Taller de Práctica en Clases: Taller 4

Arquitectura de Servicios: Creación de una API REST con NestJS, TypeORM y SQLite

Carrera:	Ingeniería de Software	**Nivel:**	Quinto	**Período Lectivo:**	2025-2026(1)	**Número de Taller:**	**4**
Docente:	John Cevallos	**Paralelos:**	A y B	**Asignatura:**	Aplicación para el Servidor Web	**Fecha/Horas:**	Octubre 2025 (2 horas académicas)

Tema del Taller

Arquitectura de Servicios: Creación de una API REST con NestJS, TypeORM y SQLite.

Objetivo del Taller

Desarrollar una **API REST completamente funcional** utilizando NestJS, estableciendo conexión a base de datos SQLite mediante TypeORM. Los estudiantes implementarán arquitectura de software por capas para múltiples entidades de su proyecto autónomo, creando controladores, servicios, DTOs y entidades correspondientes para exponer y gestionar recursos a través de *endpoints* HTTP.

Modalidad de Trabajo en Equipo

Grupos de 3 personas: Cada integrante del grupo será responsable de crear las rutas para una porción equitativa de las entidades del proyecto autónomo. La distribución debe ser balanceada según el número total de entidades del proyecto.

Distribución Sugerida:

- **Integrante 1:** Entidades maestras (catálogos, configuraciones, clasificaciones)
- **Integrante 2:** Entidades de negocio principal (operaciones *core* del dominio)
- **Integrante 3:** Entidades transaccionales (movimientos, registros, logs)

Ejemplos por tipo de entidad:

- **Maestras:** Categorías, Estados, Tipos, Países, Monedas, Configuraciones
- **Negocio:** Productos, Servicios, Clientes, Proveedores, Proyectos
- **Transaccionales:** Ventas, Compras, Movimientos, Auditorías, Reportes

Herramientas y Materiales Requeridos

Software Obligatorio:

- **Node.js:** Versión LTS más reciente desde nodejs.org
- **NPM o Yarn:** Gestor de paquetes (incluido con Node.js)
- **NestJS CLI:** Instalar globalmente

```
Bash
# Instalación global de NestJS CLI
npm install -g @nestjs/cli
```

Editores de Código (Opciones):

- **Visual Studio Code:** Editor principal recomendado
- **Cursor:** Editor con AI integrada
- **Windsurf:** Editor colaborativo
- **Extensiones de VSCode:** Cualquier *fork* o extensión compatible

Cliente API REST:

- Postman (recomendado)
- Insomnia
- Thunder Client (extensión VSCode)

Instrucciones Detalladas Paso a Paso

Paso 1: Creación del Proyecto NestJS

Crear un nuevo proyecto colaborativo donde cada integrante trabajará en su porción asignada.

```
Bash
# Crear proyecto NestJS
nest new nombre-proyecto-equipo

# Navegar al directorio
cd nombre-proyecto-equipo
```

Coordinación de Equipo: Establecer un repositorio Git compartido desde el inicio para facilitar la colaboración.

Paso 2: Instalación de Dependencias Principales

Instalar todas las dependencias necesarias para TypeORM, SQLite y validaciones.

```
Bash
# Dependencias principales de base de datos
npm install @nestjs/typeorm typeorm sqlite3

# Dependencias para validación de DTOs (recomendado)
npm install class-validator class-transformer

# Dependencias para documentación API (opcional)
npm install @nestjs/swagger swagger-ui-express
```

Paso 3: Configuración de Base de Datos

Configurar la conexión a SQLite en el módulo principal de la aplicación.

TypeScript

```typescript
// src/app.module.ts
import { Module } from '@nestjs/common';
import { TypeOrmModule } from '@nestjs/typeorm';
import { AppController } from './app.controller';
import { AppService } from './app.service';

@Module({
  imports: [
    TypeOrmModule.forRoot({
      type: 'sqlite',
      database: 'proyecto-equipo.sqlite',
      entities: [__dirname + '/**/*.entity{.ts,.js}'],
      synchronize: true, // Solo desarrollo
      logging: true, // Para debug
    }),
    // Aquí se importarán los módulos de cada entidad
  ],
  controllers: [AppController],
  providers: [AppService],
})
export class AppModule {}
```

Paso 4: Arquitectura por Capas (Para cada Entidad)

Cada integrante debe implementar la siguiente estructura para sus entidades asignadas:

4.1 Entidad (Entity)

TypeScript

```typescript
// src/usuarios/entities/usuario.entity.ts
import { Entity, PrimaryGeneratedColumn, Column } from 'typeorm';

@Entity()
export class Usuario {
  @PrimaryGeneratedColumn()
  id: number;

  @Column({ length: 100 })
  nombre: string;

  @Column({ unique: true })
  email: string;

  @Column()
  fechaCreacion: Date;
}
```

4.2 DTOs (Data Transfer Objects)

TypeScript
```
// src/usuarios/dto/create-usuario.dto.ts
import { IsEmail, IsNotEmpty, IsString } from 'class-validator';

export class CreateUsuarioDto {
  @IsNotEmpty()
  @IsString()
  nombre: string;

  @IsEmail()
  email: string;
}
```
4.3 Servicio (Service)

TypeScript
```
// src/usuarios/usuarios.service.ts
import { Injectable, NotFoundException } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { Repository } from 'typeorm';

@Injectable()
export class UsuariosService {
  constructor(
    @InjectRepository(Usuario)
    private usuarioRepository: Repository<Usuario>,
  ) {}

  async create(createUsuarioDto) {
    const usuario = this.usuarioRepository.create(createUsuarioDto);
    return await this.usuarioRepository.save(usuario);
  }
}
```
4.4 Controlador (Controller)

TypeScript
```
// src/usuarios/usuarios.controller.ts
import { Controller, Get, Post, Body } from '@nestjs/common';

@Controller('usuarios')
export class UsuariosController {
  constructor(private readonly usuariosService) {}

  @Post()
  create(@Body() createUsuarioDto) {
    return this.usuariosService.create(createUsuarioDto);
  }
}
```

Paso 5: Generación Rápida con CLI

Utilizar el CLI de NestJS para generar la estructura base de cada entidad:

Bash
```
# Generar recurso completo
nest generate resource nombre-entidad
```

Actividades Obligatorias

Deberás aplicar los conocimientos adquiridos para crear una API REST para **todas las entidades** de tu proyecto autónomo, distribuidas equitativamente entre los 3 integrantes del equipo.

Para cada una de las entidades asignadas, debes implementar:

1. Crear la Entidad de TypeORM:

- Define una clase Entity con al menos **4 propiedades** (columnas).
- Incluir una clave primaria autoincremental (`@PrimaryGeneratedColumn()`).
- Utilizar decoradores apropiados (`@Column`, `@Entity`, etc.).
- Considerar tipos de datos apropiados y restricciones.

2. Crear los DTOs:

- **`create-[entidad].dto.ts`:** Para la creación del recurso.
- **`update-[entidad].dto.ts`:** Para la actualización del recurso (propiedades opcionales).
- Implementar validaciones con **`class-validator`**.
- Documentar propiedades con comentarios.

3. Implementar el Servicio:

- Inyectar el repositorio de la entidad (`@InjectRepository(Entidad)`).
- Implementar los siguientes métodos obligatorios:

TypeScript
```
// Métodos obligatorios en cada servicio
create(createDto): // Para guardar una nueva entidad
findAll(): // Para obtener todos los registros
findOne(id): // Para obtener un registro por su ID
update(id, updateDto): // Para actualizar un registro
remove(id): // Para eliminar un registro
```

4. Implementar el Controlador:

- Inyectar el servicio correspondiente.
- Definir los siguientes *endpoints* con sus respectivos decoradores HTTP:

TypeScript
```
@Post('/'): // Llama al método create del servicio
@Get('/'): // Llama al método findAll del servicio
@Get('/:id'): // Llama al método findOne del servicio
@Patch('/:id'): // Llama al método update del servicio
@Delete('/:id'): // Llama al método remove del servicio
```

- Utilizar decoradores `@Body()` para recibir DTOs y `@Param('id')` para parámetros de ruta.
- Implementar `ParseIntPipe` para validación de IDs.

5. Configurar el Módulo de la Entidad:

- Asegurar que el módulo importa **TypeOrmModule.forFeature([Entidad])**.
- Declarar el controlador y el proveedor (servicio).
- Exportar el servicio si será usado por otros módulos.

6. Importar los Módulos en el Módulo Principal:

- En **src/app.module.ts**, añadir todos los módulos de entidades en el array de *imports*.
- Verificar que la configuración de TypeORM reconoce todas las entidades.

Importante: Cada integrante debe coordinar con el equipo para evitar conflictos en el repositorio Git y asegurar que todas las entidades se integren correctamente en el proyecto final.

