Instituto Politécnico Nacional

Escuela Superior de Cómputo

Web Client and Backend Development Frameworks

Creación de cuenta y base de datos en Yugabyte

Profesor: M. en C. José Asunción Enríquez Zárate

Alumno: Munive Hernández Erika Natalia

munivehernandezerika@gmail.com

7CM1

December 12, 2025

Contents

List of Figures

List of Tables

1 Introducción

En el presente trabajo documento de manera detallada mi experiencia realizando el proceso de creación, configuración y conexión de una base de datos distribuida en YugabyteDB, con el objetivo de integrarla a mi proyecto de Administración de Eventos desarrollado con Spring Boot. A lo largo del ejercicio tuve la oportunidad de explorar los elementos fundamentales de esta plataforma, comprender su arquitectura distribuida y experimentar directamente con las herramientas que ofrece para la administración de clústeres y bases de datos compatibles con PostgreSQL.

Elegí YugabyteDB debido a que proporciona una solución moderna para aplicaciones que requieren escalabilidad, resiliencia y disponibilidad continua, características que considero esenciales para sistemas actuales que manejan múltiples usuarios, transacciones concurrentes y posibles picos de carga. Desde mi perspectiva como desarrollador, resulta sumamente valioso trabajar con un motor que mantiene compatibilidad total con PostgreSQL, ya que esto me permitió reutilizar mis conocimientos previos en SQL y conectar mi proyecto sin realizar modificaciones significativas en la lógica de persistencia.

Durante este ejercicio, llevé a cabo el proceso completo: inicié registrándome en la plataforma Yugabyte Cloud, configuré un clúster en modo sandbox, generé mi base de datos, creé las tablas necesarias para los módulos de eventos y asistentes, y finalmente integré la base de datos con mi aplicación Spring Boot mediante JDBC. Para validar la comunicación y el correcto funcionamiento del sistema, realicé pruebas de creación de registros utilizando herramientas como Postman y verifiqué que los datos se almacenaran de manera adecuada dentro del clúster.

Esta actividad no solo me permitió cumplir con los requerimientos académicos del ejercicio, sino que también me brindó una visión más clara del papel que juegan las bases de datos distribuidas en las aplicaciones modernas. Además, pude reafirmar la importancia de utilizar servicios administrados en la nube para simplificar la infraestructura, reducir tiempos de despliegue y facilitar la escalabilidad del sistema. En general, considero que el uso de YugabyteDB representa una oportunidad para fortalecer mis competencias en el desarrollo de aplicaciones empresariales y en la adopción de tecnologías orientadas a la nube.

2 Conceptos

A continuación, se presentan los conceptos fundamentales que sustentan la práctica realizada sobre la configuración de bases de datos distribuidas mediante YugabyteDB, su compatibilidad con PostgreSQL y su integración con aplicaciones desarrolladas en Spring Boot. Estos conceptos permiten comprender el funcionamiento interno del clúster, la gestión de los datos y la conexión mediante JDBC.

- **Base de datos distribuida:** Sistema en el cual la información se almacena en múltiples nodos interconectados. Su objetivo es mejorar la disponibilidad, tolerancia a fallos y escalabilidad horizontal, permitiendo que el sistema continúe operando incluso si un nodo falla.
- **YugabyteDB:** Motor de base de datos SQL distribuida que combina compatibilidad total con PostgreSQL y una arquitectura basada en replicación y particionamiento de datos. Ofrece transacciones ACID, alta disponibilidad y escalabilidad horizontal para aplicaciones modernas.
- **PostgreSQL Distribuido:** Concepto que describe cómo YugabyteDB preserva la sintaxis y las funcionalidades de PostgreSQL, ejecutándolas sobre un motor distribuido que reparte y replica los datos entre múltiples nodos sin modificar la lógica SQL tradicional.
- **Aplicaciones nativas en la nube:** Aplicaciones diseñadas para ejecutarse de manera eficiente en entornos cloud, utilizando principios como microservicios, contenedores, automatización y escalabilidad. Se benefician de servicios administrados como YugabyteDB Managed.
- **Escalabilidad elástica:** Capacidad de un sistema para incrementar o disminuir sus recursos de forma dinámica según la demanda. En bases de datos distribuidas, se logra agregando o removiendo nodos sin interrumpir la operación.
- **Alta disponibilidad:** Propiedad que garantiza el funcionamiento continuo del sistema incluso ante fallos de hardware o software. YugabyteDB implementa alta disponibilidad mediante replicación y mecanismos automáticos de recuperación.
- **Replicación de datos:** Técnica que mantiene copias idénticas de los datos en diferentes nodos del clúster. Aumenta la tolerancia a fallos y permite responder a mayor cantidad de solicitudes con rapidez.
- **Particionamiento automático (Sharding):** Proceso mediante el cual una tabla se divide en múltiples fragmentos distribuidos entre nodos. Permite procesar consultas en paralelo y manejar grandes volúmenes de datos de forma eficiente.
- **Transacciones ACID distribuidas:** Conjunto de propiedades (atomicidad, consistencia, aislamiento y durabilidad) aplicadas en sistemas distribuidos. YugabyteDB garantiza transacciones ACID mediante el uso del protocolo de consenso Raft.
- **Consistencia fuerte:** Modelo en el cual todas las lecturas reflejan el estado más reciente de los datos. Se implementa mediante replicación síncrona y garantiza exactitud y confiabilidad en las operaciones.
- **Failover automático:** Mecanismo mediante el cual un nodo secundario asume la función del nodo principal en caso de falla, asegurando continuidad sin intervención manual.
- **Replicación síncrona:** Tipo de replicación donde una operación de escritura solo se confirma cuando se guarda exitosamente en todas las réplicas necesarias. Garantiza coherencia y durabilidad inmediata.
- **Conectividad JDBC:** API de Java que permite la comunicación entre aplicaciones y bases de datos SQL. YugabyteDB, al ser compatible con PostgreSQL, utiliza el mismo controlador JDBC para integrarse con Spring Boot.
- **Servicio administrado:** Modelo donde la infraestructura, mantenimiento, seguridad y monitoreo son gestionados por el proveedor. YugabyteDB Managed simplifica la creación y operación de clústeres.
- **Clúster distribuido:** Conjunto de nodos que trabajan de manera conjunta para almacenar y procesar datos. Permite distribuir carga, mejorar rendimiento y garantizar redundancia.

3 Desarrollo

3.1 Creación de cuenta en Yugabyte

Para comenzar con la práctica, se ingresó al portal oficial de YugabyteDB Cloud a través de la dirección: `https://cloud.yugabyte.com`. En la pantalla inicial de registro, mostrada en la Figura 1, la plataforma presenta las opciones principales para crear una cuenta nueva.

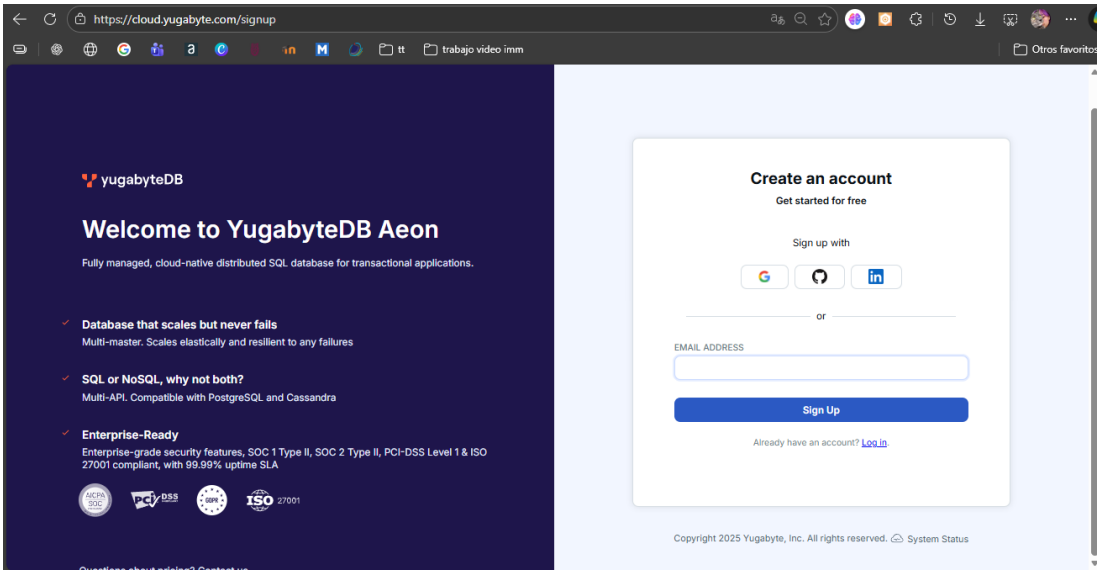

Figure 1: Pantalla inicial del portal para crear una cuenta en YugabyteDB Cloud.

Posteriormente, como se observa en la Figura 2, se accedió al formulario de creación de cuenta, donde se ingresó la dirección de correo electrónico para proceder con el registro mediante la opción *Sign Up with Google* con el fin de crear la cuenta en Yugabyte mediante una cuenta ya existente de Google.

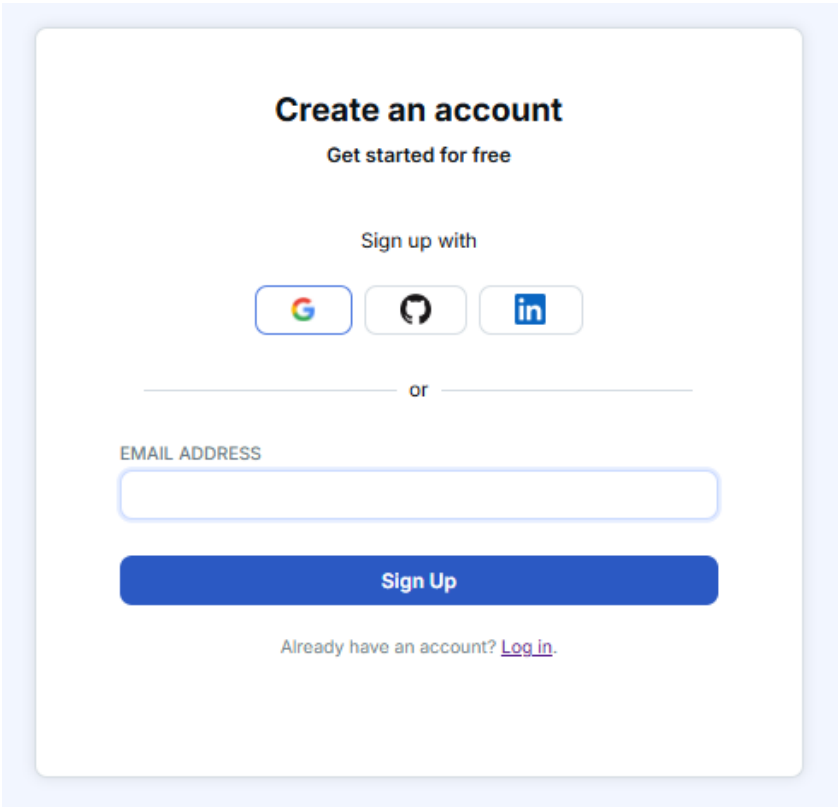

Figure 2: Formulario de registro para crear una nueva cuenta en YugabyteDB.

Una vez enviado el formulario, la plataforma mostró la pantalla de bienvenida ilustrada en la Figura 3, indicando que el proceso inicial había sido completado y que la cuenta sería configurada para su primer uso.

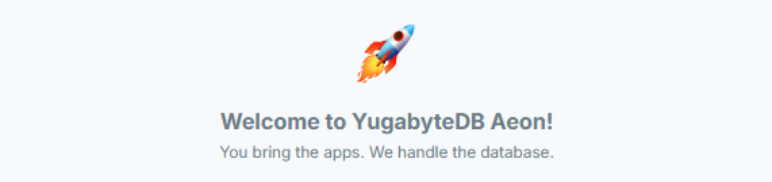

Figure 3: Pantalla de bienvenida tras completar el registro.

A continuación, se presentó un formulario adicional (Figura 4) en el cual se seleccionó el país de origen, necesario para ajustar las preferencias de la cuenta y continuar con la activación del acceso.

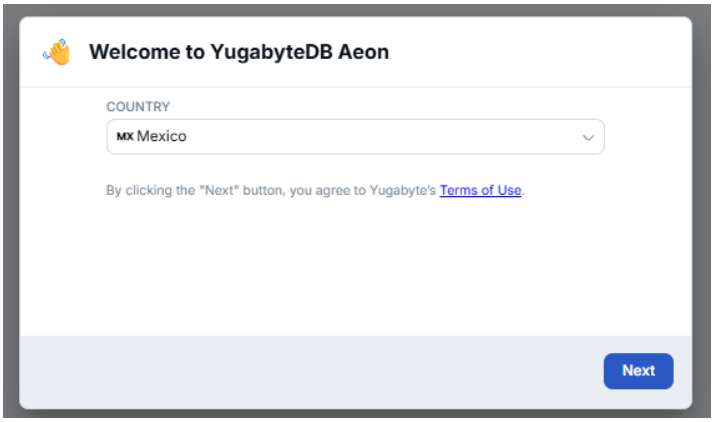

Figure 4: Formulario de información adicional solicitado por YugabyteDB.

Finalmente, completado el proceso de verificación y configuración inicial, la plataforma redirigió al panel principal de YugabyteDB Cloud como se muestra en la Figura 5, desde el cual es posible gestionar clústeres, credenciales y bases de datos. Con esto concluyó satisfactoriamente la creación de la cuenta.

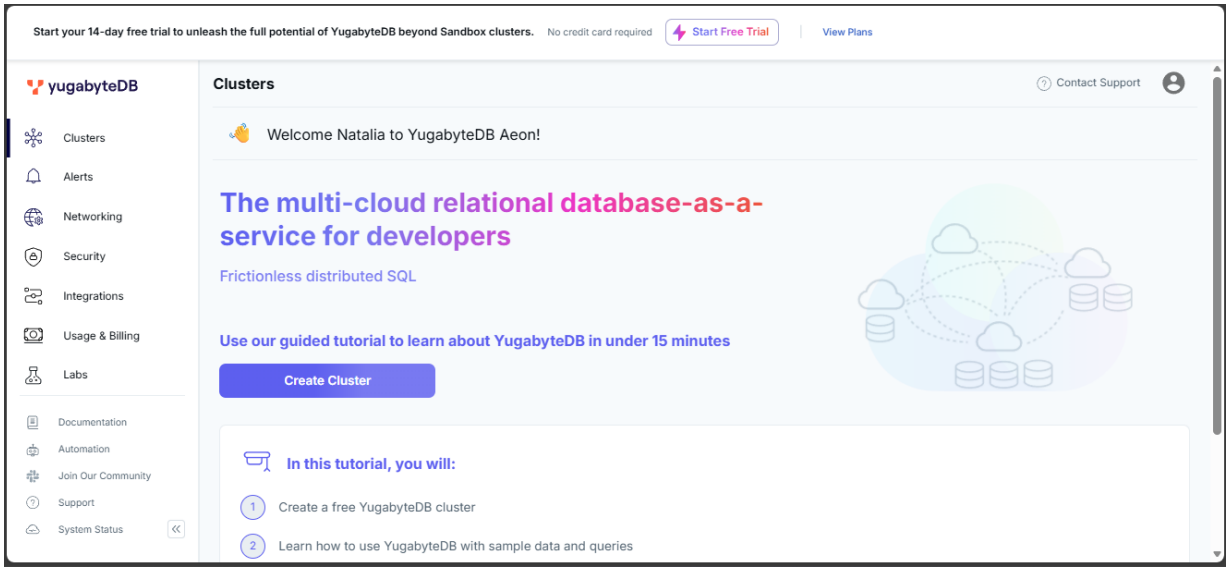

Figure 5: Acceso al panel principal de YugabyteDB Cloud tras la activación de la cuenta.

3.2 Creación de clúster

Una vez creada la cuenta en YugabyteDB Cloud, el siguiente paso consistió en generar un clúster distribuido que funcionaría como entorno de ejecución para la base de datos del sistema de Administración de Eventos. Desde el panel principal, se seleccionó la opción **Create Cluster**, como se muestra en la Figura 6.

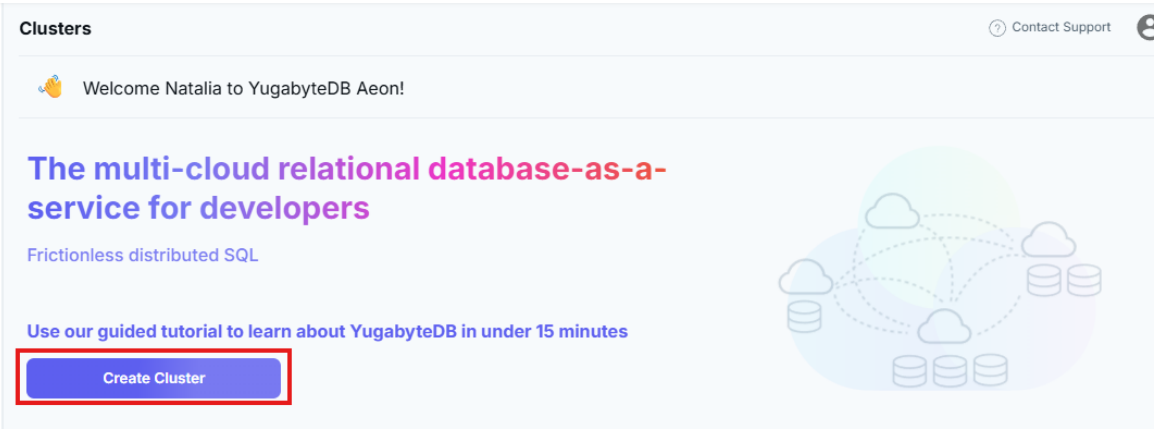

Figure 6: Acceso a la opción para crear un nuevo clúster en YugabyteDB Cloud.

Posteriormente, se presentó la pantalla de selección del tipo de clúster. Para fines académicos y de prueba, se eligió la modalidad **Sandbox**, que se observa en la Figura 7. Esta opción permite utilizar un clúster gratuito, con configuración automática y almacenamiento suficiente para entornos no productivos.

