Universidad Nacional de Colombia - sede Bogotá
Facultad de Ingeniería
Departamento de Sistemas e Industrial
Curso: Ingeniería de Software 1 (2016701)

Tutorial

Nicolás Andrés Castellanos Rico
Juan Felipe Quiroga Medina
Erfán Andrés Triana Duque
Juan David Yopasa Maldonado

1. Lenguaje y framework seleccionados

- **Lenguaje:** TypeScript
- **Framework Frontend:** React
- **ORM:** TypeORM
- **Base de datos:** MySQL (dockerizado)

2. Configuración y levantamiento de la base de datos

Paso 1: Instalar Docker y Docker Compose

Nos aseguramos de tener instalado Docker y Docker Compose en nuestro sistema. Se puede verificar ejecutando:

```
docker --version
docker-compose --version
```

Paso 2: Crear el archivo docker-compose.yml

En la raíz del proyecto, creamos un archivo llamado docker-compose.yml con el siguiente contenido:

```
version: '3.8'

services:
  mysql:
    image: mysql:8.0
    container_name: ubudget_mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: ubudget_db
      MYSQL_USER: ubudget_user
      MYSQL_PASSWORD: ubudget_pass
    ports:
      - "3306:3306"
    volumes:
      - mysql_data:/var/lib/mysql
    networks:
      - ubudget_net

  backend:
    build: ./backend
    container_name: ubudget_backend
    restart: always
```

```
    ports:
      - "3000:3000"
    depends_on:
      - mysql
    environment:
      DB_HOST: mysql
      DB_PORT: 3306
      DB_USER: ubudget_user
      DB_PASSWORD: ubudget_pass
      DB_NAME: ubudget_db
    networks:
      - ubudget_net

volumes:
  mysql_data:

networks:
  ubudget_net:
    driver: bridge
```

Paso 3: Levantar MySQL con Docker Compose

Ejecutamos el siguiente comando en la terminal desde la raíz del proyecto:

```
docker-compose up -d mysql
```

Esto levantará el contenedor de MySQL en segundo plano.

Paso 4: Instalar dependencias para TypeORM y MySQL

En tu proyecto backend, instala las siguientes dependencias:

```
npm install typeorm mysql2 reflect-metadata
npm install --save-dev @types/node typescript ts-node
```

Paso 5: Configurar la conexión con TypeORM

Creamos un archivo data-source.ts en la carpeta src del backend:

```
import { DataSource } from "typeorm";
import { Usuario } from "./entities/Usuario";

export const AppDataSource = new DataSource({
  type: "mysql",
  host: process.env.DB_HOST || "localhost",
  port: parseInt(process.env.DB_PORT || "3306"),
  username: process.env.DB_USER || "user",
  password: process.env.DB_PASSWORD || "password",
  database: process.env.DB_NAME || "myapp",
  synchronize: true,
  logging: false,
  entities: [Usuario],
  migrations: [],
  subscribers: [],
});

AppDataSource.initialize()
  .then(() => {
    console.log("✅ Conexión a MySQL establecida");
  })
  .catch((error) => {
    console.error("❌ Error al conectar con MySQL:", error);
  });
Alternativamente, puedes usar un archivo ormconfig.json:
{
  "type": "mysql",
  "host": "localhost",
  "port": 3306,
  "username": "user",
  "password": "password",
  "database": "myapp",
  "synchronize": true,
  "logging": false,
  "entities": ["src/entities/**/*.ts"],
  "migrations": ["src/migrations/**/*.ts"],
  "subscribers": ["src/subscribers/**/*.ts"]
}
```

3. Archivos .yml necesarios

Archivo docker-compose.yml completo

Dockerfile para el backend

Creamos un archivo Dockerfile en la carpeta backend:

```
FROM node:18-alpine

WORKDIR /app

COPY package*.json ./

RUN npm install

COPY . .

RUN npm run build

EXPOSE 3000

CMD ["node", "dist/index.js"]
```

