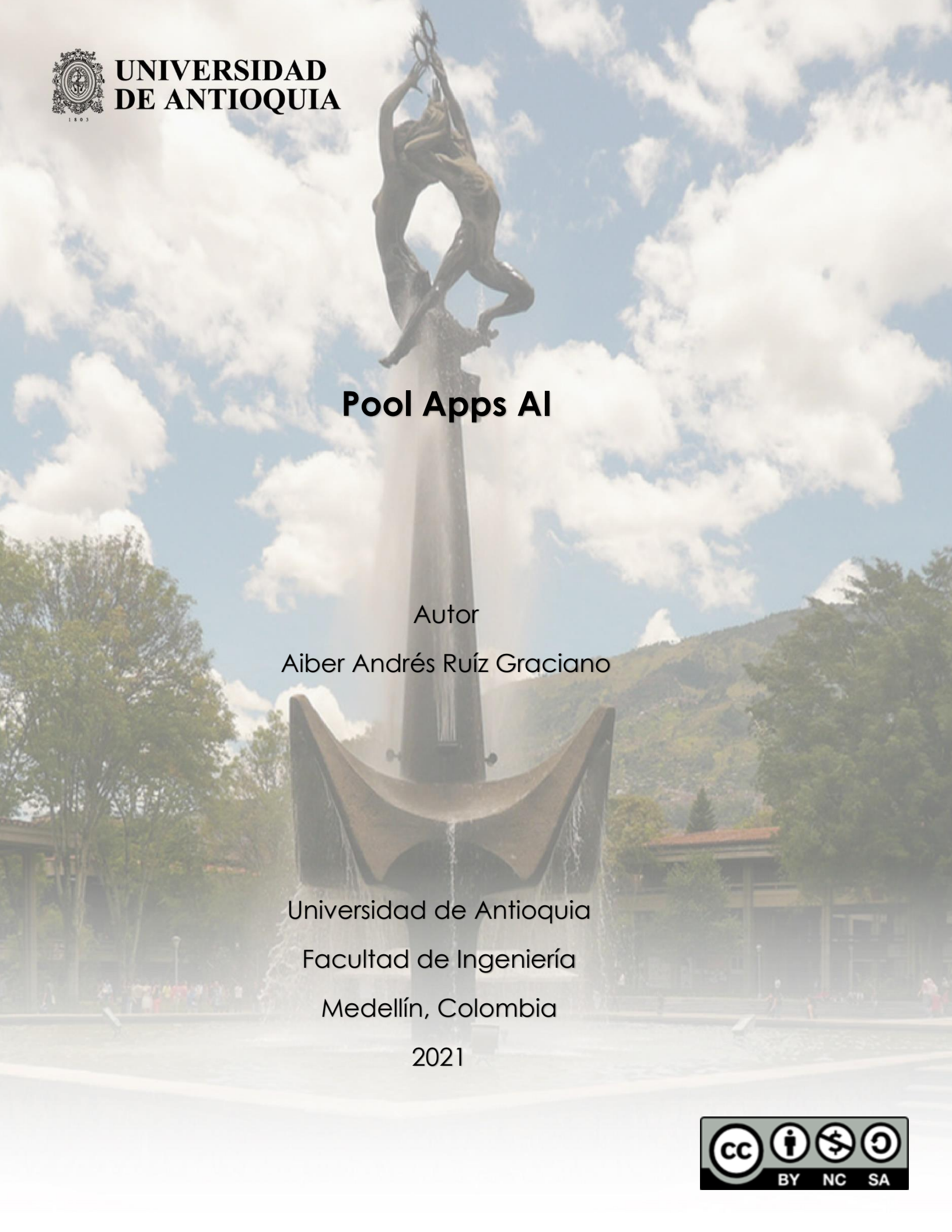


UNIVERSIDAD DE ANTIOQUIA

Pool Apps AI

Autor

Aiber Andrés Ruíz Graciano

Universidad de Antioquia

Facultad de Ingeniería

Medellín, Colombia

2021

Pool Apps AI

Aiber Andrés Ruiz Graciano

Tesis o trabajo de investigación presentada(o) como requisito parcial para optar al título de:

Ingeniero de Sistemas

Asesor:

Carlos Mauricio Duque Restrepo, Ingeniero de Sistemas

Universidad de Antioquia

Facultad de Ingeniería

Medellín, Colombia

2021

Resumen

El área de Artificial Intelligence (AI) & Data Science de Konecta, desea migrar su plataforma web a nuevas tecnologías de desarrollo para satisfacer la necesidad de exponer sus modelos y aplicaciones de forma modular y escalable, contribuyendo a mejorar la satisfacción de sus clientes. Se ha generado un plan de migración que contiene la construcción de una API de login para autorizar los modelos presentes en el área, por lo tanto, el presente proyecto comprende la estructuración de una API REST para realizar una *Autenticación* y *Autorización* de los integrantes del área, así como la inclusión de buenas prácticas de desarrollo, al involucrar el proceso de construcción de pruebas unitarias y la implementación de la cultura DevOps en los servidores de Amazon Web Services. El método de Autenticación y Autorización desarrollado en el proyecto, permitió dar cumplimiento al objetivo planteado, incrementando el interés en diseñar una nueva célula interna que se encargará de implementar la cultura DevOps en los proyectos existentes y futuros.

Palabras clave: Autenticación, Autorización, Amazon Web Services, DevOps, Desarrollo Web.

Abstract

Konecta's Artificial Intelligence (AI) & Data Science area wishes to migrate its web platform to new development technologies to satisfy its need to expose its models and applications in a modular and scalable manner, contributing to improving its clients' satisfaction. A migration plan that contains the construction of a login API to authorize the models present in the area has been generated, therefore, the present project comprises the structuration of a REST API to perform Authentication and Authorization of the area members, as well as the inclusion of good development practices by involving the unit test construction process and the implementation of the DevOps culture by using Amazon Web Services servers. The Authentication and Authorization method developed in the project allowed to comply with proposed objective, increasing the interest in designing a new internal cell that will be in charge of implementing the DevOps culture in the existing and future projects.

Keywords: Authentication, Authorization, Amazon Web Services, DevOps, Web Development.

Contenido

Introducción 4
Objetivos 4
General 4
Específicos 4
Marco Teórico 5
Metodología 8
Resultados y análisis 9
Conclusiones 22
Referencias Bibliográficas 23
Anexos 25
Anexo 1. Cuerpo de petición al endpoint */api/v1/config/base* 25
Anexo 2. Respuesta del endpoint /api/v1/config/base 28

Figuras

Figura 1. Arquitectura general Pool App AI 10
Figura 2. Arquitectura por Capas 12
Figura 3. Código fuente almacenado en AWS CodeCommit 13
Figura 4. Flujo entre las ramas de Git [26] 13
Figura 5. Cuerpo de la petición para la autenticación 14
Figura 6. Resultado exitoso de la autenticación 14
Figura 7. Resultado erróneo de la autenticación 14
Figura 8. Petición al endpoint /api/v1/modules como administrador 15
Figura 9. Petición al endpoint /api/v1/modules como un usuario regular 16
Figura 10. Retorno del endpoint /api/v1/modules para un administrador 16
Figura 11. Página de documentación del componente Country 17
Figura 12. Ejecución del método GET al endpoint /api/v1/countries desde la documentación. 18
Figura 13. Posibles respuestas documentadas del endpoint /api/v1/countries 18
Figura 14. Prueba unitaria para el servicio del componente Area 19
Figura 15. Prueba unitaria para el controlador del componente Area 19
Figura 16. Pipeline de Pool App AI en AWS CodePipeline 22

Tablas

Tabla 1. Código del archivo buildspec.yml 20

Introducción

El área de AI & Data Science de Konecta funciona actualmente con una plataforma web donde centraliza el acceso a los diferentes aplicativos y modelos cognitivos implementados (que son consumidos por los analistas y las distintas áreas de operación de la empresa). En el último año, han identificado una necesidad de mejora de la plataforma, dado que, en las revisiones de código, se encuentra que el aplicativo no cuenta con buenas prácticas de desarrollo (conocimiento interno de la compañía).

Con el objetivo de poseer una plataforma que se adapte a la evolución del área (poseer un aplicativo que permita exponer aplicaciones y modelos de forma modular y escalable), se ha propuesto implementar un aplicativo web con las tecnologías de punta [1] basado en una arquitectura de microservicios, que permita exponer sus servicios de forma independiente. Para cumplir este objetivo, se diseñó un plan de migración donde se encuentra el desarrollo de una API que permita Autenticar y Autorizar a los integrantes en la visibilidad de los modelos expuestos.

