ARQUITECTURA DE SOFTWARE BASADA EN MICROSERVICIOS PARA EL DESARROLLO DE UNA APLICACIÓN WEB DE RUTEO DE VEHÍCULOS

Jesús C. Carmona-Frausto[1], Luis F. Zurita-González[1], Adriana Mexicano-Santoyo[1], Lilia del C. García-Mundo[1] & Nelva N. Almanza-Ortega[2]*

[1]Tecnológico Nacional de México, Instituto Tecnológico de Cd. Victoria. Boulevard Emilio Portes Gil No. 1301 C. P. 87010 Ciudad Victoria, Tamaulipas, México.
*jesus.cf@cdvictoria.tecnm.mx, m21380027@cdvictoria.tecnm.mx, adriana.ms@cdvictoria.tecnm.mx, lilia.gm@cdvictoria.tecnm.mx**
[2]Secretaría de Ciencia, Humanidades, Tecnología e Innovación (SECIHTI), nelva.almanza@conahcyt.mx

RESUMEN. El presente artículo propone una arquitectura de software para el desarrollo de una aplicación web de ruteo de vehículos basada en microservicios, además de un marco de trabajo para la construcción dinámica de interfaces de usuario para resolver problemas de ruteo de vehículos, mediante la especificación de parámetros por medio de esquemas basados en el formato JSON. Para corroborar que la arquitectura de software propuesta cumple con los requisitos establecidos, se muestra la interacción entre sus componentes a través de la descripción del flujo de sus procesos. Se concluye que la arquitectura cumple con los requisitos establecidos a la vez que proporciona una mejora en las capacidades de expansión de los componentes de la aplicación, además el marco de trabajo para el modelado de problemas de ruteo reduce significativamente el número de componentes entre el cliente y el servidor.

PALABRAS CLAVE: Problema de Ruteo de Vehículos, Problema de Ruteo de Vehículos Capacitados, Problema de Ruteo de Vehículos con Múltiples Depósitos, OpenStreetMap.

ABSTRACT. This article proposes software architecture for the development of a vehicle routing web application based on microservices, as well as a framework for the dynamic construction of user interfaces to solve vehicle routing problems, by specifying parameters using schemas based on the JSON Schema format. To corroborate that the proposed software architecture meets the established requirements, the interaction between its components is shown through the description of the flow of its processes. It is concluded that architecture meets the established requirements while providing an improvement in the expansion capabilities of the application components, in addition the framework for modeling routing problems significantly reduces the number of components between the client and the server.

KEY WORDS: Vehicle Routing Problem, Capacitated Vehicle Routing Problem, Multiple Depot Vehicle Routing Problem, OpenStreetMap.

1. INTRODUCCIÓN

La mantenibilidad y escalabilidad son factores sumamente importantes cuando se decide desarrollar una aplicación, por lo cual el diseño de la arquitectura del software se vuelve un paso crucial en el proceso de desarrollo; la elección de tecnologías, métodos y prácticas adecuadas da como resultado productos de software que son altamente tolerantes a fallos, fácilmente mantenibles y capaces de escalar según las necesidades de los usuarios y del negocio. Una de las arquitecturas de software más populares para satisfacer estas necesidades es la arquitectura de microservicios, la cual es altamente utilizada por grandes empresas como Google, Facebook, Netflix o Amazon debido a las facilidades que ofrece para agilizar el desarrollo, manejar una gran cantidad de peticiones, aumentar la infraestructura de cómputo y responder eficientemente a las demandas cambiantes del mercado (Contreras *et al.*, 2018). Debido a los beneficios anteriormente mencionados una arquitectura de software basada en microservicios para una aplicación de ruteo de vehículos es necesaria ya que facilita gestionar los recursos del sistema de manera eficiente si la aplicación crece. Actualmente, las arquitecturas propuestas en la literatura para aplicaciones de ruteo de vehículos carecen de la posibilidad de escalar los recursos del sistema eficientemente debido a que están desarrolladas bajo

paradigmas antiguos que no lo permiten. Además, en estas arquitecturas, no se han desarrollado aplicaciones de ruteo con un enfoque multipropósito, es decir, que resuelva diferentes tipos de problemas de ruteo de vehículos. Este enfoque de desarrollo multipropósito trae consigo el reto del incremento entre componentes de la interfaz de usuario y el servidor, ya que, bajo los paradigmas actuales, existe un componente de la interfaz de usuario y un componente del servidor por cada problema de ruteo que se desee incluir en la aplicación. Esto trae consigo un crecimiento mayor en el número de componentes conforme más problemas de ruteo de vehículos sean integrados a la aplicación.

Para afrontar este reto de manera eficiente se propone un marco de trabajo para la construcción del *front-end* de forma dinámica a través de esquemas en formato *JSON* proporcionados por el servidor. Este marco de trabajo permitirá agregar nuevos tipos de problemas de ruteo sin incrementar los componentes del *front-end*. Tomando como base este marco de trabajo, en este artículo se presenta una propuesta de una arquitectura de software para el desarrollo de una aplicación Web de ruteo de vehículos basada en microservicios que resuelve tres problemas, los cuales son:

- Problema de Ruteo de Vehículos (VRP, por sus siglas en inglés). Consiste en encontrar la ruta más corta para visitar un conjunto de clientes con una flota de vehículos.
- Problema de Ruteo de Vehículos con Capacidad Limitada (CVRP, por sus siglas en inglés). Es una extensión del VRP en donde los clientes tienen demandas de producto y los vehículos tienen una capacidad máxima para el producto.
- Problema de Ruteo de Vehículos con Múltiples Depósitos (MDVRP, por sus siglas en inglés). Se asemeja al VRP, sin embargo, los vehículos tienen distintos depósitos desde donde se puede empezar la visita de los clientes.

