Curso: Datos II
Catedrático: Fernando José Boiton
Auxiliar: Elmer Ramiro García
Fecha de Presentación: Martes 19 de Noviembre 2024

Bitácora Completa del Proyecto Fithub

Christian Alejandro Barrios Rodríguez
Mayco Ivan Castellanos Diaz

1. Descripción General del Proyecto

Fithub es una aplicación web desarrollada como parte de un proyecto académico con el objetivo de permitir a los usuarios gestionar y monitorear su salud mediante el registro de actividad física, nutrición y sueño. La plataforma cuenta con un sistema de autenticación para los usuarios y permite realizar operaciones CRUD en los registros de salud. El proyecto comenzó con una base de datos MySQL y luego fue migrado a MongoDB para aprovechar las características de flexibilidad de una base de datos NoSQL. La arquitectura incluye un frontend en React y un backend con Node.js y Express, ambos gestionados en contenedores Docker.

2. Estructura Inicial del Proyecto en MySQL

Librerías y Configuración Inicial

- **Frontend (React):** Librerías principales instaladas:
  - axios para manejar solicitudes HTTP.
  - react-router-dom para el enrutamiento en el frontend.
  - formik y yup para el manejo de formularios y validación de datos.
- **Backend (Node.js y Express):** Librerías y configuraciones iniciales:
  - mysql2 para manejar la conexión con MySQL.
  - dotenv para la configuración de variables de entorno.
  - jsonwebtoken para autenticación JWT.
  - bcrypt para encriptar contraseñas.
- **Docker:** Se configuró un contenedor para MySQL con persistencia de datos mediante volúmenes y la conexión se estableció usando un archivo docker-compose.yml.

Desafíos en la Configuración Inicial

- **Estructura de Rutas y Modulares:** Dividir las rutas en users, nutrition, sleep y exercise generó cierta duplicación de código en los controladores. La solución fue crear repositorios y controladores separados, organizando el código en archivos más modulares y reutilizables.
- **Migración entre Bases de Datos:** Uno de los principales desafíos enfrentados fue el cambio de MongoDB a MySQL y luego de vuelta a MongoDB, ocasionado por un mal entendimiento de las instrucciones del proyecto. Esto generó atrasos y requirió ajustar nuevamente el modelo de datos y las APIs.

3. Migración a MongoDB

La decisión de migrar a MongoDB fue motivada por la flexibilidad que proporciona NoSQL, especialmente para manejar estructuras de datos variables y no estructuradas.

Configuración de MongoDB en Docker

- **Actualización de docker-compose.yml:** Se configuró un contenedor para MongoDB junto con otro contenedor para Mongo Express, lo cual permitía la administración visual de la base de datos.
- **Problemas de Conexión Inicial:** Durante las pruebas iniciales, el contenedor de MongoDB presentó problemas de autenticación. Esto se solucionó ajustando correctamente las variables de entorno MONGO_INITDB_ROOT_USERNAME y MONGO_INITDB_ROOT_PASSWORD en el archivo docker-compose.yml.

```
Unset
services:
  mongo:
    image: mongo:latest
    container_name: fithub-mongo
    environment:
      MONGO_INITDB_ROOT_USERNAME: root
      MONGO_INITDB_ROOT_PASSWORD: 123a
    ports:
      - "27017:27017"
    volumes:
      - mongo_data:/data/db
    networks:
      - fithub-network

  mongo-express:
    image: mongo-express:latest
    container_name: fithub-mongo-express
    environment:
      ME_CONFIG_MONGODB_ADMINUSERNAME: root
      ME_CONFIG_MONGODB_ADMINPASSWORD: 123a
      ME_CONFIG_MONGODB_SERVER: mongo
      ME_CONFIG_BASICAUTH_USERNAME: user
      ME_CONFIG_BASICAUTH_PASSWORD: lol1
    ports:
      - "8081:8081"  # Asegúrate de que esta línea esté fuera de environment
    networks:
      - fithub-network

networks:
  fithub-network:
```

```
    driver: bridge

volumes:
  mongo_data:

  app:
    build: .
    container_name: fithub-app
    depends_on:
      - mongo
    volumes:
      - .:/usr/src/app
    ports:
      - "3000:3000"
    networks:
      - fithub-network
```