El informe comprende las diferentes etapas de construcción del API REST. En primer lugar, se describe en forma general cómo fue utilizada la metodología por iteraciones en el desarrollo del proyecto; en segundo lugar, se especifica la estructuración arquitectural del aplicativo; en tercer lugar, se describe la implementación del proceso de autenticación contra el LDAP de Konecta; en cuarto lugar, se indica cómo se construyó un sistema de roles y permisos para dar autorización a los integrantes del área; y por último, se enuncia el uso de buenas prácticas de desarrollo, al indicar cómo fue documentada el API, la implementación de pruebas unitarias y el desarrollo de un pipeline (DevOps) que permite un despliegue continuo en los servidores de Amazon Web Services.

Objetivos

General

Desarrollar una API REST que permita al área de Artificial Intelligence & Data Science conocer los permisos asociados a un usuario autenticado.

Específicos

- Realizar un análisis de requerimientos de la aplicación a desarrollar
- Establecer una conexión con el LDAP de Konecta
- Diseñar una arquitectura parcial serverless en AWS para dar soporte al API

- Realizar un plan de pruebas unitarias para los métodos CRUD del API
- Diseñar rutas jerárquicas para exponer servicios
- Construir un pipeline que permita una integración continua (pruebas automatizadas)
- Documentar los servicios expuestos con la herramienta Swagger
- Desplegar el API en AWS

Marco Teórico

Un aplicativo web, es una aplicación desarrollada en tecnologías soportadas por el navegador (CSS, HTML, JavaScript) a la cual se accede por medio de internet [2]. Su desarrollo puede ser desafiante, pero con una elección correcta de metodologías de desarrollo, herramientas y personal, es posible desarrollar un aplicativo de alto desempeño [3].

Las metodologías se refieren a un marco de trabajo usado para estructurar y controlar el proceso de desarrollo de un software. En particular, la metodología de desarrollo Incremental e Iterativa permite planificar el aplicativo en diversos bloques temporales, llamados iteraciones (mini proyectos que repiten los mismos pasos de construcción para proporcionar nuevas funcionalidades o conocimiento al producto final) [4].

Las herramientas, por otro lado, son formas estandarizadas de resolver problemas en el proceso de desarrollo de software. Pueden ser divididas en dos aspectos, las que ayudan a la elicitación de requisitos (lo que el cliente desea) y las que permiten un diseño de aplicativos modulares (tecnologías de desarrollo).

Una herramienta de elicitación de requisitos usada en Konecta, son los diagramas UML [5] (casos de usos, diagrama de actores, diagramas de secuencia, entre otros), dado que permiten responder con facilidad qué desea desarrollar el usuario final.

En el otro campo de las herramientas, se encuentran los estilos arquitectónicos, los patrones arquitectónicos y los patrones de diseño. Un estilo arquitectónico, es una colección nombrada de decisiones de arquitectura que permiten limitar las decisiones de diseño, obteniendo beneficios en el sistema resultante, como por ejemplo el estilo *por capas*, que permite dividir con facilidad la lógica de negocio de la presentación del aplicativo. Los patrones arquitectónicos y patrones de diseño son respectivamente, soluciones recurrentes a problemas relacionados con el estilo arquitectónico y una representación de la solución a problemas recurrentes encontrados durante el desarrollo de software [6].

El personal puede ser dividido en dos grandes y bien definidos grupos, *Backend y Frontend*. El grupo *Backend* es el encargado de administrar y disponer los datos almacenados del aplicativo, así como de asegurarse que las conexiones entre componentes estén correctas; generalmente es la parte del desarrollo web que el usuario final no puede ver e interactuar directamente. Por su parte, el grupo *Frontend* se encarga del desarrollo de todos los componentes con los que el usuario final interactúa directamente (colores, estilos, diseño, experiencias, animaciones, entre otros) [7].

Aunque ambas áreas son diferentes desde sus funciones, para que el desarrollo de un aplicativo web sea exitoso, se requiere de una comunicación efectiva entre ellas, de modo que parezcan una sola unidad [7], por lo que se hace necesario, definir los métodos de envío de información, seguridad y frameworks de desarrollo. Típicamente se implementa un Application Programming Interface (API), ya que permite ocultar complejidad a los desarrolladores, extender las aplicaciones, organizar el código y reusar componentes [8].

Si no se requiere guardar la sesión de las peticiones, la API puede ser desarrollada en el paradigma REST. REST (Representational State Transfer) es un conjunto de principios, propiedades y restricciones orientados hacia los recursos; su uso fue potenciado desde la publicación del trabajo de doctorado de Roy Thomas Fielding [9]. Para la comunicación, REST usa el protocolo HTTP (Hypertext Transfer Protocol), utilizado para hipermedia distribuida colaborativa en sistemas de información [10], usando la definición de sus cuatro métodos, GET, POST, PUT y DELETE. GET es un método seguro e idempotente (siempre retorna la misma respuesta bajo las mismas condiciones) utilizado para recuperar información [10]; POST es un método utilizado para realizar la creación de un recurso [10]; PUT es un método idempotente usado para actualizar o crear un recurso en el servidor [10]; Y DELETE es un método idempotente usado para identificar un recurso y eliminarlo [10]. Cada petición HTTP contiene al menos dos campos, un *body* opcional, que es un objeto informático utilizado para contener la información de la petición o la respuesta y unos *headers* que son vistos como tuplas clave valor, separados por dos puntos [10]. Adicionalmente, para brindar seguridad en las comunicaciones, se utiliza un protocolo TLS (Transport Layer Security), que define los mecanismos para asegurar una transmisión de datos sobre internet [11], entre otros más.

Definidos los recursos que se expondrán, es ideal crear un control de acceso que permita identificar cuáles usuarios pueden interactuar con el API. Para construirlo, se genera un proceso de Autenticación, que ayuda a responder la pregunta de quién es el usuario, usando una o varias de las tres técnicas,

lo que se sabe (usuario y contraseña), lo que tiene (clave dinámica) y lo que se es (huella dactilar). En el caso de Konecta, el proceso de autenticación se logra con un Directorio Activo (framework jerárquico de objetos) [12], accedido por el protocolo LDAP (Lightweight Directory Access Protocol) [13]. Por consiguiente, se genera un proceso de Autorización, que ayuda a responder la pregunta de qué pueden hacer los usuarios sobre el API, cuya implementación depende de cada equipo de desarrollo. Generalmente se definen roles y permisos, que son un conjunto de permisos y las acciones que puede realizar un usuario con el sistema respectivamente.

El desarrollo de los componentes anteriores se facilita mediante el uso de frameworks y servicios en la nube. Un framework, es la suma de patrones y componentes reutilizables que forman un esqueleto que puede ser customizado por el desarrollador [14]. En el caso de Javascript, se cuenta con la herramienta Node Js que combinado con el framework express (una herramienta pequeña y robusta para hacer grandes proyectos) [15] se logra diseñar de forma rápida e intuitiva un API REST. Adicionalmente, como parte de buenas prácticas, las APIs son documentadas con herramientas como Swagger, dado que permite definir un estándar para que máquinas y humanos entiendan los servicios expuestos a internet, bajo la especificación OpenAPI 3.0 [16].

Los servicios en la nube son un modelo que permite el acceso a una red bajo demanda de un conjunto de aplicaciones e infraestructura configurable. Generalmente están divididas en tres categorías, IaaS (Infrastructure as a Service), donde los usuarios tienen acceso directo al almacenamiento, red y recursos del proveedor (Amazon Web Service); PaaS (Platform as a Service), donde el usuario puede desarrollar y ofrecer aplicaciones en una infraestructura controlada por el proveedor (Heroku); y SaaS (Software as a Service), donde el usuario hace uso del aplicativo ofrecido (correo electrónico) [17]. Este tipo de servicios son muy utilizados porque permiten una estabilidad y accesibilidad de las plataformas desplegadas, así como la abstracción de los distintos protocolos de comunicación.

Finalmente, como buena práctica de desarrollo, se aplica el término DevOps, que alude al creciente desarrollo de una cultura que respalda una conexión de trabajo entre las áreas de TI (Desarrolladores y Operaciones), generando un flujo rápido de trabajo basado en la confianza, la estabilidad y la flexibilidad [18]. Su ciclo de vida puede ser visto en tres etapas: Integración continua, encargada de introducir el código fuente a un repositorio central donde se puedan automatizar pruebas para hallar fallos y errores; Despliegue continuo, puede ser vista como un paso adicional al anterior, donde se busca entregar nuevas funcionalidades al usuario final, minimizando su interacción para el despliegue del producto; y Entrega

Continua, donde la interacción humana es mínima para la entrega de nuevas funcionalidades del producto [19].

