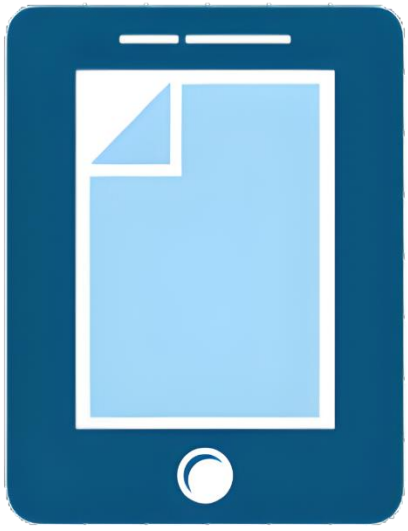
**INFORME DE IMPLEMENTACIÓN**



EVISOR

## Presentación de la solución del sistema

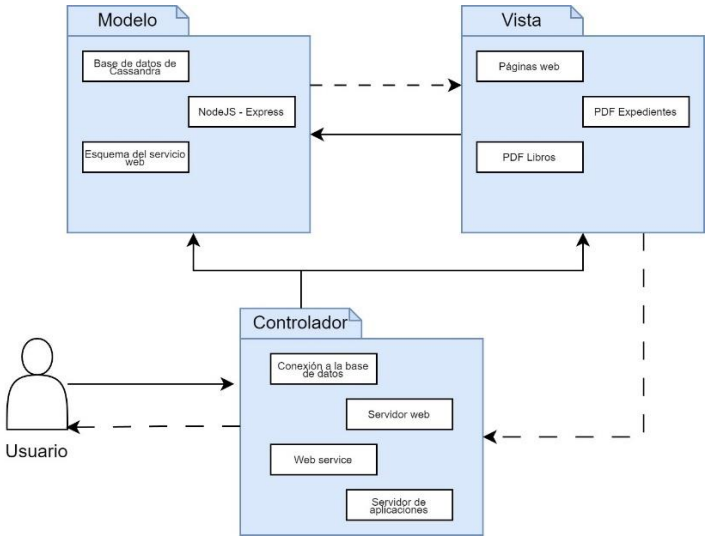
La base fundamental para obtener un sistema sólido es llevar un orden específico y apropiado para la elaboración del sistema en cuestión, ya que se cubre con todas las funcionalidades planificadas, pero a su vez, también será capaz de poder adaptarse a las necesidades y requerimientos futuros, proporcionando a los empleados muchas más características que harán más amenas las labores diarias de su día a día, por lo que dejar unas bases sostenibles es fundamental.

En este informe de implementación se procede a explicar de manera técnica como se ha resuelto el sistema en su totalidad, definiendo la arquitectura de hardware y software, así como las tecnologías que se implementaron, las metodologías de trabajo utilizadas durante el desarrollo, también algunas herramientas que fueron útiles para diferentes propósitos que aportaron a la construcción del sistema y también la estructura de datos utilizada para el almacenamiento de información.

## Patrón de software utilizado en el sistema

El patrón de software que mejor se adapta a las necesidades y los procesos a realizar es la de Modelo, Vista y controlador (MVC), por sus muchas virtudes que permiten un desarrollo completo y que, a su vez, permite una organización estable y entendible, a continuación, en la siguiente figura, se puede observar a grandes rasgos la forma distintiva en la que este patrón de software funciona:

*Patrón MVC de la solución*




Nota. Diagrama que demuestra el tipo de arquitectura con la que se fundamentó el sistema.

## Modelo

En esta capa se coordina la comunicación con la base de datos según las necesidades del sistema, se ayuda del controlador para recibir y transmitir la información para propagarla por el sistema en los lugares correspondientes.

La comunicación con la base de datos se hace por medio de consultas, estas son direccionadas a un conector (cassandra-driver) que gestiona la conexión con la base de datos y ejecuta los procedimientos solicitados.

**Controlador**

Esta capa responde a las acciones realizadas por el usuario u otros servicios, también permite efectuar la comunicación entre la capa Modelo y la capa Vista.

En la plataforma de desarrollo Angular se definieron servicios que actúan como proveedores (providers) que permitirán decidir qué información obtener de la base de datos y a que vista dirigir la información según las necesidades y acciones que efectúe el usuario.

**Vista**

Aquí se manipula y se muestra la información que el usuario solicita, de igual manera permite al usuario interactuar con las vistas para luego obtener la acción que el usuario solicita y procesarla para ejecutar acciones o procesos.

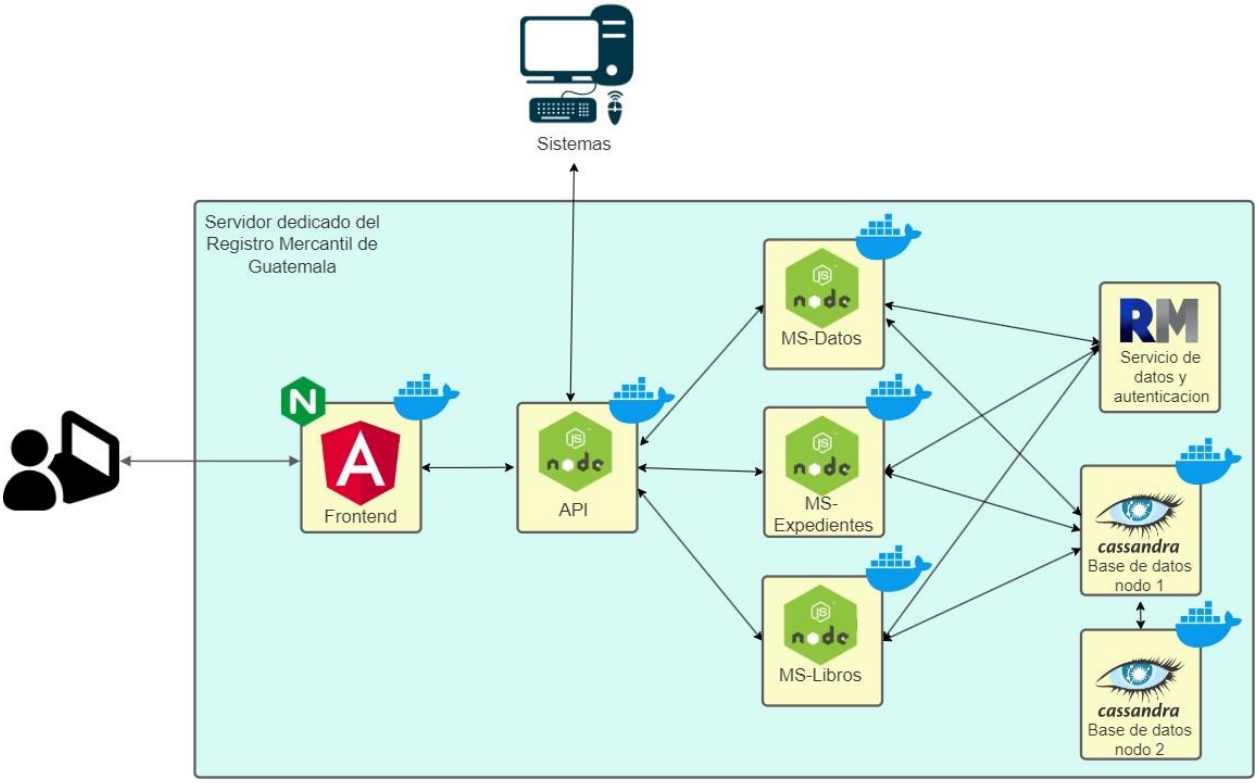
Las vistas se muestran por medio de HTML, estas pueden ser accedidas por los usuarios registrados, en base a los roles y permisos que disponga cada uno se podrá acceder a ciertas áreas de la aplicación.