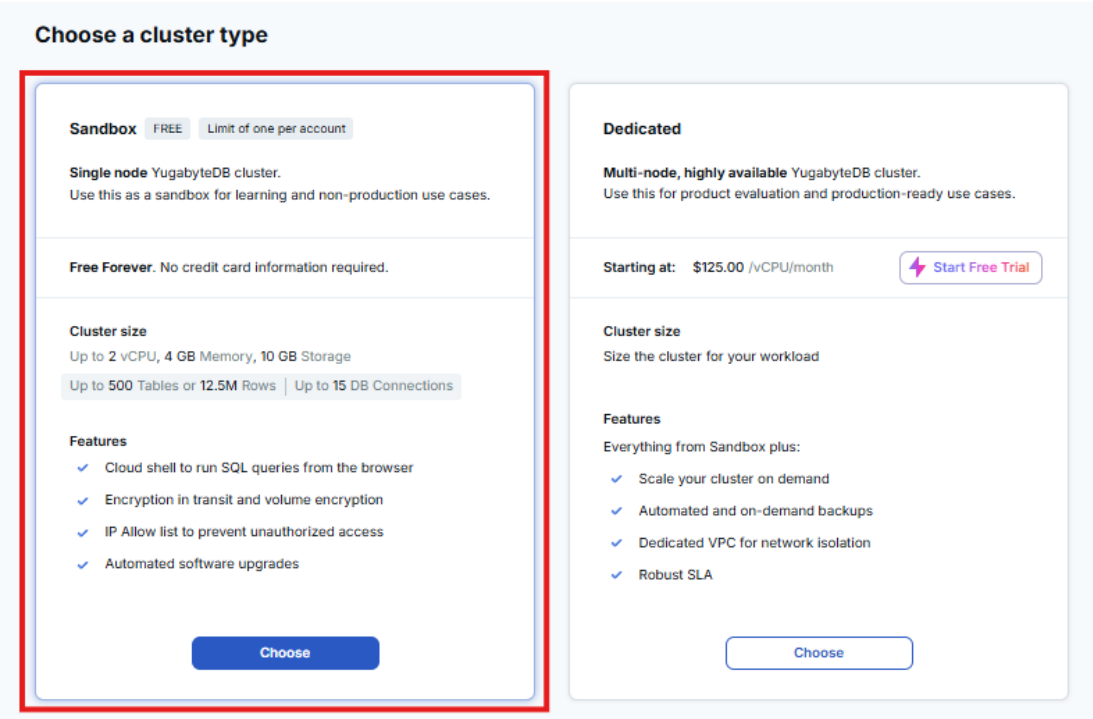

Figure 7: Selección del tipo de clúster en modalidad Sandbox.

A continuación, se configuraron los parámetros generales del clúster: nombre asignado, proveedor de infraestructura y la región de despliegue. Estos valores se ilustran en la Figura 8.

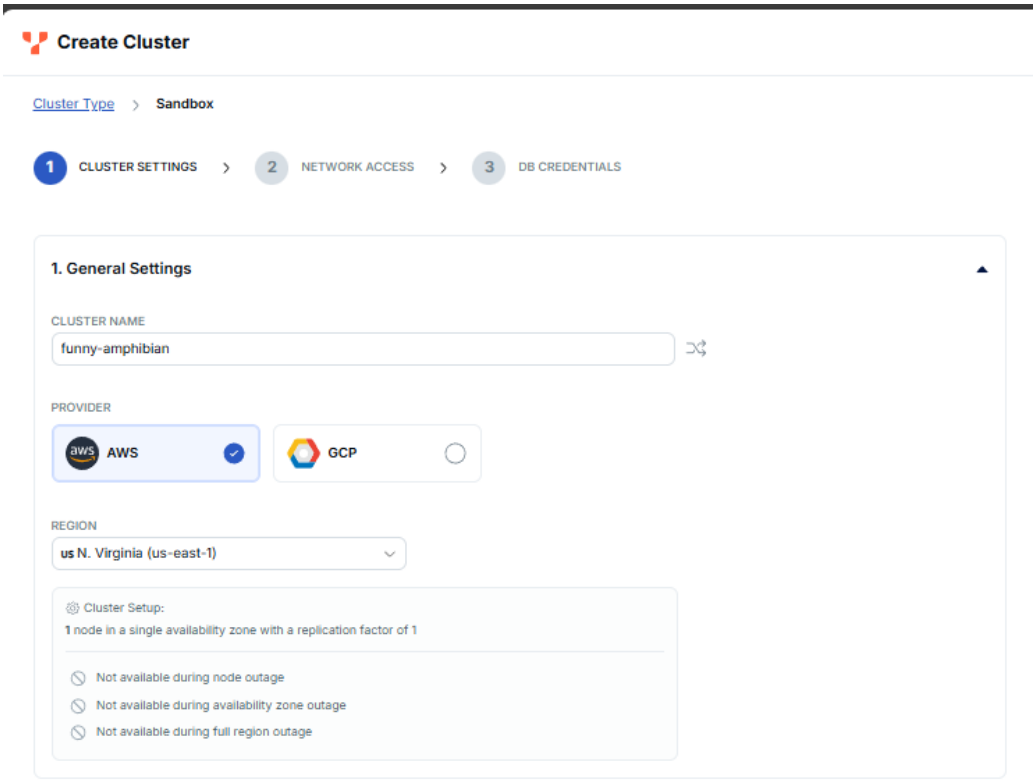

Figure 8: Configuración inicial del clúster: nombre, proveedor y región.

Dentro de las configuraciones adicionales, la plataforma permitió seleccionar la versión de la base de datos y otras opciones, como se muestra en la Figura 9.

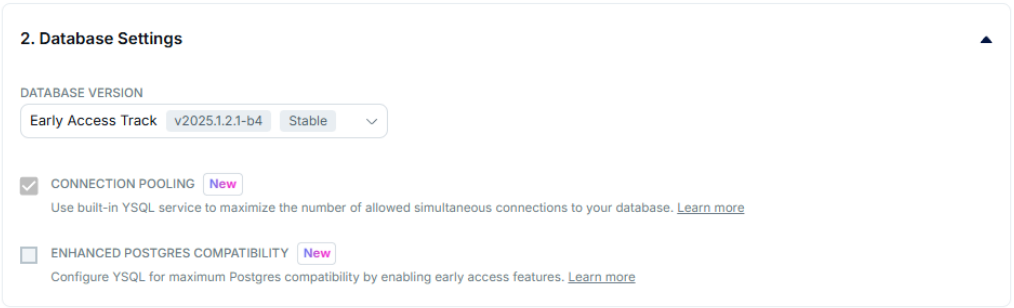

Figure 9: Selección de versión y configuraciones adicionales de la base de datos.

En el paso de configuración de red, se añadió la dirección IP desde la cual se realizaría la conexión al clúster mediante la opción **Add Current IP Address**, como se aprecia en la Figura 10.

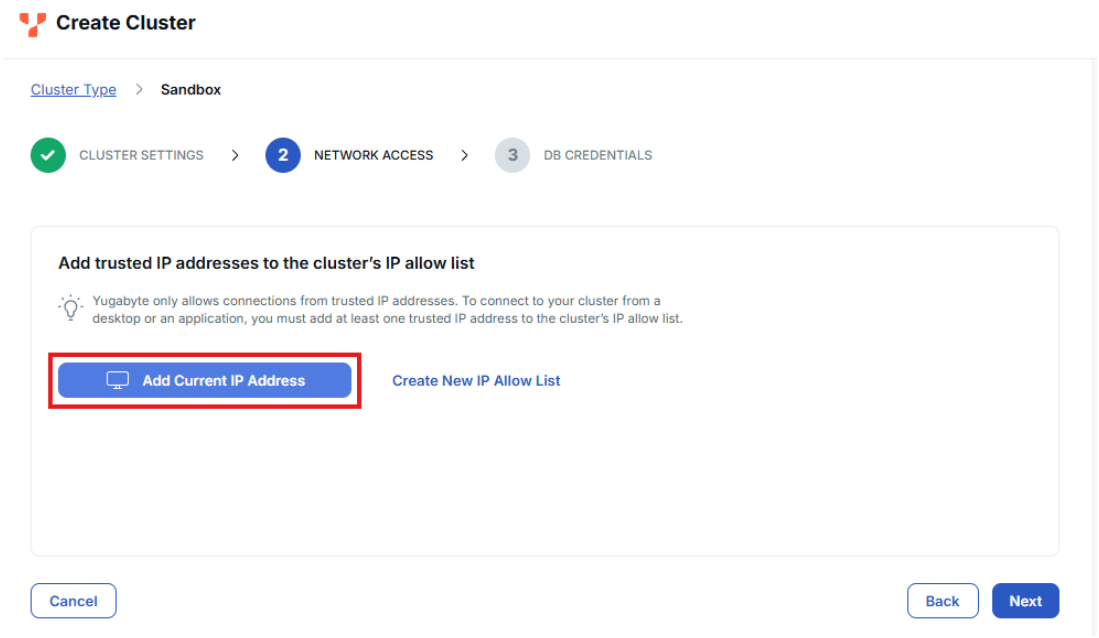

Figure 10: Autorización de acceso mediante la IP actual del dispositivo.

Después de agregar la IP permitida, el panel mostró la lista de direcciones autorizadas, como se aprecia en la Figura 11.

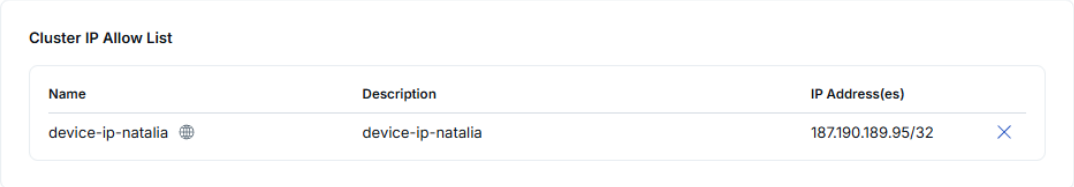

Figure 11: Listado de direcciones IP autorizadas para acceder al clúster.

Finalmente, en la sección de credenciales, YugabyteDB generó un usuario, contraseña, después se seleccionó la opción de descargar las credenciales necesarias para conectarse desde herramientas externas, y luego en la opción de "Create Cluster". La Figura 12 muestra esta etapa.

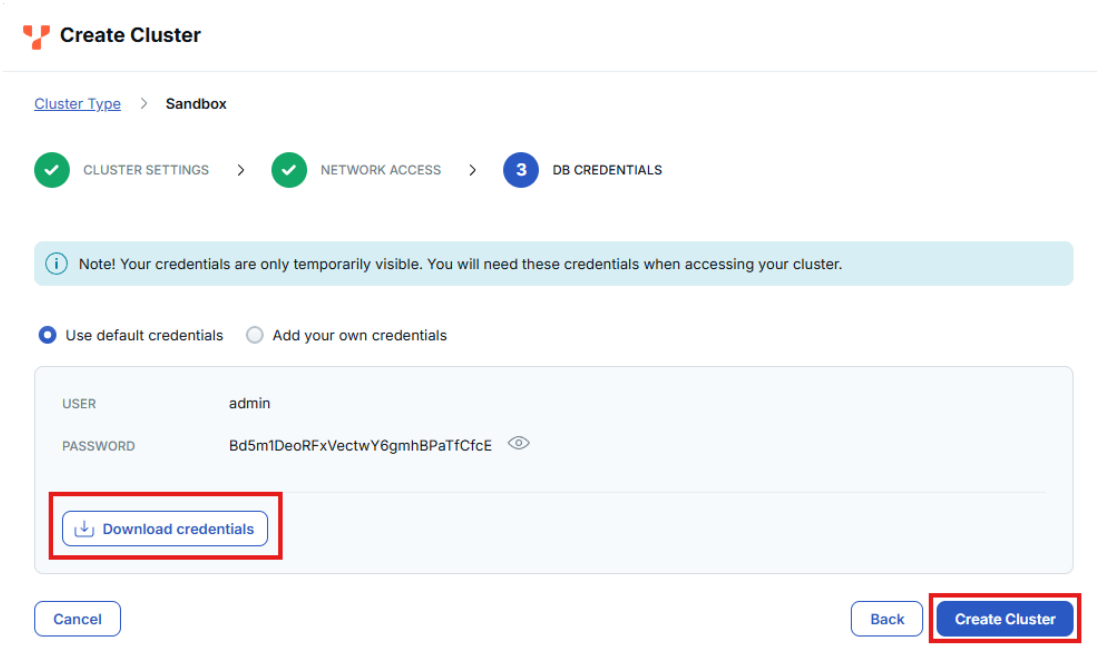

Figure 12: Credenciales generadas para la conexión al clúster.

Tras confirmar la configuración, la creación del clúster inició automáticamente. En la Figura 13 se observa el proceso de inicialización, aprovisionamiento y configuración final.

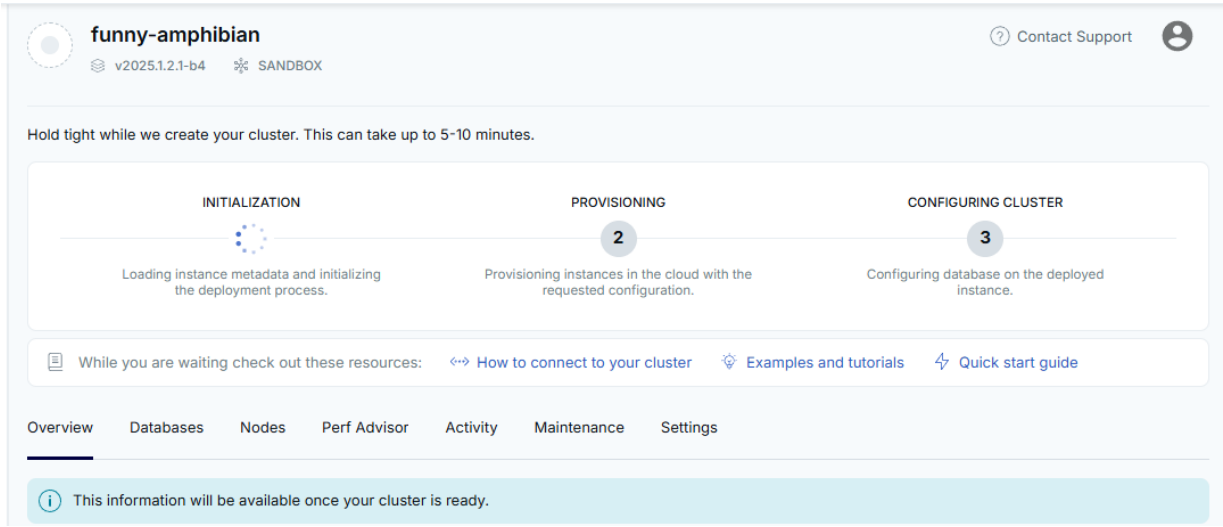

Figure 13: Proceso de creación e inicialización del clúster Sandbox.

Una vez transcurridos unos minutos, el clúster pasó a estado operativo y quedó disponible para su uso. En la Figura 14 se observa la vista general del clúster ya creado, incluyendo información del proveedor, región, recursos asignados y parámetros de seguridad. Con esto se confirma que la configuración fue realizada exitosamente y que la instancia se encuentra lista para gestionar bases de datos y aceptar conexiones externas.

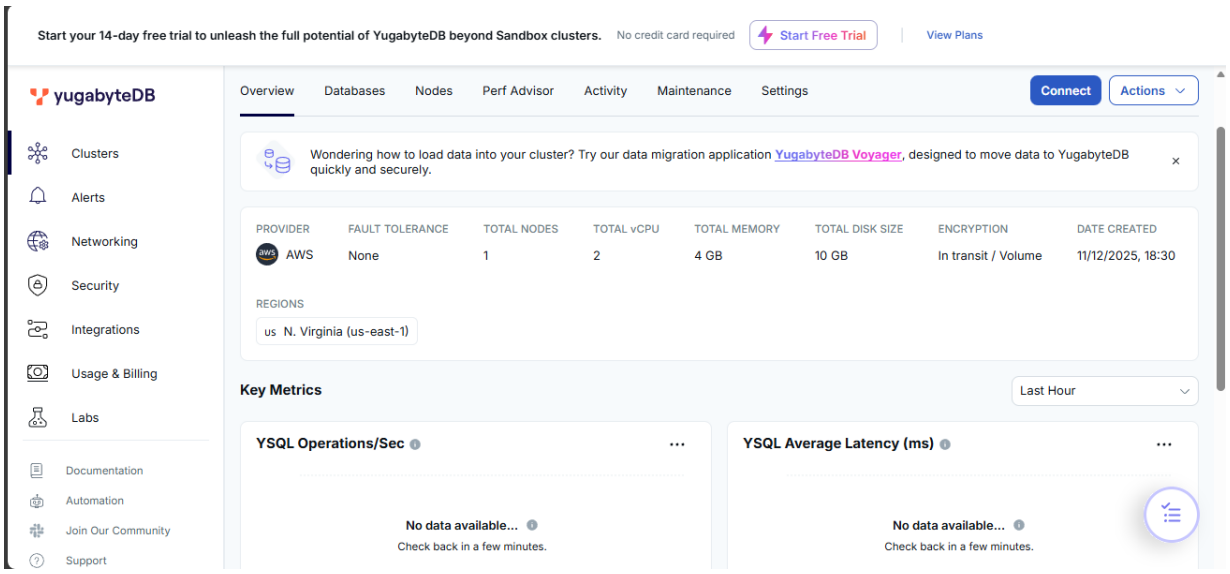

Figure 14: Clúster Sandbox completamente inicializado y listo para su uso.

3.3 Conexión del proyecto Spring Boot con YugabyteDB

Para establecer la comunicación entre mi aplicación desarrollada en Spring Boot y la base de datos distribuida creada en YugabyteDB, fue necesario obtener los parámetros de conexión proporcionados por la plataforma. Para ello, desde el panel principal del clúster seleccioné la opción *Connect*, tal como se muestra en la Figura 15.

