MÓDULO 7. DESARROLLO AVANZADO DE BACKEND Y APIS

Tarea 3: Interacción con bases de datos en Node.js

Diego Matilla De La Cierva

Descripción del Desarrollo

En este ejercicio se ha desarrollado una API REST en Node.js utilizando Express que interactúa con una base de datos MySQL a través de Sequelize y optimiza las consultas más frecuentes mediante Redis como sistema de cache. El objetivo ha sido implementar un CRUD completo sobre un modelo de “producto” y, al mismo tiempo, incorporar un mecanismo de cache con TTL de 30 minutos para reducir la carga en la base de datos y mejorar el rendimiento.

.env

```
PORT=3000



SQL_HOST=localhost

SQL_PORT=3306

SQL_DATABASE=tablaName

SQL_USER=user

SQL_PASSWORD=pass



REDIS_HOST=127.0.0.1

REDIS_PORT=6379
```

La aplicación arranca en app.js, donde se inicializan las conexiones a MySQL y Redis, se configura el parseo de JSON y se monta el router de productos en /api/products. Al iniciar, sequelize.authenticate() verifica la conexión y sequelize.sync({ alter: true }) sincroniza el esquema de la tabla.

En routes/productRoutes.js cada petición GET / y GET /:id define antes un cacheKey (all_products o product_<id>) y pasa por el middleware checkCache, que intenta responder directamente desde Redis si los datos ya están cacheados. Solo cuando no hay “cache hit” se invoca el controlador correspondiente.

Los controladores (productController.js) utilizan los métodos CRUD de Sequelize:

- getAllProducts: lee todos los registros con findAll(), llena el cache con setEx(key, TTL, data) y devuelve el array.
- getProductById: recupera un producto por findByPk(), almacena el resultado en Redis y lo retorna.
- createProduct, updateProduct y deleteProduct: tras modificar la base de datos, llaman a la función clearCache para invalidar las claves afectadas (all_products y, en actualizaciones/borrados, también product_<id>), asegurando que las siguientes lecturas reflejen datos actualizados.

La capa de configuración consta de dos ficheros:

- config/database.js: instancia de Sequelize con credenciales leídas de .env.
- config/redisClient.js: cliente Redis que se conecta al arranque y emite logs de estado.

Conclusiones y Observaciones

En conclusión, este ejercicio te ha permitido consolidar de forma práctica cómo trabajar con Sequelize para mapear y gestionar un modelo relacional en MySQL, automatizando la sincronización de esquemas y aprovechando métodos como findAll, findByPk, create, update y destroy para realizar las operaciones CRUD de manera sencilla y declarativa. Al incorporar Redis como sistema de cache, aprendiste a reducir la carga de la base de datos en endpoints de lectura frecuente, implementando un TTL de 30 minutos con setEx y gestionando la invalidación del cache tras cualquier modificación de datos con una función de limpieza (clearCache), lo cual asegura que la información siempre esté actualizada sin renunciar al rendimiento.

Además, la estructura modular —separando configuración, rutas, controladores y middleware— refuerza la mantenibilidad del código y facilita futuras ampliaciones, mientras que el uso de variables de entorno centraliza la configuración sensible y hace la aplicación más portable y segura.