Modificaciones de Código para MongoDB

1. **Actualización de la Conexión de Mongoose:**
   - **Problema:** La conexión de Mongoose arrojaba errores intermitentes debido a configuraciones incorrectas en useNewUrlParser y useUnifiedTopology, opciones ya obsoletas en versiones recientes de mongoose.
   - **Solución:** Se modificó el código para eliminar estas opciones y se añadió un sistema de reintentos para asegurar que el backend se conecte con la base de datos.
2. **Adaptación de los Modelos y Controladores:**
   - **Problema:** La estructura de datos de MySQL estaba basada en un modelo relacional rígido, lo cual no se adaptaba bien a MongoDB.
   - **Solución:** Se crearon modelos de Mongoose para cada entidad (User, Exercise, Nutrition, Sleep) que permitieron manejar datos más flexibles y estructurados en colecciones.
3. **Reconfiguración de Rutas para Manejo de ObjectIds:**
   - **Problema:** MongoDB utiliza ObjectId en lugar de enteros para sus identificadores, lo cual requirió ajustes en las rutas para aceptar estos identificadores.
   - **Solución:** Se actualizó el manejo de IDs en todas las rutas y se realizaron pruebas exhaustivas para evitar problemas de conversión de tipo de datos.

Obstáculos y Errores durante la Migración

- **Autenticación Fallida en Mongo Express:** Al intentar conectarse con Mongo Express, surgieron problemas de autenticación constantes. Estos errores se debían a la configuración incorrecta del usuario y la contraseña en las variables de entorno. Finalmente, se corrigieron las credenciales tanto en el archivo docker-compose.yml como en los archivos de configuración de Mongo Express.
- **Problemas con la Persistencia de Datos en Docker:** Para garantizar que los datos no se borraran después de detener los contenedores, se configuró un volumen de Docker. Inicialmente, los datos se perdían después de cada reinicio. La solución fue asegurar que el volumen estuviera correctamente configurado para la base de datos y Mongo Express.

4. Pruebas de Integración y Rendimiento

Pruebas de la API con Postman

- **Usuarios:** Se probaron las rutas para la creación de usuarios, inicio de sesión, actualización y eliminación de cuentas. También se verificaron los permisos mediante JWT en rutas protegidas.
- **Ejercicio, Nutrición, y Sueño:** Cada entidad fue probada para CRUD completo. Esto incluyó manejar errores en los campos de entrada y asegurarse de que cada solicitud HTTP respondiera con el código de estado adecuado.

Evaluación del Rendimiento con Clinic.js

Para analizar el rendimiento de la aplicación se usó clinic doctor, una herramienta de profiling de Node.js.

Resultados Clave:

1. **Uso de CPU:**
    - **MongoDB:** Picos de hasta **159%** en operaciones intensivas.
    - **MySQL:** Picos más frecuentes pero menos pronunciados (**108%**).

2. **Uso de Memoria:**
    - **MongoDB:** RSS **55 MB**, Heap Usado **19 MB**. Más consumo debido a la flexibilidad estructural.
    - **MySQL:** RSS **48 MB**, Heap Usado **13 MB**. Más eficiente en recursos.

3. **Event Loop Delay:**
    - **MongoDB:** Promedio **10 ms**, picos de **30 ms** por operaciones bloqueantes.
    - **MySQL:** Promedio **7 ms**, picos menores (**25 ms**).

4. **Handles Activos:**
    - **MongoDB: 6-8 handles**, más conexiones simultáneas.
    - **MySQL: 4-6 handles**, mejor optimización de recursos.

Problemas Detectados:

- **MongoDB:** Picos en CPU y Event Loop por escrituras masivas.
- **MySQL:** Picos frecuentes en CPU, limitado para alta concurrencia

Recomendaciones:

- **MongoDB:** Optimizar índices, usar bulkWrite y proyecciones en consultas.
- **MySQL:** Usar EXPLAIN, implementar caché y ajustar el pool de conexiones.