En Amazon Web Services es posible alcanzar un despliegue continuo utilizando los servicios de AWS CodeCommit, un servicio administrado de control de código fuente que aloja repositorios basados en git[1] [20]; AWS Codebuild, un servicio administrado para realizar integraciones continuas al compilar código fuente, ejecutar pruebas, y generar paquetes de software listos para su implementación [21]; AWS Elastic Container Registry, un servicio administrado de registro de contenedores[2] que facilita a los desarrolladores el almacenamiento, la administración, el uso compartido y la implementación de artefactos e imágenes[3] de contenedores [22]; AWS Elastic Container Service, un servicio administrado de orquestación de contenedores que permite dividir los proyectos en diferentes *Clusters* con los recursos necesarios para entrar en operación [23]; AWS Fargate un motor informático sin servidor que permite aprovisionar y administrar servidores [24] y AWS CodePipeline, un servicio de Entrega Continua completamente administrado por Amazon, que permite automatizar las fases de compilación, prueba e implementación del proceso del lanzamiento de nuevas funcionalidades en la modificación de código [25].

Metodología

El proyecto fue realizado bajo la metodología Iterativa e incremental, lo cual permitió dividir el producto en diferentes iteraciones (mini proyectos que repiten los mismos pasos de construcción para proporcionar nuevas funcionalidades o conocimiento al producto final) con entregas tempranas de Software. Cada iteración, siguió los lineamientos establecidos de desarrollo en Konecta y dejó un artefacto que permitía avanzar en la culminación exitosa del proyecto. Las iteraciones pueden ser agrupadas en las siguientes etapas:

- **Análisis y elicitación de requisitos:** Antes de abordar la codificación del proyecto, se realizó un análisis del objetivo propuesto, alcance del mismo y las tecnologías de desarrollo, logrando establecer una arquitectura general de los microservicios.

---

[1] Sistema distribuido, libre y gratuito de control de versiones https://git-scm.com/

[2] Unidad estándar de software que empaqueta el código fuente con sus dependencias https://www.docker.com/resources/what-container

[3] Una imagen es un ejecutable ligero e independiente que incluye los pasos necesarios para correr un aplicativo https://www.docker.com/resources/what-container

- **Arquitectura y estructuración de código:** Una vez definidas las tecnologías (Node Js, express, Amazon, MySQL) y plataformas de desarrollo (visual studio code, MySQL workbench), se planteó una arquitectura general para estructurar el proyecto asignado.
- **Unificación de proyecto en un repositorio central:** Para tener un control de versión de código, se creó un repositorio basado en git donde se protegieron las ramas para dar mayor seguridad.
- **Método de autenticación:** Por políticas internas de la empresa, no se puede almacenar contraseñas de los usuarios activos de la empresa, por lo tanto, se realiza una estructuración de conexión contra el LDAP de Konecta.
- **Método de autorización:** Se definió un esquema relacional, bajo la jerarquía de roles, que permitió brindar permisos a los modelos según el usuario autenticado. Este mecanismo se logra mediante la generación de un Json Web Token (JWT) que contiene la información del usuario autenticado.
- **Automatización de creación de registros:** Para agilizar el tiempo de uso del API, se decidió diseñar un Endpoint que ayudará a los administradores a crear los registros iniciales de configuración (países, módulos, aplicaciones, entre otros).
- **Documentación del API:** Cada aplicativo debe contar con documentación que indique tanto a usuarios como a desarrolladores, cómo funciona el producto desarrollado. En *Pool App AI,* se trabajó con la herramienta *Swagger* para construir una documentación basada en el estándar Open API 3.0, donde se registró el funcionamiento del aplicativo.
- **Integración y Despliegue continuos:** Para darle un valor agregado al producto desarrollado, se diseñó un flujo (pipeline) que permitió automatizar las pruebas unitarias, instalar dependencias, generar como artefacto una imagen Docker, y, por último, la instanciación de un contenedor donde se despliega el API.

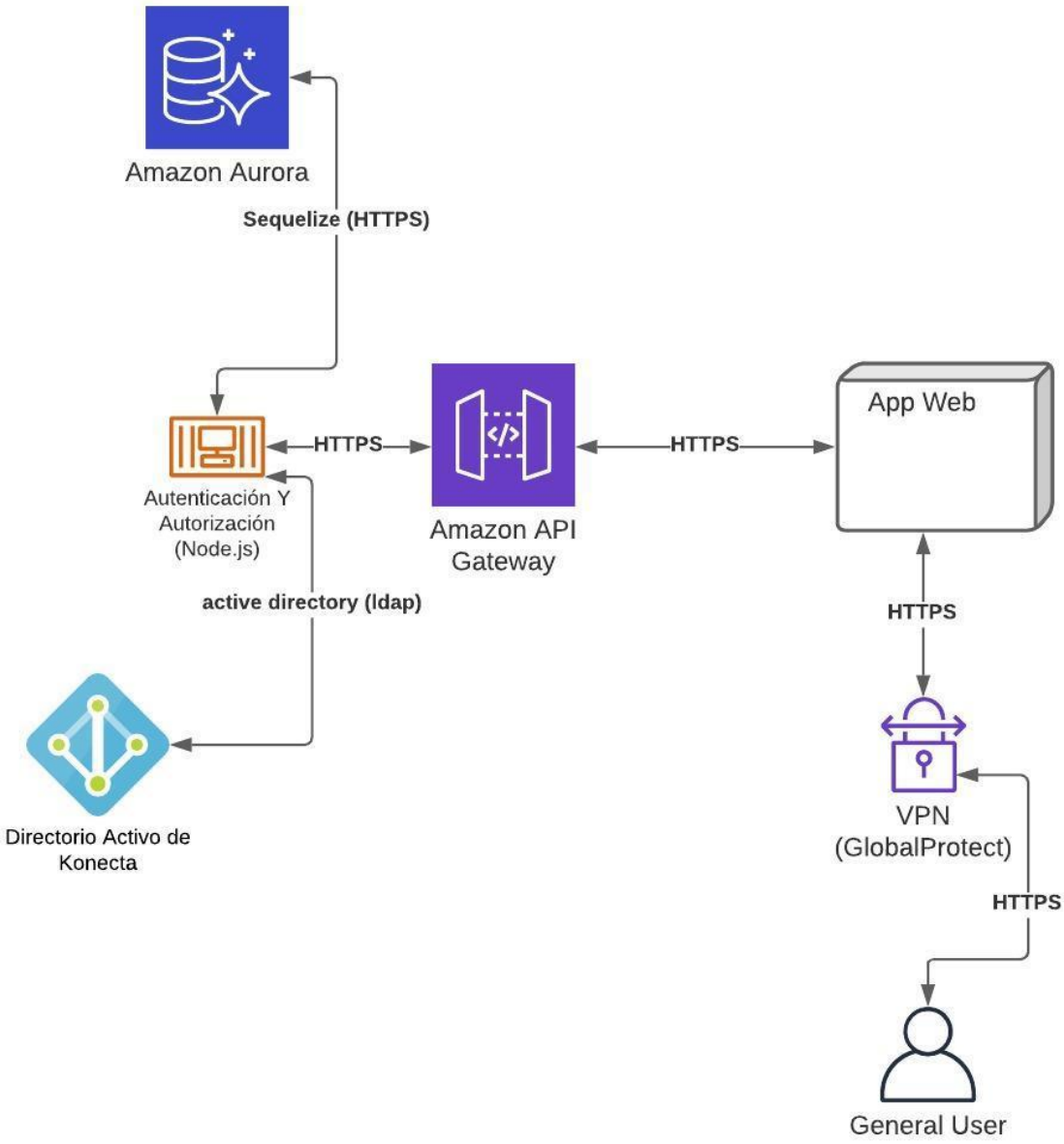
Cada etapa definida anteriormente, tomó entre una a tres iteraciones de la metodología global del proyecto a migrar.

Resultados y análisis

En la construcción de los distintos componentes del API, se logra evidenciar varios resultados; las fases iniciales del proyecto (análisis y arquitectura de código), dejaron un banco de historias de usuarios donde se explicaba a groso modo cómo debería comportarse el API, y el diseño de una arquitectura base del proyecto (que se puede observar en la **Figura 1**). Si

bien el desarrollo se basó únicamente en los componentes de **Amazon Aurora**, **Autenticación y Autorización** con sus conexiones, entender la arquitectura inicial brindó al área la iniciativa de cómo dividir los próximos microservicios.