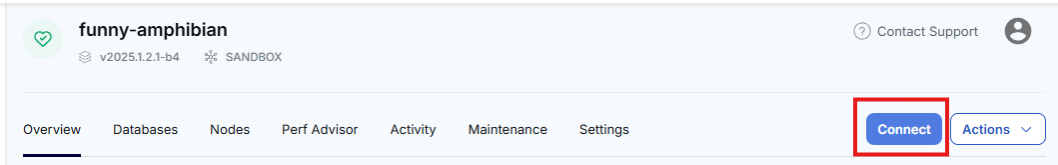

Figure 15: Acceso al menú de conexión del clúster en YugabyteDB.

Una vez dentro de las opciones de conexión, seleccioné la alternativa *Connect to your Application*, ya que es la que proporciona la cadena JDBC compatible con Spring Boot. Esta vista se muestra en la Figura 16.

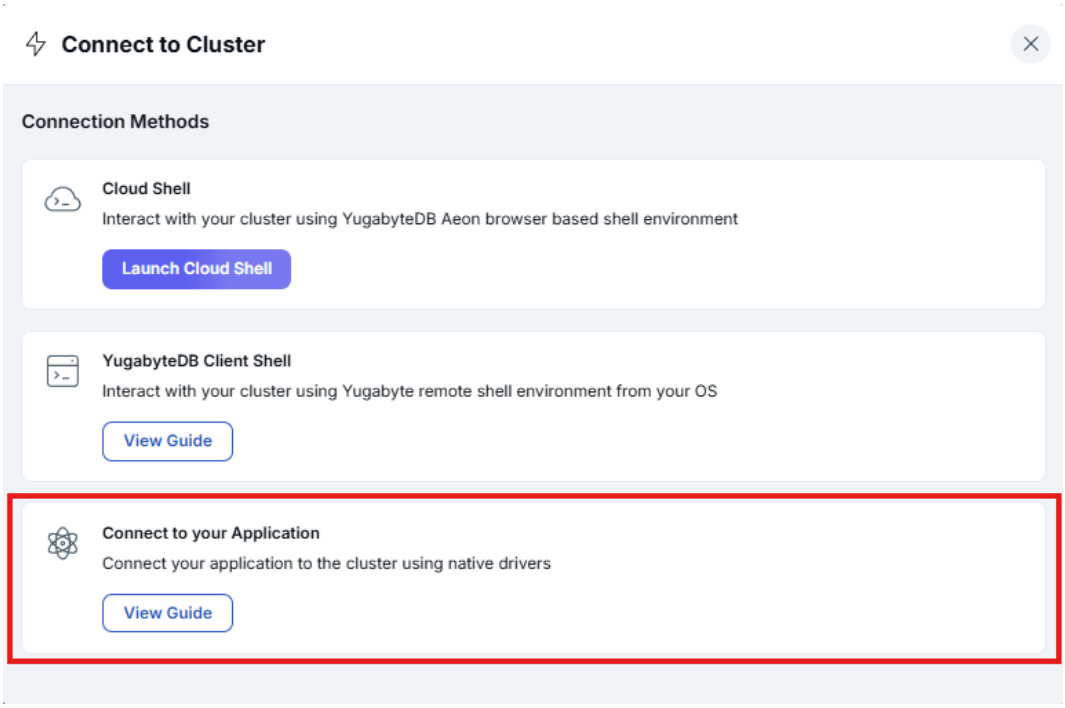

Figure 16: Selección del método de conexión para aplicaciones.

Al abrir esta sección, el primer paso consistió en descargar el certificado raíz `root.crt`, necesario para habilitar la comunicación segura mediante SSL entre la aplicación y el clúster. En la Figura 17 se observa el botón para descargar el certificado junto con la cadena de conexión YSQL generada automáticamente por YugabyteDB.

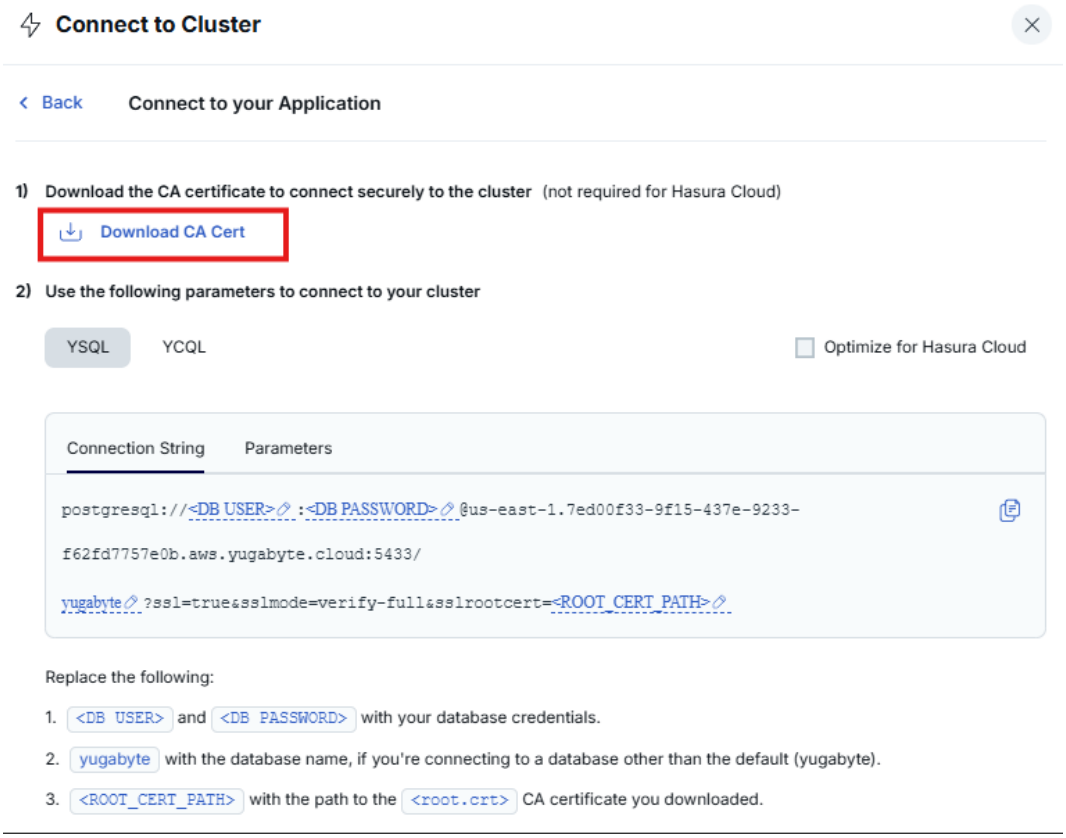

Figure 17: Parámetros de conexión YSQL y descarga del certificado raíz.

Una vez descargado, el archivo `root.crt` fue colocado en mi equipo local y posteriormente se utilizó su ruta absoluta dentro del archivo `application.properties`. En la Figura 18 se muestra dicho archivo, el cual

contiene la configuración completa del origen de datos: la URL JDBC, el usuario, la contraseña y los parámetros de Hibernate necesarios para la creación y actualización automática de tablas.

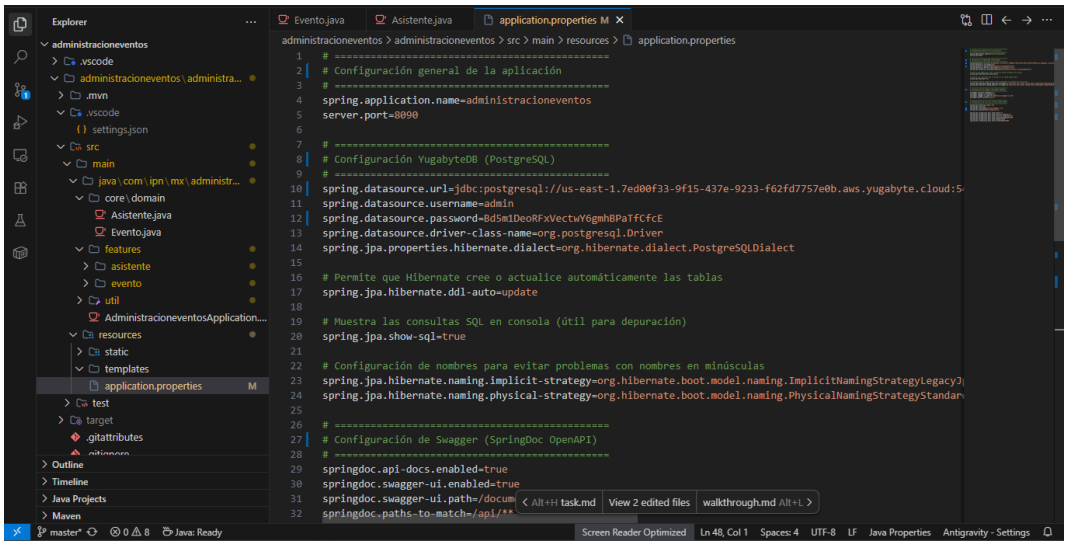

Figure 18: Archivo `application.properties` configurado con los parámetros de conexión.

La configuración final utilizada fue la siguiente:

```
spring.datasource.url=jdbc:postgresql://us-east-1.7ed00f33-9f15-437e-9233-
    f62fd7757e0b.aws.yugabyte.cloud:5433/yugabyte?ssl=true&sslmode=verify-full&
    sslrootcert=c:/Users/muniv/Downloads/administracioneventos/root.crt
spring.datasource.username=admin
spring.datasource.password=Bd5m1DeoRFxVectwY6gmhBPaTfCfcE
spring.datasource.driver-class-name=org.postgresql.Driver
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

Para ejecutar el proyecto desde Antigravity utilicé el siguiente comando Maven:

```
.\mvnw.cmd spring-boot:run
```

Este comando compila el proyecto e inicia el servidor embebido Tomcat.

En la Figura 19 se aprecia la confirmación de que la aplicación inició correctamente, estableciendo conexión con YugabyteDB y quedando disponible en el puerto 8090 sin presentar errores.

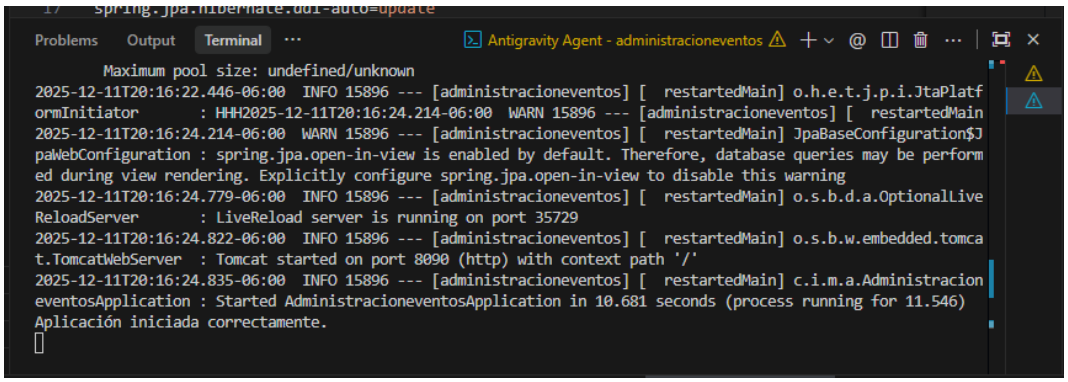

Figure 19: Ejecución correcta del proyecto Spring Boot y confirmación de inicio.

Con esta configuración, la aplicación quedó lista para interactuar con el clúster YugabyteDB, permitiendo la gestión de registros de eventos y asistentes del sistema.

3.4 Instalación y configuración inicial de Postman

Para realizar las pruebas de los servicios REST desarrollados en el proyecto de Spring Boot, utilicé la herramienta Postman, la cual permite enviar solicitudes HTTP de manera sencilla y estructurada. A continuación, se describe el proceso de instalación y configuración inicial.

El primer paso consistió en descargar la aplicación desde el sitio oficial de Postman. En la Figura 20 se muestra la página de descarga, donde seleccioné la versión correspondiente para Windows de 64 bits.

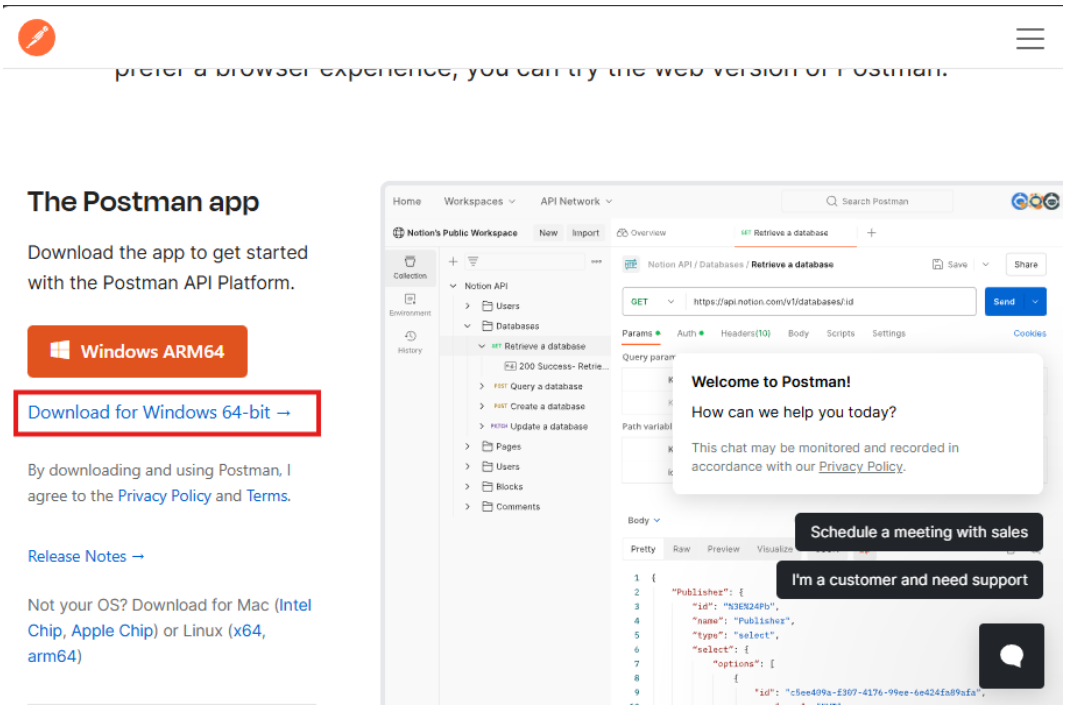

Figure 20: Página oficial de descarga de Postman para Windows 64 bits.

Una vez descargado el instalador, procedí a ejecutar la aplicación. Postman solicita crear una cuenta para sincronizar colecciones y configuraciones en la nube. En la Figura 21 se muestra la pantalla de inicio de sesión, donde ingresé mi correo institucional para continuar.

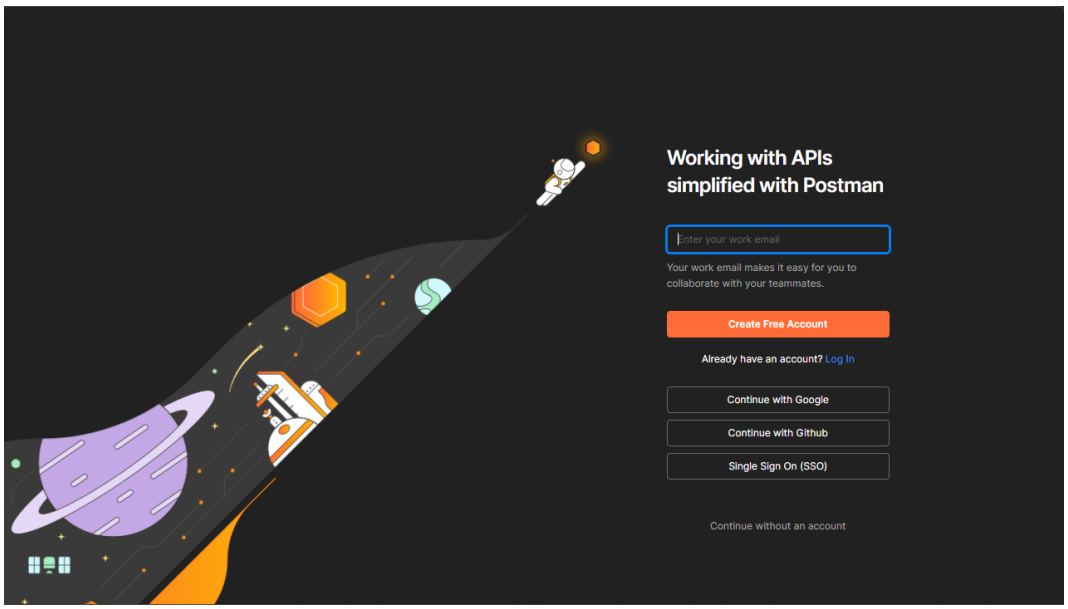

Figure 21: Pantalla de inicio de sesión en Postman.

Posteriormente, la plataforma solicita algunos datos generales para personalizar el entorno de trabajo. En mi caso, proporcioné mi nombre y seleccioné el rol que mejor describe mi perfil como usuaria, tal como se aprecia en la Figura 22.

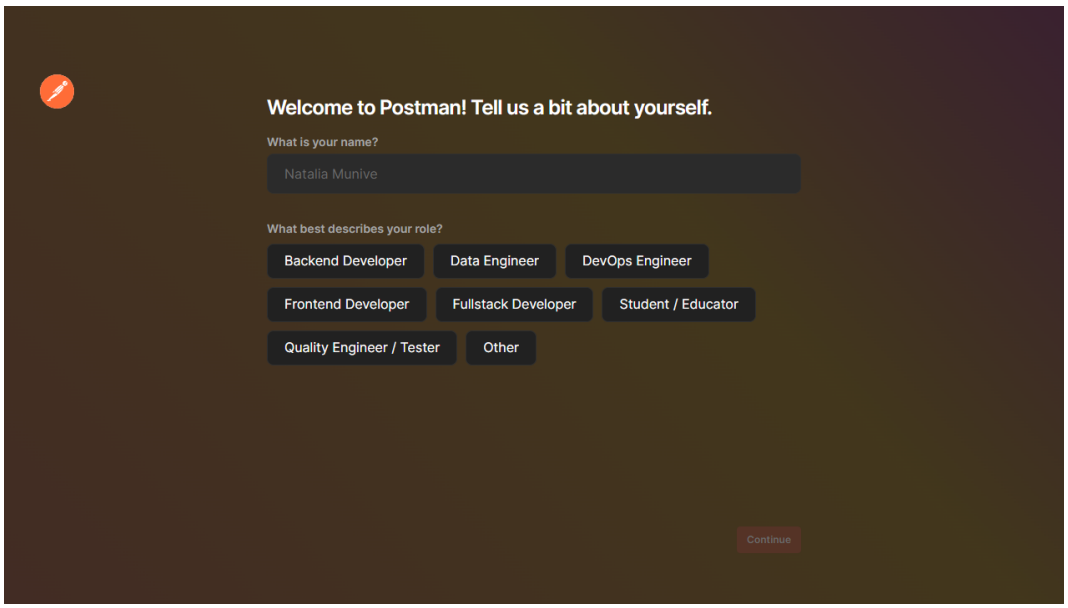

Figure 22: Configuración inicial del perfil de usuario en Postman.

Tras completar esta información, Postman muestra los diferentes planes disponibles. Para fines educativos, seleccioné la opción gratuita *Continue with Free Plan*, como se muestra en la Figura 23.

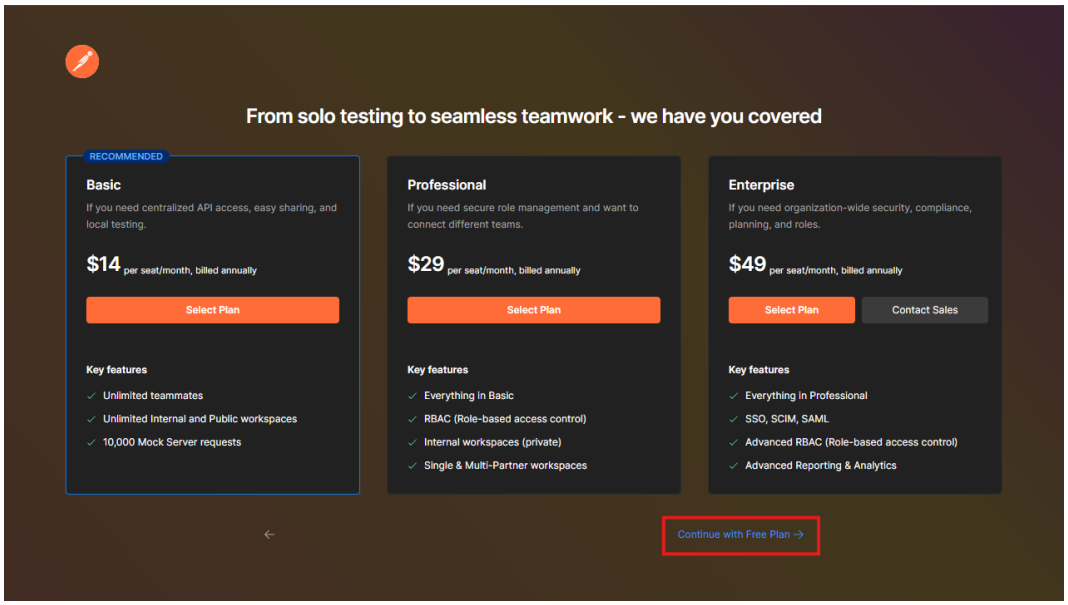

Figure 23: Selección del plan gratuito de Postman.

Finalmente, la plataforma presenta un asistente de bienvenida en el cual se detallan algunas acciones iniciales sugeridas. Para acceder al entorno principal, seleccioné la opción *Create your first API request*, tal como aparece en la Figura 24.

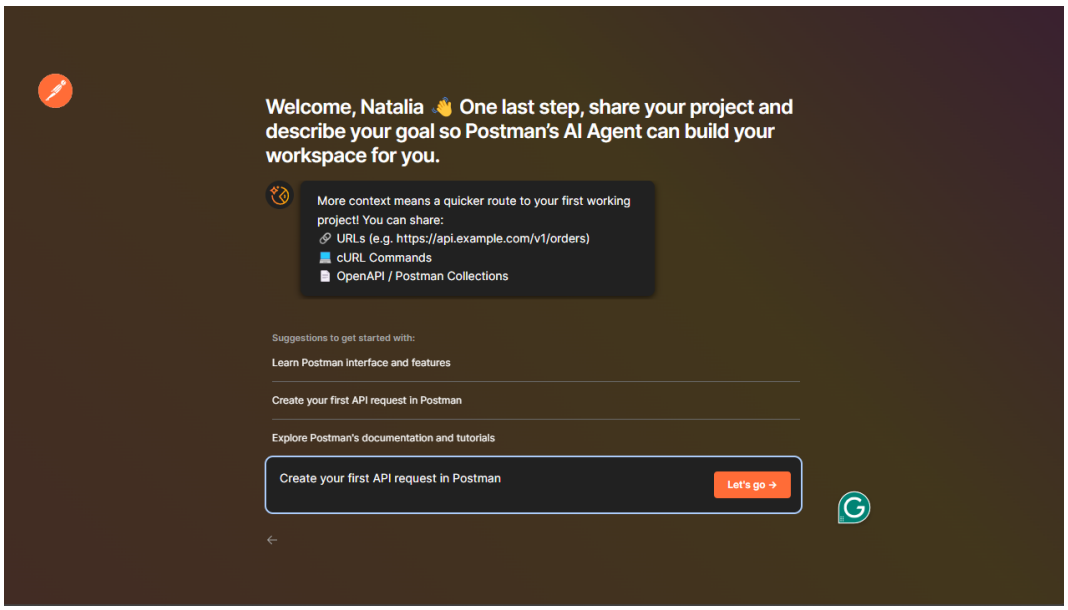

Figure 24: Pantalla final del asistente de bienvenida en Postman.

Una vez dentro del entorno de trabajo, pude visualizar la interfaz principal de Postman, donde es posible crear colecciones, configurar solicitudes GET, POST, PUT o DELETE, y observar las respuestas generadas por la API. La Figura 25 muestra la vista inicial con mi primer espacio de trabajo ya configurado.

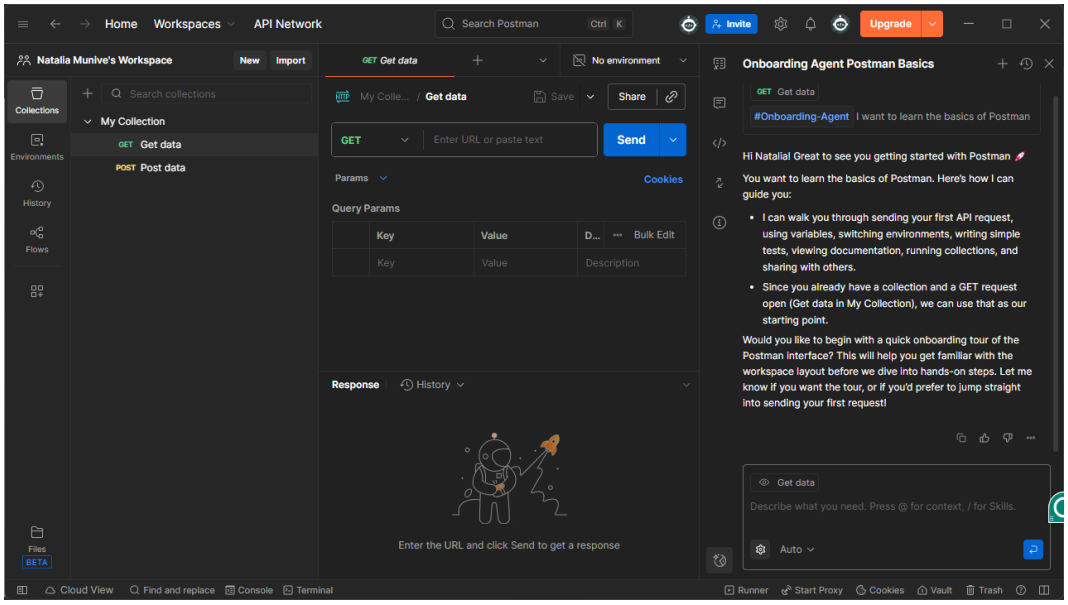

Figure 25: Interfaz principal de Postman lista para realizar solicitudes HTTP.

Con esta configuración inicial, la herramienta quedó lista para comenzar a realizar pruebas sobre los endpoints construidos en el sistema de Administración de Eventos.

3.5 Pruebas de Registro y Consulta de Eventos en Postman

Para iniciar las pruebas de los servicios REST desarrollados en el proyecto, se utilizó la herramienta **Postman**. El primer paso consistió en crear una colección que permitiera organizar todas las solicitudes relacionadas con la administración de eventos.

Desde la pantalla principal de Postman, se seleccionó la opción **Collection** dentro del menú de creación (véase Figura 26). Esta opción permite agrupar múltiples solicitudes HTTP dentro de un mismo proyecto, facilitando su administración y documentación.

A continuación, se creó una nueva colección asignándole el nombre:

AdministracionEventos

Dentro de esta colección posteriormente se añadieron las solicitudes para registrar eventos, obtener la lista de eventos y consultar asistentes.