El artículo se encuentra estructurado de la siguiente forma, en la Sección 2, se presenta un resumen de evolución de las arquitecturas de software. En la Sección 3 se muestran diversos trabajos desarrollados que muestran aplicaciones y herramientas para resolver problemas de ruteo de vehículos. En la Sección 4 se detalla la arquitectura de software propuesta. En la Sección 5 se muestra el flujo de los procesos de la aplicación Web. Finalmente, en la Sección 7 se presentan las conclusiones de este trabajo.

2. ANTECEDENTES

Las arquitecturas de software han ido evolucionando a lo largo del tiempo debido a las cambiantes necesidades de las aplicaciones. En las etapas tempranas del desarrollo de software las aplicaciones eran concebidas como una única unidad que involucraba todo el código necesario para ser desplegado en conjunto. A este estilo de arquitectura de software se le denomina arquitectura monolítica. Este enfoque facilita algunos procesos como el desarrollo de la aplicación, la ejecución de pruebas automatizadas y el despliegue de la aplicación (Barrios Contreras, 2018). Sin embargo, este estilo arquitectural puede ser insuficiente para algunas aplicaciones, ya que presenta grandes limitaciones al momento de escalar la aplicación, es decir, para aumentar los recursos de manera eficiente cuando aumenta la carga de trabajo.

Las aplicaciones que tienen éxito suelen crecer rápidamente, lo que ocasiona que el número de funcionalidades aumente y por consecuencia el tamaño. Esto puede ocasionar demoras en los tiempos de compilación y el aumento de la complejidad de la aplicación, lo cual conduce a tropiezos en el mantenimiento y entendimiento de la aplicación (Barrios Contreras, 2018).

La arquitectura de microservicios es un estilo de arquitectura de software que busca solventar la necesidad de escalabilidad y mantenibilidad de las aplicaciones actuales. A diferencia de la arquitectura monolítica, la arquitectura de microservicios desarrolla todos los componentes del software como pequeñas piezas independientes unas de otras. Este enfoque de independencia entre componentes permite a los microservicios priorizar recursos escasos para los componentes más relevantes. Por ejemplo, los servicios más demandantes pueden ser alojados, con facilidad en servidores más potentes. También posibilita la integración de las tecnologías más adecuadas

para cada servicio, por lo que un servicio puede ser desarrollado en Java, Python, C# o con la tecnología que se requiera. Además, permite aumentar la resiliencia de la aplicación ya que se pueden mantener múltiples instancias de un mismo componente. Esto aumenta la disponibilidad de la aplicación, ya que, en el caso de que un componente fallara, otra instancia del mismo componente podría asumir su funcionalidad de forma automática, garantizando así la continuidad del servicio sin afectar la experiencia del usuario ni comprometer la integridad del sistema. En la Figura 1 se puede observar la diferencia entre la arquitectura monolítica y la de microservicios. Mientras que, en una aplicación monolítica, los diferentes componentes de la aplicación como la interfaz de usuario, la capa de acceso a datos y la capa de negocios interactúan directamente entre sí dentro de un mismo proceso y espacio de ejecución, en la arquitectura de microservicios se muestra cómo los diferentes componentes de la aplicación son separados en pequeñas partes independientes unas de otras.

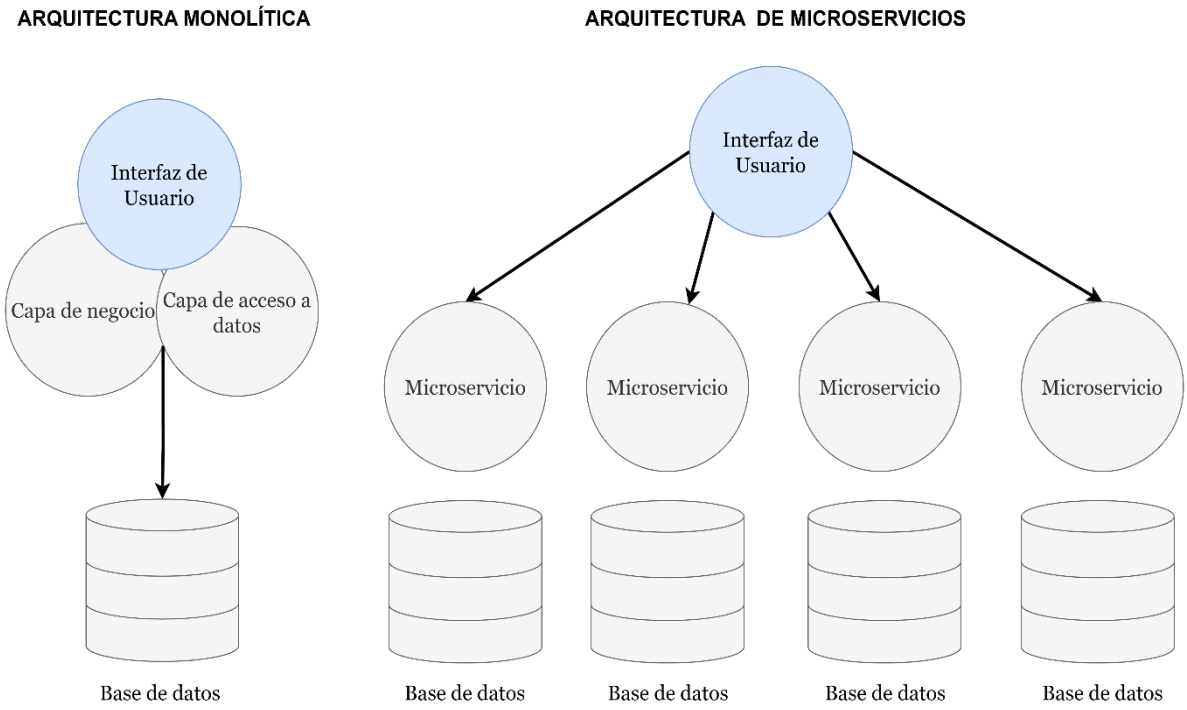

Figura 1. Arquitectura monolítica y de microservicios Barrios (2018).