Figura 1. *Arquitectura general Pool App AI*



Fuente: Elaboración propia utilizando Lucid

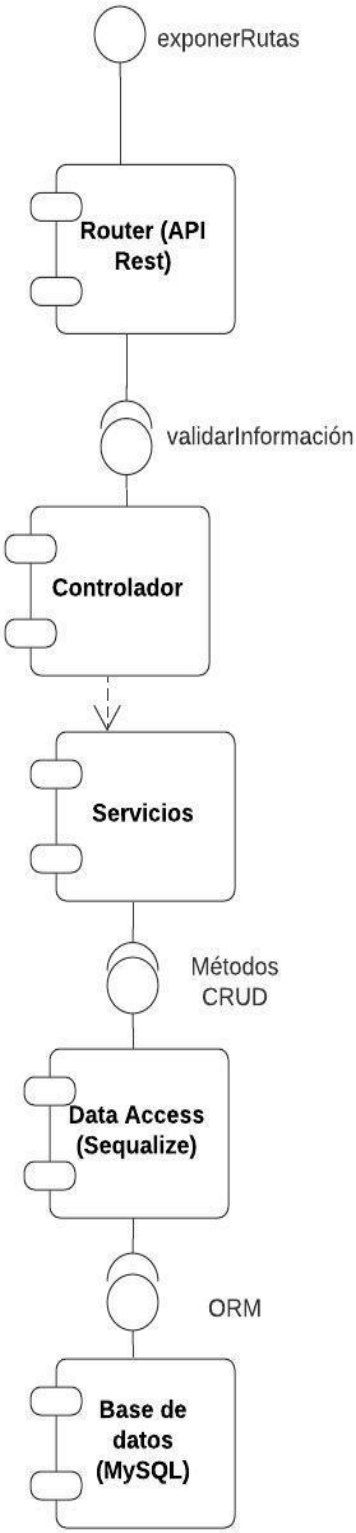
Como se observa en la **Figura 1**, los usuarios del área se autentican en la Virtual Private Network (VPN) de GlobalProtect (software externo de la compañía) que les permite visualizar la plataforma actual que posee el

área. Por medio del protocolo HTTPS, la plataforma consume los servicios expuestos en el componente de Amazon API Gateway, que sirve como orquestador hacia los diferentes microservicios construidos, en particular a la Autorización y Autenticación. Para realizar la autenticación, el microservicio consume por medio del protocolo LDAP el activo directo de la empresa, y accede a la base de datos Amazon Aurora para brindar la autorización, por medio del ORM (Object Request Mapping) Sequelize.

El componente **Autenticación y autorización** a su vez fue desarrollado con el patrón arquitectónico *Por capas,* para lograr un aplicativo modular, en el cual, los cambios o agregar nuevas funcionalidades sea un proceso fácil de realizar. La arquitectura puede ser vista en la **Figura 2**, donde cada elemento representa:

- **Base de datos:** este componente representa la base de datos relacional desplegada en Amazon Aurora. Allí se almacenan los registros de las tablas del modelo entidad relación, y se expone una interfaz para la conexión de la misma.
- **Data Access:** representa el middleware utilizado (sequelize) para acceder a los registros de la base de datos. Gracias al middleware, se evita la creación de querys crudos que pueden evitar posibles inyecciones SQL. Expone una interfaz bien definida con los métodos CRUD de las tablas en la base de datos.
- **Servicios:** este componente es usado para quitar acoplamiento en el sistema, ya que se encarga de definir una interfaz que consume los métodos expuestos en la capa anterior. Es bastante útil, ya que permite cambiar el acceso a la base de datos con solo modificar la firma de sus métodos. No expone una interfaz, debido a que su capa superior depende completamente de su implementación.
- **Controlador:** el componente contiene toda la lógica de negocio para validar la información entrante al API. Se verifica, que los tipos de datos de la petición sean correctos, y que el usuario posee los permisos pertinentes. Expone una interfaz que permite verificar las peticiones entrantes.
- **Router (API Rest):** se encarga de recibir y retornar todas las peticiones HTTP realizadas por los clientes. Se indica claramente cuáles rutas y qué métodos son soportados en el API. Gracias a la capa de *controlador*, puede acceder hasta la información almacenada en la base de datos.

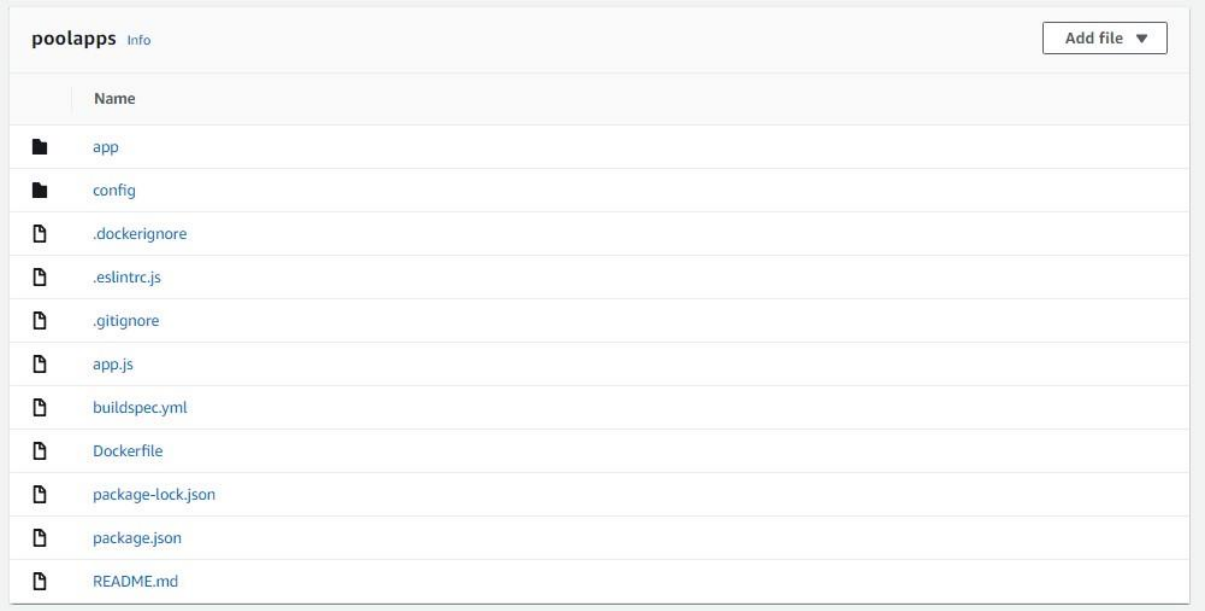
Figura 2. *Arquitectura por Capas*



Fuente: Elaboración propia utilizando Lucid

Todo el código fuente fue almacenado en ***AWS CodeCommit*** como repositorio privado (**Figura 3**) para agregar la seguridad que brinda Amazon en sus servicios.

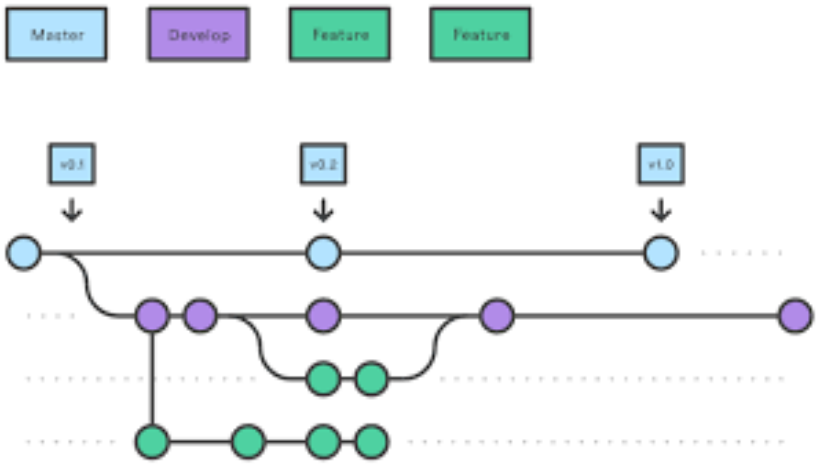
Figura 3. *Código fuente almacenado en AWS CodeCommit*



Fuente: Elaboración propia utilizando AWS CodeCommit

El flujo entre las ramas puede ser visto en la **Figura 4**, donde se tienen dos ramas principales, **master** y **develop**; con ramas secundarias (**features**) donde se crean nuevas funcionalidades para el aplicativo.

Figura 4. *Flujo entre las ramas de Git [26]*



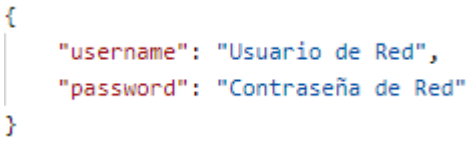
Fuente: Tomado de Atlassian [26]

La rama **master** solo se modifica cuando se tiene suficiente desarrollo para publicar una nueva versión del aplicativo (cambios en el motor de base de

datos), agregando los cambios que se encuentren en la rama **develop.** Por su parte, **develop** es modificada con los cambios que se encuentren en las múltiples ramas **feature;** por último, las ramas **feature** se desprenden de la versión más actual de **develop** y se agregan nuevamente a ella cuando se tiene un código funcional. Trabajar de este modo, brindó independencia y paralelismo en el desarrollo del API.