Rúbrica de Evaluación

Criterio de Evaluación	Excelente (4)	Bueno (3)	Satisfactorio (2)	Necesita Mejora (1)
Implementación Técnica	API completa con todas las capas correctamente implementadas, validaciones y manejo de errores	API funcional con mayoría de capas bien implementadas	API básica funcional con algunas capas implementadas	Implementación incompleta o con errores críticos
Comunicación Efectiva	Documentación clara, código comentado apropiadamente, presentación organizada y comprensible	Buena documentación y comunicación con mínimos detalles faltantes	Comunicación básica, documentación suficiente pero podría ser más clara	Comunicación pobre, documentación insuficiente o confusa
Pensamiento Crítico	Decisiones de diseño bien fundamentadas, análisis profundo de problemas, soluciones innovadoras	Buenas decisiones de diseño con justificación adecuada	Decisiones básicas de diseño con alguna justificación	Decisiones poco fundamentadas o análisis superficial
Trabajo en Equipo	Colaboración ejemplar, distribución equitativa del trabajo, integración fluida entre módulos	Buena colaboración con distribución apropiada del trabajo	Colaboración básica, trabajo distribuido pero con algunas deficiencias	Poca colaboración evidente o distribución desigual del trabajo

Criterio de Evaluación	Excelente (4)	Bueno (3)	Satisfactorio (2)	Necesita Mejora (1)
Gestión del Tiempo	Entrega puntual, planificación evidente, uso eficiente del tiempo de taller	Entrega puntual con buena gestión del tiempo	Entrega en tiempo con gestión básica	Entrega tardía o gestión deficiente del tiempo
Resolución de Problemas	Identificación proactiva de problemas, soluciones creativas, debugging efectivo	Buena identificación y resolución de problemas	Resolución básica de problemas con algo de ayuda	Dificultad para identificar o resolver problemas

Puntuación Total: 24 puntos máximos (4 puntos por criterio)

Escala de Calificación: 22-24 = Excelente | 18-21 = Bueno | 14-17 = Satisfactorio | 10-13 = Necesita Mejora

---

Entregables del Taller

1. Código del repositorio Git Colaborativo completo debe ser copiado a todos los repositorios individuales de prácticas

- Código fuente completo del proyecto NestJS en los repositorios individuales.
- Implementación de todas las entidades del proyecto autónomo.
- Commits organizados por integrante del equipo.
- *Branching strategy* clara.

2. Archivo README.md

- Descripción breve del proyecto.
- Instrucciones de instalación y ejecución.
- Documentación de entidades implementadas.
- Ejemplos de uso de la API.

3. Colección de Postman/Ejemplos cURL

- Ejemplos de todos los *endpoints* implementados.
- Casos de prueba para cada operación CRUD.
- Datos de ejemplo para *testing*.

Nota: En caso de presentar su proyecto en clases no es necesario evidenciar sus colecciones de prueba.

Consideraciones Técnicas

Validación de Datos: Para implementación avanzada, configurar `ValidationPipe` global para validar automáticamente los DTOs entrantes.

```TypeScript
// src/main.ts
import { ValidationPipe } from '@nestjs/common';

async function bootstrap() {
  const app = await NestFactory.create(AppModule);

  app.useGlobalPipes(new ValidationPipe({
    whitelist: true,
    forbidNonWhitelisted: true,
    transform: true,
  }));

  await app.listen(3000);
}
```

Manejo de Errores: NestJS maneja excepciones comunes. Para entidades no encontradas, usar **`NotFoundException`** para respuestas HTTP apropiadas.

Estructura del Proyecto: Mantener organización coherente para cada módulo. Usar la estructura generada por CLI de NestJS como base.

Versionado de API: Considerar prefijar rutas con versión para futuras actualizaciones. Configurar en **`main.ts`** con **`app.setGlobalPrefix('api/v1')`**.

Enlaces de Referencia

Documentación Oficial de NestJS:

- **Página Principal:** https://nestjs.com/
- **Técnicas: TypeORM:** https://docs.nestjs.com/techniques/database
- **Controladores:** https://docs.nestjs.com/controllers
- **Proveedores (Servicios):** https://docs.nestjs.com/providers

Documentación de TypeORM:

- **Página Principal:** https://typeorm.io/
- **Decoradores de Entidades:** https://typeorm.io/#/entities
- **Relaciones:** https://typeorm.io/#/relations

Documentación de SQLite:

- **Página Oficial:** https://www.sqlite.org/index.html