**Arquitectura de hardware**

La aplicación se encarga de dar el flujo completo de consulta e impresión de expedientes y libros que actualmente se encuentran en el anterior sistema, y dárselos a los usuarios previamente definidos en el sistema, también da acceso a los administradores para que estos puedan gestionar los permisos correspondientes a los roles de usuario.

En el siguiente diagrama se ilustran los componentes de la arquitectura de hardware del sistema de gestión de documentos.

*Arquitectura de hardware*

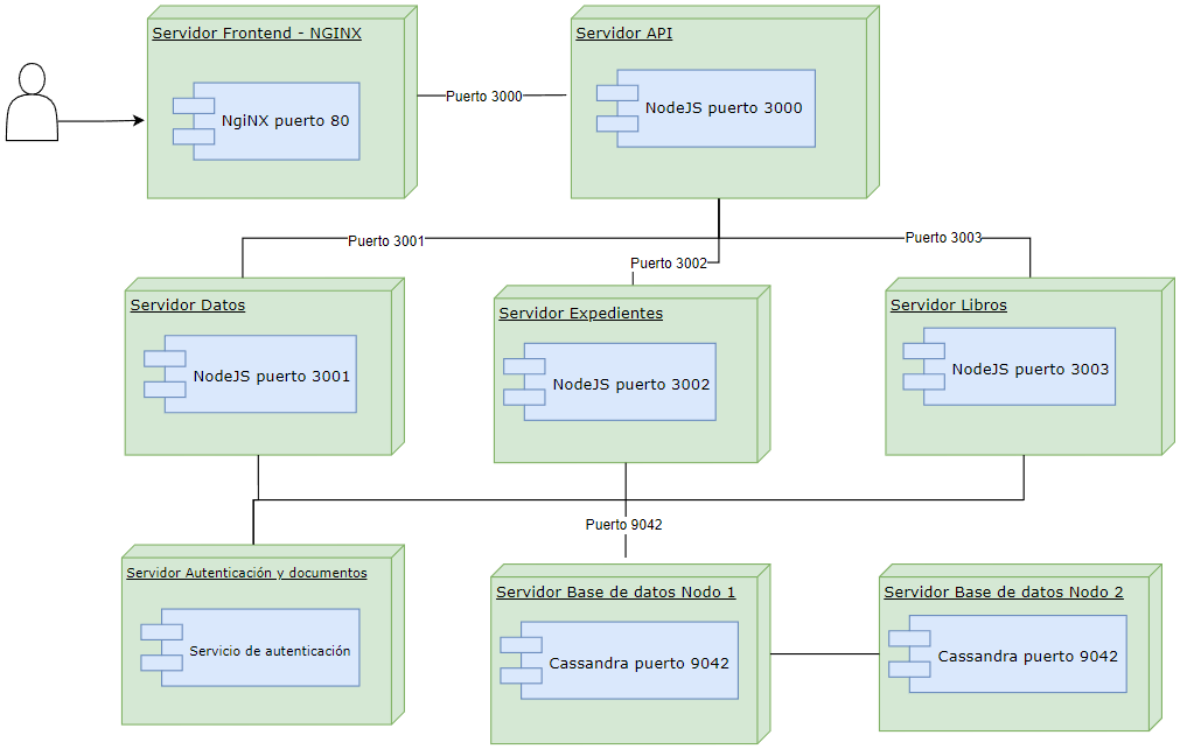



Nota. Flujograma que muestra la arquitectura de la solución.

El despliegue de la aplicación se ve en el siguiente diagrama, la aplicación se ejecuta bajo los servidores internos, por lo que los usuarios solo podrán acceder cuando el dispositivo se encuentre conectado a la misma red en la que se encuentra la aplicación.

Esta es una medida de seguridad general, de manera que, ningún usuario ajeno a las funcionalidades maniobradas en el sistema pueda tener algún tipo de interacción con él, la ruta por la cual pueden acceder los usuarios es: http://evisor.registromercantil.gob.gt/.