Conclusión:

- **MongoDB:** Más flexible pero demandante en recursos, ideal para datos dinámicos como en Fithub.
- **MySQL:** Más eficiente para datos estructurados y cargas moderadas.

5. Load Testing: Comparativo MongoDB vs MySQL

Resultados Clave del Load Testing:

1. **MySQL:**
    - **Promedio General:** 1,503 ms
    - **Post Users:** 1,489 ms
    - **Get Users:** 1,469 ms
    - **Error Rate:** 0% en todas las operaciones.

- **Rendimiento:** 61.5 solicitudes/seg.

2. **MongoDB:**
   - **Promedio General:** 5,818 ms
   - **Post Users:** 2,636 ms
   - **Get Users:** 4,536 ms
   - **Error Rate:** 32.38%, debido a operaciones complejas y menor tolerancia a carga simultánea.
   - **Rendimiento:** 12.8 solicitudes/seg.

Análisis de Resultados:

1. **Velocidad:**
   - MySQL demostró mejor rendimiento en operaciones comunes, con tiempos más bajos en solicitudes GET y POST.
   - MongoDB presentó tiempos más altos, especialmente en operaciones GET, debido al manejo de grandes volúmenes de datos y estructuras más flexibles.

2. **Concurrencia y Consistencia:**
   - **MySQL:** Soportó 8,000 solicitudes sin errores, mostrando alta estabilidad.
   - **MongoDB:** Error del 32.38%, reflejando un desafío en el manejo de conexiones intensivas.

3. **Rendimiento General:**
   - MySQL sobresale en rendimiento y manejo de transacciones simples.
   - MongoDB es más adecuado para datos no estructurados, pero requiere optimización para alta concurrencia.

Recomendaciones:

- **MongoDB:**
  - Optimizar índices y consultas frecuentes.

    - Ajustar el pool de conexiones para reducir la tasa de error bajo carga.
    - Implementar técnicas como bulkWrite para manejar inserciones masivas.

- **MySQL:**
    - Verificar y ajustar índices para consultas más rápidas.
    - Revisar conexiones concurrentes para maximizar el rendimiento en escenarios de alta demanda.

- **Conclusion**:

  MySQL es ideal para aplicaciones que requieren rendimiento constante y transacciones estructuradas, mientras que MongoDB se adapta mejor a datos flexibles como los de **Fithub**, aunque necesita ajustes para soportar alta carga.

6. Configuración Final y Documentación

Configuración Final del Proyecto

El proyecto cuenta con una arquitectura completa y modular, incluyendo:

- **Backend:** Node.js y Express con Mongoose para MongoDB.
- **Frontend:** React, con axios para solicitudes HTTP y formik/yup para validación de formularios.
- **Autenticación:** JWT para la autenticación de usuarios y rutas protegidas.
- **Docker Compose:** Configuración para MongoDB y Mongo Express con volúmenes persistentes.
- **Herramientas de Prueba y Profiling:** Clinic.js para evaluación de rendimiento y Postman para pruebas de la API.

Documentación de Errores y Soluciones

1. **Errores de Conexión en Docker:** Configuraciones de puertos y credenciales en Docker.
2. **Configuración de Rutas y Validación JWT:** Middleware de autenticación y manejo de errores con JWT.

3. **Pruebas de API y Validación de Datos:** Pruebas exhaustivas con Postman y validación con Formik/Yup.

Estructura y Uso de Modelos en MongoDB

Todos los modelos de datos fueron adaptados a MongoDB para aprovechar sus características NoSQL, permitiendo flexibilidad en la gestión de datos y simplificando operaciones complejas.

7. Conclusiones

La migración a MongoDB trajo varias ventajas, como la simplificación en la estructura de datos y la posibilidad de realizar cambios sin preocuparse por restricciones de esquema, al contrario de MySQL. Las pruebas de rendimiento indicaron que el uso de una base de datos NoSQL mejoró el manejo de grandes volúmenes de datos, especialmente en las operaciones de sueño, ejercicio y nutrición.

