Informe del Proyecto: Aplicación Java Spring Boot con WebFlux, MongoDB Reactivo, Log4j2 y Actuator (Métricas Prometheus)

Autor: Juan Junior Abad Yacila.
Fecha: 15-03-2025

---

1. Introducción y Objetivos

Este proyecto tiene como finalidad construir una **API REST** con **Spring Boot** que emplee un modelo de **programación reactiva** para conectarse a una base de datos **MongoDB** de forma no bloqueante. Se aprovechan diversas tecnologías y componentes para cubrir los siguientes requerimientos:

1. **Gradle** como sistema de construcción (en lugar de Maven).
2. **Java 8** como versión del lenguaje.
3. **Spring Data MongoDB Reactivo** para persistencia asíncrona en MongoDB.
4. **Spring WebFlux** para crear un servidor embebido (Netty) con un modelo reactivo.
5. **Spring Log4j2** para manejo de logs (excluyendo Logback).
6. **Spring Actuator** para exponer endpoints de monitoreo, métricas y estado de la aplicación, incluyendo Prometheus.
7. **Toda la configuración** en un archivo **application.yml** (en lugar de application.properties).
8. Uso de **Lombok** (opcional) para reducir código repetitivo en getters, setters y constructores.

Además, se solicitan dos endpoints:

- **POST** para insertar un objeto TraceMsg en MongoDB.
- **GET** para consultar objetos TraceMsg en un rango de fechas.

Se agrega también un **contador** (hacom.test.developer.insert.rx) que se incrementa cada vez que se inserta un nuevo TraceMsg, exponiendo dicha métrica en **Prometheus** a través de Actuator.

---

2. Tecnologías Empleadas

A continuación, se listan y describen las principales tecnologías y librerías integradas en el proyecto.

2.1. Gradle

- Sistema de construcción y manejo de dependencias.
- Definido en el archivo build.gradle.
- Permite compilar, ejecutar pruebas y empacar la aplicación de manera automatizada.

2.2. Java 8

- Versión mínima del lenguaje Java utilizada.
- Habilita características como **lambdas**, **Stream API** y el uso de **OffsetDateTime**.

2.3. Spring Boot (versión 2.4.3 preferentemente)

- Facilita la **autoconfiguración** y reduce la necesidad de configuración manual.
- Proporciona un servidor embebido (Netty) para WebFlux.

2.4. Spring Data MongoDB Reactivo

- Permite acceder a **MongoDB** de forma no bloqueante mediante un modelo **reactivo**.
- Provee repositorios como ReactiveMongoRepository<T, ID> para CRUD y consultas personalizadas.

2.5. Spring WebFlux

- Orientado a la **programación reactiva**, usando un bucle no bloqueante y backpressure.
- Ideal para alta concurrencia y escalabilidad.
- Sustituye a Spring MVC tradicional en entornos reactivos.

2.6. Spring Log4j2

- Se excluye el spring-boot-starter-logging y se configura Log4j2.
- Se utiliza un archivo log4j2.yml para definir **appenders** (consola y archivo) y niveles de log.
- Evita conflictos con Logback.

2.7. Spring Actuator (Métricas Prometheus)

- Expone endpoints de monitoreo (health, info, metrics, etc.).
- Se habilita el endpoint de **Prometheus** (/actuator/prometheus) para recolectar métricas.
- Permite registrar un **contador** personalizado (hacom.test.developer.insert.rx).

2.8. Validación con Spring

- Uso de anotaciones @NotNull, @Valid, etc.
- Facilita la validación de DTOs como DateRangeRequest.

2.9. Lombok

- Facilita la creación de getters/setters/constructores (@Data, @RequiredArgsConstructor, etc.).
- Reduce el boilerplate code.

2.10. Configuración en application.yml

- Definición del **puerto** (server.port=9898),
- Propiedades para la **conexión** a MongoDB (mongodbUri, mongodbDatabase),
- Exposición de **Actuator** (management.endpoints.web.exposure.include="*"),
- Ajustes de **logging** (nivel DEBUG para org.springframework.web).

