My Eagle Test (Backend)

Implementación de un API que cumpla los requisitos exigidos para Mi Águila.

Diego Andrés Baquero

Agosto, 2019

1. GENERALIDADES

1.1. INTRODUCCIÓN

El presente documento busca presentar de forma general el diseño del API construida para Mi Águila, donde se buscar resolver y mejorar la escalabilidad de la aplicación con el fin de mejorar los servicios prestados por la compañía. Por lo tanto, se exige que se realicen algunas funcionalidades que podrán mejorar su operación.

1.2. PLANTEAMIENTO DEL PROBLEMA

Mi Águila lleva a cabo cerca de 5.000 viajes diarios y satisfacer la demanda es un punto primordial para la compañía.

1.3. OBJETIVOS

1.3.1. Objetivo General

Implementar una solución basada en Tecnologías de la Información y Comunicación que mejores los procesos operativos para Mi Águila.

1.3.2. Objetivos Específicos

- Brindar información a tiempo
- Generar seguridad de la información
- Analizar y actuar sobre los procesos operativos

1.4. JUSTIFICACIÓN

La ejecución de este proyecto está contemplada en el marco de ser un desarrollador *Backend* en Mi Águila. Por lo tanto, se han planteado una serie de historias de usuario que se tuvieron que llevar a cabo. Adicionalmente, se debe presentar la correspondiente documentación que permita obtener más información sobre las aptitudes al momento de desarrollar y documentar cada uno de los proyectos o soluciones de software que se puedan desarrollar para Mi Águila.

2. IMPLEMENTACIÓN

El API tiene como objetivo primordial mejorar los procesos operativos en Mi Águila, basado en las TIC. De esta manera, es necesario definir los requisitos del negocio y los requisitos de software que darán solución al problema planteado anteriormente. Por otro lado, será necesario presentar las historias de usuario descritas por especialistas en el área de la salud.

2.1. REQUISITOS

Los requisitos son esas necesidades a las que se debe enfrentar la solución de software planteada y que serán de vital importancia para su implementación.

2.1.1. Requisitos del negocio

Inicialmente, se deben plantear los requisitos de negocio, los cuales son las necesidades que se deben resolver para mejorar un proceso o reducir gastos en una organización, en este caso, Mi Águila.

- Mejorar su servicio principal de viajes.

2.1.2. Características

Por otro lado, las características son de forma general las funcionalidades básicas que tendrá la aplicación móvil.

- Consultar viajes
- Crear viajes
- Editar viajes
- Eliminar viajes

2.1.3. Historias de usuario

De acuerdo con las características descritas anteriormente, junto a un equipo de especialistas en las ciencias de la salud, se plantean las historias de usuario, las cuales se encargan de describir las funcionalidades de forma muy general, las cuales serán uno de los insumos para el diseño de la aplicación, junto con el mapa de entradas y salidas.

Por otro lado, de acuerdo a las siguientes historias de usuario se plantea el MVP:

- Consultar la cantidad de viajes totales
- Consultar la cantidad de viajes totales por ciudad
- Consultar la cantidad de viajes totales por país
- Crear un viaje

- Actualizar un viaje
- Consultar los viajes actuales
- Eliminar un viaje

2.2. PROPUESTA TÉCNICA

En esta sección, se detallan los elementos de software y herramientas que se utilizarán para la implementación del API.

2.2.1. API

El API está basada en el patrón de arquitectura de software Modelo-Vista-Controlador (MVC), la cual separa los datos y la lógica de negocio de una aplicación, de la interfaz de usuario, y el módulo encargado de gestionar los eventos y las comunicaciones.

A continuación, se presenta brevemente cada una de estas, con el fin de brindar una introducción al concepto de Web API, el cual es fundamental para el funcionamiento de la aplicación.

Modelo: Es la abstracción de la información en la aplicación y se encarga de gestionar todos los accesos a dicha información. Es decir, es el componente que se relaciona directamente con la base de datos a través de los diferentes tipos de consultas, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación**.**

Vista: Se encarga de presentar el modelo, es decir la información y la lógica del negocio de una forma adecuada para interactuar con el usuario. Sin embargo, la aplicación solo cuenta con dos vistas. La primera, que renderiza un mensaje de que la aplicación se ha ejecutado exitosamente; y la segunda, que muestra un mensaje si ha ocurrido un error al ejecutarla.

Controlador: Este módulo se encarga de responder a los diferentes eventos, los cuales, generalmente son accionados por el usuario de la aplicación a través de la vista (en este caso una petición *http*) e invoca peticiones al modelo cuando se hace algún tipo de solicitud a la información.