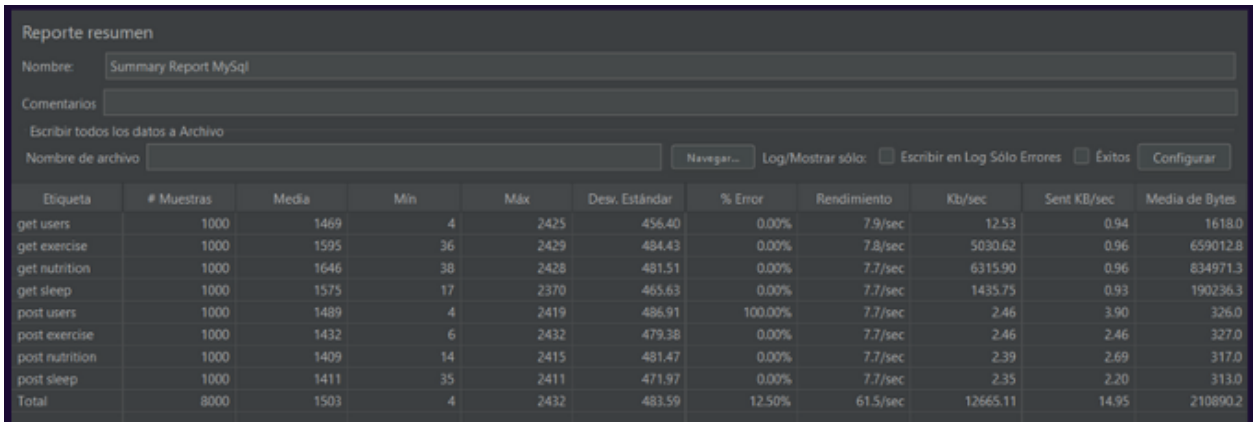
Esta bitácora sirve como una guía exhaustiva para futuras modificaciones y permite una mejor comprensión del proceso de desarrollo, los desafíos enfrentados y las soluciones implementadas en el proyecto Fithub.

