	<code>npm run dev</code>	 Servidor ejecutándose en puerto 4000

TRANSICIÓN AL SIGUIENTE HITO: ISS-02 (INFRAESTRUCTURA DE BD)

Con el esqueleto HTTP funcionando, el ISS-02 conectará el ORM Sequelize, configurará la BD multi-motor mediante .env y completará el método dbConnection() de la clase App.