---

3. Estructura y Configuración de la Aplicación

3.1. build.gradle

Contiene todas las **dependencias** y plugins. Destacan:

- spring-boot-starter-webflux para WebFlux.
- spring-boot-starter-data-mongodb-reactive para MongoDB no bloqueante.
- spring-boot-starter-actuator para métricas y monitoreo.
- Exclusión de logback-classic y uso de spring-boot-starter-log4j2.

3.2. application.yml

Se configuran las propiedades clave:

yaml
CopiarEditar

```
server:
  port: 9898

spring:
  data:
    mongodb:
      uri: mongodb://127.0.0.1:27017
      database: exampleDb

management:
  endpoints:
    web:
      exposure:
        include: "*"
  metrics:
    export:
      prometheus:
        enabled: true

logging:
  file:
    name: logs/app.log
```

- server.port=9898 para el puerto del servidor WebFlux.
- spring.data.mongodb.uri y spring.data.mongodb.database para la conexión a MongoDB.
- management.endpoints.web.exposure.include="*" habilita todos los endpoints Actuator.
- logging.file.name=logs/app.log para registrar logs en un archivo local, adicionalmente a la consola.

3.3. log4j2.yml

Define la configuración de **Log4j2**. Se ubica en src/main/resources/log4j2.yml:

yaml
CopiarEditar

```
Configuration:
  status: INFO
  Appenders:
    Console:
      name: ConsoleAppender
      target: SYSTEM_OUT
      PatternLayout:
        Pattern: "%d{yyyy-MM-dd HH:mm:ss} [%t] %-5level %logger{36} - %msg%n"
    File:
      name: FileAppender
      fileName: logs/app.log
      PatternLayout:
        pattern: "%d{yyyy-MM-dd HH:mm:ss} [%t] %-5level %logger{36} - %msg%n"
  Loggers:
    Root:
```

```
    level: INFO
    AppenderRef:
      - ref: ConsoleAppender
      - ref: FileAppender
```

- **Console Appender** para imprimir logs en la consola.
- **File Appender** para guardar logs en logs/app.log.

---

4. Modelo y Persistencia

4.1. Entidad: TraceMsg

```
java
CopiarEditar
@Data
@Document(collection = "trace_msgs")
public class TraceMsg {
   @Id
   private ObjectId _id;
   private String sessionId;
   private String payload;
   private OffsetDateTime ts;
}
```

- Representa un documento en la colección trace_msgs.
- Campo _id con ObjectId (de MongoDB).
- ts para guardar la fecha/hora.

4.2. Repositorio Reactivo: TraceMsgRepository

```
java
CopiarEditar
public interface TraceMsgRepository extends ReactiveMongoRepository<TraceMsg, String> {
   Flux<TraceMsg> findByTsBetween(OffsetDateTime from, OffsetDateTime to);
}
```

- Permite CRUD asíncrono (save, findById, delete, etc.).
- Método personalizado findByTsBetween(...) para filtrar por rango de fechas.

4.3. Servicio: TraceMsgService

```
java
CopiarEditar
@Service
@Log4j2
@RequiredArgsConstructor
public class TraceMsgService {
   private final TraceMsgRepository repository;

   public Mono<TraceMsg> insert(TraceMsg traceMsg) {
      log.info("Insertando mensaje de rastreo: {}", traceMsg);
      return repository.save(traceMsg);
   }
```

```
  public Flux<TraceMsg> findByDateRange(DateRangeRequest request) {
    log.info("Buscando rastros desde {} hasta {}", request.getFrom(), request.getTo());
    return repository.findByTsBetween(request.getFrom(), request.getTo());
  }
}
```

- **Encapsula** la lógica de negocio.
- Inserta un TraceMsg y consulta por rango de fechas.
- **Registra logs** (nivel INFO) en cada operación.