Por un lado, para la construcción del modelo y los controladores, se realiza la implementación con el lenguaje de programación C#, respaldado por el *Framework* de .NET (.NET Core 2.2), propietario de Microsoft Corporation. Por otro lado, las vistas son realizadas con el uso del lenguaje de marcas de hipertexto (HTML), para el maquetado de estas, CSS y el *Framework* Bootstrap para la implementación de estilos y, por último, el lenguaje de programación JavaScript.

Finalmente, el API cuenta con una clase (y su interfaz) dedicada a la implementación de los servicios REST, que serán expuestos para *API Client* a través de solicitudes HTTP e implementando el formato JSON para el cambio de datos.

2.2.2. Base de Datos

Inicialmente, es necesario explicar las razones por las cuales se plantea un modelo no relacional (No SQL), de acuerdo a las necesidades presentadas para el API. Por lo tanto, se considera que una estructura JSON podrá facilitar y acelerar el proceso de desarrollo. Adicionalmente, la escalabilidad de una aplicación juega un papel muy importante a la hora de decidir, debido a que la gran demanda de usuarios y servicios permiten que se pueda realizar una escalabilidad tanto vertical, como horizontal. Además, la flexibilidad permite que el esquema pueda cambiar fácilmente, sin tener agregar campos y modificar de forma drástica el diseño actual de la base de datos.

Finalmente, se realiza la elección de MongoDB como motor de bases de datos, de acuerdo a la fácil integración con muchos lenguajes a través de su conector. Adicionalmente, MongoDB desde la versión 4, presenta transacciones ACID (*Atomicity, Consistency, Isolation, Durability*) entre múltiples documentos, una de las desventajas (hasta ahora) contra las bases de datos relacionales. Por último, MongoDB es de código abierto y el tema de licenciamiento se convierte en una ventaja sobre la mayoría de motores de bases de datos SQL.

2.3. DISEÑO DE LA APLICACIÓN

En esta sección se detalla la planeación y el diseño de la aplicación, desde un punto de vista de la ingeniería y la arquitectura de software, con el objetivo de construir una solución de software que permita escalabilidad, interoperabilidad y un buen rendimiento, para dar las necesidades del negocio.

5.3.1. Modelo de datos

Como se mencionó anteriormente, una de las ventajas de MongoDB es que su estructura está basada en un documento JSON, el cual, es el siguiente:

```
{
    "start" : {
    "date" : {
          "$date" : "2019-01-25T19:06:27.936+0000"
     },
    "pickup_address" : "Cl. 90 #19-41, Bogotá, Colombia" ,
    "pickup_location" : {
        "type" : "Point" ,
        "coordinates" : [-74.0565192, 4.6761959]
     }
```

```
    },
    "end" : {
        "date" : null ,
        "pickup_address" : "Ac. 100 #13-21, Bogotá, Colombia" ,
        "pickup_location" : {
            "type" : "Point" ,
            "coordinates" : [-74.0465655 , 4.6830892]
        }
    },
    "country" : {
        "name" : "Colombia"
    },
    "city" : {
        "name" : "Bogotá"
    },
   "passenger" : {
        "first_name" : "Ricardo" ,
        "last_name" : "Sarmiento"
    },
    "driver" : {
        "first_name" : "Julio Alberto" ,
        "last_name" : "Mesa Rodriguez"
    },
    "car" : {
        "plate" : "ESM308"
    },
    "status" : "started" ,
    "check_code" : "66" ,
    "createdAt" : {
         "$date" : "2019-01-25T19:03:53.251+0000"
    },
   "updatedAt" : {
         "$date" : "2019-01-25T19:47:04.397+0000"
    },
    "price" : 13800.0 ,
    "driver_location" : {
         "type" : "Point" ,
         "coordinates" : [-74.06017631292343, 4.669553302335373]
    }
}
```


5.3.2. Casos de Uso

A continuación, se describe el diagrama de casos de uso, para el desarrollo del proyecto se identifican un único actor, el *API Client*. El primer caso de uso corresponde a las acciones a realizar por el cliente dentro de la implementación del API.

Los casos de uso generales son las principales acciones con las que puede interactuar el actor con el sistema, estos a su vez pueden extender o incluir otros casos de uso en el desarrollo de los primeros.