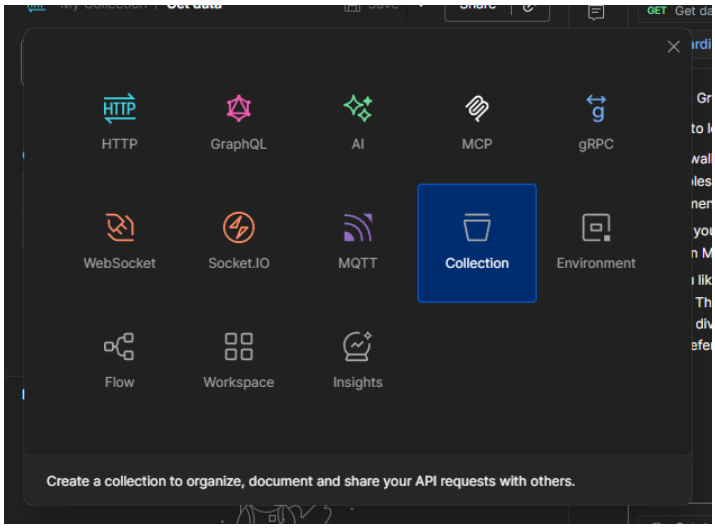

Figure 26: Selección de la opción *Collection* para crear una nueva colección en Postman.

Después de crear la colección, se procedió a realizar las pruebas de registro y consulta de eventos utilizando los métodos **POST** y **GET** proporcionados por el controlador correspondiente.

3.5.1 Consulta inicial de eventos

Antes de registrar nuevos eventos, se ejecutó la solicitud **GET** hacia la ruta:

`http://localhost:8090/api/v1/eventos`

Tal como se muestra en la Figura 27, la respuesta inicial fue un arreglo vacío, lo cual confirma que la base de datos no contenía registros previos.

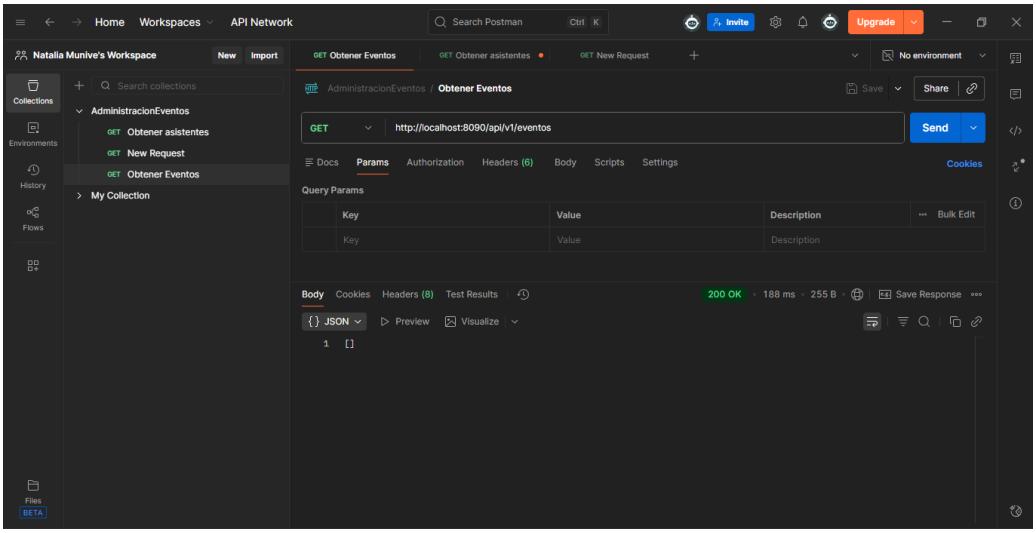

Figure 27: Resultado inicial de la consulta de eventos: la base se encuentra vacía.

3.5.2 Registro de eventos en Postman

Una vez verificado que no existían registros, se procedió a crear diferentes eventos mediante solicitudes **POST**. Cada solicitud fue enviada en formato `JSON` a la ruta:

`http://localhost:8090/api/v1/eventos`

A continuación se describen las pruebas realizadas para cada evento registrado.

Registro del evento *Beauty Glam Fest* En la Figura 28 se muestra la solicitud **POST** correspondiente al evento de belleza denominado **Beauty Glam Fest**. La API respondió con un objeto JSON que incluye el campo `idEvento`, confirmando que el registro fue exitoso.

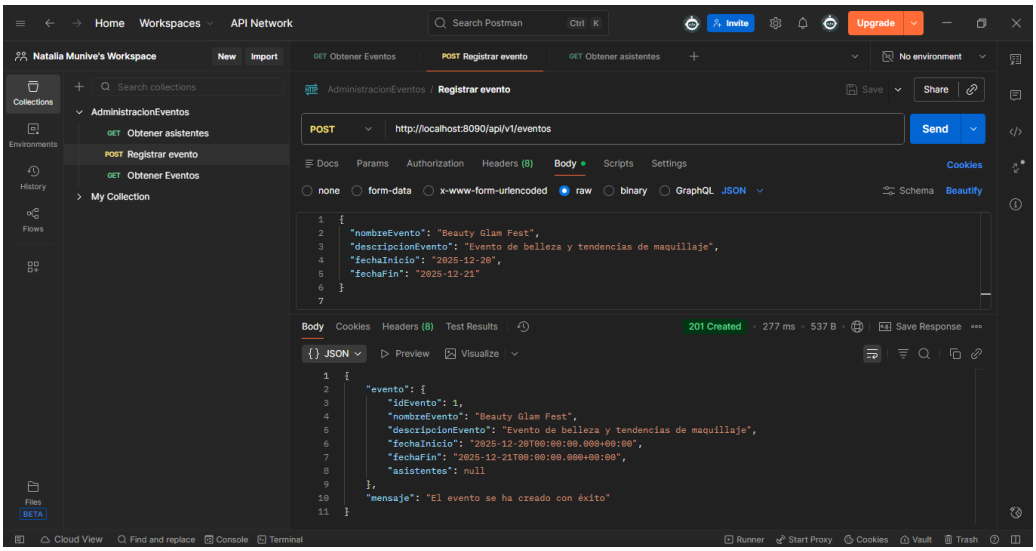

Figure 28: Registro del evento *Beauty Glam Fest*.

Registro del evento *Skincare Day* El siguiente registro corresponde al evento **Skincare Day**, enfocado en rutinas de cuidado facial. La Figura 29 muestra tanto la solicitud como la respuesta satisfactoria del servidor.

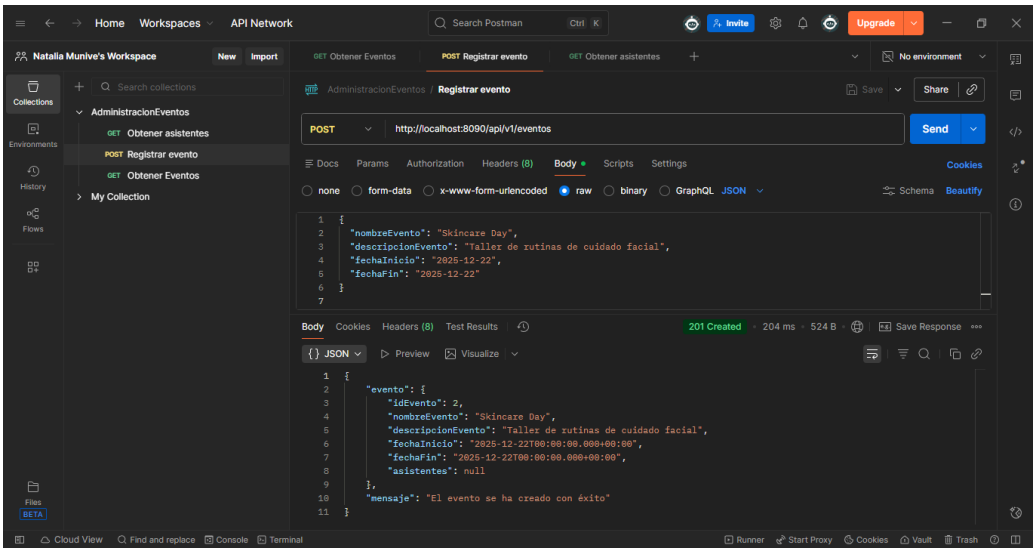

Figure 29: Registro del evento *Skincare Day*.

Registro del evento *Urban Fashion Expo* Posteriormente se registró el evento **Urban Fashion Expo**, relacionado con moda urbana y tendencias streetwear. La Figura 30 muestra la estructura enviada y la confirmación de creación.

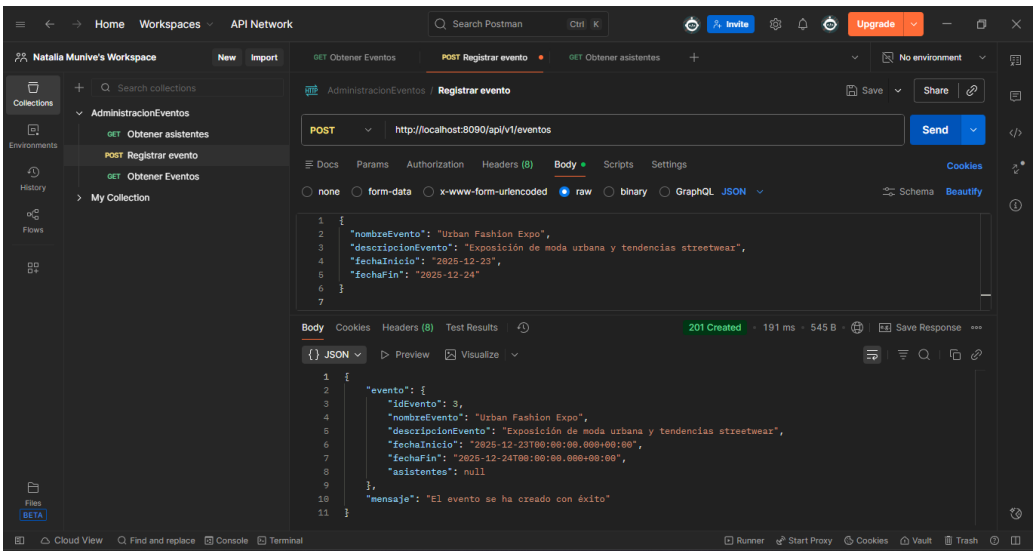

Figure 30: Registro del evento *Urban Fashion Expo*.

Registro del evento *Gaming Experience* La Figura 31 muestra la solicitud de creación del evento **Gaming Experience**, orientado a actividades relacionadas con videojuegos. El servidor devolvió un código `201 Created`, indicando éxito en la operación.

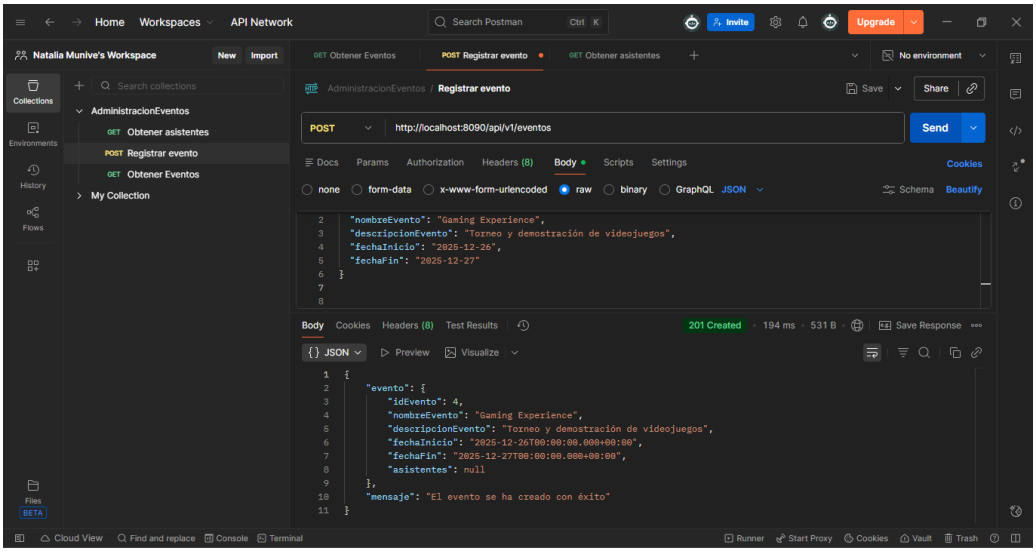

Figure 31: Registro del evento *Gaming Experience*.

Registro del evento *Yoga & Wellness Retreat* Finalmente, en la Figura 32 se presenta el registro del evento de bienestar **Yoga & Wellness Retreat**. El servicio respondió adecuadamente, asignando su respectivo identificador.

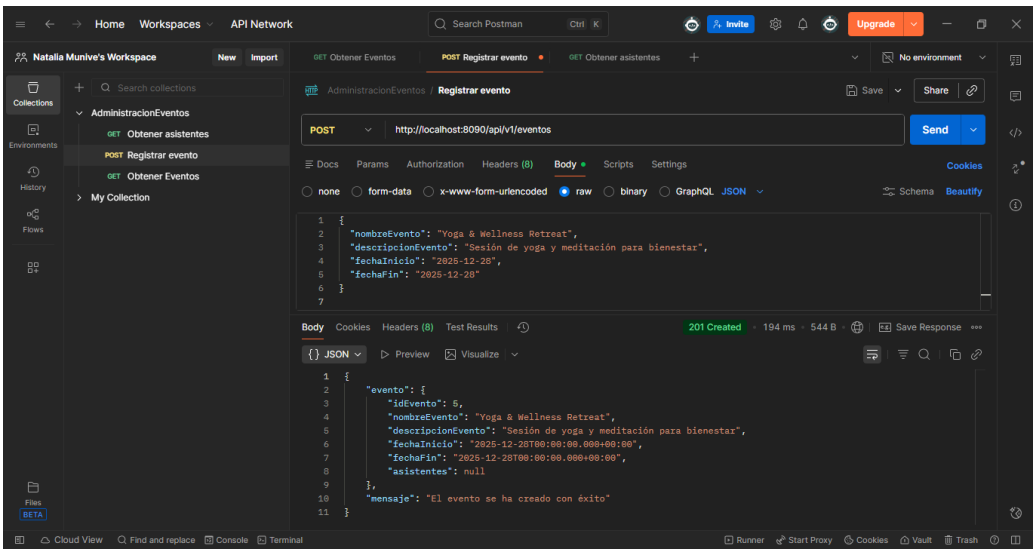

Figure 32: Registro del evento *Yoga & Wellness Retreat*.

3.5.3 Consulta de la lista completa de eventos

Una vez registrados los cinco eventos, se ejecutó nuevamente la solicitud **GET** para obtener la lista actualizada. La Figura 33 muestra la respuesta generada, donde pueden observarse todos los eventos previamente registrados, cada uno con su respectivo `idEvento` y la estructura definida en el modelo.

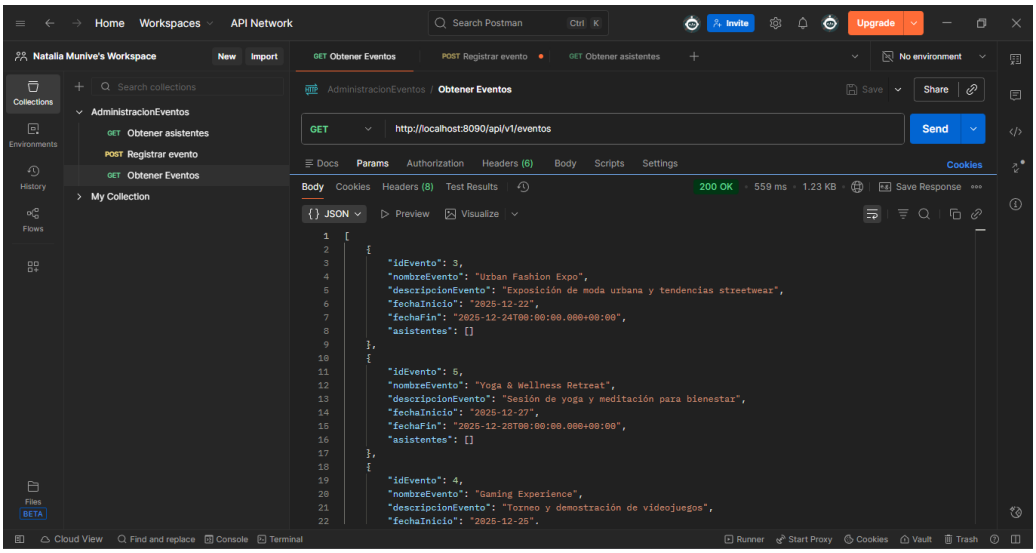

Figure 33: Consulta de la lista completa de eventos registrados.

3.6 Pruebas de Registro y Consulta de Asistentes en Postman

Para validar el correcto funcionamiento del módulo de administración de asistentes dentro del sistema, se empleó nuevamente la herramienta **Postman**. El proceso consistió en consultar inicialmente la lista de asistentes, registrar nuevos participantes y verificar posteriormente que los datos fueron almacenados de manera adecuada.

3.6.1 Consulta inicial de asistentes

Antes de realizar cualquier registro, se ejecutó una solicitud `GET` hacia el endpoint:

`http://localhost:8090/api/v1/asistentes`

La respuesta obtenida fue una lista vacía, lo cual confirma que aún no existían asistentes registrados en la base de datos al inicio de las pruebas (véase Figura 34).

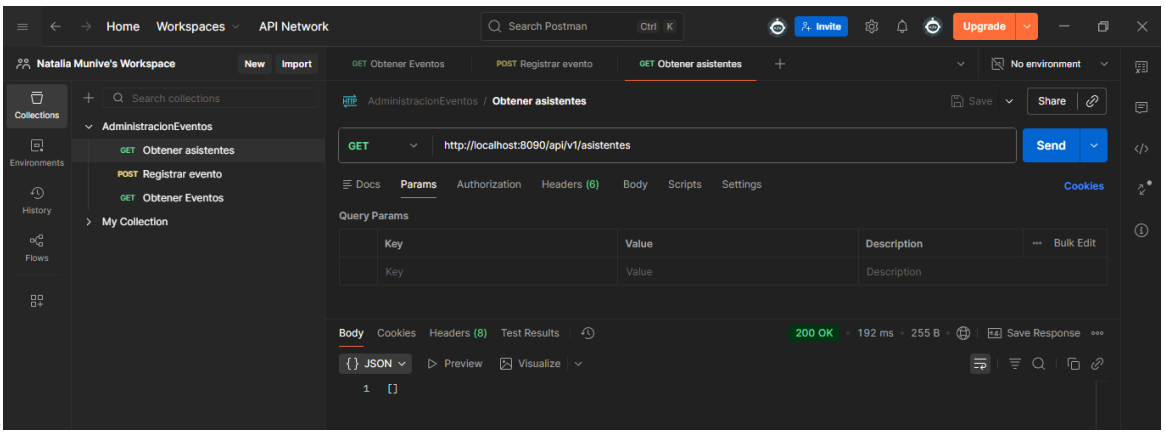