*Despliegue de la solución*



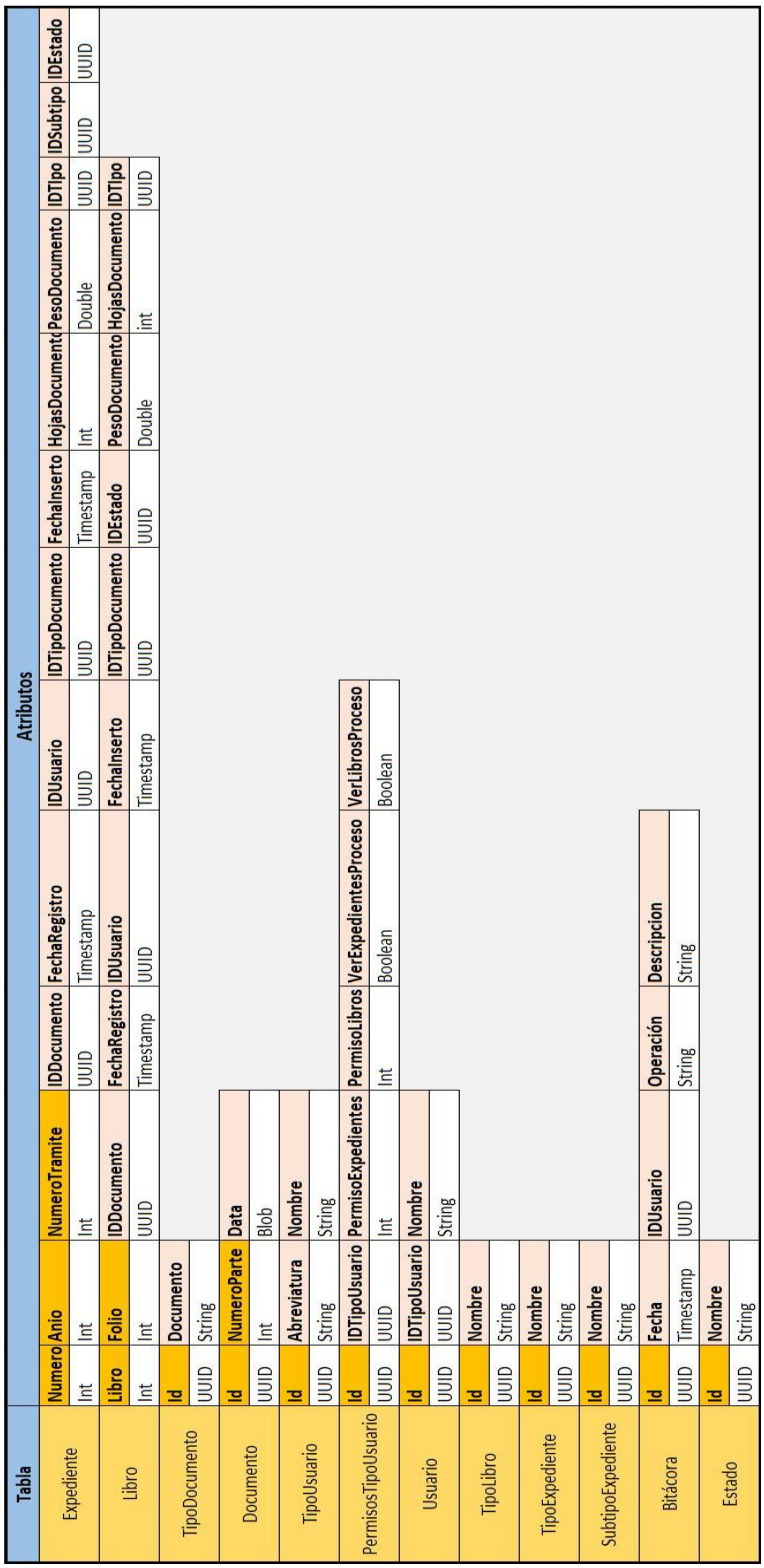

Nota. Diagrama de despliegue de la solución.

### Base de datos del sistema

Se implementó un manejador de base de datos, por sus características de escalabilidad, alto rendimiento, open source, manejo de altos volúmenes de información y alta capacidad de respuesta se utilizará Apache Cassandra.

Se definió un esquema de base de datos no relacional en donde se tiene las colecciones de datos que representan las tablas en donde se almacenará la información, las colecciones se definen en el siguiente diagrama.

*Esquema no relacional*



| Tabla | Atributos | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Expediente | Numero | Anio | NumeroTramite | IDDocumento | FechaRegistro | IDUsuario | IDTipoDocumento | FechaInserto | HojasDocumento | PesoDocumento | IDTipo | IDSubtipo | IDEstado | | |
| | Int | Int | Int | UUID | Timestamp | UUID | UUID | Timestamp | Int | Double | UUID | UUID | UUID | | |
| Libro | Libro | Folio | IDDocumento | FechaRegistro | IDUsuario | FechaInserto | IDTipoDocumento | IDEstado | PesoDocumento | HojasDocumento | IDTipo | | | | |
| | Int | Int | UUID | Timestamp | UUID | Timestamp | UUID | UUID | Double | int | UUID | | | | |
| TipoDocumento | Id | Documento | | | | | | | | | | | | | |
| | UUID | String | | | | | | | | | | | | | |
| Documento | Id | NumeroParte | Data | | | | | | | | | | | | |
| | UUID | Int | Blob | | | | | | | | | | | | |
| TipoUsuario | Id | Abreviatura | Nombre | | | | | | | | | | | | |
| | UUID | String | String | | | | | | | | | | | | |
| PermisosTipoUsuario | Id | IDTipoUsuario | PermisoExpedientes | PermisoLibros | VerExpedientesProceso | VerLibrosProceso | | | | | | | | | |
| | UUID | UUID | Int | Int | Boolean | Boolean | | | | | | | | | |
| Usuario | Id | IDTipoUsuario | Nombre | | | | | | | | | | | | |
| | UUID | UUID | String | | | | | | | | | | | | |
| TipoLibro | Id | Nombre | | | | | | | | | | | | | |
| | UUID | String | | | | | | | | | | | | | |
| TipoExpediente | Id | Nombre | | | | | | | | | | | | | |
| | UUID | String | | | | | | | | | | | | | |
| SubtipoExpediente | Id | Nombre | | | | | | | | | | | | | |
| | UUID | String | | | | | | | | | | | | | |
| Bitácora | Id | Fecha | IDUsuario | Operación | Descripcion | | | | | | | | | | |
| | UUID | Timestamp | UUID | String | String | | | | | | | | | | |
| Estado | Id | Nombre | | | | | | | | | | | | | |
| | UUID | String | | | | | | | | | | | | | |

Nota. Esquema no relacional que representa la estructura de la nueva base de datos en el sistema.