Para autenticar los miembros del área **AI & Data Science**, se diseñó un endpoint que solicita el usuario y la contraseña de red del integrante (datos que tiene cada empleado en la compañía para acceder a los servicios internos) para generar un JWT en caso de que se encuentre registrado en el LDAP, o generar un mensaje de error en el caso contrario. El cuerpo de la petición HTTP POST se puede observar en la **Figura 5**, y el mensaje de éxito o error en la **Figura 6** y **7** respectivamente.

Figura 5. *Cuerpo de la petición para la autenticación*

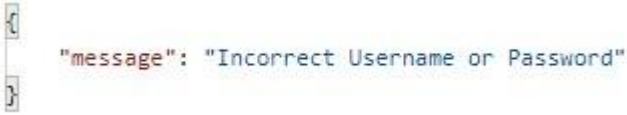


Fuente: Elaboración propia utilizando Postman

Figura 6. *Resultado exitoso de la autenticación*

Fuente: Elaboración propia utilizando Postman

Figura 7. *Resultado erróneo de la autenticación*



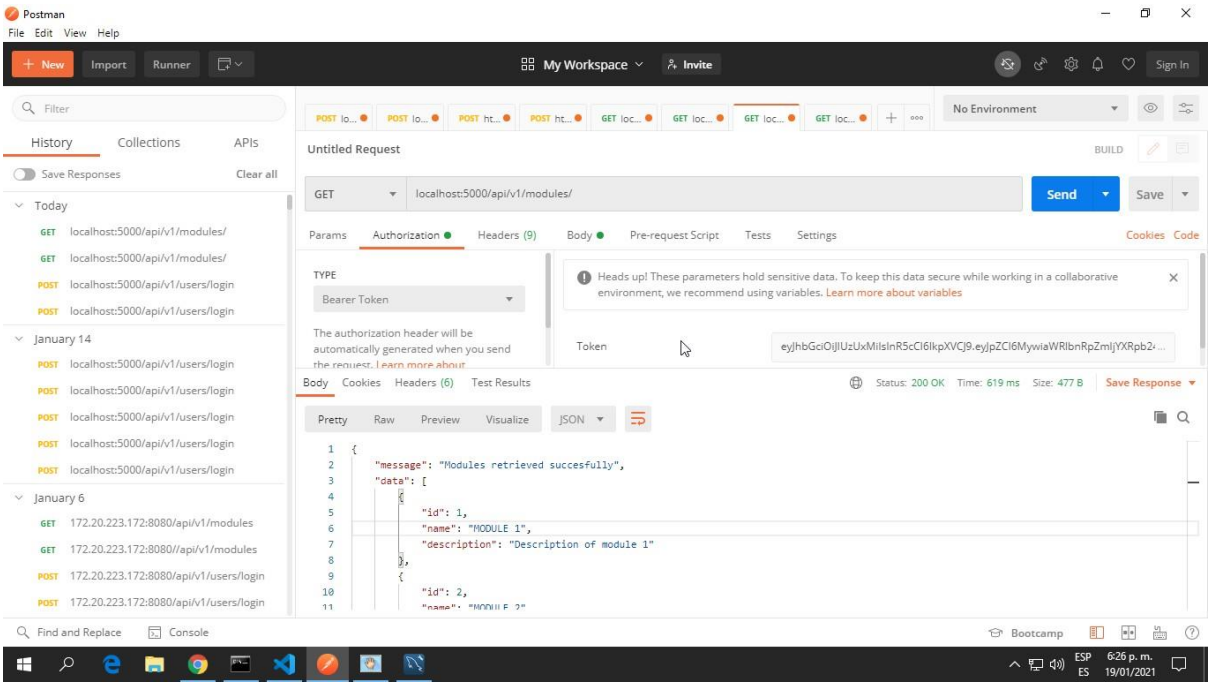
Fuente: Elaboración propia utilizando Postman

El token generado contiene la información del usuario autenticado (id en la base de datos, nombre, área, equipo, país, entre otros datos), que se envía en la cabecera **Authorization** en las peticiones realizadas al API. Allí, al decodificar el token, se logra una autorización en dos simples pasos:

1. La petición pasa por tres interceptores; el primero, se encarga de verificar que la cabecera Authorization exista en la petición y que el token fue generado desde el API; el segundo, consulta los roles y permisos asignados al usuario; y el último verifica si el usuario es un administrador.
2. Una vez obtenidos los permisos, se retorna la respuesta con los datos que el usuario puede ver

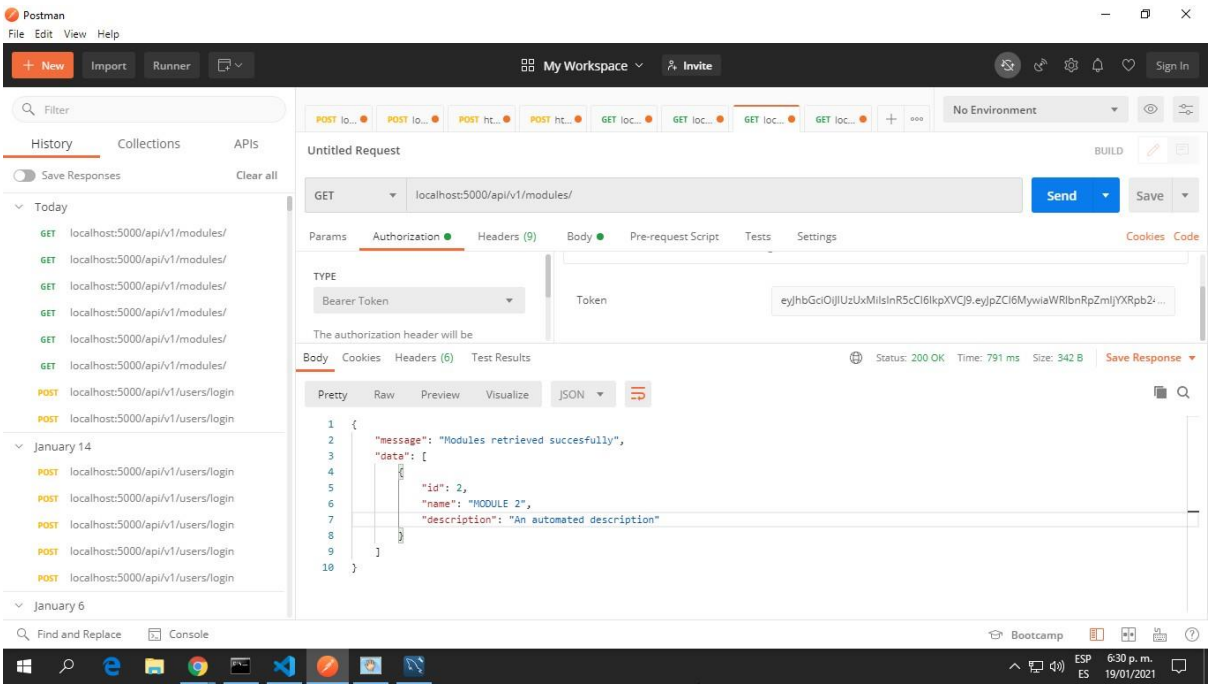
En la **Figura 8**, se muestra una petición al endpoint */api/v1/modules* con un usuario administrador, y en la **Figura 9**, se muestra la petición con un usuario regular. Note que, al usuario de la **Figura 9**, solo se le retorna un dato (información del módulo 2) ya que solo tiene permisos para acceder a él. En la **Figura 10**, se puede observar los datos retornados para el administrador.