---

5. Endpoints y Controladores

5.1. DTO de Rango de Fechas: DateRangeRequest

```
java
CopiarEditar
@Data
@AllArgsConstructor
public class DateRangeRequest {
  @NotNull private OffsetDateTime from;
  @NotNull private OffsetDateTime to;
}
```

- Define from y to como fechas/hora.
- Se valida que no sean nulos (@NotNull).

5.2. Controlador: TraceController

```
java
CopiarEditar
@RestController
@RequestMapping("/api/traces")
@RequiredArgsConstructor
@Log4j2
public class TraceController {
  private final TraceMsgService traceMsgService;

  @PostMapping("/insert")
  public Mono<TraceMsg> insertTrace(@RequestBody TraceMsg traceMsg) {
    traceMsg.setTs(OffsetDateTime.now());
    log.info("Solicitud de inserción recibida: {}", traceMsg);
    return traceMsgService.insert(traceMsg);
  }

  @GetMapping("/range")
  public Flux<TraceMsg> getTracesInRange(@RequestParam("from") String from,
                                         @RequestParam("to") String to) {
    log.info("Solicitud de rango recibida de {} a {}", from, to);
    OffsetDateTime fromDate = OffsetDateTime.parse(from);
    OffsetDateTime toDate = OffsetDateTime.parse(to);
    return traceMsgService.findByDateRange(new DateRangeRequest(fromDate, toDate));
  }
}
```

- **Rutas**:
    - POST /api/traces/insert: Inserta un TraceMsg.
    - GET /api/traces/range: Consulta mensajes entre dos fechas/hora.
- Asigna ts = OffsetDateTime.now() en cada inserción.
- **Registra logs** y llama a los métodos del servicio.
- Se podría integrar un **contador** con Micrometer (hacom.test.developer.insert.rx) para incrementarlo en cada inserción.

---

6. Uso de Métricas y Actuator (Prometheus)

1. **Spring Actuator** expone endpoints en http://localhost:9898/actuator.
2. Para **Prometheus**, se habilita management.metrics.export.prometheus.enabled=true.
3. El endpoint de métricas se ubica en http://localhost:9898/actuator/prometheus.
4. Se puede crear un **contador** en el controlador/servicio:

```
java
CopiarEditar
private final Counter insertCounter;

public TraceController(TraceMsgService service, MeterRegistry meterRegistry) {
   this.traceMsgService = service;
   this.insertCounter = meterRegistry.counter("hacom.test.developer.insert.rx");
}
// ...
insertCounter.increment();
```

De este modo, cada vez que se realice un **POST** para insertar un TraceMsg, se incrementa hacom.test.developer.insert.rx.

---

7. Ejecución y Pruebas con Postman

1. **Iniciar MongoDB** localmente:

```
bash
CopiarEditar
mongod --dbpath /path/to/data
```

2. **Ejecutar la aplicación**:

```
bash
CopiarEditar
./gradlew bootRun
```

    - Por defecto, inicia en el puerto 9898 (definido en application.yml).
3. **Probar Inserción**:
    - **POST**: http://localhost:9898/api/traces/insert
    - **Body (JSON)**:

```
json
CopiarEditar
{
  "sessionId": "abc123",
```

```
  "payload": "Mensaje de ejemplo"
}
```

   - Debe devolver un TraceMsg con _id, sessionId, payload, ts.
4. **Probar Consulta por Rango**:
   - **GET**: http://localhost:9898/api/traces/range?from=2023-01-01T00:00:00Z&to=2023-12-31T23:59:59Z
   - Devuelve un array JSON de TraceMsg dentro de ese rango.
5. **Revisar Logs**:
   - Consola y archivo logs/app.log.
   - Se registran mensajes de nivel INFO.
6. **Revisar Métricas**:
   - **Prometheus**: http://localhost:9898/actuator/prometheus
   - **Otros endpoints**: http://localhost:9898/actuator/health, http://localhost:9898/actuator/metrics, etc.