*Descripción de esquema no relacional*

| Entidad | Descripción | |
|---|---|---|
| Expediente | Almacena los datos de cada expediente registrado, cuenta con los datos del usuario que lo registro y el tipo de trámite. | Tiene como llaves primarias combinadas el número de expediente, el año y el número de trámite a tratar, referencia a las colecciones Documento, TipoDocumento, Usuario, TipoExpediente, SubtipoExpediente y Estado. |
| Libro | Almacena los datos de cada libro registrado, cuenta con los datos del usuario que lo registro. | Tiene como llaves primarias combinadas el número de libro, y el folio, referencia a las colecciones Documento, TipoDocumento, Usuario, TipoLibro y Estado. |
| TipoDocumento | Almacena los tipos de documentos existentes en el sistema | |
| Documento | Almacena los documentos digitales, tanto de expedientes como de libros, en la base de datos. | Tiene como llaves primarias el id del documento al que referencian y el número de parte de cada documento, referencia a la colección TipoDocumento. |
| TipoUsuario | Almacena los tipos de usuario existentes en el sistema. | |
| PermisosTipoUsuario | Almacena los permisos que posee cada tipo de usuario en el sistema. | Referencia a la colección de TipoUsuario para darle permisos a cada rol en el sistema. |
| Usuario | Almacena a cada usuario en el sistema. | Referencia a la colección de TipoUsuario para saber qué tipo de usuario es y así obtener los permisos de este. |
| TipoLibro | Almacena los tipos existentes para los libros en el sistema. | |
| TipoExpediente | Almacena los tipos existentes para los expedientes en el sistema. | |
| SubTipoExpediente | Almacena los subtipos existentes para los expedientes en el sistema. | |
| Bitacora | Almacena todas las acciones que se ejecutan en el sistema. | Referencia a la colección de Usuario para saber que usuarios realizaron que operaciones. |
| Estado | Almacena los estados posibles para los documentos en el sistema. | |

Nota. Esquema no relacional utilizado en la base de datos de Cassandra.

**Roles dentro del sistema**

En el sistema existen diferentes tipos de roles, cada usuario en el sistema tiene asignado un tipo de rol, con los cuales se identifican los permisos que este usuario posee, vemos a continuación, en la tabla 1, los roles existentes en el sistema:

*Roles dentro del sistema*

| Abreviatura | Nombre |
|---|---|
| I | Informes |
| CC | Call Center |
| M | Modificaciones |
| D | Despacho |
| A | Auxiliar |
| J | Jurídico |
| C | Certificaciones |
| T | Tecnología |
| PV | Ventanilla |
| S | Sociedades |
| E | Empresas |
| L | Libros |
| F | Archivo |

Nota. Detalle de los roles que poseen los usuarios dentro del sistema.

Finalmente, cada rol les da a los usuarios ciertos permisos para poder consultar documentos dentro del sistema, estos permisos son:

- Consultar documentos
- Imprimir documentos
- Visualizar expedientes en proceso
- Ver libros en proceso

**Herramientas de software**

Para la implementación y desarrollo se utilizaron diferentes tecnologías y frameworks, los cuales son los siguientes:

- Framework Angular, versión 16.2.9
- Entorno de ejecución NodeJS, versión 18.18.1
- Apache Cassandra, versión 5.0
- Docker, versión 24.0.6
- Nginx, versión 1.25.3
- Typescript, versión 5.1.3

Hardware para alojamiento de microservicio Frontend

- Sistema operativo Debian GNU/Linux 11
- CPU 2 núcleos
- Memoria RAM 8 GB
- Disco duro 20 GB

Hardware para alojamiento de microservicio API

- Sistema operativo Debian GNU/Linux 11
- CPU 2 núcleos
- Memoria RAM 8 GB
- Disco duro 20 GB

Hardware para alojamiento de microservicio Datos

- Sistema operativo Debian GNU/Linux 11
- CPU 1 núcleos
- Memoria RAM 8 GB

- Disco duro 20 GB

Hardware para alojamiento de microservicio Expedientes

- Sistema operativo Debian GNU/Linux 11
- CPU 2 núcleos
- Memoria RAM 8 GB
- Disco duro 20 GB

Hardware para alojamiento de microservicio Libros

- Sistema operativo Debian GNU/Linux 11
- CPU 2 núcleos
- Memoria RAM 8 GB
- Disco duro 20 GB

Hardware para alojamiento de Base de datos nodo 1

- Sistema operativo Debian GNU/Linux 11
- CPU 2 núcleos
- Memoria RAM 16 GB
- Disco duro 2 TB

Hardware para alojamiento de Base de datos nodo 2

- Sistema operativo Debian GNU/Linux 11
- CPU 2 núcleos
- Memoria RAM 16 GB
- Disco duro 2 TB

Las librerías administradas por NodeJS son:

- cors
- dotenv
- express
- jsonwebtoken
- swagger-jsdoc
- swagger-ui-express
- cassandra-driver
- node-fetch
- pdfkit
- pdf-merger-js
- uuid
- jest
- supertest

Las librerías administradas por Angular son:

- ng2-pdf-viewer
- primeicons
- primeng
- rxjs
- sweetaler2
- tslib
- zonejs

Las imágenes administradas por Docker son:

- cassandra:5.0.

## Tecnologías

Angular

Es un framework que gracias a todas las herramientas que proporciona se ha vuelto en una plataforma de desarrollo basada en componentes, estas virtudes ayudan al equipo completo a tener un control total de lo que se quiera llevar a cabo, también a mantener un orden bien estructurado para que todo sea transparente, de este modo es fácil adentrarse en un proyecto realizado en Angular.

Proporciona una colección de librerías que cubren la mayoría de las funcionalidades más comúnmente utilizadas para facilitar la integración con el proyecto, incluyendo enrutamiento, formularios, manejo de estados, comunicación cliente-servidor, entre otras, de igual manera proporciona un entorno bien integrado para que todo pueda ser desarrollado, construido, probado y actualizado de una manera sencilla.

Agregado a esto, existen millones de compañías, equipos de desarrollo o usuarios, que continuamente desarrollan componentes para compartirlos con la comunidad de desarrolladores, estos componentes permiten facilitar la implementación de ciertas características dentro del propio ecosistema donde se está trabajando, de esta manera se facilitan virtudes que si se realizaran desde el principio sería tedioso.

Angular hace uso de Typescript, el cual es un superset de JavaScript, este mismo añade características a JavaScript que lo hacen más entendible y escalable, proporcionando un tipado estático que inicialmente no existe en JavaScript, estas características permiten evitar errores sutiles de programación que a la larga quitan tiempo valioso para el desarrollo de los proyectos.

NodeJS

Es un entorno de ejecución de JavaScript open source y multiplataforma, tiene diversas herramientas que permiten adaptarse a casi cualquier proyecto en el que desee aplicarse su uso, sus librerías son creadas con el uso de paradigmas que evitan los bloqueos de ejecución, por lo que ejecuta un único proceso, pero permite crear eventos asíncronos.