Figura 8. *Petición al endpoint /api/v1/modules como administrador*



Fuente: Elaboración propia utilizando Postman

Figura 9. *Petición al endpoint /api/v1/modules como un usuario regular*



Fuente: Elaboración propia utilizando Postman

Figura 10. *Retorno del endpoint /api/v1/modules para un administrador*


```
{
  "message": "Modules retrieved succesfully",
  "data": [
    {
      "id": 1,
      "name": "MODULE 1",
      "description": "Description of module 1"
    },
    {
      "id": 2,
      "name": "MODULE 2",
      "description": "An automated description"
    },
    {
      "id": 3,
      "name": "MODULE 3",
      "description": "Some amazing description"
    }
  ]
}
```

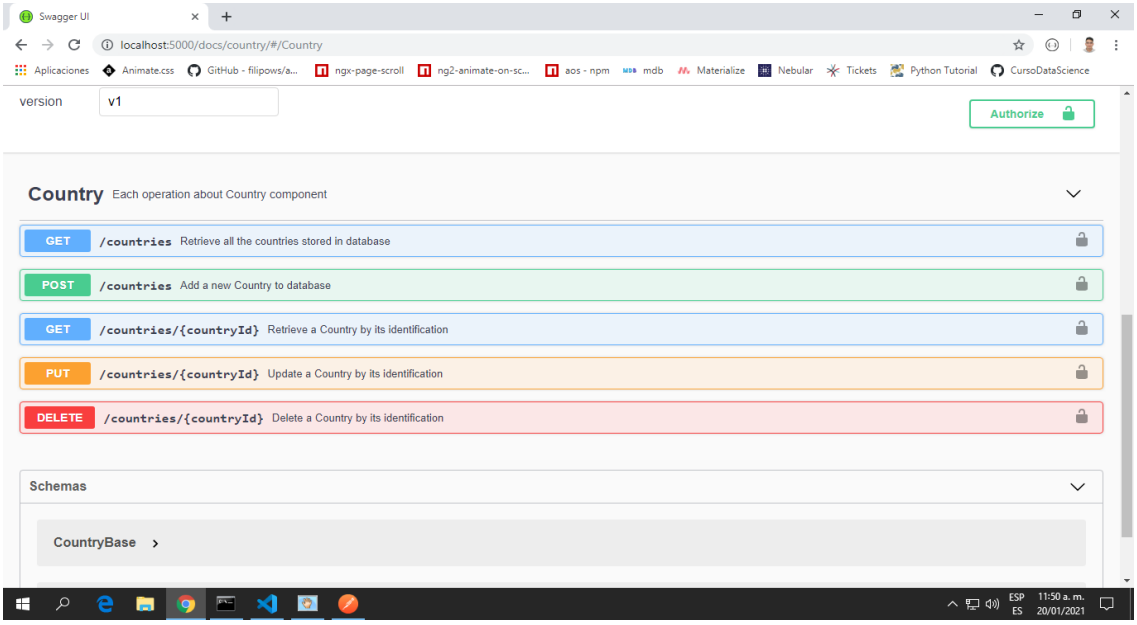

Fuente: Elaboración propia utilizando Postman

Todas las rutas expuestas están protegidas con el proceso anterior, lo que facilita saber quién es el usuario autenticado y qué puede hacer dentro del aplicativo. Solo la ruta de */api/v1/users/login* no cuenta con protección, debido que allí son generados los tokens.

Existe una ruta especial que ayuda a configurar la creación de registros bases. Permite crear de forma ágil registros de países, áreas, equipos, módulos, submódulos y aplicaciones con sus relaciones internas, facilitando al administrador para que solo se preocupe en generar los registros de autorización (roles y permisos especiales). El **Anexo 1**, muestra el contenido del cuerpo de la petición POST al endpoint */api/v1/config/base*, donde su respuesta se observa en el **Anexo 2**; note que el API retorna información útil sobre el estado de creación de los nuevos registros.

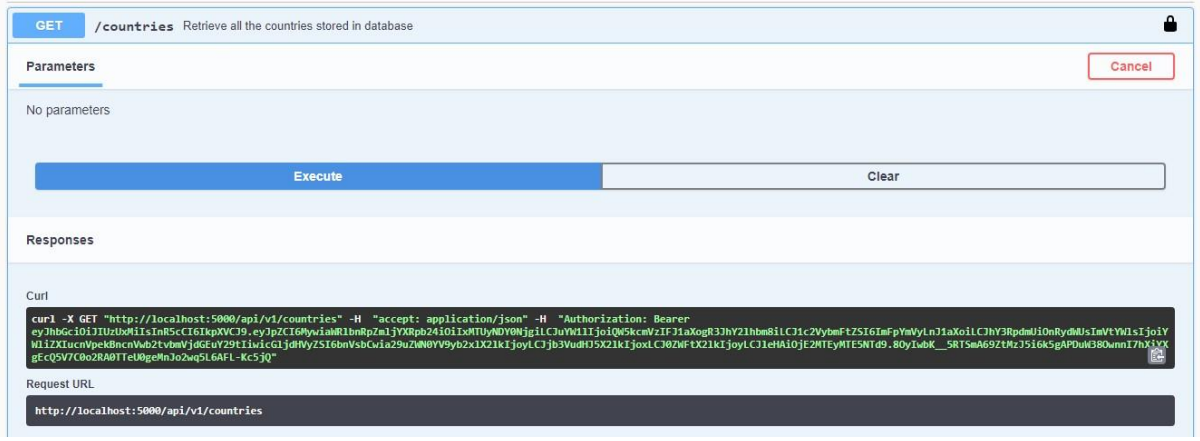
Adicionalmente, se han definido rutas en el API para exponer la documentación de sus servicios, permitiendo que futuros desarrolladores entiendan el funcionamiento de la misma. La documentación está expuesta en los endpoint con el patrón de */docs/{componente},* donde componente se refiere al nombre de las tablas principales del modelo relacional. En la **Figura 11**, se puede observar la documentación para el componente *Country;* note que, desde la misma documentación se pueden hacer pruebas al presionar el botón ***Try it out*** (**Figura 12**), se explica cada código de retorno HTTP (**Figura 13**), y se protegen las rutas para que solo usuarios autorizados puedan interactuar con ella.

Figura 11. *Página de documentación del componente Country*



Fuente: Elaboración propia utilizando Swagger UI

Figura 12. *Ejecución del método GET al endpoint /api/v1/countries desde la documentación.*



Fuente: Elaboración propia utilizando Swagger UI

Figura 13. *Posibles respuestas documentadas del endpoint /api/v1/countries*

Fuente: Elaboración propia utilizando Swagger UI

Finalmente, como valor agregado, se ha creado todo un flujo de trabajo (pipeline) para realizar un despliegue continuo en los servicios de Amazon Web Services. Primero, se definieron pruebas unitarias a nivel de servicio (encargadas de verificar el funcionamiento de la capa de servicio) y pruebas unitarias a nivel de controlador (encargadas de verificar que las respuestas de la API sean correctas). Las **Figuras 14** y **15** muestran un ejemplo de prueba unitaria de servicio y de controlador respectivamente; cada prueba sigue el patrón AAA (Arrange, Act, Assert), es decir, las primeras líneas de código preparan las variables necesarias para la ejecución de la

prueba, las líneas intermedias ejecutan la sección de código a probar, y las últimas líneas verifican que se cumpla la condición.

Figura 14. *Prueba unitaria para el servicio del componente Area*

```
it('Should retrieve Area with id 1', async (done) => {
    const areaId = 1;
    const expectedValue = {
        id: 1,
        name: "ANALITICA",
        description: "Test area"
    }

    const area = await areaService.getAreaById(areaId);

    expect(area["dataValues"]).toEqual(expectedValue);
    done();
});
```

Fuente: Elaboración propia utilizando Visual Studio Code

Figura 15. *Prueba unitaria para el controlador del componente Area*

```
it('Should return 401 HTTP status code due to missing Token', async (done) => {
    const expectedStatus = 401;
    const expectedValue = {
        message: errorMessages.UNAUTHORIZED
    }

    const response = await request(app)
        .get(areasUrl);

    expect(response.status).toEqual(expectedStatus);
    expect(response.body).toEqual(expectedValue);
    done();
});
```

Fuente: Elaboración propia utilizando Visual Studio Code

Como segundo paso, se creó un proyecto en AWS CodeBuild para realizar una integración continua. Para ello, en el código fuente se produce un archivo llamado *buildspec.yml* que indica a CodeBuild cómo operar. El contenido del archivo completo puede observarse en la **Tabla 1**, donde sus etapas se encargan de:

- install
    - Instalación de software requerido (Node.js en la versión 12)

- pre_build
    - Instalar dependencias del proyecto
    - Hacer un login al servicio de AWS Elastic Container Registry
    - Establecer el repositorio de imágenes Docker
    - Definir tags para imagen Docker a crear
- build
    - Automatizar las pruebas unitarias generadas
    - Crear la imagen Docker si las pruebas se ejecutan exitosamente
    - Etiquetar la imagen Docker con el tag creado en el paso anterior
- post_build
    - Publicar la imagen en el repositorio de AWS Elastic Container Registry
    - Generar un archivo llamado ***imagedefinitions.json*** para levantar un contenedor en AWS Elastic Container Service