---

8. Metodología y Referencias

8.1. Metodología de Desarrollo

- **Ágil (Scrum/Kanban)**: se recomiendan **sprints** cortos para ir añadiendo funcionalidad.
- **Integración Continua (CI)**: configurar pipelines (GitHub Actions, Jenkins) para compilar, testear y desplegar.
- **Revisión de Código**: promover un proceso de Pull Requests para mejorar calidad.

8.2. Referencias

- **Documentación de Spring WebFlux**: https://docs.spring.io/spring-framework/docs/current/reference/html/web-reactive.html
- **Documentación de Spring Data MongoDB (reactivo)**: https://docs.spring.io/spring-data/mongodb/docs/current/reference/html/#mongo.reactive
- **Spring Boot Actuator**: https://spring.io/guides/gs/actuator-service
- **Micrometer + Prometheus**: https://micrometer.io/docs/registry/prometheus
- **Log4j2 Configuration**: https://logging.apache.org/log4j/2.x/manual/configuration.html

---

9. Conclusiones

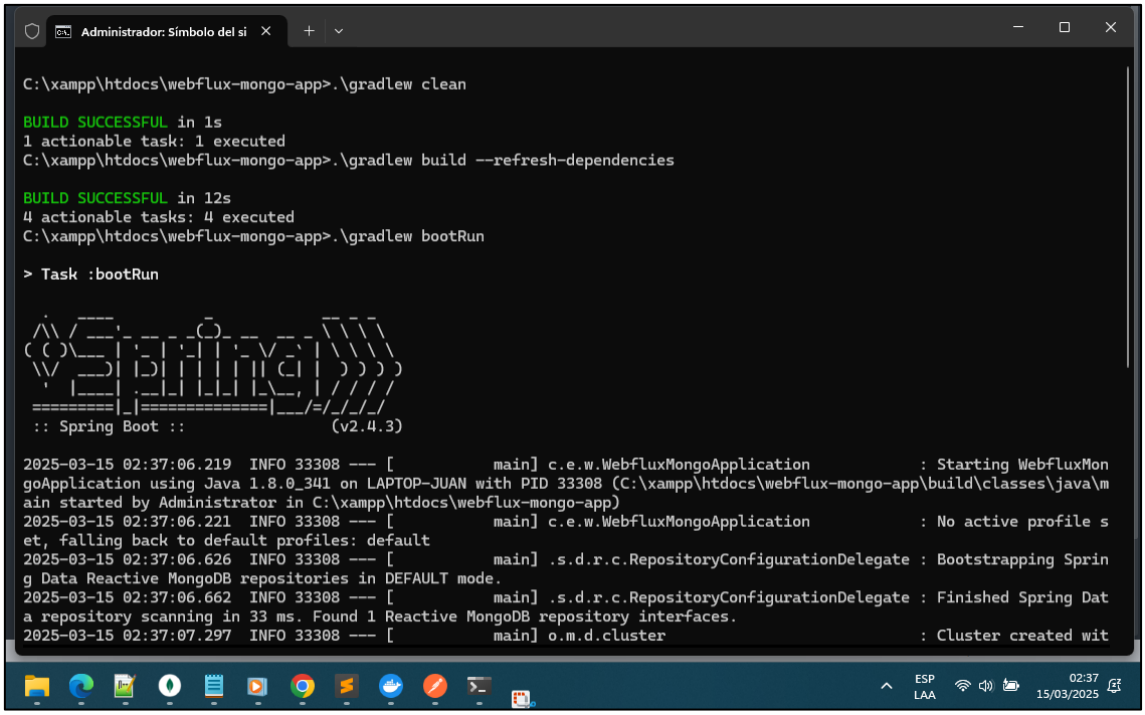
Este proyecto cumple con los requisitos de:

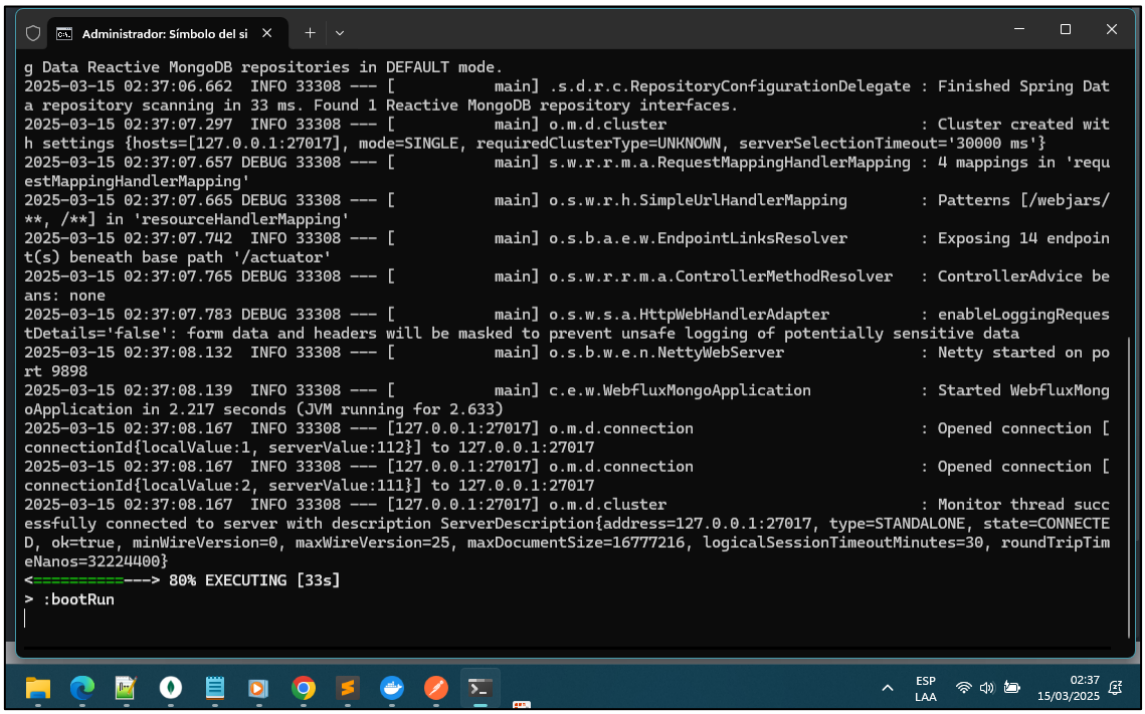
1. **Uso de Gradle** y **Java 8**.
2. **Spring Data MongoDB Reactivo** y **Spring WebFlux** para un modelo no bloqueante.
3. **Log4j2** configurado con un archivo log4j2.yml, excluyendo Logback.
4. **Spring Actuator** para exponer métricas a **Prometheus** y monitoreo general.
5. **application.yml** como archivo único de configuración, definiendo URI de MongoDB, base de datos y puerto de la API.
6. **Dos endpoints** principales: POST para insertar TraceMsg y GET para consultar por rango de fechas (DateRangeRequest).
7. **Contador** hacom.test.developer.insert.rx que se incrementa en cada inserción.

Con esta arquitectura y metodología, se obtiene una aplicación **reactiva, escalable y fácilmente monitoreable**, integrable con **Prometheus** y **Grafana** para visualización de métricas, y apta para un ciclo de desarrollo ágil (Scrum/Kanban) con despliegues continuos.