NodeJS utiliza el motor V8 de JavaScript, el mismo que ejecuta Google Chrome, y el mismo con el que millones de aplicaciones han sido desarrolladas, por lo que un usuario que conoce este lenguaje para el desarrollo frontend puede hacer uso del mismo lenguaje del lado del servidor sin ningún problema al utilizar NodeJS.

Este entorno de desarrollo provee muchas características que facilitan la vida de los desarrolladores, desde herramientas para desarrollar, probar, ejecutar y actualizar su código, hasta un cliente que permite ejecutar todas estas acciones de manera automática y sencilla con un comando, también posee un manejador de paquetes que permite descargar y utilizar librerías que otros desarrolladores han puesto a disposición de nuestras aplicaciones.

Docker

Es una plataforma especializada para desarrollar, desplegar y ejecutar aplicaciones de manera aislada, permitiendo crear infraestructuras sencillas o complejas en entornos específicos, también nos provee herramientas para poder maniobrar entre estas aplicaciones de manera sencilla y sin tener que estar preocupándonos de que cada aplicación tenga las librerías, versiones y recursos compatibles para funcionar.

Una de las ventajas principales de Docker es que se puede reducir drásticamente el tiempo en el que se el desarrollador crea o actualiza una

funcionalidad y la traslada al entorno real, esto debe suceder mediante la realización de muchos pasos, mismos que pueden ser automatizados y con la ayuda de Docker validar que todo funcione para posteriormente darlo a producción.

Docker es fácilmente instalable en múltiples plataformas, y posee una amplia documentación con la cual podemos encontrar como instalarlo en el entorno que deseemos, Docker aísla cada aplicación que deseemos en un entorno o contenedor, este contenedor es fácilmente ejecutable y producible, por lo que millones de desarrolladores día a día crean sus propias aplicaciones, las introducen en un contenedor y las ponen a disposición de los interesados.

Nginx

Es un software open source para servicios web, servidores proxy inversos, cache, balanceadores de carga, streaming, y mucho más, entre sus principales características se destacan:

- Rendimiento: Cuando se trata de gestionar un gran número de conexiones simultaneas Nginx es la mejor opción, ya que permie gestionarlas con alto rendimiento.

- Diseñado para ser ligero: En comparación con otros, Nginx es bastante ligero, por lo que es seguro que su consumo de recursos será menor, lo que permite aprovechar estos recursos en el rendimiento de las aplicaciones.

- Proxy inverso y balanceador de carga: Nginx es conocido por ser un servidor de proxy inverso eficiente y por ofrecer funciones de balanceadores de carga.

Git

Es un sistema de control de versiones, este permite manejar el histórico de los cambios efectuados en el desarrollo y también a organizar el progreso que vamos realizando de la manera que deseemos, permite distribuir el trabajo de manera rápida y adecuada para posteriormente unificarlo sin causar incongruencias en el resultado final.

**Herramientas utilizadas durante el desarrollo**

Postman

Es una plataforma API que nos permite consumir servicios para poder interactuar con ellos sin necesidad de tener un cliente real, de esta manera se puede probar la funcionalidad de un servicio y entender cómo se desempeña, que necesitamos hacer para recibir las respuestas que deseamos.

Postman busca que todo lo que se haga sea transparente en los grupos de trabajo, por lo que crea igualmente grupos en donde se puede probar funcionalidades en conjunto, compartir las pruebas para identificar errores o mejores en los servicios a evaluar.

Visual Studio Code

Es un editor de código que busca facilitar al desarrollador las herramientas necesarias que un IDE posee, pero combinándolo con muchas otras características, esto ayuda a que el desarrollador únicamente deba preocuparse por la programación, es capaz de aplicar sus extensiones a todo lenguaje de programación conocido.

Si un equipo de trabajo está desarrollando un sistema con varios lenguajes de programación, todos pueden usar Visual Studio Code para programar ya que este editor se adaptará al lenguaje de programación que se esté usando, y aún fuera de la programación, también sus extensiones permiten brindarle al usuario una experiencia integrada con muchas otras tecnologías como git, Postman, Docker, etc.

Swagger

Es una herramienta que nos permite realizar documentación de nuestras APIS, ya que luego de que la funcionalidad esté completa, siempre existe la interrogante de cómo hacer uso de esta, por lo que cuando usamos Swagger podemos definir como un usuario puede hacer uso de nuestra API y hacerla entendible.

Es fácil de usar ya que ofrece una interfaz gráfica que nos permite visualizar el funcionamiento y nos explica detalladamente que cosas debemos hacer para ejecutar acciones y cuando obtengamos una respuesta de estas acciones también nos indica que posibles respuestas vamos a recibir, además, también es fácil de crear, ya que tiene una sintaxis minimalista pero completa, que da la libertad al desarrollador de crear sin necesidad de usar interfaces o códigos complejos.

Gitlab

Es una plataforma de alojamiento de código open source, es especializada en manejar grandes grupos de trabajo y proporciona herramientas que facilitan realizar el flujo devops que permitirá mantener una integración y despliegue continuos de nuestras aplicaciones.

**Despliegue**

El sistema almacena su código fuente en dos repositorios proporcionados por el personal del Registro Mercantil, uno para manejar el código fuente del frontend, y otro para manejar el código fuente del backend, estos están alojados dentro de repositorios locales que usan el motor de Gitlab, por lo que sus virtudes son aprovechadas para la integración y despliegue continuos.

En el caso del frontend, este posee en el repositorio el respectivo archivo ‘dockerfile’ con el cual se creará la imagen de Docker con el código fuente del mismo, en el caso del backend, al contar con microservicios, cada microservicio está separado por carpetas y cada una identificada con su nombre, dentro de cada microservicio existe el respectivo archivo ‘dockerfile’ con el cual se crea la imagen de cada microservicio, los microservicios que se manejan son los siguientes:

- API: Es el api que maneja todas las solicitudes que procesa el sistema, sin embargo, solo es el intermediario que se encarga de la validación de la información enviada y recibida, también se encarga de verificar la autenticidad de los usuarios.
- Datos: Es el microservicio encargado de manejar toda la información de usuarios, procesos y acciones técnicas dentro del sistema.
- Expedientes: Es el microservicio que se encarga de sobrellevar las tareas que tengan relación con expedientes.
- Libros: Es el microservicio que se encarga de sobrellevar las tareas que tengan relación con libros.