3. TRABAJO RELACIONADO

Existen diversos trabajos en la literatura sobre la implementación de aplicaciones de ruteo de vehículos. En (Nugroho *et al.*, 2021) se presenta el desarrollo de una aplicación para dispositivos móviles para resolver el VRP. La aplicación se ejecuta bajo un sistema operativo Android. Mediante la integración del API de Mapbox y el API de Google Maps, proporciona una ruta óptima. El autor menciona que esta aplicación resulta útil para los mensajeros de marketing y para la planeación de viajes turísticos.

En (Li *et al.*, 2016) se desarrolla una aplicación para resolver el Problema Dinámico de Ruteo de Vehículos (DVRP, por sus siglas en inglés). La aplicación presenta en tiempo real la demanda de los clientes y la información de tráfico para ajustar las rutas de los vehículos. Cada vez que una demanda es atendida, el sistema vuelve a calcular un nuevo plan de rutas. El autor concluye que el sistema ha probado ser eficiente en el ahorro de costos, la mejora de los efectos visuales de la aplicación y el apoyo a la capacidad de expansión del negocio.

En (Alves *et al.*, 2021) se construye una aplicación para resolver el VRP usando una librería de optimización de código abierto llamada Google OR-Tools la cual sirve para resolver problemas de ruteo de vehículos. Además, se integran servicios de Google como Google Maps y la Distance Matrix API. Como arquitectura de sistema se propone una aplicación cliente-servidor en la nube. Los componentes de la arquitectura propuesta son la interfaz de usuario, los microservicios, los servicios externos como la Distance Matrix API y Google OR-Tools.

Además del desarrollo de aplicaciones, existen trabajos relacionados en donde se muestra interés por la creación de herramientas que faciliten este desarrollo. Tal es el caso de (Peng y Murray, 2020) en donde se desarrolla Veroviz, un paquete de herramientas para la visualización de pruebas de problemas de ruteo en ambientes reales. Por otra parte (Errami *et al.*, 2023) desarrollan una librería para el lenguaje Python la cual provee de una interfaz de programación sencilla para resolver diferentes variantes populares del VRP. (Pessoa *et al.*, 2020) también desarrollan una biblioteca en los lenguajes C++ y Julia la cual genera un modelo genérico para resolver problemas de ruteo; debido a la complejidad de implementar algoritmos iguales para diferentes variantes del VRP.

Las aplicaciones de ruteo de vehículos en la literatura no se han estudiado con un enfoque multipropósito, es decir, que tengan la posibilidad de resolver más de un tipo de problema, lo cual representa un área de oportunidad para el estudio de aplicaciones de ruteo multipropósito.

4. ARQUITECTURA DE SOFTWARE PROPUESTA

La arquitectura de software es una de las partes más importantes del desarrollo de una aplicación. Una buena arquitectura de software permite desarrollar aplicaciones de calidad que sean tolerantes a fallos, fácilmente mantenibles y escalables, estas características son indispensables en aplicaciones modernas donde la carga de trabajo de la aplicación puede crecer en poco tiempo. Como estrategia para añadir estas características, se propone que la aplicación se desarrolle bajo la arquitectura de microservicios. Además, del uso de un marco de trabajo para la comunicación entre los microservicios encargados de la solución de problemas de ruteo. Como estándar para la transmisión de información entre la aplicación se propone el uso de la arquitectura API REST, la cual es un estilo de arquitectura que define un conjunto de reglas para el intercambio de información en una aplicación mediante el protocolo HTTP, este estilo de arquitectura facilita la identificación y operación de los recursos de una aplicación (Surwase, 2016).

La interfaz de usuario es la parte que permite interactuar a los usuarios con la aplicación, ésta, puede ser desarrollada como una aplicación de escritorio, una aplicación para dispositivos móviles o una aplicación web. Actualmente, el enfoque más usado para desarrollo de interfaces de usuario es a través de la web, ya que este tipo de aplicaciones ofrece ventajas, como la facilidad de obtención, porque solo necesitamos de un navegador para poder usarla. Además, permiten llevar los cambios a los usuarios de manera rápida y que la aplicación sea accesible por medio de diferentes dispositivos. Por estos motivos se decide que la construcción de la interfaz de usuario para esta arquitectura sea una aplicación web.

Para el diseño de una arquitectura, también debemos tener en cuenta los requisitos del software, ya que estos dictan la construcción de todos los servicios disponibles por la aplicación. Los requisitos definidos para la construcción de la presente arquitectura son los siguientes:

1. La aplicación debe permitir al usuario ingresar, proporcionando una dirección de correo electrónico y contraseña.
2. La aplicación debe permitir al usuario registrarse con una dirección de correo electrónico y contraseña, el correo electrónico debe de ser confirmado a través de un mensaje de correo electrónico, para habilitar su uso.
3. La aplicación debe de permitir resolver el VRP. El usuario podrá visualizar un mapa del mundo, donde debe indicar el punto de partida, el número de vehículos de la flota y los puntos de destino a visitar. Una vez ingresados los datos deberá de presionar un botón y las rutas deberán de ser marcadas en pantalla.

