Hank Erubiel Cárdenas Zamora

2-1

Examen – Concierto

Primero, veamos el archivo de conexión:

```
1. const { Sequelize } = require('sequelize')
2. const sequelize = new Sequelize({
3.     dialect: 'sqlite',
4.     storage: './boletos.sqlite'
5. })
6.
7. module.exports = sequelize;
```

Ahora veamos el archivo principal de la aplicación:

```
const express = require ('express')
const boletos = require('./models/boletos')
const bodyParser = require('body-parser')
const app= express()
const puerto = 3000

app.use(bodyParser.json())
//iniciar servidor
app.listen(puerto, () => {
    console.log('servidor iniciado')
})
//ruta
app.post('/boletos', async(req, res) =>{
    const {localidad, fecha, estudiante} = req.body;
    const data = await boletos.findOne({
        where : {
            localidad,
            fecha
        }
    })
    let {precio,descuento}=data;
    if(!estudiante){
        descuento=0;
    }
    const total=precio-(precio*descuento);
    res.send({
        localidad,
        fecha,
        precio,
```

```
        descuento,
        total
    });

});
```

Finalmente, veamos el modelo de boletos:

```
const { DataTypes } = require ("sequelize");
const sequelize = require("../conexion");

const boletos = sequelize.define('boletos', {
    id: {
        type : DataTypes.INTEGER,
        primaryKey: true
    },
    localidad: {
        type: DataTypes.TEXT
    },
    fecha: {
        type: DataTypes.TEXT
    },
    precio: {
        type: DataTypes.DOUBLE
    },
    descuento: {
        type: DataTypes.DOUBLE
    }
}, {
    timestamps: false
})

module.exports = boletos;
```

Los detalles importantes del código:

1. **Sistema de Base de Datos (conexion.js)**:
    - Utiliza SQLite como base de datos
    - Implementa Sequelize como ORM (Object-Relational Mapping)
    - La base de datos se almacena en un archivo local llamado boletos.sqlite
2. **Modelo de Boletos (boletos.js)**:

- Define la estructura de la tabla 'boletos' con los siguientes campos:
- id: Identificador único (clave primaria)
- localidad: Texto que indica la ubicación del asiento
- fecha: Texto que almacena la fecha del evento
- precio: Número decimal que indica el costo del boleto
- descuento: Número decimal que representa el porcentaje de descuento
- No incluye timestamps (created_at, updated_at)

3. **Aplicación Principal (app.js)**:

- Es una aplicación Express.js que corre en el puerto 3000
- Implementa un endpoint POST en /boletos que:
- Recibe datos del boleto (localidad, fecha, si es estudiante)
- Busca en la base de datos el boleto correspondiente
- Calcula el precio final aplicando descuento si es estudiante
- Devuelve un objeto con:
- Localidad
- Fecha
- Precio original
- Descuento aplicado
- Precio total final