Tabla 1. *Código del archivo buildspec.yml*

```
version: 0.2

phases:
  install:
    commands:
      - echo Preparing Runtime Environment...
      - echo Installing Node V12...
      - curl -sL https://deb.nodesource.com/setup_12.x | bash -
      - apt install -y nodejs
      - echo Node V12 installed.

  pre_build:
    commands:
      - echo Installing dependencies...
      - npm install
      - echo Dependencies installed.
      - echo Logging in to Amazon ECR...
      - aws --version
      - $(aws ecr get-login --region us-east-1 --no-include-email)
      - REPOSITORY_URI=751970947055.dkr.ecr.us-east-
1.amazonaws.com/back-pool-apps
      - COMMIT_HASH=$(echo $CODEBUILD_RESOLVED_SOURCE_VERSION | cut -c
1-7)
```

```
      - IMAGE_TAG=${COMMIT_HASH:=latest}

  build:
    commands:
      - echo Build started on `date`
      - echo Start running Unit Tests...
      - npm run test
      - echo Unit Tests finished.
      - echo Creating Docker Image...
      - docker build -t $REPOSITORY_URI:latest .
      - docker tag $REPOSITORY_URI:latest $REPOSITORY_URI:$IMAGE_TAG
      - echo Docker image created successfully.

  post_build:
    commands:
      - echo Build completed on `date`
      - echo Pushing the Docker images...
      - docker push $REPOSITORY_URI:latest
      - docker push $REPOSITORY_URI:$IMAGE_TAG
      - echo Docker images published.
      - echo Writing image definitions file...
      - printf '[{"name":"back-pool-apps-container","imageUri":"%s"}]'
$REPOSITORY_URI:$IMAGE_TAG > imagedefinitions.json
      - echo Imagedefinitions file created successfully.

artifacts:
  files: imagedefinitions.json
```

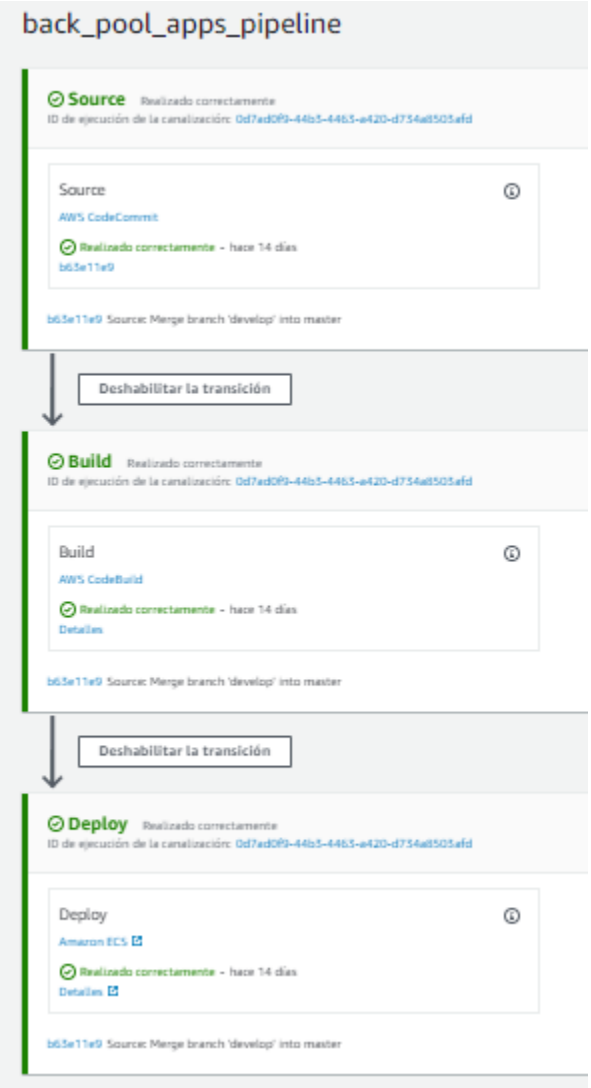
Fuente: Elaboración propia utilizando Visual Studio Code

El tercer y último paso, fue la creación de un proyecto en AWS CodePipeline que se encargue de unificar los pasos anteriores para lograr un despliegue continuo, que permitió agilizar los tiempos de entrega del servicio (véase la **Figura 16**). Lo pasos generales del flujo pueden ser descritos así:

1. Verifica cuando se realiza un cambio a la rama master del proyecto
2. Descarga el código fuente del repositorio disponerlo a AWS CodeBuild
3. Ejecutar los pasos del AWS Codebuild y producir el artefacto *imagedefinition.json*
4. Con el archivo anterior se levanta un contenedor en AWS Elastic Container Service y AWS Fargate

5. El usuario administrador toma la IP generado para el contenedor y el aplicativo queda desplegado

Figura 16. *Pipeline de Pool App AI en AWS CodePipeline*



Fuente: Elaboración propia utilizando AWS CodePipeline

Conclusiones

- Los resultados obtenidos en la realización del proyecto son bastante satisfactorios, debido a que se cumple con los objetivos propuestos y además permitió crear el interés en el diseño de una nueva célula interna, que se encargará de implementar la cultura DevOps en los proyectos existentes y futuros.
- Si bien los objetivos del proyecto plantean alcanzar una integración continua, al utilizar la metodología incremental e iterativa, se logró

superar el planteamiento inicial, debido que, al poder dividir el proyecto en iteraciones, fue fácil diseñar y probar los nuevos elementos construidos del API, incrementando la velocidad de desarrollo hasta permitir alcanzar un Despliegue Continuo.

- El uso correcto de un Sistema de Control de Versiones (Git) facilitó el desarrollo del proyecto, dado que permitió trabajar de forma paralela al independizar los cambios de cada desarrollador.
- El método de autenticación y autorización cumplió el objetivo planteado, satisfaciendo la necesidad del área e incluso permitió la construcción de un nuevo proyecto a nivel global para autenticar los empleados en los distintos países donde reside Konecta.
- Se evidencio que el uso de una metodología ágil permite realizar cambios en el proyecto sin causar el fracaso del mismo. Es así como en un principio se planteó una arquitectura *serverless*, pero que por poco conocimiento y experiencia en la estructuración de la misma, se tuvo que cambiar el planteamiento a una arquitectura *por capas,* cambiando completamente la elicitación de requisitos. Sin embargo, gracias a los beneficios de la metodología incremental e iterativa, se logró cambiar el planteamiento arquitectural y se culminó con éxito el proyecto.

Referencias Bibliográficas

[1] "Languages and Frameworks | Technology Radar", Thoughtworks, 2020. [Online]. Available: https://www.thoughtworks.com/radar/languages-and-frameworks. [Accessed: 30- Sep- 2020]

[2] S. Al-Fedaghi, "Developing Web Applications", ResearchGate, 2011. [Online]. Available: https://www.researchgate.net/publication/228849246_Developing_Web_Applications. [Accessed: 02- Oct- 2020]

[3] H. B Prajapati and V. Dabhi, "High Quality Web-Application Development on Java EE Platform", ResarchGate, 2009. [Online]. Available: https://www.researchgate.net/publication/224398744_High_Quality_Web-Application_Development_on_Java_EE_Platform. [Accessed: 02- Oct- 2020]

[4] "Desarrollo iterativo e incremental", Proyectos Ágiles. [Online]. Available: https://proyectosagiles.org/desarrollo-iterativo-incremental/. [Accessed: 02- Oct- 2020]

[5] K. Fakhroutdinov, "The Unified Modeling Language", Uml-diagrams. [Online]. Available: https://www.uml-diagrams.org/. [Accessed: 03- Oct- 2020]

[6] H. Mu and S. Jiang, "Design patterns in software development", Ieee Xplore, 2011. [Online]. Available: https://ieeexplore.ieee.org/document/5982228. [Accessed: 03- Oct- 2020]

[7] "Frontend vs Backend", GeeksforGeeks, 2019. [Online]. Available: https://www.geeksforgeeks.org/frontend-vs-backend/. [Accessed: 03- Oct- 2020]

[8] J. Freeman, "What is an API? Application programming interfaces explained", InfoWorld, 2019. [Online]. Available: https://www.infoworld.com/article/3269878/what-is-an-api-application-programming-interfaces-explained.html. [Accessed: 05- Oct- 2020]