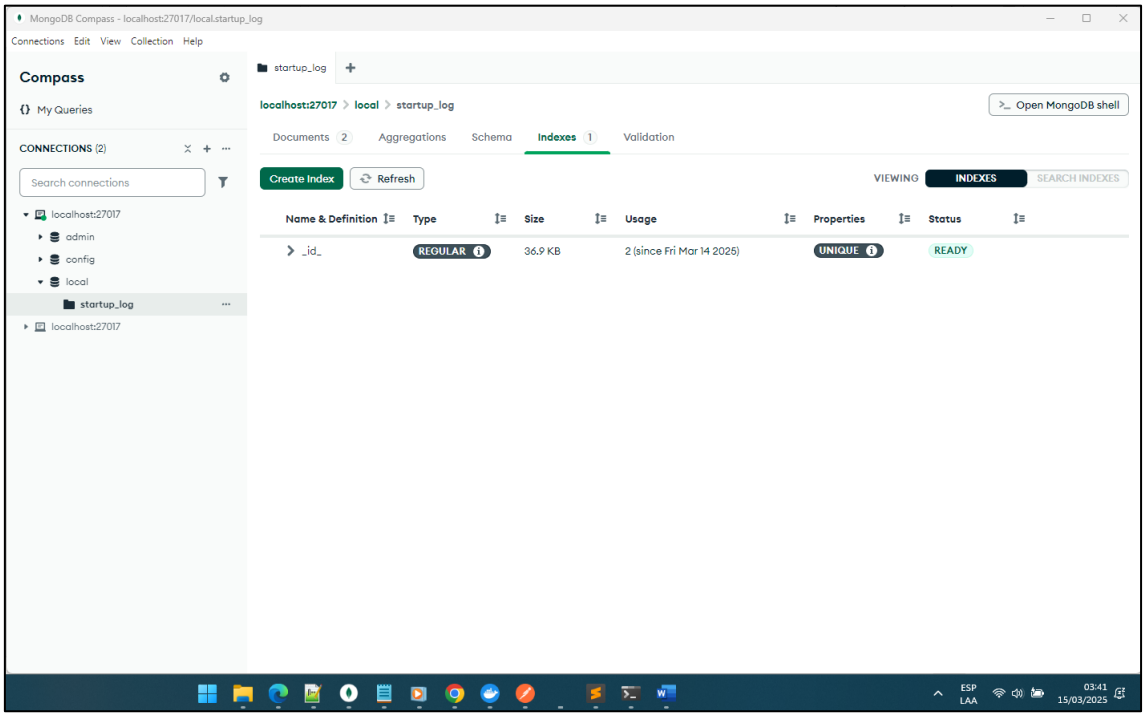
Evidencias:

Spring Boot:





MongoDB:



Postman:

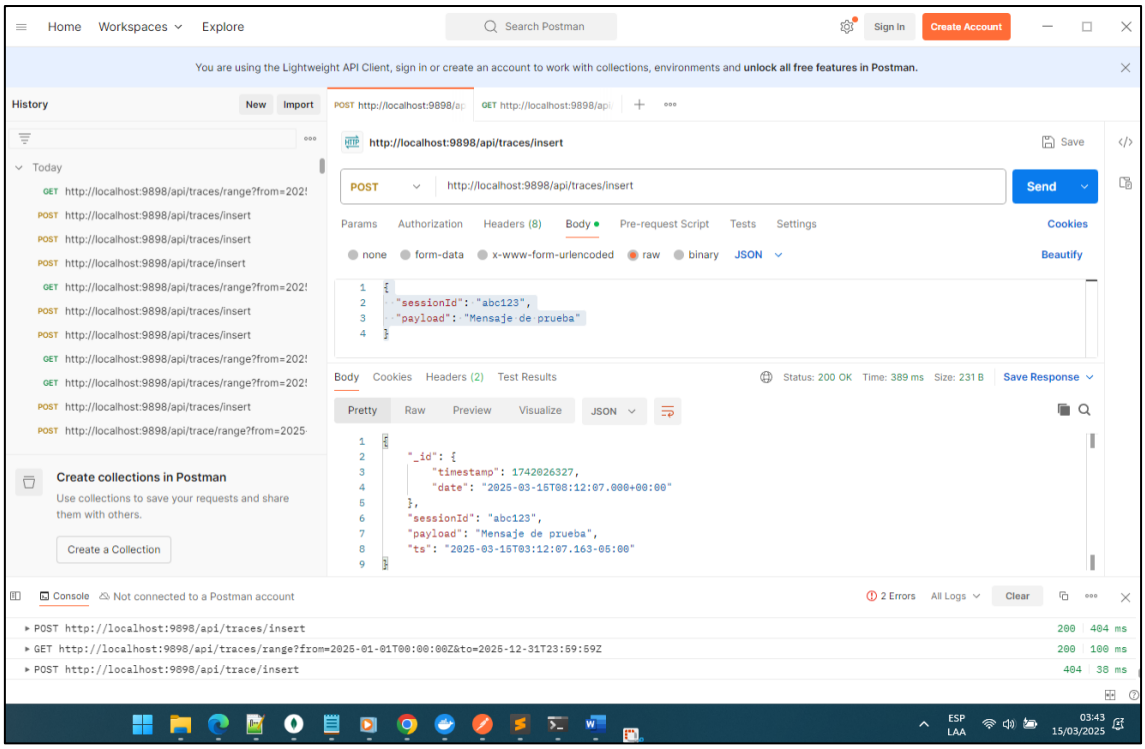
(POST) http://localhost:9898/api/traces/insert:

JSON (Ingreso):

```
{
  "sessionId": "abc123",
  "payload": "Mensaje de prueba"
}
```

Respuesta:


```
{
    "_id": {
        "timestamp": 1742026327,
        "date": "2025-03-15T08:12:07.000+00:00"
    },
    "sessionId": "abc123",
    "payload": "Mensaje de prueba",
    "ts": "2025-03-15T03:12:07.163-05:00"
}
```



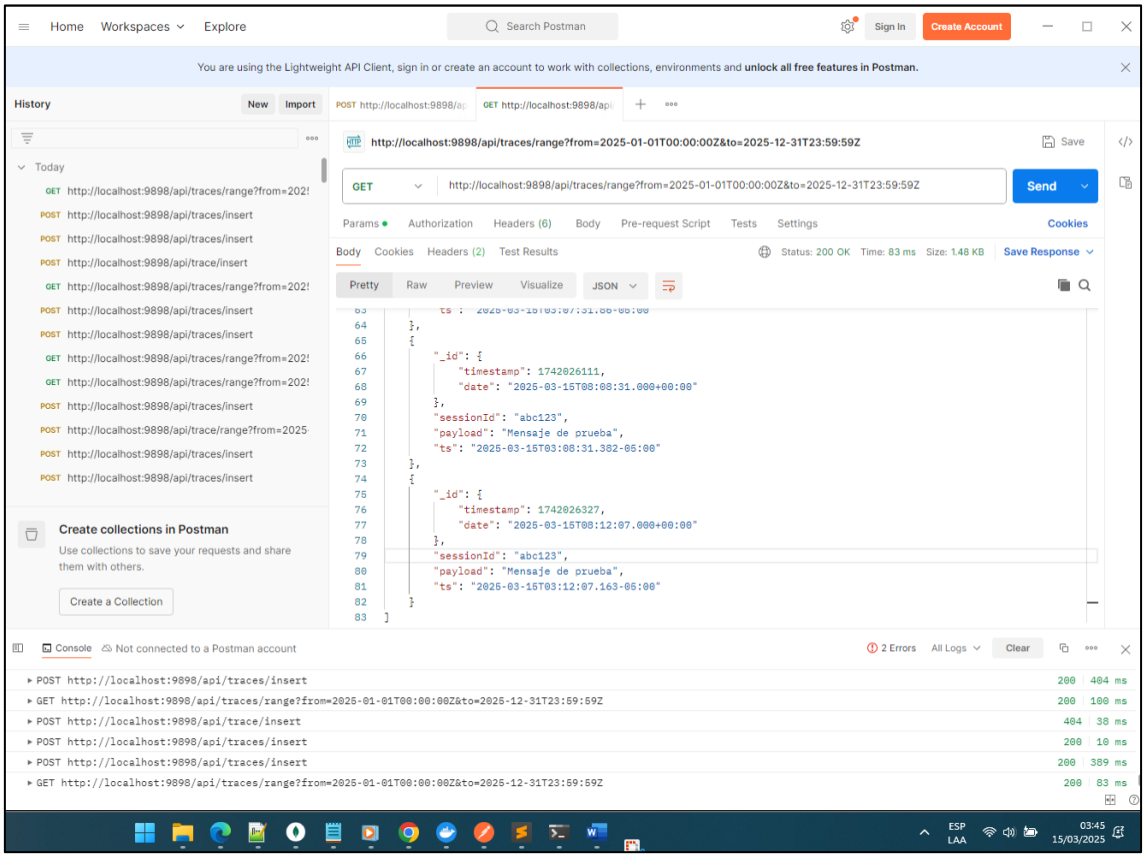
(GET) http://localhost:9898/api/traces/range?from=2025-01-01T00:00:00Z&to=2025-12-31T23:59:59Z :