4. La aplicación debe de permitir resolver el CVRP. El usuario podrá visualizar un mapa del mundo, donde debe indicar el punto de partida, el número de vehículos de la flota, los puntos de destino a visitar, la demanda de los clientes y la capacidad de la flota. Una vez ingresados los datos deberá de presionar un botón y las rutas deberán de ser marcadas en pantalla.
5. La aplicación debe de permitir resolver el MDVRP. El usuario podrá visualizar un mapa el mundo, donde debe indicar los depósitos disponibles, la capacidad de los depósitos, el número de vehículos por depósito, la capacidad de los vehículos, la ubicación de los clientes a satisfacer y la demanda de los clientes. Una vez ingresados los datos deberá de presionar un botón y las rutas deberán de ser marcadas en pantalla.
6. La aplicación debe permitir guardar, eliminar o editar un problema de ruteo en cualquier momento.

Una vez definidos los requisitos se establecen los diferentes servicios que tendrá la aplicación. La arquitectura propuesta considera 6 microservicios los cuales son descritos en las secciones 4.1, 4.2, 4.3 y 4.4 (servicios: VRP, CVRP y MDVRP).

4.1 Microservicio de usuarios

El microservicio de usuarios es el responsable de mantener la colección de usuarios de la aplicación. Para la estructura de la base de datos de este microservicio se debe tener en cuenta que los usuarios se registran mediante una contraseña y un correo electrónico, además que su cuenta no puede ser usada si no ha sido confirmada. Por lo cual es necesario contemplar los campos “email”, para el correo electrónico, “password”, para la contraseña y “verified”, para indicar si el usuario ha confirmado su cuenta.

Los recursos necesarios para la implementación de este microservicio se muestran en la Tabla 1.

Tabla 1. Definición de recursos para la implementación del microservicio de usuarios**.**

Ruta	Método	Descripción
/users/{email-del-usuario}	GET	A través de esta ruta se pueden obtener los datos almacenados de un usuario, proporcionando su correo electrónico en la url. Esta ruta es privada y está destinada exclusivamente para ser utilizada entre microservicios. Por lo tanto, está protegida por una API key, la cual es un mecanismo popular para autorización de operaciones en aplicaciones.
/users/confirm-account/{email-del-usuario }/{token-de-confirmación}	GET	A través de esta ruta, un usuario puede confirmar su cuenta utilizando el correo electrónico y el token de confirmación de cuenta. Al crear la cuenta, el servicio de usuarios enviará un mensaje de correo electrónico con un enlace para esté servicio, después de esta acción la cuenta se confirmará y podrá ser usada.
/users	POST	A través de esta ruta, el cliente puede registrar a un usuario en el sistema. A esta ruta se deberá enviar una petición que contenga el correo electrónico y contraseña, posteriormente se enviará un mensaje de correo electrónico para confirmar la cuenta.
/users/{email-del-usuario }	DELETE	A través de esta ruta, se puede borrar un usuario del sistema. Mediante una petición, con el correo electrónico del usuario, se debe de verificar que el usuario que hace la petición sea el mismo que quiere borrar el correo electrónico mencionado.

4.2 Microservicio de autenticación

La autenticación es una característica obligatoria y sumamente importante en cualquier aplicación que maneje datos sensibles. Mediante la autenticación se debe asegurar la identidad del usuario que interactúa con la aplicación. De esta manera se pueden restringir operaciones que el usuario no está autorizado para realizar.

La autenticación en aplicaciones es frecuentemente implementada a través del uso de *tokens.* Un token es un recurso digital que es obtenido cuando un usuario se autentica en una aplicación. Con éste, el usuario puede validar su identidad con la aplicación para cada acción que realice. La implementación de este sistema en aplicaciones monolíticas es sencilla y práctica debido a la centralización de la aplicación, ya que el servidor siempre tiene a la mano la información del usuario para realizar cualquier validación que se requiera. En el ámbito de microservicios esto es diferente, debido a que cada parte del sistema es independiente y no tiene un acceso directo a los demás recursos. Los JSON Web Tokens (JWT) (Otka, 2024) se vuelven una buena alternativa para el transporte de información entre microservicios. Un JWT es un estándar de seguridad para la transmisión segura de información, la autenticación y la autorización. Este *token* está conformado por tres partes: el *header,* el cual típicamente incluye el tipo de token y el algoritmo de firma usado, el *payload,* el cual contiene la información de la entidad que se autentica y la *signature,* la cual verifica que los datos contenidos en el *header* y el *payload* no han sido modificados. Este mecanismo es definido para la implementación de la autenticación de la aplicación. Los recursos necesarios para la implementación de este microservicio se muestran en la Tabla 2.

Tabla 2. Definición de recursos para la implementación del microservicio de autenticación.

Ruta	Método	Descripción
/auth	GET	Esta ruta permite a los usuarios autenticarse mediante los siguientes pasos: 1. Enviar una petición con el correo electrónico y contraseña del usuario. 2. El microservicio consulta con el servicio Users si los datos proporcionados son válidos. 3. Se genera un token, se almacena en la base de datos y se envía al usuario.
/auth	DELETE	A través de esta ruta el cliente podrá desautenticarse a través de los siguientes pasos: 1. Enviar una petición con el *token* de sesión. 2. El microservicio borra de su base de datos el token de sesión. 3. Se envía una respuesta al cliente si la operación fue exitosa.
/auth/session	GET	A través de esta ruta los demás microservicios podrán consultar si un usuario se encuentra autenticado a través de una petición que proporcione el token de acceso. Esta ruta es privada y exclusiva para su uso entre microservicios y estará protegida con un API key, el cual es un mecanismo para la autorización de uso de servicios en una API.