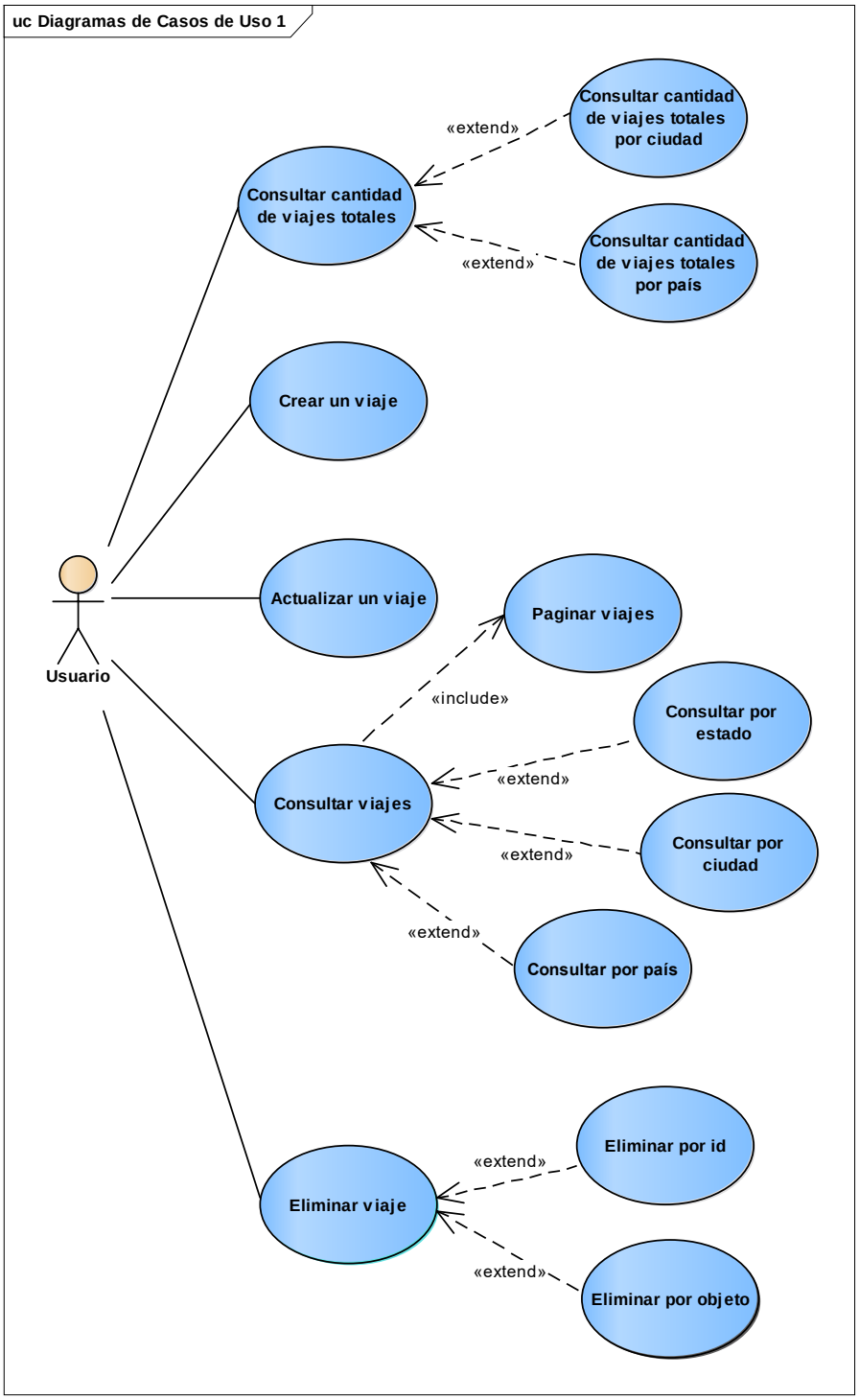

Imagen 1. Diagrama de Casos de Uso Generales

2.3.3. Diagrama de Descomposición

El siguiente diagrama presenta los módulos funcionales del API.

- **Trip:** En este módulo se realiza la gestión de los viajes.
    - Trip Handler: Este módulo realiza la gestión de viajes.
    - Trip Search: Este módulo realiza la paginación, filtrado y consulta de los viajes en el documento *Trips* de la base de datos.