Respuesta:


```
{
        "_id": {
            "timestamp": 1742022921,
            "date": "2025-03-15T07:15:21.000+00:00"
        },
        "sessionId": "abc123",
        "payload": "Mensaje de prueba",
        "ts": "2025-03-15T02:15:20.956-05:00"
    },
    {
        "_id": {
            "timestamp": 1742023443,
            "date": "2025-03-15T07:24:03.000+00:00"
        },
        "sessionId": "abc123",
        "payload": "Mensaje de prueba",
        "ts": "2025-03-15T02:24:03.876-05:00"
    },
    {
        "_id": {
            "timestamp": 1742023600,
            "date": "2025-03-15T07:26:40.000+00:00"
        },
        "sessionId": "abc123",
        "payload": "Mensaje de prueba",
        "ts": "2025-03-15T02:26:40.234-05:00"
```

```
},
{
    "_id": {
        "timestamp": 1742023859,
        "date": "2025-03-15T07:30:59.000+00:00"
    },
    "sessionId": "abc123",
    "payload": "Mensaje de prueba",
    "ts": "2025-03-15T02:30:59.387-05:00"
},
{
    "_id": {
        "timestamp": 1742024793,
        "date": "2025-03-15T07:46:33.000+00:00"
    },
    "sessionId": "abc123",
    "payload": "Mensaje de prueba",
    "ts": "2025-03-15T02:46:33.326-05:00"
},
{
    "_id": {
        "timestamp": 1742025946,
        "date": "2025-03-15T08:05:46.000+00:00"
    },
    "sessionId": "abc123",
    "payload": "Mensaje de prueba",
    "ts": "2025-03-15T03:05:46.958-05:00"
},
{
    "_id": {
        "timestamp": 1742026051,
        "date": "2025-03-15T08:07:31.000+00:00"
    },
    "sessionId": "abc123",
    "payload": "Mensaje de prueba",
    "ts": "2025-03-15T03:07:31.86-05:00"
},
{
    "_id": {
        "timestamp": 1742026111,
        "date": "2025-03-15T08:08:31.000+00:00"
    },
    "sessionId": "abc123",
    "payload": "Mensaje de prueba",
    "ts": "2025-03-15T03:08:31.382-05:00"
},
{
    "_id": {
        "timestamp": 1742026327,
        "date": "2025-03-15T08:12:07.000+00:00"
    },
```

```
        "sessionId": "abc123",
        "payload": "Mensaje de prueba",
        "ts": "2025-03-15T03:12:07.163-05:00"
    }
]
```

Estructura de proyecto:

```
.
├── build.gradle
├── src
│   ├── main
│   │   ├── java
│   │   │   └── com.example.webfluxmongo
│   │   │       ├── WebfluxMongoApplication.java
│   │   │       ├── config
│   │   │       │   ├── MongoConfig.java
│   │   │       │   ├── AppConfig.java
│   │   │       │   └── ...
│   │   │       ├── controller
│   │   │       │   ├── TraceController.java
│   │   │       │   └── ...
│   │   │       ├── dto
│   │   │       │   └── DateRangeRequest.java
│   │   │       ├── model
│   │   │       │   └── TraceMsg.java
│   │   │       ├── repository
│   │   │       │   └── TraceMsgRepository.java
│   │   │       └── service
│   │   │           └── TraceMsgService.java
│   │   └── resources
│   │       ├── application.yml
│   │       └── log4j2.yml
└── ...
```