Figure 34: Consulta inicial de asistentes en Postman.

3.6.2 Registro de nuevos asistentes

El registro de asistentes se realizó mediante solicitudes `POST` hacia el endpoint:

`http://localhost:8090/api/v1/asistentes`

Cada solicitud incluyó un cuerpo en formato **JSON** con los datos del asistente, incluyendo su nombre, apellidos, correo electrónico, fecha de registro y el identificador del evento al que está asociado.

A continuación se muestran los registros realizados:

Registro de Sofía Martínez Delgado En la Figura 35 se observa la solicitud con los datos de la asistente Sofía Martínez Delgado. El sistema respondió con código **201 Created**, confirmando el registro exitoso.

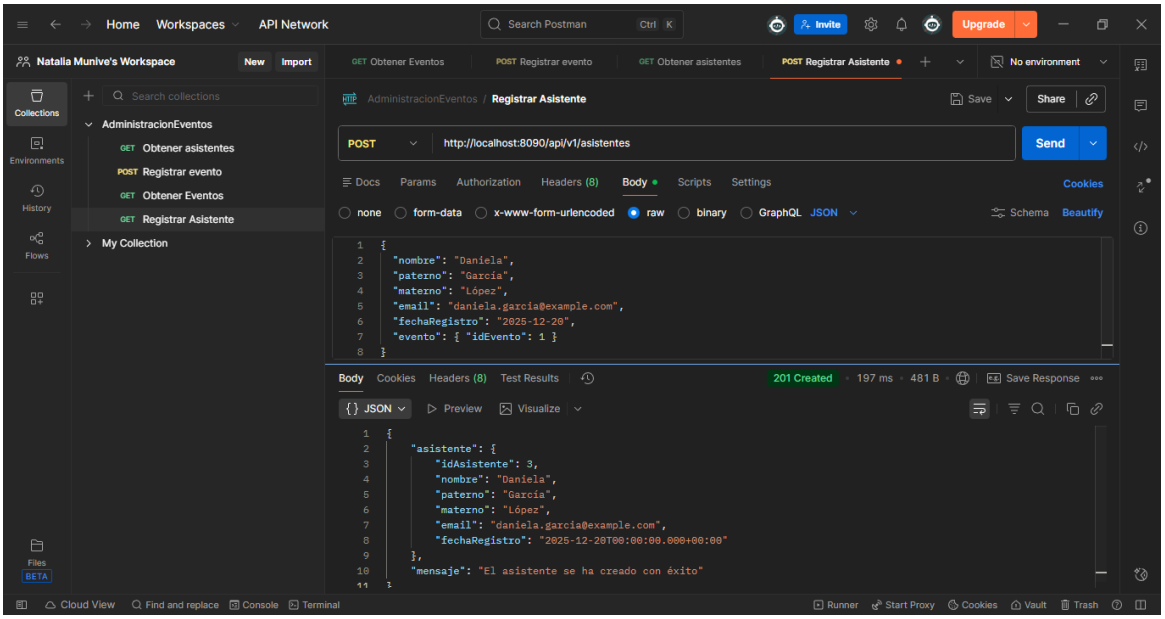

Figure 35: Registro de la asistente Sofía Martínez Delgado.

Registro de Daniela García López De manera similar, en la Figura 36 se muestra el registro de la asistente Daniela García López. El servidor devolvió una respuesta satisfactoria, indicando que los datos fueron almacenados correctamente.

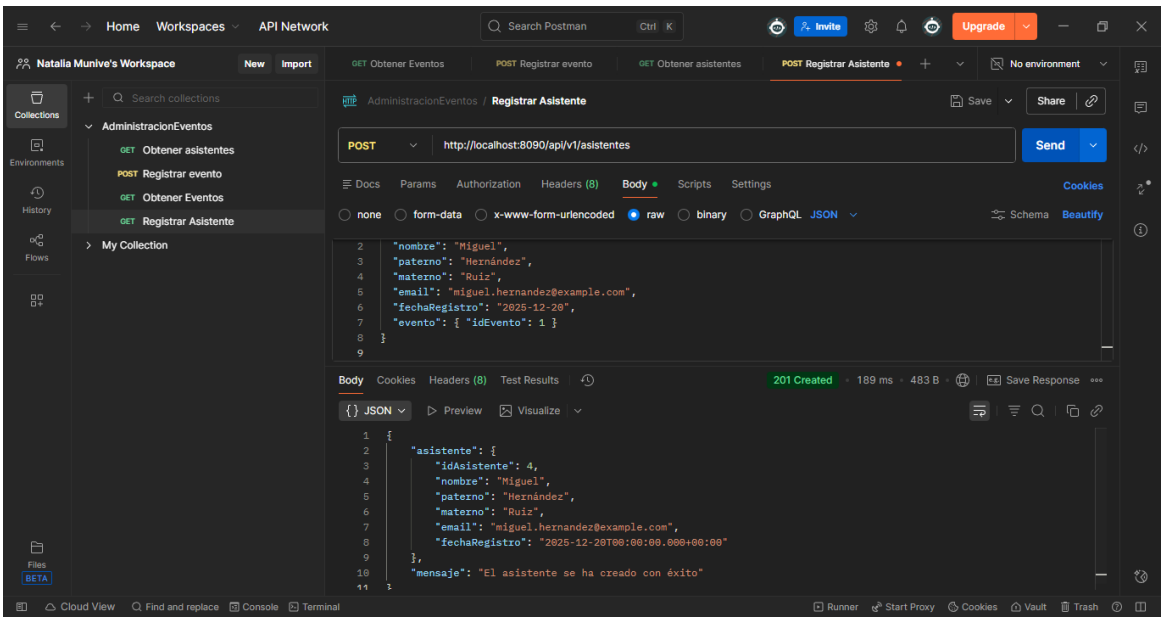

Figure 36: Registro de la asistente Daniela García López.

Registro de Miguel Hernández Ruiz La Figura 37 corresponde al registro del asistente Miguel Hernández Ruiz, quien también quedó vinculado al evento con identificador 1. El sistema confirmó nuevamente el éxito del registro.

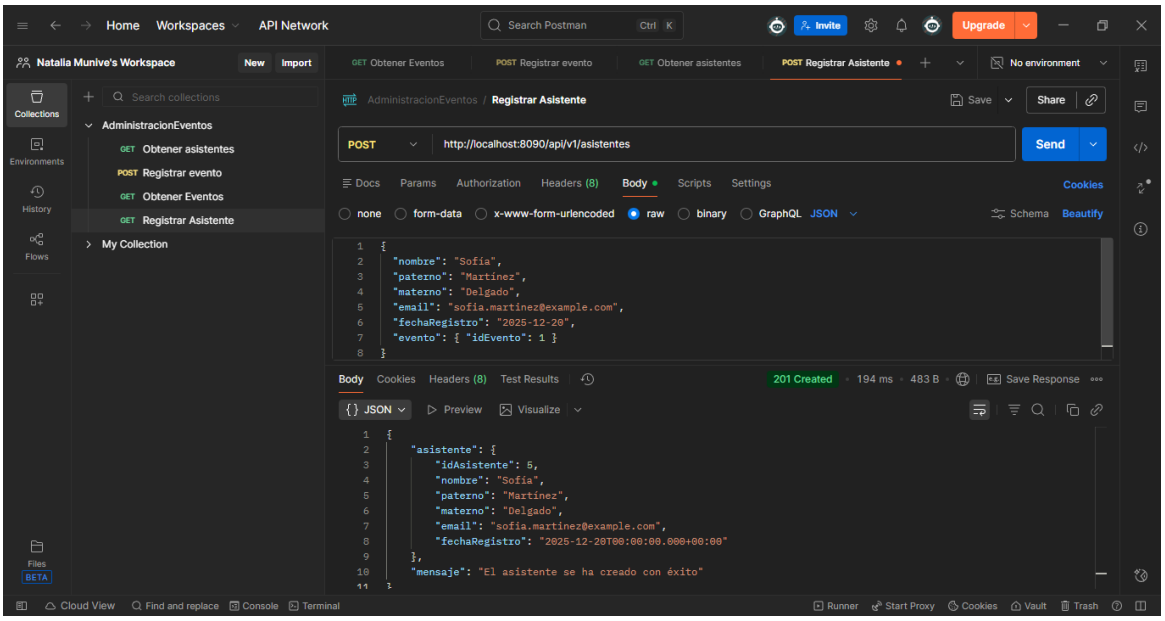

Figure 37: Registro del asistente Miguel Hernández Ruiz.

Registro de un cuarto asistente La Figura 38 muestra el registro de un cuarto asistente adicional. El comportamiento del servicio fue consistente, devolviendo código **201**.

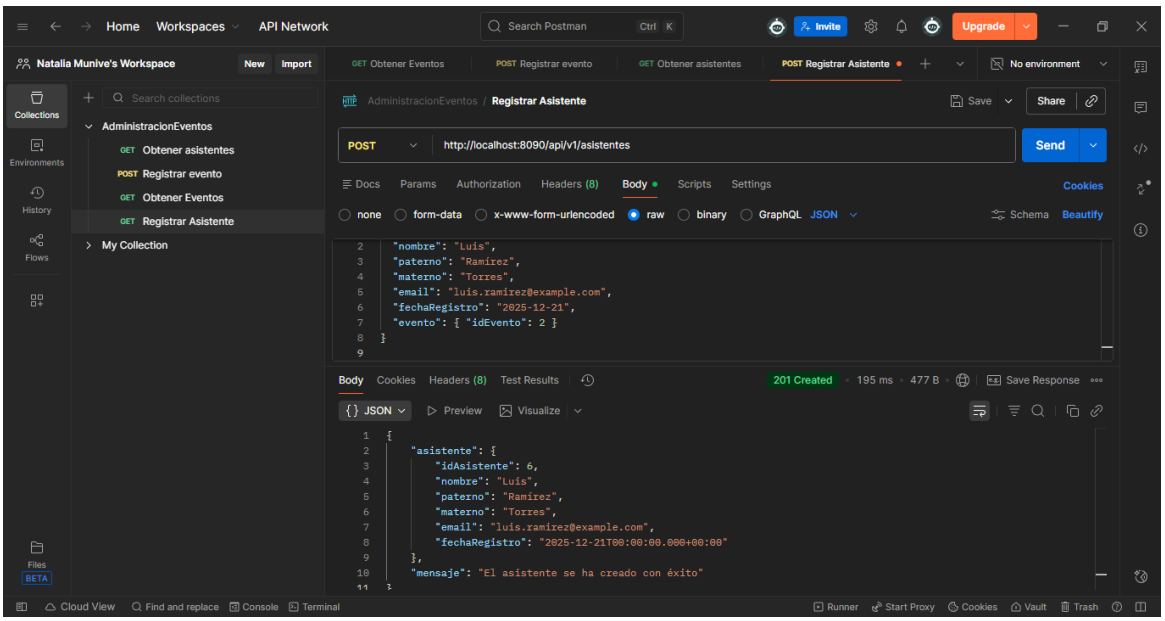

Figure 38: Registro de un cuarto asistente en el sistema.

Registro de un quinto asistente En la Figura 39 se observa el registro de otro asistente, empleando nuevamente el mismo formato JSON. El servicio respondió correctamente, indicando que el asistente fue guardado en la base de datos.

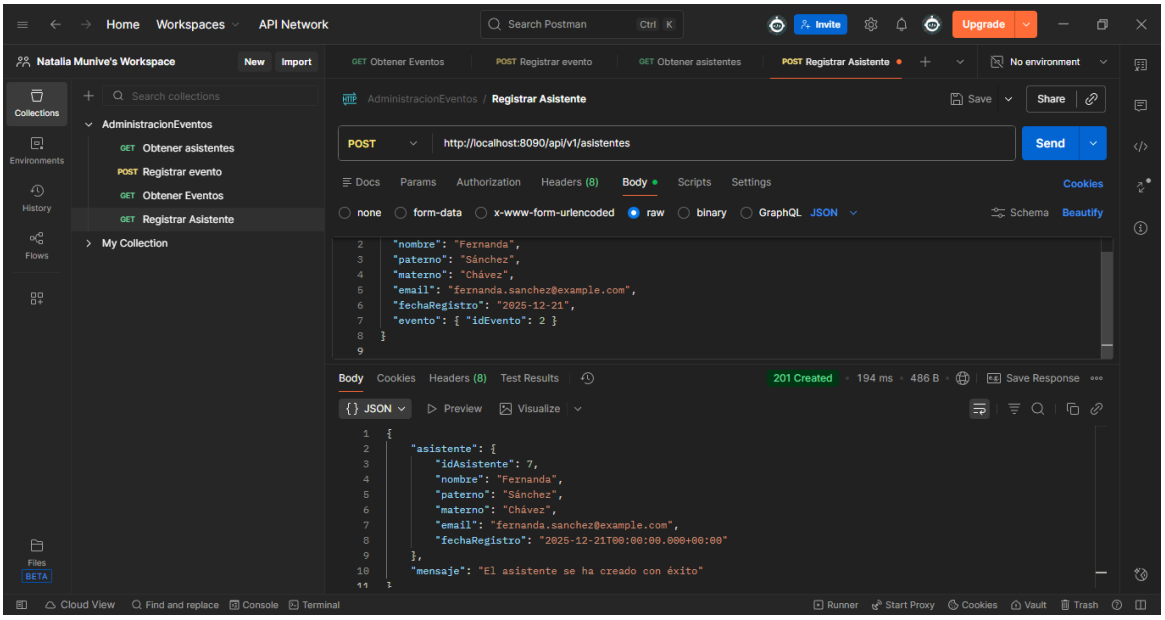

Figure 39: Registro de un quinto asistente.

Registro de un sexto asistente Como se aprecia en la Figura 40, el proceso de registro fue exitoso y coherente con las pruebas anteriores.

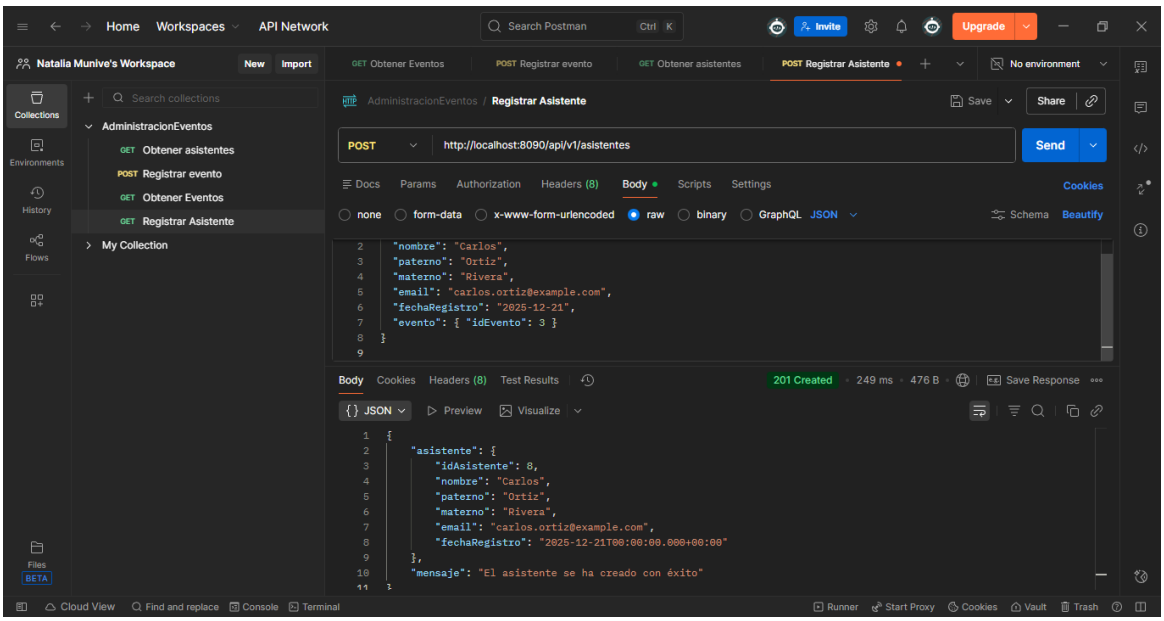

Figure 40: Registro de un sexto asistente.

Registro de un séptimo asistente La Figura 41 muestra el registro de un séptimo asistente, validando nuevamente la consistencia del módulo.

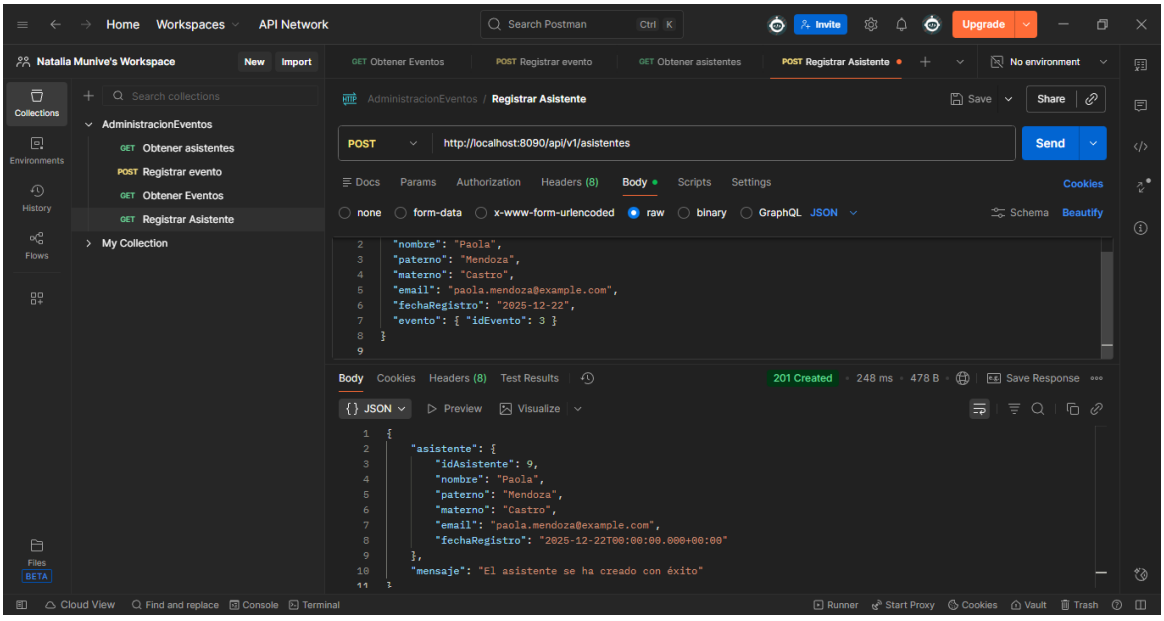

Figure 41: Registro de un séptimo asistente.

Registro de un octavo asistente En la Figura 42 se presenta el registro correspondiente al octavo asistente.

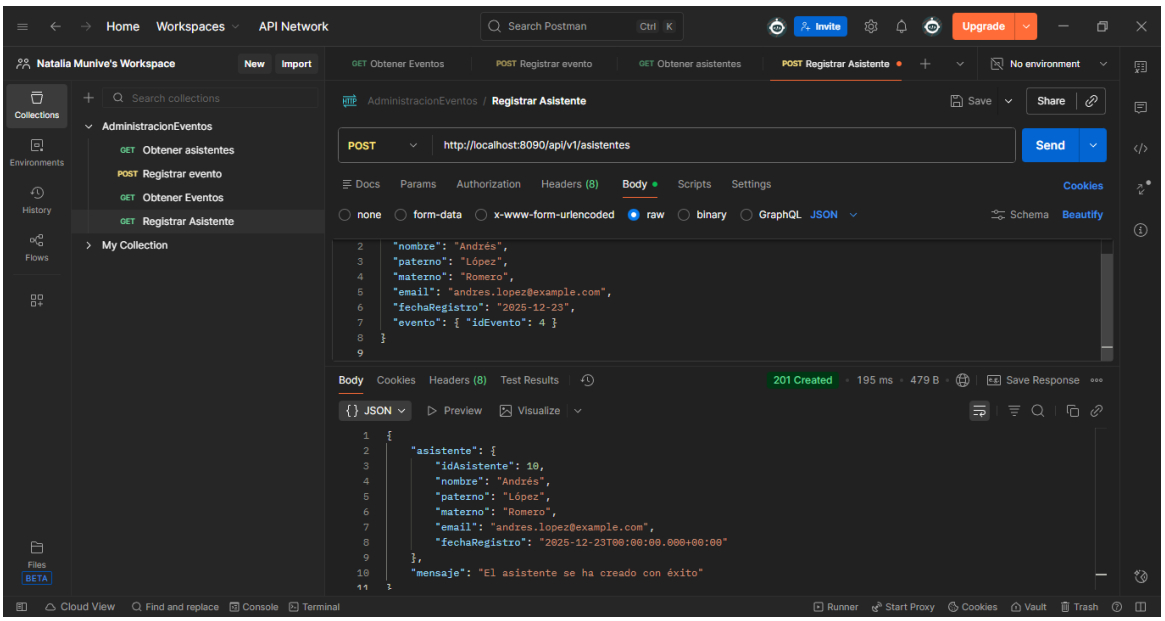

Figure 42: Registro de un octavo asistente.

Registro de un noveno asistente Como se muestra en la Figura 43, se registró correctamente un noveno asistente.

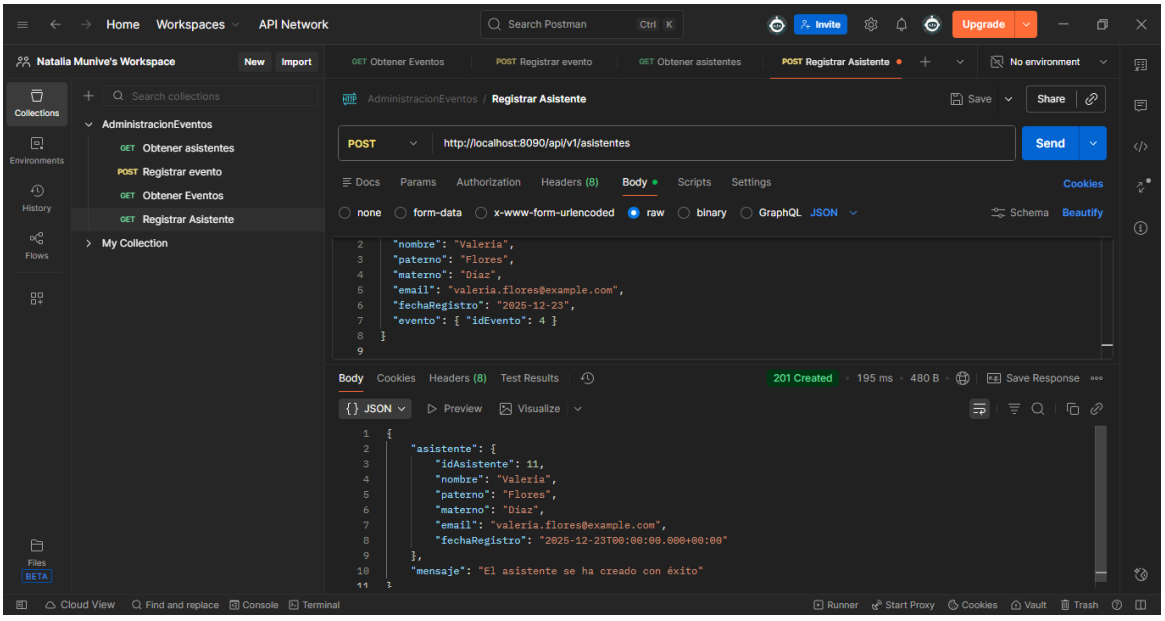

Figure 43: Registro de un noveno asistente.

Registro de un décimo asistente Finalmente, la Figura 44 corresponde al registro del décimo asistente, cerrando la etapa de pruebas de inserción.

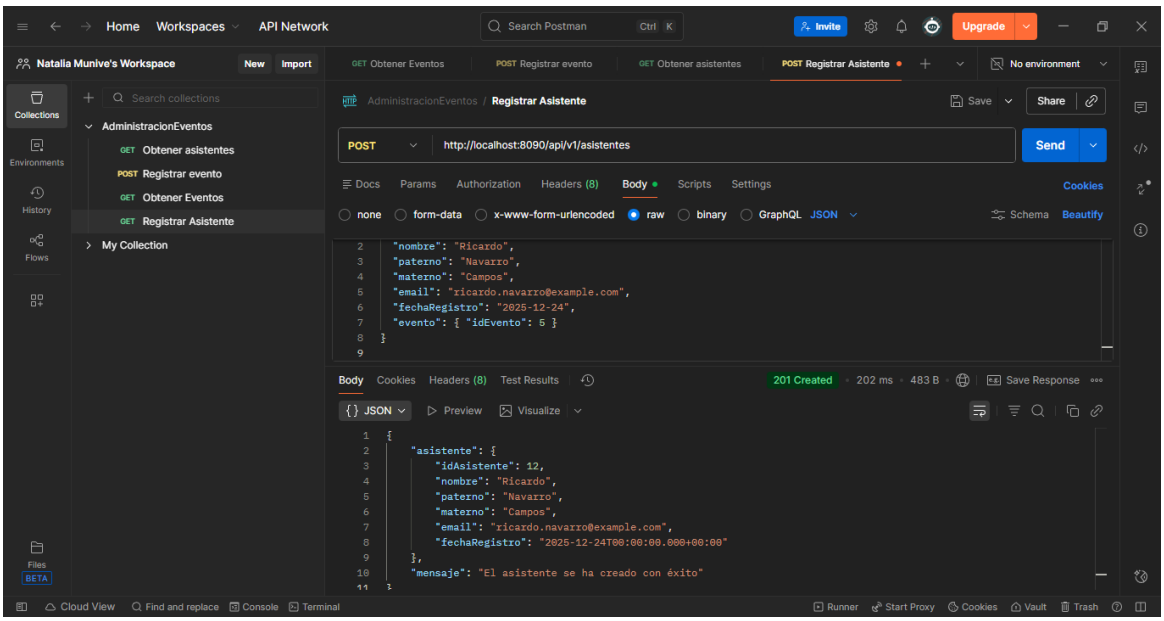

Figure 44: Registro de un décimo asistente.

3.6.3 Consulta final de asistentes

Una vez completado el registro de todos los asistentes, se realizó nuevamente una solicitud `GET` al endpoint:

`http://localhost:8090/api/v1/asistentes`

En esta consulta fue posible observar la lista completa de los asistentes previamente registrados, cada uno con su identificador, datos personales, fecha de registro y la referencia al evento correspondiente. La Figura 45 muestra la respuesta completa obtenida desde Postman, confirmando que el servicio de consulta funciona correctamente y que los datos fueron almacenados de manera íntegra en la base de datos.

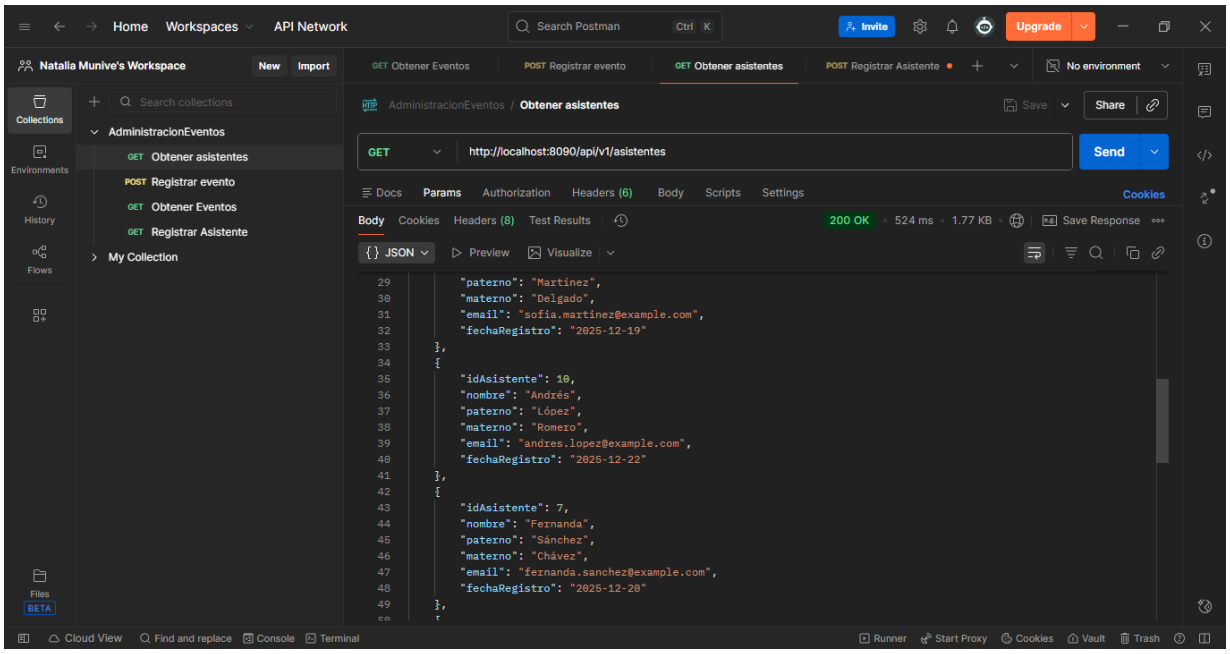

Figure 45: Consulta final de asistentes registrados en el sistema.

3.6.4 Eliminación de Asistentes