4.3 Microservicio de problemas de ruteo

El microservicio de problemas de ruteo es el responsable de almacenar la colección de los problemas guardados por los usuarios. Debido a la variedad de problemas que la aplicación debe de manejar, se recomienda guardar la información en formatos que no dicten una estructura rígida, como formatos JSON contenidos en un valor en el caso de gestores SQL o en su defecto usar bases de datos basadas en documentos. Los recursos necesarios para la implementación de este microservicio se muestran en la Tabla 3.

Tabla 3. Definición de recursos para la implementación del microservicio de problemas de ruteo.

Ruta	Método	Descripción
/routing-problems	GET	A través de esta acción el cliente podrá listar los problemas de ruteo guardados.
/routing-problems/{identificador-del-problema-de-ruteo}	GET	A través de esta acción el cliente podrá obtener un problema de ruteo específico.
/routing-problems	POST	A través de esta acción el cliente podrá publicar un problema de ruteo.
/routing-problems	PUT	A través de esta acción el cliente podrá modificar la información de un problema de ruteo.
/routing-problems/{id-del-problema-de-ruteo}	DELETE	A través de esta acción el cliente podrá borrar un problema de ruteo guardado proporcionando el identificador mediante la url.

4.4 Microservicios de resolución de problemas de ruteo: VRP, CVRP y MDVRP

Los microservicios de resolución de problemas de ruteo se refieren a los microservicios que solucionan los problemas de ruteo VRP, CVRP y MDVRP. Estos servicios son la esencia de la aplicación porque cumplen el propósito principal de resolver problemas de ruteo. Los recursos necesarios para implementar cualquier servicio de este tipo se muestran en la Tabla 4.

Tabla 4. Definición de recursos para la implementación del servicio de resolución de problemas de ruteo.

Ruta	Método	Descripción
/	GET	Mediante esta acción el cliente puede obtener la definición del problema de ruteo solicitado, el cual será proporcionado a través de un esquema JSON
/	POST	Mediante esta acción el cliente puede resolver un problema de ruteo a través del envío de información solicitada mediante el esquema. El servicio debe de leer los parámetros enviados y consultar con el servicio de información geográfica los datos necesarios para devolver el orden correcto de las rutas.

Para este tipo de servicios solo se pueden realizar las dos posibles acciones que se muestran en la Tabla 4. La acción del método GET está relacionada con la definición de un problema de ruteo, que incluye todos los posibles parámetros y configuraciones que deben de ser especificados para su resolución. Por ejemplo, para la resolución del VRP, los parámetros necesarios son el número de vehículos de la flota, el punto de inicio y los clientes a satisfacer. Este marco de trabajo propuesto permitirá a la aplicación del cliente mostrar los componentes necesarios para introducir los datos para resolver el problema en cuestión. Bajo marcos de trabajo tradicionales, la creación de nuevos servicios resultaría en una gran cantidad de componentes de la aplicación del cliente y del servidor, es decir, se tendrían *n* número de servicios de resolución de problemas de ruteo y *n* pantallas para la captura de datos. Mientras que, bajo este marco de trabajo propuesto, los componentes del cliente se pueden reducir a un solo componente que interprete la definición del esquema; el cual contiene la definición del problema de ruteo. Con lo anterior, si queremos agregar un nuevo problema a la aplicación no es necesario crear otro componente de interfaz,

únicamente se agregaría un microservicio. Además, esta novedad permite agregar nuevos datos con relativa facilidad y ayuda a la formulación de nuevos problemas de ruteo, debido a que el problema se formula solamente desde el servidor, ahorrando esfuerzo en la creación de nuevas pantallas. En la Figura 2 se muestra la definición del VRP, mediante la utilización de un esquema basado en el estándar JSON Schema, el cual define un conjunto de características para la especificación de documentos.

```
{
  "title": "Problema de Ruteo de Vehículos",
  "description": "El problema de ruteo de vehículos (VRP) es un problema de
    optimización combinatoria donde el objetivo esdeterminar las rutas más
    eficientes para que una flota de vehículos entregue bienes o servicios a un
    conjunto de clientes, comenzando y terminando en un depósito central. El
    objetivo es minimizar el costo total, como la distancia recorrida, el tiempo
    o el consumo de combustible, y al mismo tiempo satisfacer limitaciones como
    la capacidad del vehículo, los plazos de entrega y la demanda de los
    clientes.",
  "locationTypes": [
    {
      "key": "start",
      "title": "Depósito",
      "description": "La localización desde donde parte la flota de vehículos.",
      "max": 1,
      "min": 1,
      "props": [
        {
          "key": "number_of_vehicles",
          "type": "integer",
          "min": 1,
          "required": true,
          "title": "Tamaño de la flota de vehículos",
          "description": "El número de vehículos de la flota"
        }
      ]
    },
    {
      "key": "client",
      "title": "Cliente",
      "description": "Los destinos a visitar por la flota de vehículos.",
      "min": 1
    }
  ],
  "settings": []
}
```

Figura 2. Definición de los parámetros del VRP mediante un esquema.