Cada uno de estos repositorios tiene un archivo ‘.gitlab-ci.yml’ con el cual se crea un pipeline que tiene las instrucciones para la creación y subida a un repositorio de las imágenes de Docker, tanto el frontend, como cada uno de los microservicios, estos procesos son ejecutados automáticamente después de haber detectado algún cambio en el repositorio respectivo.

Las imágenes son subidas a un repositorio privado, el cual solo se puede acceder por los usuarios que posean las credenciales de este, de esta manera se tiene restringido el acceso al código de las aplicaciones almacenadas en dichos contenedores, el repositorio privado tiene como nombre ‘evisorrm’.

Cada contenedor es puesto en marcha en una máquina específica para cada tarea, estas máquinas deben poder ejecutar contenedores de Docker, por lo que es necesario instalarlo dependiendo el sistema operativo que se ejecute.

Una vez instalado Docker, procedemos a ejecutar los comandos respectivos para comenzar la ejecución de cada aplicación.

Para el frontend:

- docker pull evisorrm/frontend
- docker run -d --name frontend -p 80:80 evisorrm/frontend

Detalles de los comandos:

- La imagen lleva por nombre ‘evisorrm/frontend’.
- -p: Expone el puerto por defecto, en este caso el frontend se trabaja en un estándar el cual utiliza el puerto 80 para trabajar.

Para el microservicio API

- docker pull evisorrm/api
- docker run -d -p 3000:3000 --name api -e MS_DATOS=http://131.107.5.84:3001 -e MS_EXPEDIENTES=https://m19r4pch-3002.use2.devtunnels.ms -e DATABASE_ADDR='131.107.5.92' -e KEYSPACE='evisor' evisorrm/api

Detalles de los comandos:

- La imagen lleva por nombre ‘evisorrm/api’.

- -p: Este servicio trabaja en el puerto 3000, por lo que se expone dentro del contenedor para ser usado.
- -e MS_DATOS: Variable de entorno la cual debe ser la ruta que aloja el microservicio de datos.
- -e MS_EXPEDIENTES: Variable de entorno la cual debe ser la ruta que aloja el microservicio de expedientes.
- -e MS_LIBROS: Variable de entorno la cual debe ser la ruta que aloja el microservicio de libros.
- -e DATABASE_ADDR: Variable de entorno la cual debe ser la dirección en donde está alojada la base de datos de Cassandra con la que va a trabajar.
- -e KEYSPACE: Variable de entorno la cual debe ser el keyspace que maneja cassandra y tiene asignado para el sistema.
- --name:: Nombre que se le asigna al contenedor.

Para el microservicio Datos

- docker pull evisorrm/ms-datos
- docker run -d -p 3001:3001 -e DATABASE_ADDR='131.107.5.92' -e KEYSPACE='evisor' --name ms-datos evisorrm/ms-datos

Detalles de los comandos:

- La imagen lleva por nombre ‘evisorrm/ms-datos.
- -p: Este servicio trabaja en el puerto 3001, por lo que se expone dentro del contenedor para ser usado.
- -e DATABASE_ADDR: Variable de entorno la cual debe ser la dirección en donde está alojada la base de datos de Cassandra con la que va a trabajar.
- -e KEYSPACE: Variable de entorno la cual debe ser el keyspace que maneja cassandra y tiene asignado para el sistema.
- --name:: Nombre que se le asigna al contenedor.

Para el microservicio Expedientes

- docker pull evisorrm/ms-expedientes
- docker run -d -p 3002:3002 -e DATABASE_ADDR='131.107.5.92' -e KEYSPACE='evisor' --name ms-expedientes evisorrm/ms-expedientes

Detalles de los comandos:

- La imagen lleva por nombre ‘evisorrm/ms-expedientes.
- -p: Este servicio trabaja en el puerto 3002, por lo que se expone dentro del contenedor para ser usado.
- -e DATABASE_ADDR: Variable de entorno la cual debe ser la dirección en donde está alojada la base de datos de Cassandra con la que va a trabajar.
- -e KEYSPACE: Variable de entorno la cual debe ser el keyspace que maneja cassandra y tiene asignado para el sistema.
- --name:: Nombre que se le asigna al contenedor.

Para el microservicio Libros

- docker pull evisorrm/ms-libros
- docker run -d -p 3003:3003 -e DATABASE_ADDR='131.107.5.92' -e KEYSPACE='evisor' --name ms-libros evisorrm/ms-libros

Detalles de los comandos:

- La imagen lleva por nombre ‘evisorrm/ms-libros.
- -p: Este servicio trabaja en el puerto 3003, por lo que se expone dentro del contenedor para ser usado.
- -e DATABASE_ADDR: Variable de entorno la cual debe ser la dirección en donde está alojada la base de datos de Cassandra con la que va a trabajar.
- -e KEYSPACE: Variable de entorno la cual debe ser el keyspace que maneja cassandra y tiene asignado para el sistema.
- --name: Nombre que se le asigna al contenedor.

**Despliegue de base de datos**

La base de datos de cassandra es desplegada utilizando Docker, tiene ciertas configuraciones específicas para que la información sea persistente y también el apartado para agregar la seguridad que se necesite. En el repositorio de base de datos se encuentra la carpeta ‘Database’, en esta se encuentra el archivo ‘docker-compose.yml’ con el cual se realiza el despliegue automático de la base de datos.

Es necesario colocar este archivo en la máquina que se encargará de manejar la base de datos, asegurarse que tenga Docker instalado, crear la carpeta que contendrá los volúmenes de información y ejecutar el compose, esto puede realizarse ejecutando los siguientes comandos:

- mkdir cassandra-data
- docker compose up -d

El comando se encargará de exponer los puertos, mapear los volúmenes y cualquier otra configuración que se ingrese. También, en la misma carpeta, encontrarán el archivo ‘create-database.sql’, en este se encuentra el script para la creación de la base de datos, igualmente se encontrará la carpeta ‘inserts’, en ella se encuentran más scripts que sirven para poblar la base de datos con información necesaria para el funcionamiento del sistema.

Para poder ejecutar estos scripts es necesario acceder al contenedor de la base de datos, esto se realiza con el siguiente comando:

- docker exec -it cassandra cqlsh

Con este comando accedemos a la interfaz de línea de comandos que nos ofrece Cassandra, aquí podremos ejecutar línea por línea los scripts o bien cargarlos directamente.

**Documentación de la API**

Endpoints:

1. /auth/login – POST

Descripción: Endpoint utilizado para autenticarse dentro del sistema.