Además de registrar y consultar información, el sistema permite eliminar asistentes vinculados a un evento. Para realizar esta operación, se añadió en Postman la solicitud **DELETE** correspondiente dentro de la colección **AdministracionEventos**.

En la Figura 46, se muestra la eliminación del asistente con **idAsistente = 12**, quien pertenecía al evento con **idEvento = 5**. La respuesta del servicio confirma la operación mediante el mensaje:

"El asistente se ha eliminado con éxito"

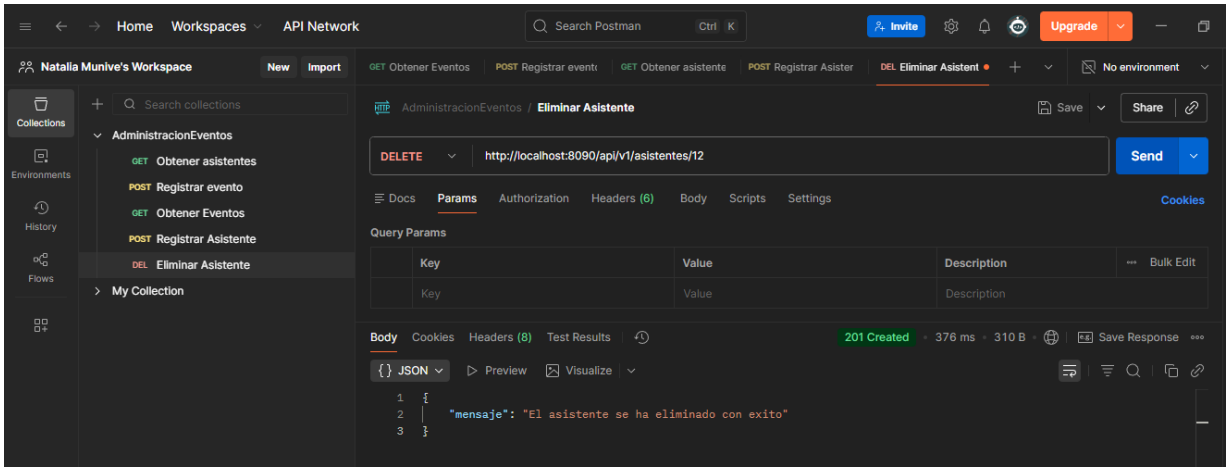

Figure 46: Solicitud DELETE para eliminar al asistente con id 12.

3.6.5 Eliminación de Eventos

Una vez eliminado el asistente asociado, se procedió a eliminar el evento correspondiente. En el ejemplo mostrado en la Figura 47, se utilizó la ruta:

`/api/v1/eventos/5`

lo que indica la eliminación del evento con **idEvento = 5**. La API respondió con el mensaje:

"El evento se ha eliminado con éxito"

confirmando la correcta ejecución de la operación.

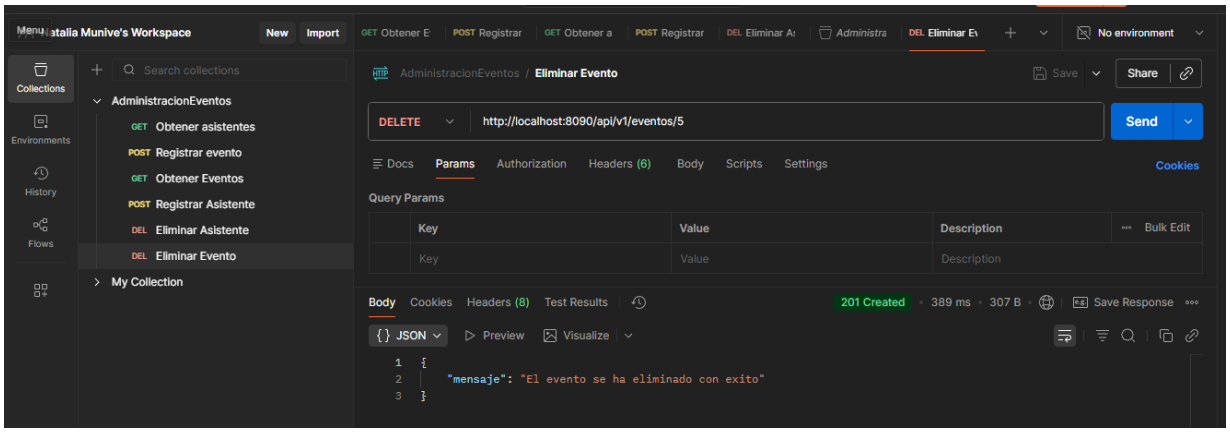

Figure 47: Eliminación del evento con id 5 mediante solicitud DELETE.

3.6.6 Verificación de Tablas en YugabyteDB

Una vez realizadas las operaciones de registro y consulta de eventos y asistentes, se procedió a verificar directamente en la plataforma **YugabyteDB Managed** que las tablas correspondientes hubieran sido creadas correctamente dentro de la base de datos configurada para este proyecto.

Desde el panel principal del clúster (véase Figura 48), se accedió a la sección **Databases**, donde se muestra la lista de bases disponibles. En este caso, la base denominada **yugabyte** aparece como activa y con un tamaño aproximado de 2 MB, lo que indica que contiene datos generados por las pruebas realizadas.

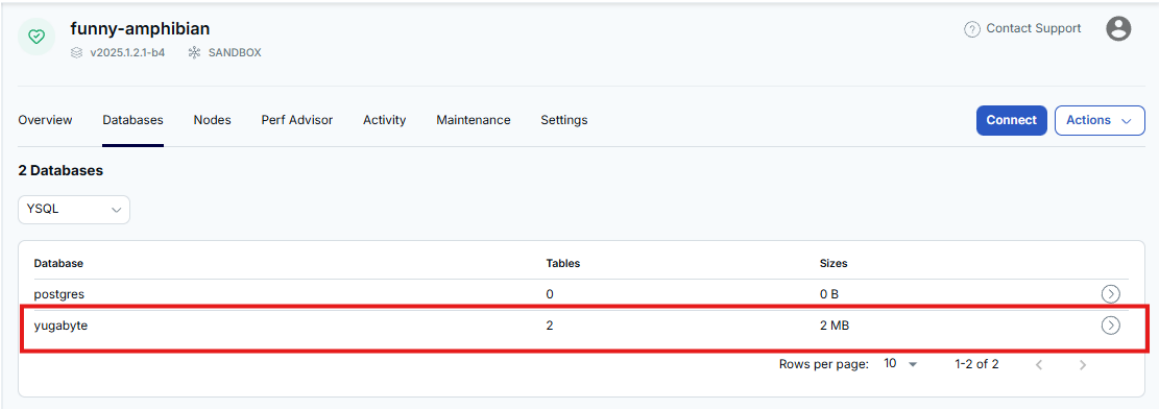

Figure 48: Vista general de bases de datos en el clúster YugabyteDB.

Al seleccionar la base de datos **yugabyte**, el sistema muestra información detallada de sus tablas internas (Figura 49). En este proyecto se crearon correctamente las tablas:

- **evento** – Contiene los datos de los eventos registrados.
- **asistente** – Almacena los registros de asistentes asociados a cada evento.

Ambas tablas se encuentran en buen estado, con un tamaño aproximado de 1 MB cada una, confirmando la correcta persistencia de los datos enviados mediante la API REST.

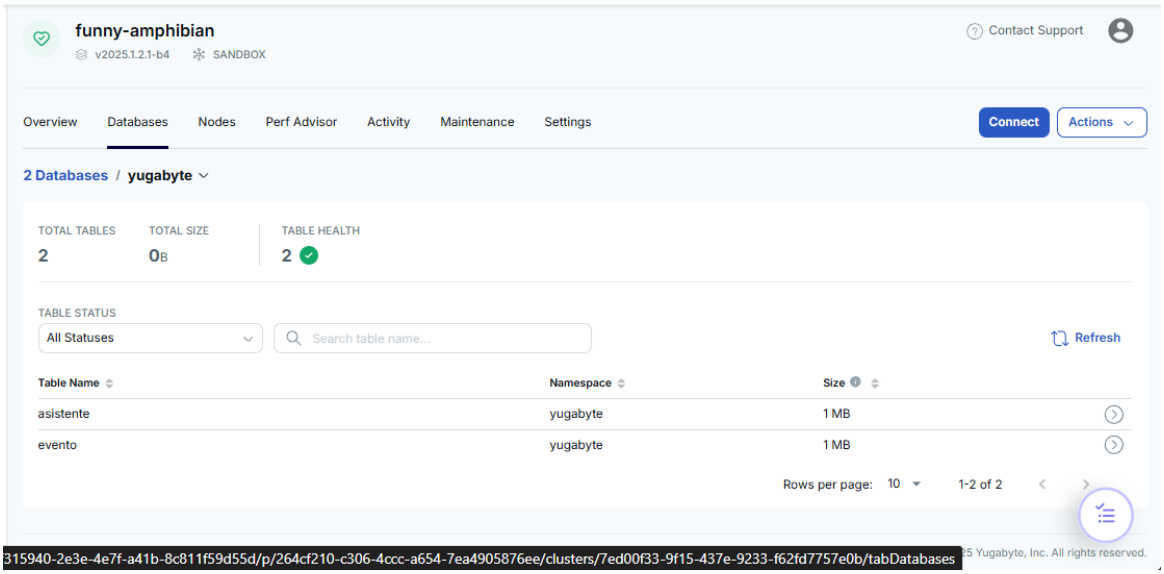

Figure 49: Tablas *evento* y *asistente* creadas en YugabyteDB.

Finalmente, también se verificó la respuesta JSON desde el navegador web (Figura 50), donde se observan los asistentes almacenados en la base de datos, junto con sus respectivos IDs y fechas de registro.

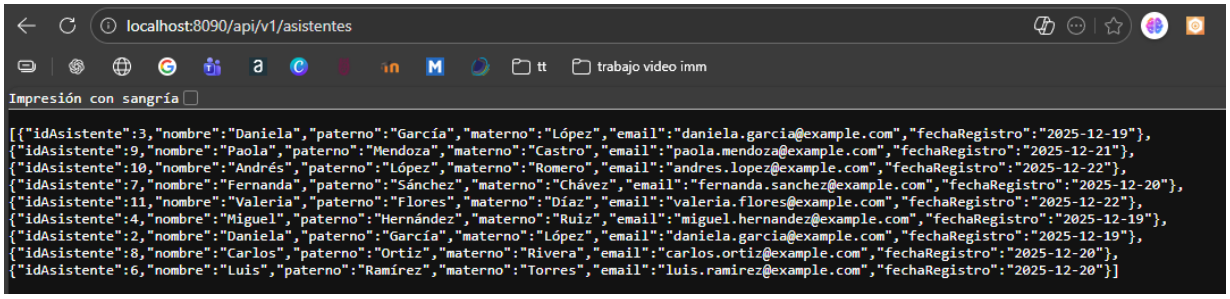

Figure 50: Consulta de asistentes desde el navegador utilizando la API REST.

De igual forma, se confirmaron los datos de todos los eventos junto con sus asistentes asociados (Figura 51), validando que la relación *uno a muchos* entre `evento` y `asistente` funciona correctamente.

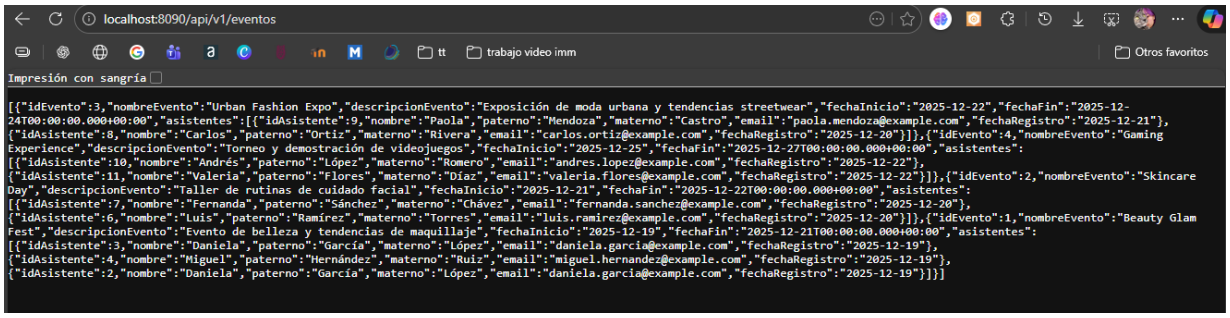

Figure 51: Consulta de eventos y asistentes desde el navegador.

4 Resultados

En esta sección se presentan las evidencias visuales correspondientes al correcto funcionamiento de la aplicación, así como la verificación de la persistencia de datos en YugabyteDB.

1. Ejecución correcta del proyecto Spring Boot

La aplicación inició de manera satisfactoria, estableciendo conexión con YugabyteDB mediante el controlador JDBC compatible con PostgreSQL.

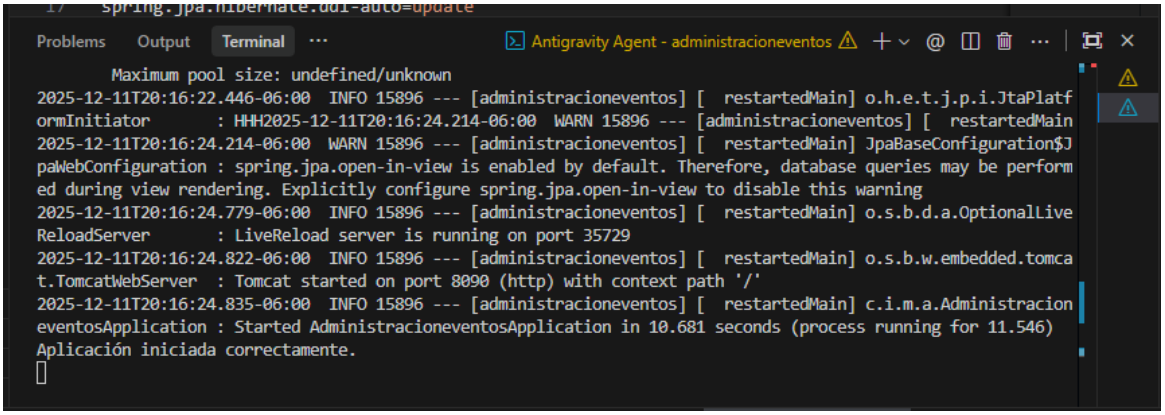

Figure 52: Ejecución correcta del proyecto Spring Boot sin errores de conexión.

2. Registro exitoso de eventos mediante Postman

Se registraron cinco eventos utilizando solicitudes `POST`. Postman devolvió códigos 201 y los objetos creados. Se hizo la consulta para ver que esten completos y la API respondió exitosamente con código HTTP 200.

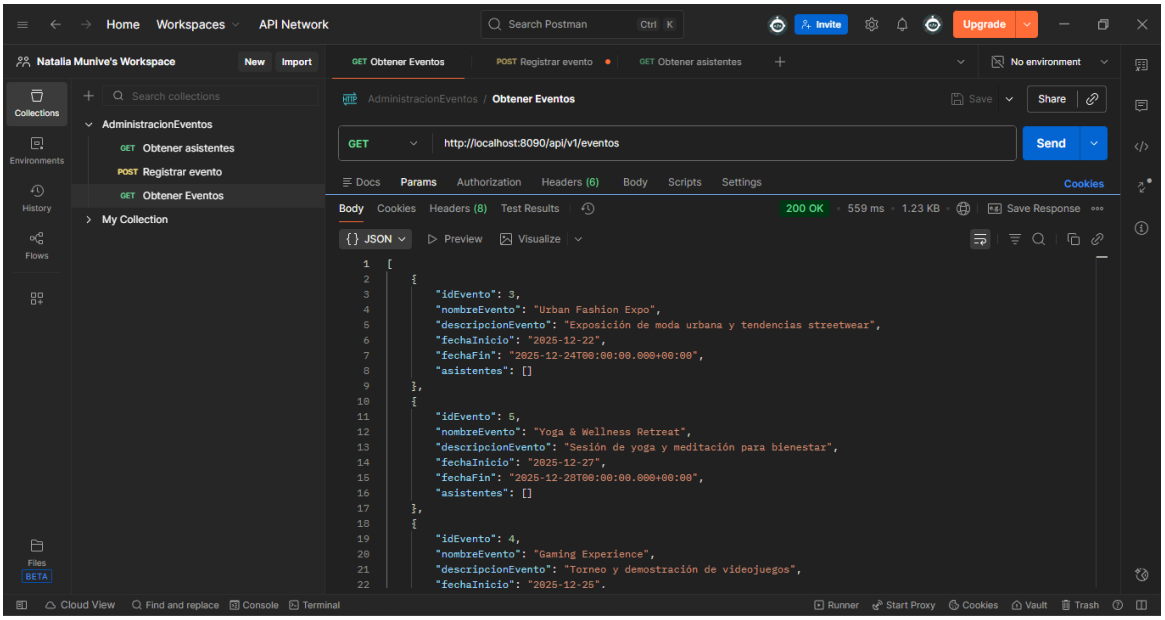

Figure 53: Registro de un evento utilizando Postman.

3. Registro exitoso de asistentes

Se crearon diez asistentes asociados a distintos eventos, la API respondió exitosamente con código HTTP 201. Se hizo la consulta para ver que esten completos y la API respondió exitosamente con código HTTP 200.