8. Anexos

1. Profiler MySql

2. Profiler MongoDB

3. Load Testing MySql



Reporte resumen

Nombre: Summary Report MySql

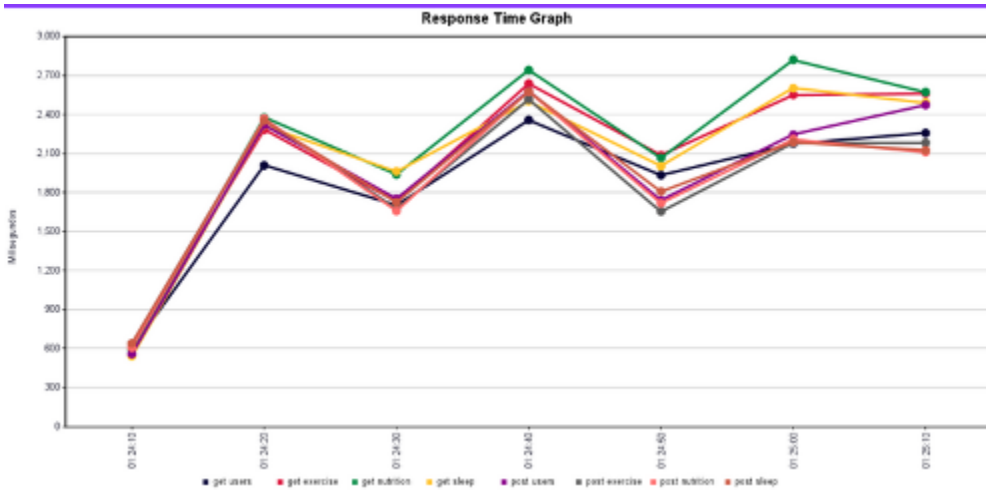
Comentarios

Escribir todos los datos a Archivo

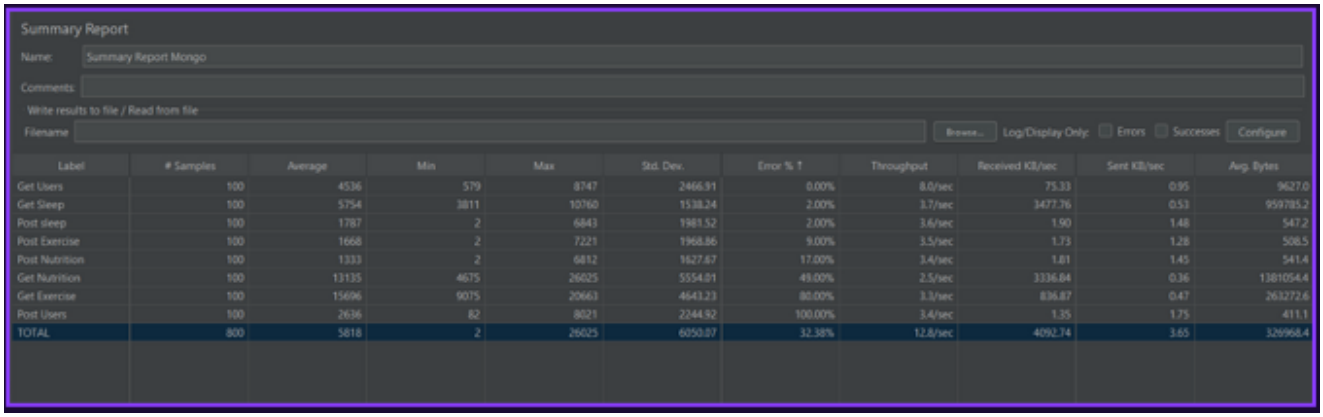
Nombre de archivo | Navegar... | Log/Mostrar sólo: | Escribir en Log Sólo Errores | Éxitos | Configurar

Etiqueta	# Muestras	Media	Min	Máx	Desv. Estándar	% Error	Rendimiento	Kb/sec	Sent KB/sec	Media de Bytes
get users	1000	1469	4	2425	456.40	0.00%	7.9/sec	12.53	0.94	1618.0
get exercise	1000	1595	36	2429	484.43	0.00%	7.8/sec	5030.62	0.96	659012.8
get nutrition	1000	1646	38	2428	481.51	0.00%	7.7/sec	6315.90	0.96	834971.3
get sleep	1000	1575	17	2370	465.63	0.00%	7.7/sec	1435.75	0.93	190236.3
post users	1000	1489	4	2419	486.91	100.00%	7.7/sec	2.46	3.90	326.0
post exercise	1000	1432	6	2432	479.38	0.00%	7.7/sec	2.46	2.46	327.0
post nutrition	1000	1409	14	2415	481.47	0.00%	7.7/sec	2.39	2.69	317.0
post sleep	1000	1411	35	2411	471.97	0.00%	7.7/sec	2.35	2.20	313.0
Total	8000	1503	4	2432	483.59	12.50%	61.5/sec	12665.11	14.95	210890.2

4. Gráfica Load Testing MySql



5. Load Testing MongoDB



Summary Report

Name: Summary Report Mongo

Comments:

Write results to file / Read from file

Filename: Browse... Log/Display Only: Errors Successes Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error % ↑	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Get Users	100	4536	579	8747	2466.91	0.00%	8.0/sec	75.33	0.95	9627.0
Get Sleep	100	5754	3811	10760	1538.24	2.00%	3.7/sec	3477.76	0.53	959785.2
Post sleep	100	1787	2	6843	1981.52	2.00%	3.6/sec	1.90	1.48	547.2
Post Exercise	100	1668	2	7221	1968.86	9.00%	3.5/sec	1.73	1.28	508.5
Post Nutrition	100	1333	2	6812	1627.67	17.00%	3.4/sec	1.81	1.45	541.4
Get Nutrition	100	13135	4675	26025	5554.01	49.00%	2.5/sec	3336.84	0.36	1381054.4
Get Exercise	100	15696	9075	20663	4643.23	80.00%	3.3/sec	836.87	0.47	263272.6
Post Users	100	2636	82	8021	2244.92	100.00%	3.4/sec	1.35	1.75	411.1
TOTAL	800	5818	2	26025	6050.07	32.38%	12.8/sec	4092.74	3.65	326968.4

6. Gráfica Load Testing MongoDB