Los parámetros de un problema de ruteo se pueden englobar dentro de dos grandes grupos, las configuraciones y las localizaciones. Las configuraciones, denotadas como “settings” en la Figura 2, se refieren a las configuraciones generales para la resolución del problema que se esté abordando. Dentro de ellas se pueden especificar aspectos como, la capacidad de los vehículos de la flota de vehículos (en caso de que la capacidad de la flota sea uniforme, esto puede variar según el problema), el tiempo máximo de búsqueda, el algoritmo a implementar, entre otras configuraciones. Las localizaciones, denotadas como “locations”, en la Figura 2, representan los puntos de un problema de ruteo, sean clientes, puntos de partida, puntos de fin, depósitos, entre otras cosas. Este marco de trabajo ofrece flexibilidad en la implementación, ya que los esquemas permiten definir múltiples datos y validaciones.

4.5 Selección de tecnologías

En este apartado hablaremos de las tecnologías usadas para la implementación de la arquitectura propuesta. La selección de tecnologías puede ser un tema complejo de abordar debido a la amplia gama de tecnologías disponibles que se tienen en la actualidad. En las secciones 4.1, 4.2 y 4.3 se habla de la estructura y recursos que deben de tener los microservicios de la arquitectura. Estos microservicios se pueden implementar con diferentes tecnologías, dependiendo de la naturaleza del proyecto, el equipo con el que se cuente y los recursos disponibles en general.

Actualmente en el mundo del desarrollo web existen diferentes tecnologías disponibles para el desarrollo de interfaces. Para la construcción de la interfaz de usuario, se eligió la tecnología React (React, 2024) debido a las ventajas que ofrece, ya que:

- Mejora el rendimiento de la aplicación.
- Permite, la creación de componentes reutilizables.
- Ofrece, una gran cantidad de documentación, bibliotecas y soporte por parte de la comunidad.
- Es capaz de afrontar los retos de desarrollar una interfaz compleja.

Para la implementación de microservicios también existen diferentes tecnologías. Se eligió Fastapi por las ventajas y características que ofrece, ya que:

- Proporciona un alto rendimiento.
- Mejora en la velocidad de implementación.
- Genera documentación automática.

En la arquitectura propuesta, hay 4 microservicios que necesitan de una base de datos para el almacenamiento de información. Para el microservicio de usuarios y el microservicio de problemas de ruteo, se va a usar una base de datos MongoDB. Esta base de datos está basada en documentos y permite almacenar datos no estructurados. Se eligió MongoDB por las ventajas que ofrece:

- Mayor velocidad respecto a otras tecnologías.
- Facilita el desarrollo y modificación de la aplicación ya que no se tiene que alterar el esquema establecido para agregar nuevos campos.

Para el servicio de autenticación, se va a usar una base de datos Redis. La cual es una base de datos veloz que almacena los datos en formatos clave-valor. Esto es conveniente debido a que el sistema de autenticación solo guardará los tokens de sesión, por lo que no requiere de un sistema gestor demasiado robusto.

Un software para la solución de problemas de ruteo necesita de información confiable y en tiempo real para la formulación y resolución de estos problemas. Una alternativa para esta necesidad es el servicio de Open Route Service (Openrouteservice, 2024), el cual provee de una serie de APIs para el cálculo de distancias entre ubicaciones. Existen otros servicios para solventar esta necesidad, sin embargo, se optó por Open Route Service por ser un servicio confiable y totalmente gratis.

Uno de los retos que presenta una aplicación basada en la arquitectura de microservicios es el manejo de la diversidad de tecnología, ya que diferentes servicios pueden requerir de diferentes versiones del lenguaje o tecnologías y esto puede ocasionar errores al momento del despliegue de las aplicaciones. Docker es una tecnología para la administración de contenedores, un contenedor es una pieza aislada de software que permite ejecutar aplicaciones sin tener conflictos de requerimientos entre ellas. Esta tecnología es elegida con el objetivo de separar la implementación de los microservicios.

En la Figura 3 se puede observar el resultado de la definición de los componentes y tecnologías dentro de ella se encuentran los microservicios de resolución de problemas de ruteo (VRP, CVRP y MDVRP), el microservicio de usuarios (Users) y el microservicio de autenticación (Auth), así como sus respectivas bases de datos. También se muestra el servidor web de la aplicación (Web Server) y el servicio de información geográfica Open Route (Open Route API) (Openrouteservice, 2024).

5. FLUJO DE LOS PROCESOS DE LA APLICACIÓN

En esta sección se detallan los procesos de interacción entre el cliente (aplicación web de react), el usuario y los microservicios, lo cual sirve para demostrar que la arquitectura puede ser implementada y está alineada a los requisitos previamente establecidos. El primer paso para comenzar a usar la aplicación es el proceso de registro de usuario. En este proceso el usuario llena los datos de registro a través de la interfaz de usuario (Cliente), posteriormente se envía la solicitud al servidor y el servidor envía un mensaje de correo electrónico para activar la cuenta al usuario, después el usuario visita el enlace contenido en el correo y finalmente el servidor confirma la cuenta para poder ser utilizada. Este proceso se puede observar en la Figura 4.