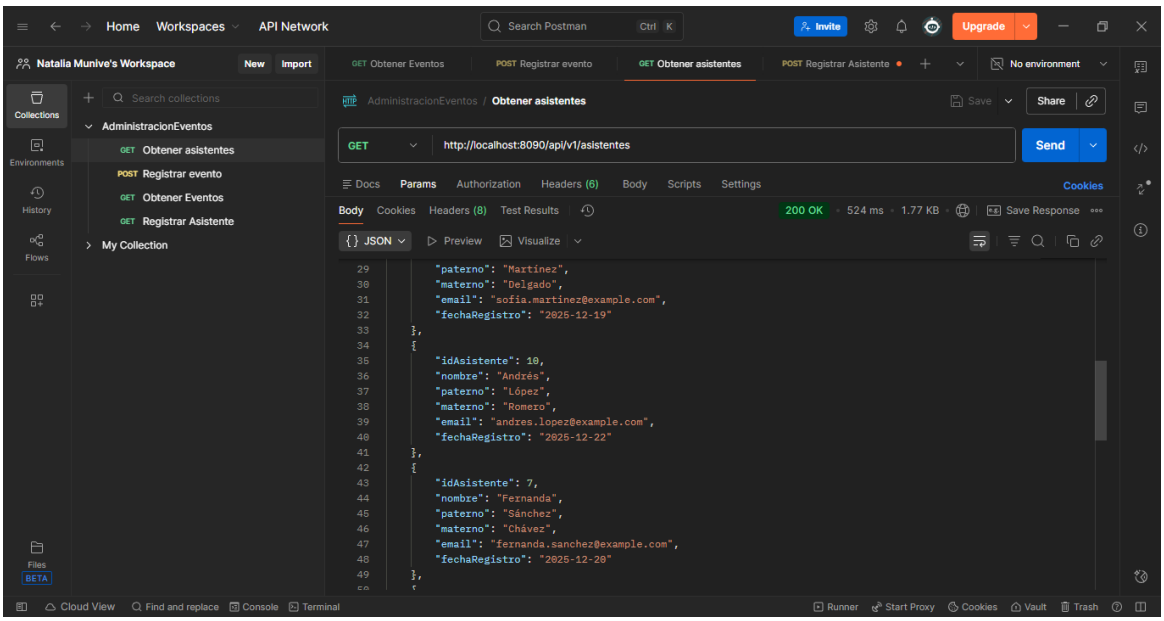

Figure 54: Registro de asistentes mediante Postman.

4. Eliminación de registros

Se probó la operación `DELETE` para eliminar un asistente (ID 12) y un evento (ID 5).

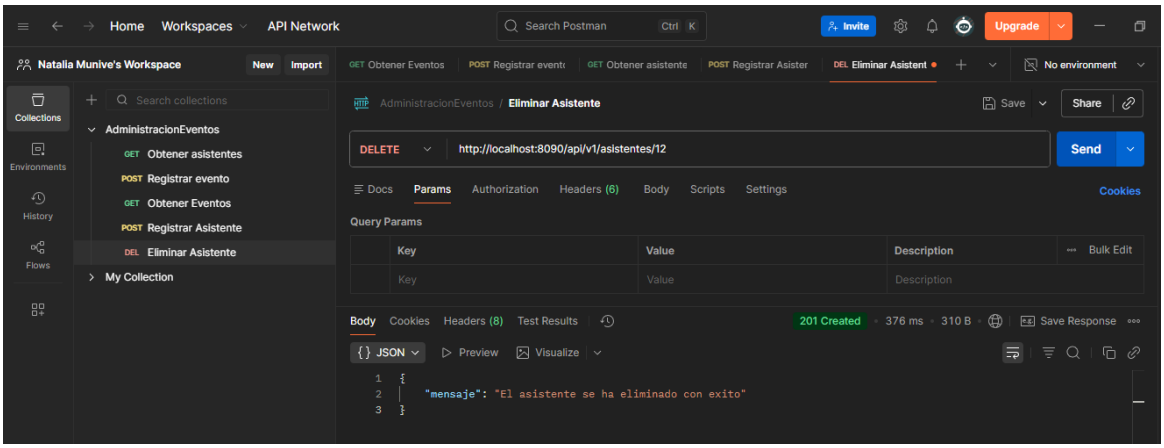

Figure 55: Eliminación del asistente con ID 12.

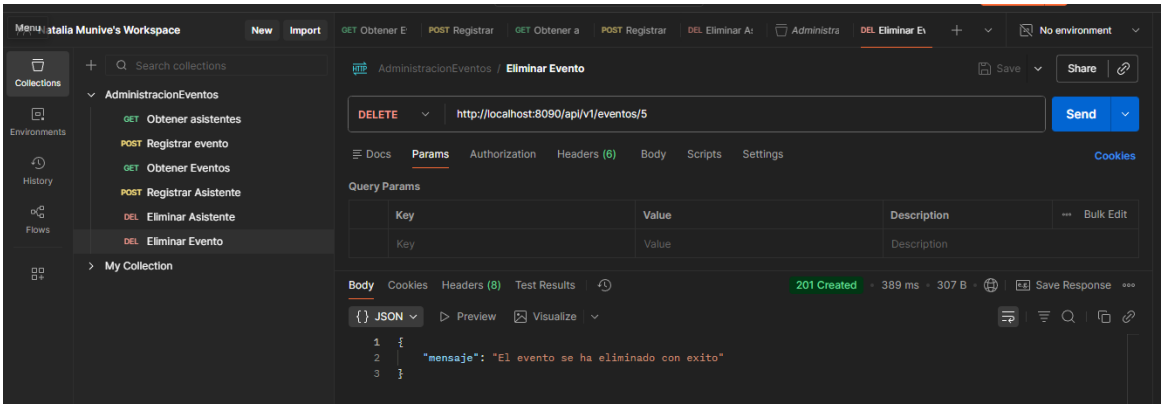

Figure 56: Eliminación del evento con ID 5.

5. Verificación de la creación de tablas en YugabyteDB

Se comprobó desde la consola de Yugabyte Cloud que las tablas `evento` y `asistente` fueron generadas de forma correcta con Hibernate.

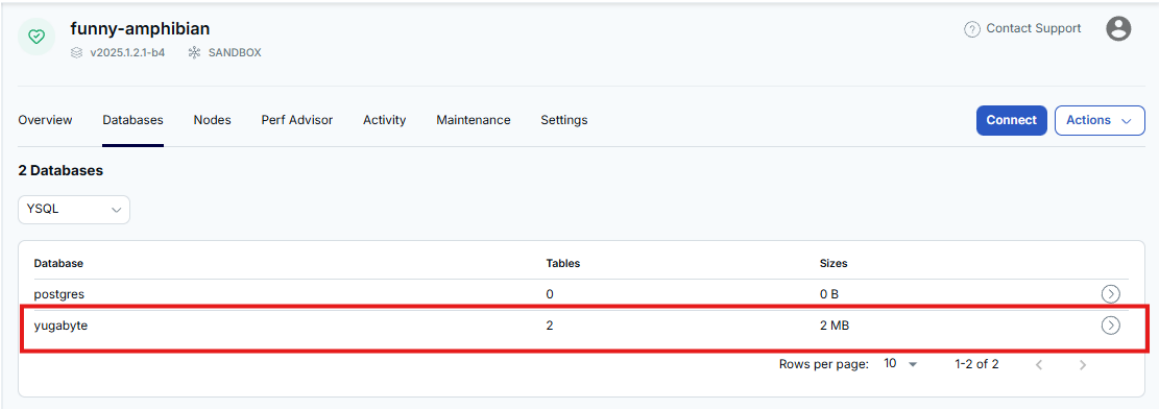

Figure 57: Base de datos `yugabyte` creada dentro del clúster.

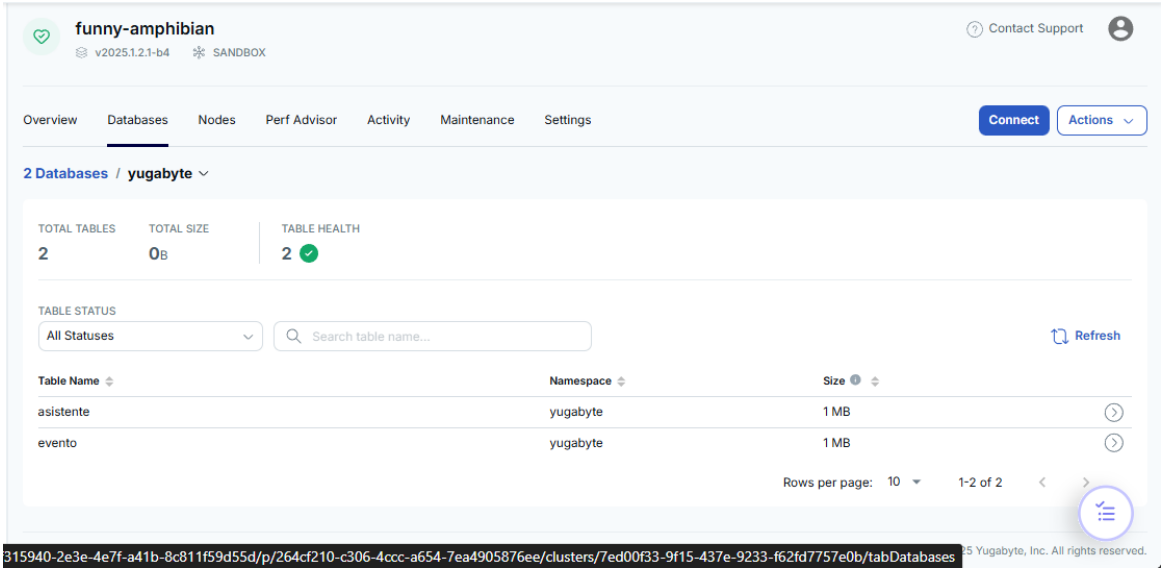

Figure 58: Verificación de las tablas `asistente` y `evento`.

6. Visualización de datos desde el navegador

Las rutas públicas expuestas por los controladores REST permiten consultar la información en formato JSON.

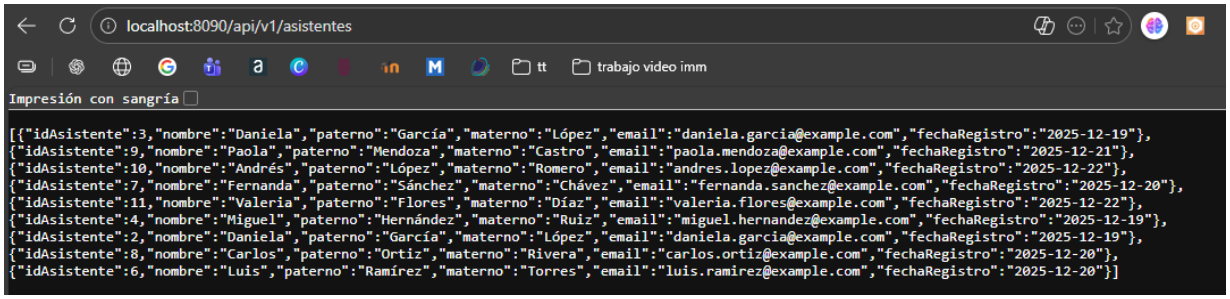

Figure 59: Consulta de asistentes desde el navegador.

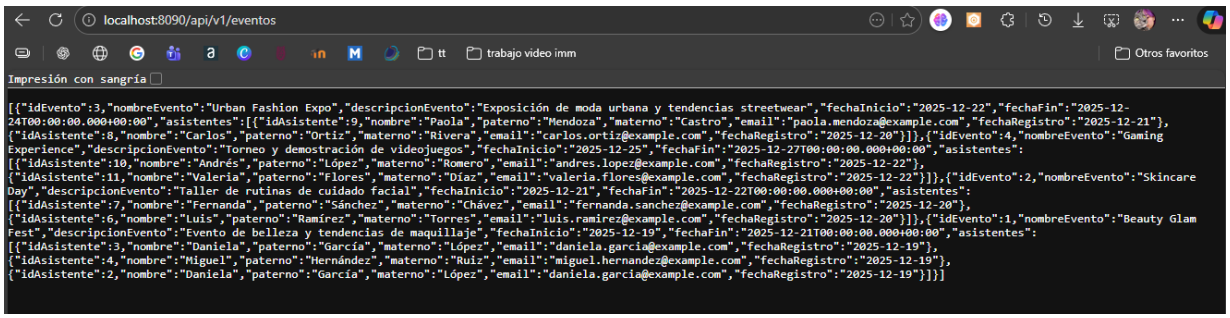

Figure 60: Consulta de eventos desde el navegador.

Estos resultados permiten confirmar el funcionamiento integral de la API, la persistencia en YugabyteDB y la correcta interacción de todos los componentes.

5 Conclusión

La práctica realizada permitió comprender de manera integral el proceso completo para la creación, configuración y utilización de una base de datos distribuida mediante YugabyteDB, así como su integración con una aplicación desarrollada en Spring Boot. A través de las diferentes etapas —creación del clúster, configuración de la conexión segura mediante certificados SSL, definición de entidades, ejecución del proyecto y pruebas con Postman— fue posible validar el funcionamiento adecuado de los servicios REST y la persistencia de los datos en una base distribuida.

Los resultados confirmaron que YugabyteDB mantiene compatibilidad total con PostgreSQL, permitiendo utilizar controladores JDBC estándar y conservando el soporte para operaciones ACID incluso dentro de un entorno distribuido. Las pruebas de inserción, consulta y eliminación de eventos y asistentes demostraron que la API responde de forma consistente y que la estructura de relaciones entre tablas se gestionó correctamente.

Además, el uso de herramientas como Postman facilitó la verificación del comportamiento de los endpoints, permitiendo observar de forma clara la comunicación entre la aplicación y la base en la nube. Del mismo modo, la inspección directa en el panel de YugabyteDB evidenció la creación automática de tablas y la correcta persistencia de los registros generados durante las pruebas.

En conjunto, esta práctica fortaleció las habilidades necesarias para trabajar con arquitecturas distribuidas, bases administradas en la nube y servicios REST, proporcionando una experiencia completa que integra configuraciones backend, pruebas de API y verificación directa en la plataforma de bases de datos.

Concepto	**Puntos**
1. Portada	Sin
2. Introducción	2
3. Conceptos	2
4. Desarrollo (Tablas, imágenes, etc.)	2
5. Resultados (Imágenes, etc.)	2
6. Conclusiones	1
7. Referencias Bibliográficas	1

Table 1: Criterios de evaluación del reporte

Munive Hernández Erika Natalia

6 Referencias Bibliográficas

[1] YugabyteDB. (2024). *YugabyteDB Documentation*. Disponible en: https://docs.yugabyte.com

[2] PostgreSQL Global Development Group. (2024). *PostgreSQL 16 Documentation*. Disponible en: https://www.postgresql.org/docs/

[3] Spring. (2024). *Spring Boot Reference Documentation*. Disponible en: https://docs.spring.io/spring-boot/

[4] Postman Inc. (2024). *Postman Learning Center*. Disponible en: https://learning.postman.com/

[5] Hibernate. (2024). *Hibernate ORM User Guide*. Disponible en: https://hibernate.org/orm/documentation/

[6] OpenSSL Foundation. (2024). *TLS/SSL Basics*. Disponible en: https://www.openssl.org/docs/