Esquema del body que recibe:

```
{
    “username”: string,
    “password”: string
}
```

Descripción del esquema:

- username: Nombre de usuario que intenta loguearse.
- password: Contraseña del usuario que intenta loguearse.

Respuestas:

Status 200: Petición exitosa

Tipo de respuesta: application/json

Esquema del body que devuelve:

```
{
    "message": string,
    "token": string
}
```

Descripción del esquema:

- message: Mensaje que brinda más información acerca del estado de la petición.
- token: Json Web Token que da acceso a las operaciones dentro del sistema, es necesario enviarlo para cada petición que se realice.

Status 400: Ha ocurrido algún error en la autenticación

2. /expedientes/getExpediente – POST

Descripción: Obtener un expediente en específico.

Esquema del body que recibe:

```
{
    "numero_expediente": number,
    "anio_expediente": number,
    "watermark": boolean
}
```

Descripción del esquema:

- numero_expediente: Número del expediente que se busca.
- anio_expediente: Año del expediente que se busca.
- watermark: Opcional, si se desea que el documento pdf del expediente tenga marca de agua “COPIA”, enviar como true, de lo contrario enviar false o no enviar este parámetro.

Respuestas:

Status 200: Petición exitosa

Tipo de respuesta: application/pdf

Esquema del body que devuelve:

string

Descripción del esquema:

- String que devuelve el binario del pdf obtenido.

Status 400: Ocurrió algún error

3. /filter/getUsuarios – GET

Descripción: Obtener a los usuarios en el sistema.

Respuestas:

Status 200: Petición exitosa

Tipo de respuesta: application/json

Esquema del body que devuelve:

```
[
```

```
        {
            "id": string,
            "nombre": string
        },…
    ]
```

Descripción del esquema:

- id: Id del usuario que lo identifica dentro del sistema.
- nombre: Nombre del usuario.

Status 400: Ocurrió algún error

Status 401: No autorizado

4. /filter/filterLogs – POST

Descripción: Filtrar la bitácora del sistema.

Esquema del body que recibe:

```
{
    "begin": string,
    "end": string,
    "user": string
    "limit": number
}
```

Descripción del esquema:

- begin: Opcional, fecha límite inicial con la que se realizará la búsqueda, debe ir en formato YYYY-MM-DD.
- end: Opcional, fecha límite final con la que se realizará la búsqueda, debe ir en formato YYYY-MM-DD.
- user: Opcional, id del usuario que se va a buscar.
- limit: Opcional, limite de resultados que se desea obtener.

Respuestas:

Status 200: Petición exitosa

Tipo de respuesta: application/json

Esquema del body que devuelve:

```
[
```

{

"id": string,

"descripcion": string,

"fecha": string,

"idUsuario": string,

"operacion": string

},…

]

Descripción del esquema:

- id: Id del registro.
- descripcion: Descripción para dar más detalle de la acción realizada.
- fecha: Fecha en la que se realizó la acción, en formato YYYY-MM-DD.
- idUsuario: Id del usuario que realizó la acción, de ser un login el id es null.
- operación: Tipo de operación que se realizó.

Status 400: Ocurrió algún error

Status 401: No autorizado

5. /permisos/getPermisos – GET

Descripción: Obtener los permisos actuales en el sistema.

Respuestas:

Status 200: Petición exitosa

Tipo de respuesta: application/json

Esquema del body que devuelve:

[

{

"id": string,

"idTipoUsuario": string"

"permisoExpedientes": number,

```
            "permisoLibros": number,
            "verExpedientesProceso": boolean,
            "verLibrosProceso": boolean,
            "abreviatura": string,
            "nombre": string
        },…
    ]
```

Descripción del esquema:

- id: Id del permiso que lo identifica en el sistema.
- idTipoUsuario: Id del tipo de usuario que identifica su rol en el sistema.
- permisoExpedientes: Permiso que posee este rol para trabajar con expedientes.
- permisoLibros: Permiso que posee este rol para trabajar con libros.
- verExpedientesProceso: Permiso que posee este rol para ver expedientes que se encuentren en proceso.
- verLibrosProceso: Permiso que posee este rol para ver libros que se encuentren en proceso.
- abreviatura: Abreviatura que posee el rol.
- nombre: Nombre del rol en el sistema.

Status 400: Ocurrió algún error

Status 401: No autorizado

6. /permisos/updatePermiso – PUT

Descripción: Actualizar un permiso en el sistema.

Esquema del body que recibe:

```
{
    "id": string,
    "permisoExpedientes": number,
    "permisoLibros": number,
    "verExpedientesProceso": boolean,
```

"verLibrosProceso": boolean

}

Descripción del esquema:

- id: Id del permiso a actualizar.
- permisoExpedientes: Permiso que se dará a este rol para trabajar con expedientes.
- permisoLibros: Permiso que se dará a este rol para trabajar con libros.
- verExpedientesProceso: Permiso que se le dará a este rol para trabajar con expedientes en proceso.
- verLibrosProceso: Permiso que se le dará a este rol para trabajar con libros en proceso.

Nota: En el caso de permisoExpedientes y permisoLibros, se trabajan con valores numéricos, estos aceptan los siguientes valores:

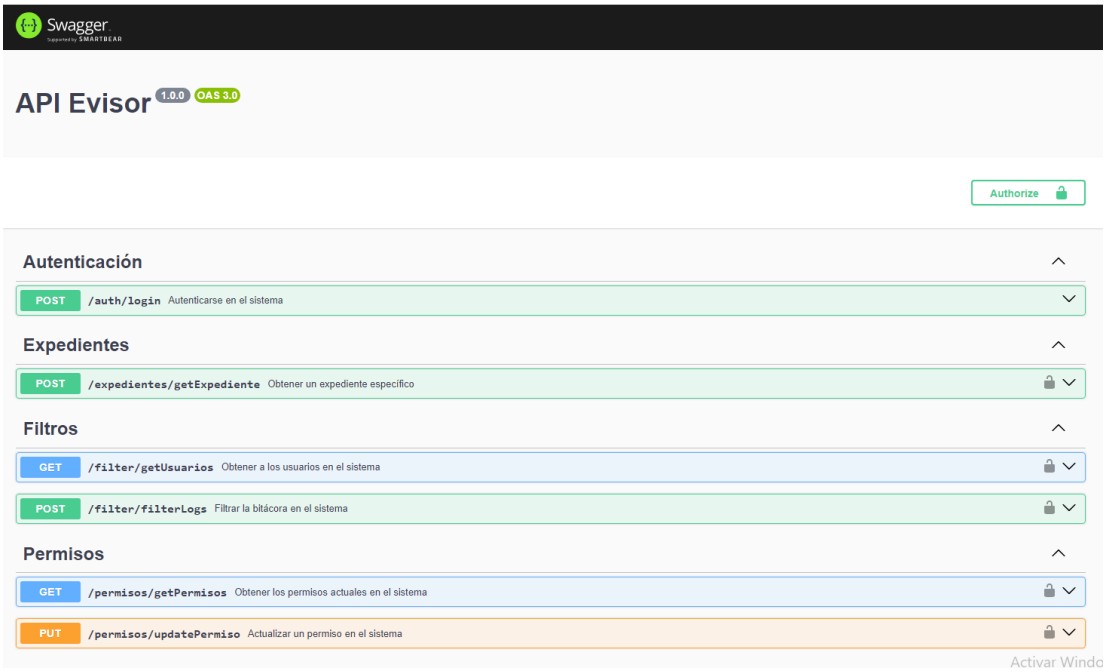
0: No tiene permisos.
1: Consulta.
2: Impresión.
3: Consulta e impresión.