A continuación, se detalla el diagrama de descomposición, el cual detalla el único módulo implementado hasta la fecha:

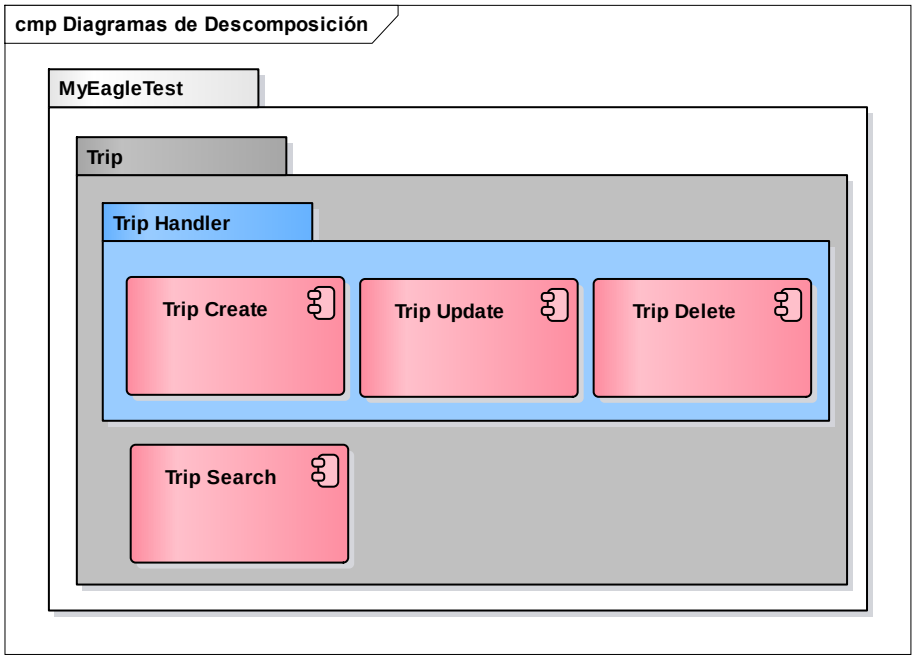

Imagen 2. Diagrama de Descomposición

2.3.4. Diagrama de Despliegue

Los Diagramas de Despliegue modelan la arquitectura en tiempo de ejecución de un sistema. Es decir, muestra los nodos (o máquinas) que trabajan para llevar a cabo la ejecución de la aplicación.

Anteriormente, se ha especificado que el sistema cuenta con un API, la cual expone los servicios su respectivo cliente. Por lo tanto, tendremos un nodo de aplicación que ejecuta estas implementaciones. El API es desplegada en un

ambiente altamente disponible en Microsoft Azure, donde podrá ser fácilmente escalable.

Por otro lado, se cuenta con una máquina dedicada como servidor de base de datos, en la cual se ejecuta la instancia que contiene la base de datos del sistema. De acuerdo con lo mencionado previamente, se ha implementado la base de datos usando el motor MongoDB, utilizando *CosmosDB*, que es la implementación de SaaS (*Software as a Service*), que Azure provee para sus bases de datos de MongoDB, de forma distribuida. Por otro lado, Microsoft Azure provee información detallada de recurrencia y peticiones a la base de datos. Adicionalmente, permite restringir las peticiones a la base de datos, para que solo sea consultada desde nuestro ambiente desplegado en el nodo de aplicación.

Finalmente, se modela el cliente del API, que podrá ser desplegada en cualquier tipo de dispositivo, con diferentes Sistemas Operativos y ambientes de ejecución.

A continuación, se muestra el diagrama de despliegue del sistema:

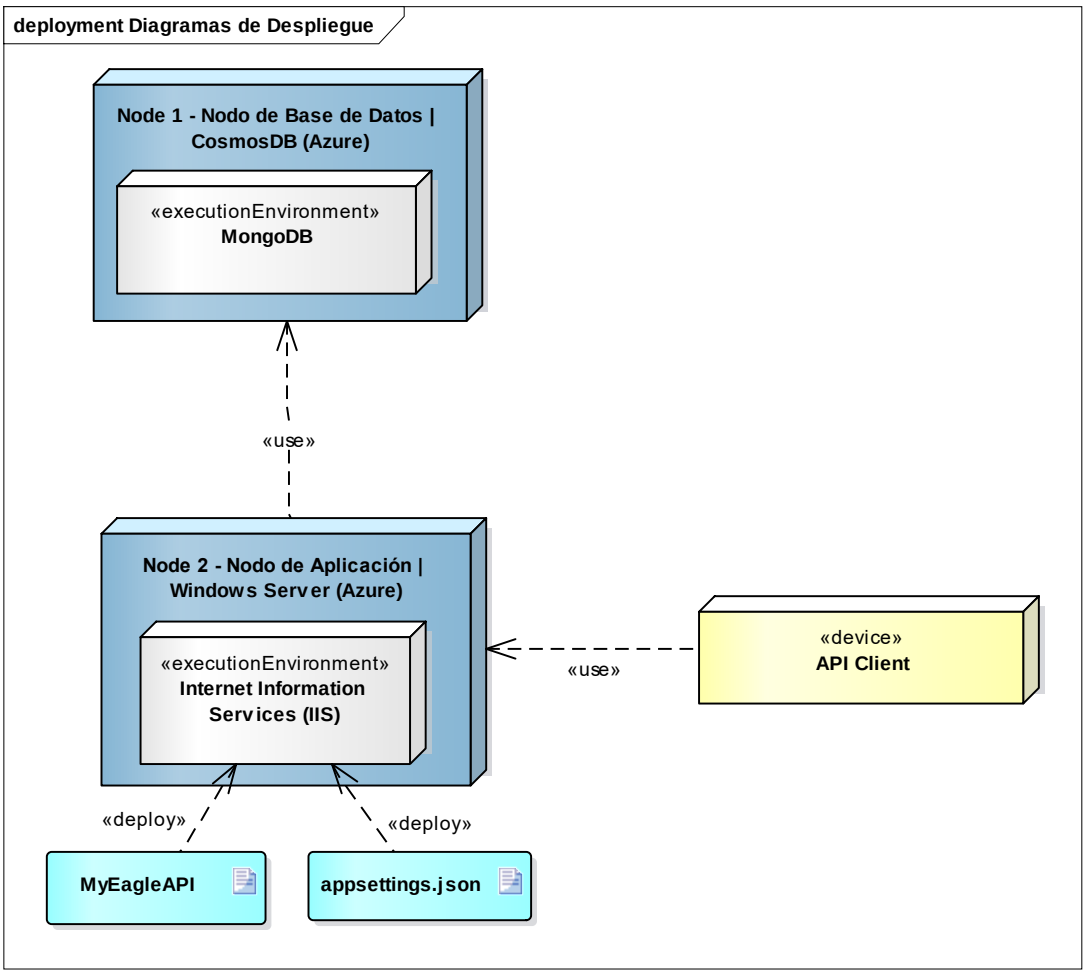

Imagen 3. Diagrama de Despliegue

2.3.5. Diagrama de la Arquitectura

El API cuenta con cuatro módulos esenciales, los cuales son: controladores, servicios, repositorios y modelo. Inicialmente, los controladores son la puerta de entra al API y es donde se definen los *endpoints*. Los servicios (y sus interfaces) se encargan de procesar la lógica de negocios de la aplicación. Los repositorios (y sus interfaces) son la puerta de acceso a la base de datos, los cuales son separados de los servicios con el objetivo de cumplir con el Principio de Responsabilidad Única. Por último, las clases de dominio son las que mapean los elementos de bases de datos.

El API interactúa con su cliente a través de peticiones *HTTP* y su correspondiente respuesta en formato JSON. Adicionalmente, interactúa con el documento alojado en la base de datos, con la información de todos los viajes.

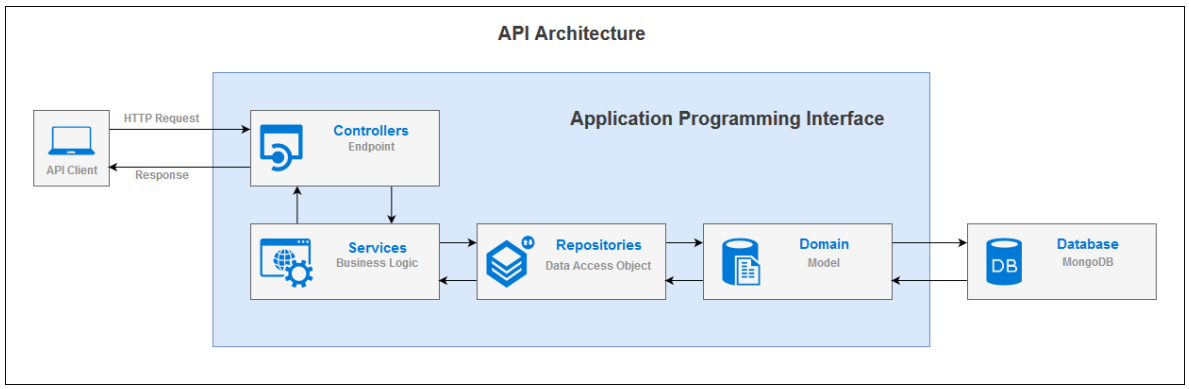

Imagen 4. Arquitectura general del API