4. Ejecución de Hola Mundo

Paso 1: Definir la entidad Budget

Creamos un archivo Budget.ts en la carpeta src/entities:

```
import { Entity, PrimaryGeneratedColumn, Column } from "typeorm";

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

  @Column()
  categoria: string;

  @Column("decimal", { precision: 10, scale: 2 })
```

```
  montoPlaneado: number;

  @Column("decimal", { precision: 10, scale: 2 })
  montoGastado: number;
}
```

Creamos el archivo data-source.ts

```
import { DataSource } from "typeorm";
import { Budget } from "./entities/Budget";

export const AppDataSource = new DataSource({
  type: "mysql",
  host: process.env.DB_HOST || "localhost",
  port: parseInt(process.env.DB_PORT || "3306"),
  username: process.env.DB_USER || "ubudget_user",
  password: process.env.DB_PASSWORD || "ubudget_pass",
  database: process.env.DB_NAME || "ubudget_db",
  synchronize: true,
  logging: false,
  entities: [Budget],
});

AppDataSource.initialize()
  .then(() => console.log("✅ Conectado a la base de datos UBudget"))
  .catch((error) => console.error("❌ Error al conectar:", error));
```

Paso 2: Crear el servidor Express con TypeORM

Creamos un archivo index.ts en la carpeta src:

```
import "reflect-metadata";
import express from "express";
import cors from "cors";
import { AppDataSource } from "./data-source";
import { Budget } from "./entities/Budget";

const app = express();
app.use(cors());
app.use(express.json());
```

```
// Crear nuevo presupuesto
app.post("/budgets", async (req, res) => {
  try {
    const { categoria, montoPlaneado, montoGastado } = req.body;
    const repo = AppDataSource.getRepository(Budget);
    const nuevo = repo.create({ categoria, montoPlaneado, montoGastado });
    await repo.save(nuevo);
    res.status(201).json(nuevo);
  } catch (error) {
    res.status(500).json({ error: "Error al crear presupuesto" });
  }
});

// Listar presupuestos
app.get("/budgets", async (_req, res) => {
  try {
    const repo = AppDataSource.getRepository(Budget);
    const budgets = await repo.find();
    res.json(budgets);
  } catch {
    res.status(500).json({ error: "Error al obtener presupuestos" });
  }
});

const PORT = process.env.PORT || 3000;
app.listen(PORT, () =>
  console.log(`🚀 Servidor UBudget en http://localhost:${PORT}`)
);
```

Paso 3: Instalar dependencias adicionales

```
npm install express cors
npm install --save-dev @types/express @types/cors
```

Paso 4: Configurar scripts en package.json

```
{
  "scripts": {
    "dev": "ts-node src/index.ts",
    "build": "tsc",
```

```
    "start": "node dist/index.js"
  }
}
```

Paso 5: Crear el componente React

En el frontend , creamos un componente BudgetsList.tsx:

```
import React, { useEffect, useState } from "react";

interface Budget {
  id: number;
  categoria: string;
  montoPlaneado: number;
  montoGastado: number;
}

function BudgetsList() {
  const [budgets, setBudgets] = useState<Budget[]>([]);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    fetch("http://localhost:3000/budgets")
      .then((res) => res.json())
      .then((data) => {
        setBudgets(data);
        setLoading(false);
      })
      .catch(() => setLoading(false));
  }, []);

  if (loading) return <div>Cargando...</div>;

  return (
    <div>
      <h2>Presupuestos UBudget</h2>
      {budgets.length === 0 ? (
        <p>No hay presupuestos registrados.</p>
      ) : (
        <ul>
          {budgets.map((b) => (
```

```
            <li key={b.id}>
              <strong>{b.categoria}</strong> -- planeado:
${b.montoPlaneado} | gastado: ${b.montoGastado}
            </li>
          ))}
        </ul>
      )}
    </div>
  );
}

export default BudgetsList;
```

Paso 6: Ejecutar todo el proyecto

1. Levantamos la base de datos:

```
docker-compose up -d mysql
```

2. Compilamos y ejecutamos el backend:

```
npm run dev
```

3. Iniciamos el frontend:

```
npm start
```

4. Probamos en el navegador:

http://localhost:3000/budgets