Respuestas:
Status 200: Petición exitosa
Status 400: Ocurrió algún error
Status 401: No autorizado

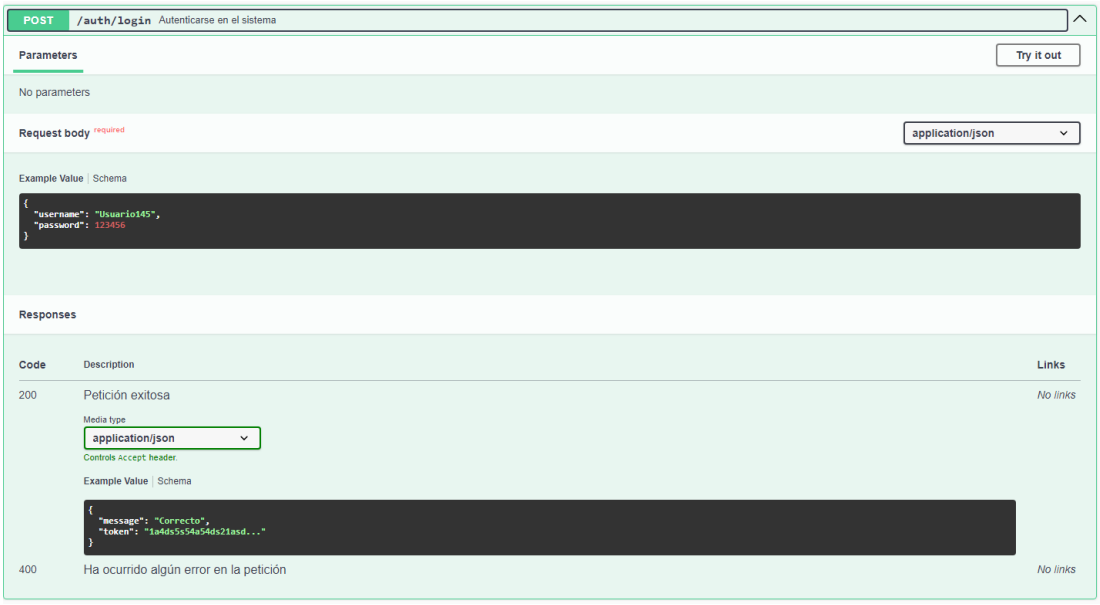
**Swagger**

Swagger se ha utilizado como una alternativa a documentar los endpoints que trabaja el sistema, Swagger nos permite obtener una documentación más visual e iterativa del sistema, esta documentación puede accederse mediante la

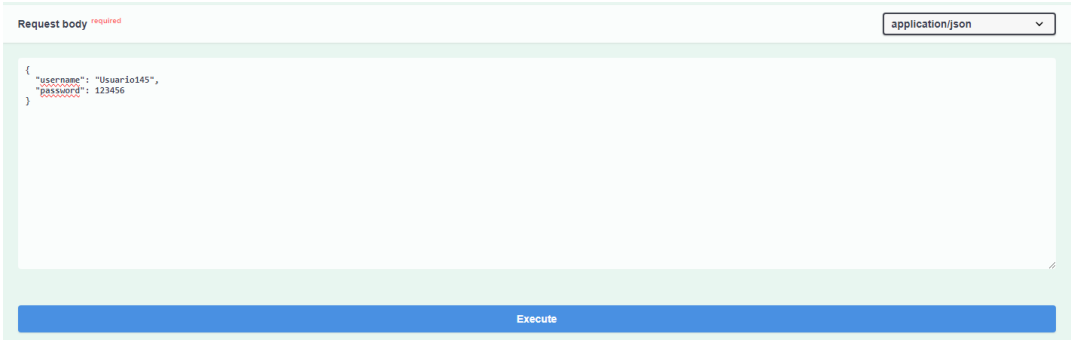
ruta /documentation que aloja el microservicio API, al ingresar a la ruta podemos observar una interfaz gráfica como la siguiente:




En la cual podemos observar cada uno de los endpoints que maneja el sistema, cada uno separado por sus respectivas funcionalidades. Si desglosamos uno de los endpoints tendremos un detalle más profundo de cómo trabaja el endpoint:




Podemos observar la estructura del body que recibe este endpoint, también la respuesta y códigos de estado que devuelve. Si deseamos, podemos realizar una petición desde Swagger, presionando el botón Try it out, el cual nos desplegará una interfaz para poder llenar los respectivos campos y ejecutar una solicitud:




Y finalmente podremos visualizar el resultado de la ejecución:


## Pruebas

Cada uno de los microservicios cuenta con un área de pruebas que deberían ser ejecutadas previo a la generación de las imágenes de Docker del respectivo microservicio, las pruebas son puntuales para determinar el funcionamiento básico de cada microservicio y que este esté trabajando correctamente.

Las pruebas por cada microservicio pueden ser ejecutados con el siguiente comando:

- npm run test

Este comando debe ser ejecutado en la carpeta correspondiente para cada microservicio, posteriormente veremos una serie de pasos que se ejecutarán para determinar que el microservicio esté trabajando correctamente, al finalizar las pruebas veremos un resultado que demostrará si todo funciona correctamente o si deberíamos revisar algo en específico.