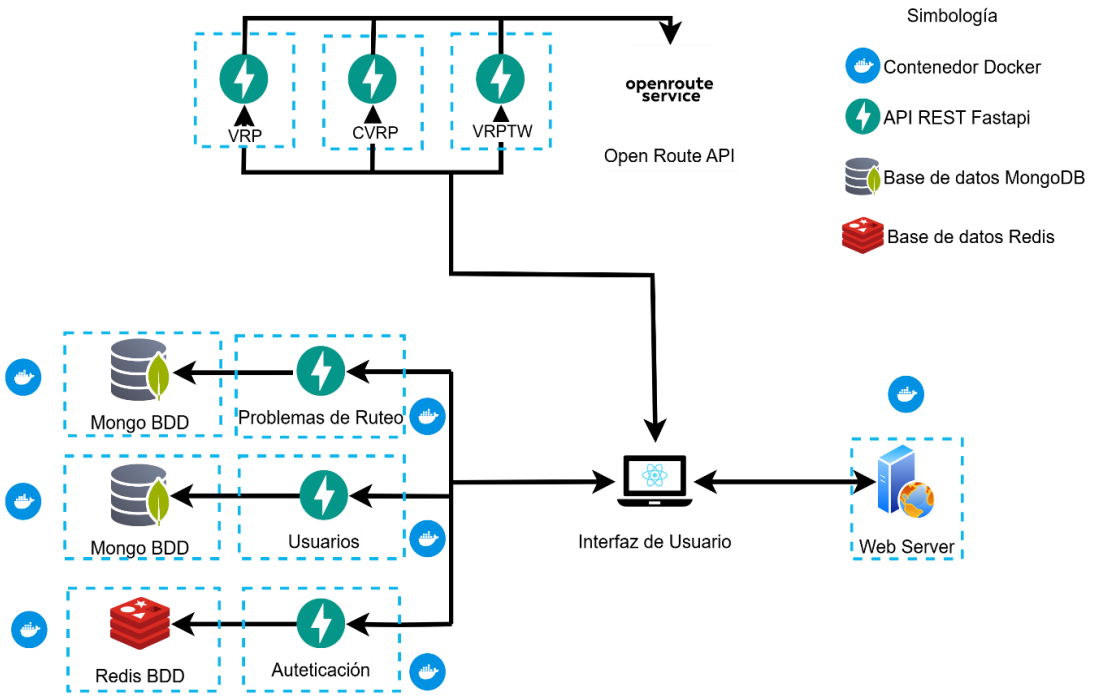

Figura 3. Diagrama de la arquitectura de software. Los logos Redis (2025), React (2024), Docker (2025), FastAPI (2025) y Openrouteservice (2025) mostrados son propiedad de sus respectivos dueños. Se utilizan con fines exclusivamente académicos y sin fines comerciales, el resto de las imágenes son de elaboración propia.

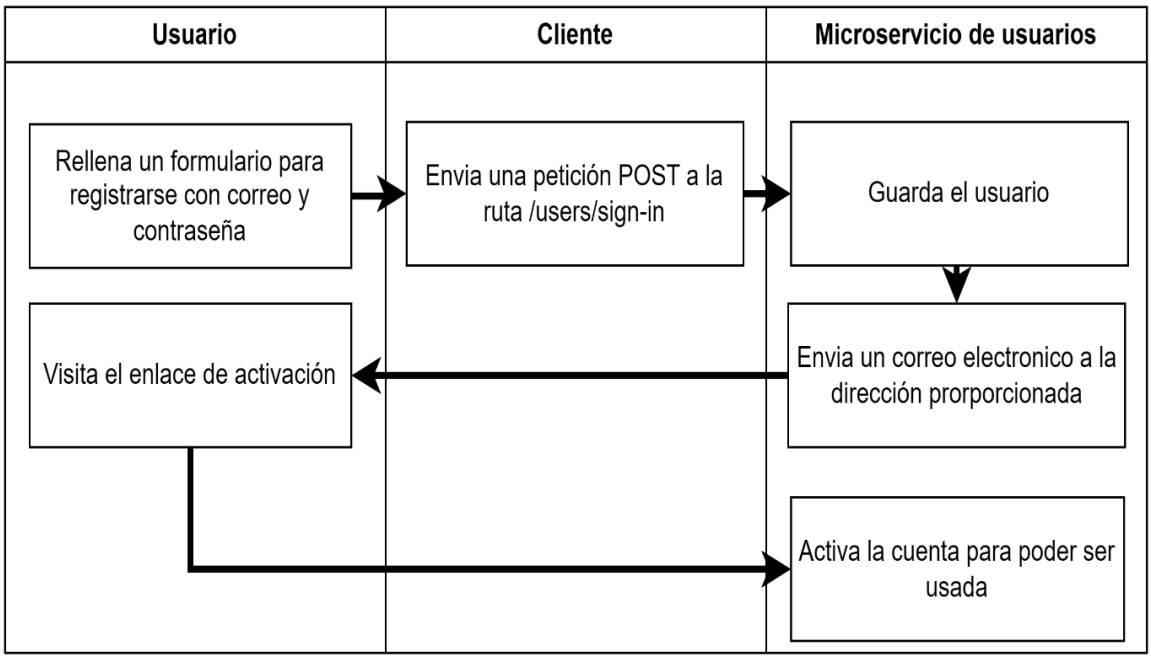

Figura 4. Diagrama de flujo del proceso de registro de usuarios.

Después de la creación y confirmación de la cuenta, el usuario debe autenticarse para poder usar la aplicación, ya que ningún usuario sin cuenta tiene permitido su uso. Primero el usuario debe llenar el formulario de inicio de sesión, después el microservicio de autenticación consulta con el microservicio de usuarios los datos del usuario, posteriormente comprueba las credenciales enviadas, genera y guarda el token de acceso y lo envía al usuario, finalmente el usuario guarda el token para poder autenticarse con los otros microservicios. El proceso de autenticación se muestra en la Figura 5.