[9] R. Thomas Fielding, "Architectural Styles and the Design of Network-based Software Architectures", UNIVERSITY OF CALIFORNIA, IRVINE, 2000. [Online]. Available:

https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf. [Accessed: 05- Oct- 2020]
[10] "Hypertext Transfer Protocol -- HTTP/1.1", W3.org, 1999. [Online]. Available: https://www.w3.org/Protocols/HTTP/1.1/rfc2616.pdf. [Accessed: 05- Oct- 2020]
[11] "Transport Layer Security (TLS)", IEEE OC CyberSecurity SIG, 2020. [Online]. Available: https://site.ieee.org/ocs-cssig/?page_id=658. [Accessed: 05- Oct- 2020]
[12] S. Sharad, A. Singh, Divyanshu, and A. Rai, "Research Paper on Active Directory", International Research Journal of Engineering and Technology (IRJET), 2019. [Online]. Available: https://www.irjet.net/archives/V6/i4/IRJET-V6I4761.pdf. [Accessed: 30- Mar- 2021]
[13] "LDAP", LDAP.com. [Online]. Available: https://ldap.com/. [Accessed: 30- Mar- 2021]
[14] D. Stamer, O. Zimmermann and K. Sandkuhl, "What Is a Framework? - A Systematic Literature Review in the Field of Information Systems", ResearchGate, 2016. [Online]. Available: https://www.researchgate.net/publication/307911744_What_Is_a_Framework_-_A_Systematic_Literature_Review_in_the_Field_of_Information_Systems. [Accessed: 05- Oct- 2020]
[15] "express", npm, 2019. [Online]. Available: https://www.npmjs.com/package/express. [Accessed: 06- Oct- 2020]
[16] "OpenAPI Specification - Version 3.0.3 | Swagger", Swagger.io, 2020. [Online]. Available: https://swagger.io/specification/. [Accessed: 06- Oct- 2020]
[17] "Modelos de servicios IaaS, PaaS y SaaS de IBM Cloud", IBM. [Online]. Available: https://www.ibm.com/co-es/cloud/learn/iaas-paas-saas. [Accessed: 06- Oct- 2020]
[18] P. Jha and R. Khan, "A Review Paper on DevOps: Beginning and More To Know", ResearchGate, 2018. [Online]. Available: https://www.researchgate.net/publication/325792247_A_Review_Paper_on_DevOps_Beginning_and_More_To_Know. [Accessed: 01- Oct- 2020]
[19] "Integración Continua, Entrega Continua, y Despliegue Continuo", Viewnext. [Online]. Available: https://www.viewnext.com/integracion-continua/. [Accessed: 06- Oct- 2020]
[20] "AWS CodeCommit | Servicio administrado de control de código fuente", Amazon Web Services, Inc., 2021. [Online]. Available: https://aws.amazon.com/es/codecommit/. [Accessed: 13- Jan- 2021]
[21] "AWS CodeBuild – Servicio de compilación completamente administrado", Amazon Web Services, Inc., 2021. [Online]. Available: https://aws.amazon.com/es/codebuild/. [Accessed: 13- Jan- 2021]
[22] "Amazon ECR | Registro de contenedores de Docker | Amazon Web Services", Amazon Web Services, Inc., 2021. [Online]. Available: https://aws.amazon.com/es/ecr/. [Accessed: 13- Jan- 2021]
[23] "AWS | Gestión de contenedores (ECS)", Amazon Web Services, Inc., 2021. [Online]. Available: https://aws.amazon.com/es/ecs/. [Accessed: 13- Jan- 2021]
[24] "AWS Fargate | Motor informático sin servidor | Amazon Web Services", Amazon Web Services, Inc., 2021. [Online]. Available: https://aws.amazon.com/es/fargate/. [Accessed: 13- Jan- 2021]
[25] "AWS CodePipeline | Integración y entrega continua", Amazon Web Services, Inc., 2021. [Online]. Available: https://aws.amazon.com/es/codepipeline/. [Accessed: 13- Jan- 2021]
[26] "Flujo de trabajo de Gitflow | Atlassian Git Tutorial", Atlassian, 2021. [Online]. Available: https://www.atlassian.com/es/git/tutorials/comparing-workflows/gitflow-workflow. [Accessed: 30- Mar- 2021]

Anexos

Anexo 1. Cuerpo de petición al endpoint */api/v1/config/base*


```
{
    "countries": [
        {
            "description": "Some amazing description"
        },
        {
            "name": "Colombia"
        },
        {
            "name": "Venezuela"
        }
    ],
    "konectaRoles": [
        {
            "description": "Amazing one"
        },
        {
            "name": "Analista"
        },
        {
            "name": "Aprendiz",
            "description": "Automated konecta role"
        }
    ],
    "areas": [
        {
            "description": "Area description"
        },
        {
            "name": "AI & Data Science"
        },
        {
            "name": "Tesorería",
            "description": "Automated description"
        }
    ],
    "teams": [
```

```
        {
            "description": "Team descrption"
        },
        {
            "name": "Team 1",
            "description": "Description"
        },
        {
            "name": "Team 1",
            "areaName": "Tesoreria"
        },
        {
            "name": "Team 2",
            "areaName": "AI & Data Science",
            "description": "Automated description"
        },
        {
            "name": "Team 3",
            "areaName": "Not found"
        }
    ],
    "modules": [
        {
            "name": "Module 1"
        },
        {
            "description": "Some description"
        },
        {
            "name": "Module 2",
            "description": "An automated description"
        }
    ],
    "submodules": [
        {
            "description": "failure"
        },
        {
            "name": "Submodule 1"
        },
        {
```

```
            "name": "Submodule 1",
            "moduleName": "module 1"
        },
        {
            "name": "Submodule 2",
            "moduleName": "Not valid"
        }
    ],
    "submoduleApps": [
        {
            "description": "Some description"
        },
        {
            "name": "Sub App 1"
        },
        {
            "name": "Sub App 1",
            "moduleName": "Module 1"
        },
        {
            "name": "Sub App 1",
            "moduleName": "Module 1",
            "submoduleName": "Submodule 1"
        },
        {
            "name": "Sub App 2",
            "moduleName": "Not found",
            "submoduleName": "Submodule 1"
        },
        {
            "name": "Sub App 3",
            "moduleName": "Module 1",
            "submoduleName": "Not found"
        },
        {
            "name": "Sub App 4",
            "moduleName": "Module 1",
            "submoduleName": "Submodule 1",
            "icon": "https://www.icon.com",
            "description": "Some automated description"
        }
```

```
    ]
}
```


Fuente: Elaboración propia utilizando Postman

Anexo 2. Respuesta del endpoint /api/v1/config/base


```
{
    "message": "Data of tables created successfully",
    "information": {
        "countries": [
            {
                "countryName": "Not specified",
                "message": "Country name is required"
            },
            {
                "countryName": "Colombia",
                "message": "Country already exists"
            },
            {
                "countryName": "Venezuela",
                "message": "Country created succesfully"
            }
        ],
        "konectaRoles": [
            {
                "konectaRole": "Not specified",
                "message": "Konecta Role name is required"
            },
            {
                "konectaRole": "Analista",
                "message": "Konecta Role already exists"
            },
            {
                "konectaRole": "Aprendiz",
                "message": "Konecta Role already exists"
            }
        ],
        "areas": [
            {
                "areaName": "Not specified",
                "message": "Area name is required"
            },
```

```
            {
                "areaName": "AI & Data Science",
                "message": "Area already exists"
            },
            {
                "areaName": "Tesorería",
                "message": "Area already exists"
            }
        ],
        "teams": [
            {
                "teamName": "Not specified",
                "message": "Team name and Area name are required"
            },
            {
                "teamName": "Not specified",
                "message": "Team name and Area name are required"
            },
            {
                "teamName": "Team 1",
                "message": "Team already exists"
            },
            {
                "teamName": "Team 2",
                "message": "Team created successfully"
            },
            {
                "teamName": "Team 3",
                "message": "Area name not found"
            }
        ],
        "modules": [
            {
                "moduleName": "Module 1",
                "message": "Module already exists"
            },
            {
                "moduleName": "Not specified",
                "message": "Module name is required"
            },
            {
```

```
                "moduleName": "Module 2",
                "message": "Module already exists"
            }
        ],
        "submodules": [
            {
                "submoduleName": "Not specified",
                "message": "Submodule name and Module name are required"
            },
            {
                "submoduleName": "Not specified",
                "message": "Submodule name and Module name are required"
            },
            {
                "submoduleName": "Submodule 1",
                "message": "Submodule already exists"
            },
            {
                "submoduleName": "Submodule 2",
                "message": "Module not found"
            }
        ],
        "submoduleApps": [
            {
                "subAppName": "Not specified",
                "message": "Submodule APP name, Module name and Submodule name
are required"
            },
            {
                "subAppName": "Not specified",
                "message": "Submodule APP name, Module name and Submodule name
are required"
            },
            {
                "subAppName": "Not specified",
                "message": "Submodule APP name, Module name and Submodule name
are required"
            },
            {
                "subAppName": "Sub App 1",
                "message": "Submodule App already exists"
```

```
            },
            {
                "subAppName": "Sub App 2",
                "message": "Module name not found"
            },
            {
                "subAppName": "Sub App 3",
                "message": "Submodule name not found"
            },
            {
                "subAppName": "Sub App 4",
                "message": "Submodule App already exists"
            }
        ]
    }
}
```


Fuente: Elaboración propia utilizando Postman