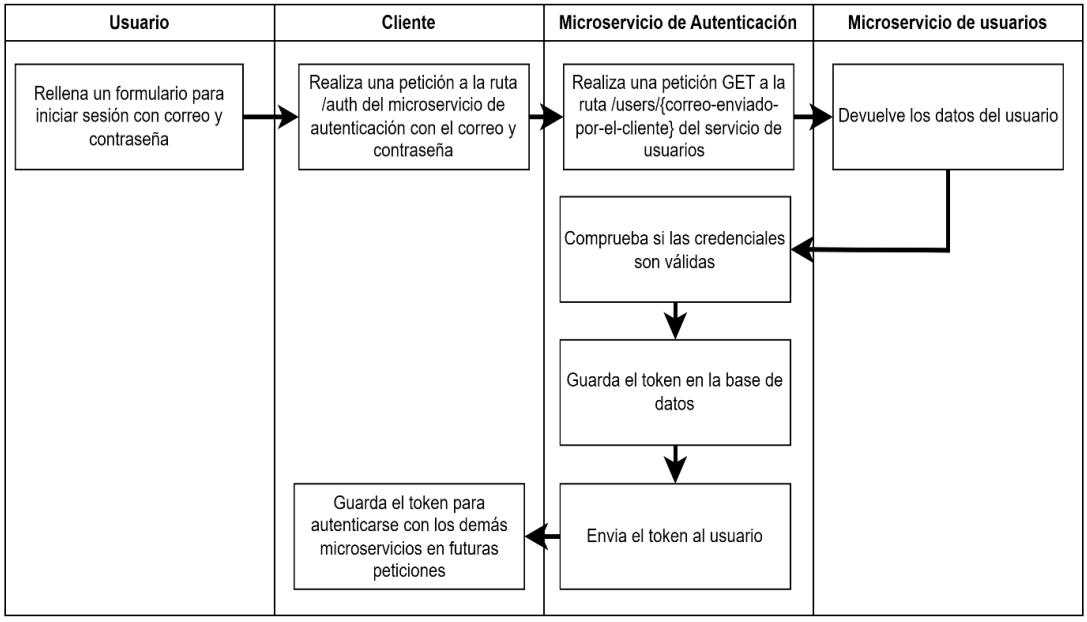

Figura 5. Diagrama de flujo del proceso de autenticación de usuario.

Bajo este modelo de autenticación, los demás microservicios deberán de consultar el microservicio de autenticación para corroborar la identidad del usuario que realiza la petición, el microservicio responderá si el token es válido y continuará con la petición. El comportamiento de los microservicios para la autenticación se muestra en la Figura 6.

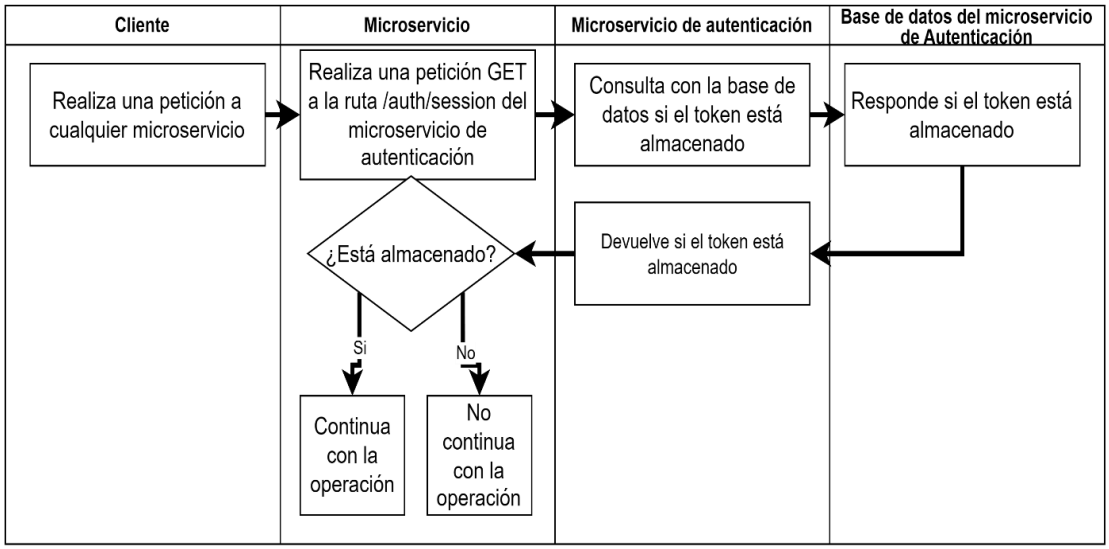

Figura 6. Diagrama de flujo del proceso de autenticación en microservicios.

Además de validar la información del token recibida por el servicio de autenticación, los microservicios deben de implementar sus propias reglas de seguridad, ya que este método solo permite validar la información del solicitante. Por ejemplo, el microservicio de usuarios solo debe permitir borrar, eliminar y editar los datos del usuario que los solicita, no de los demás. Esto también sucede con el servicio de problemas de ruteo, ya que solo el usuario dueño de los problemas debe de poder eliminar, editar y crear problemas de ruteo en su colección.

Finalmente, el usuario podrá usar la aplicación, permitiéndole formular problemas de ruteo, así como guardarlos para su futura consulta, eliminarlos o editarlos. Dentro del proceso de modelado de un problema de ruteo intervienen varios procesos como la construcción de la interfaz, primeramente, el usuario debe de ingresar a la pantalla de edición de problemas de ruteo, después la aplicación identifica el tipo de problema seleccionado, consulta el esquema y construye los elementos de la interfaz de usuario, posteriormente el usuario ingresa los parámetros y finalmente se muestra en pantalla el orden de recorrido de las rutas. Este proceso se puede observar en la Figura 7.

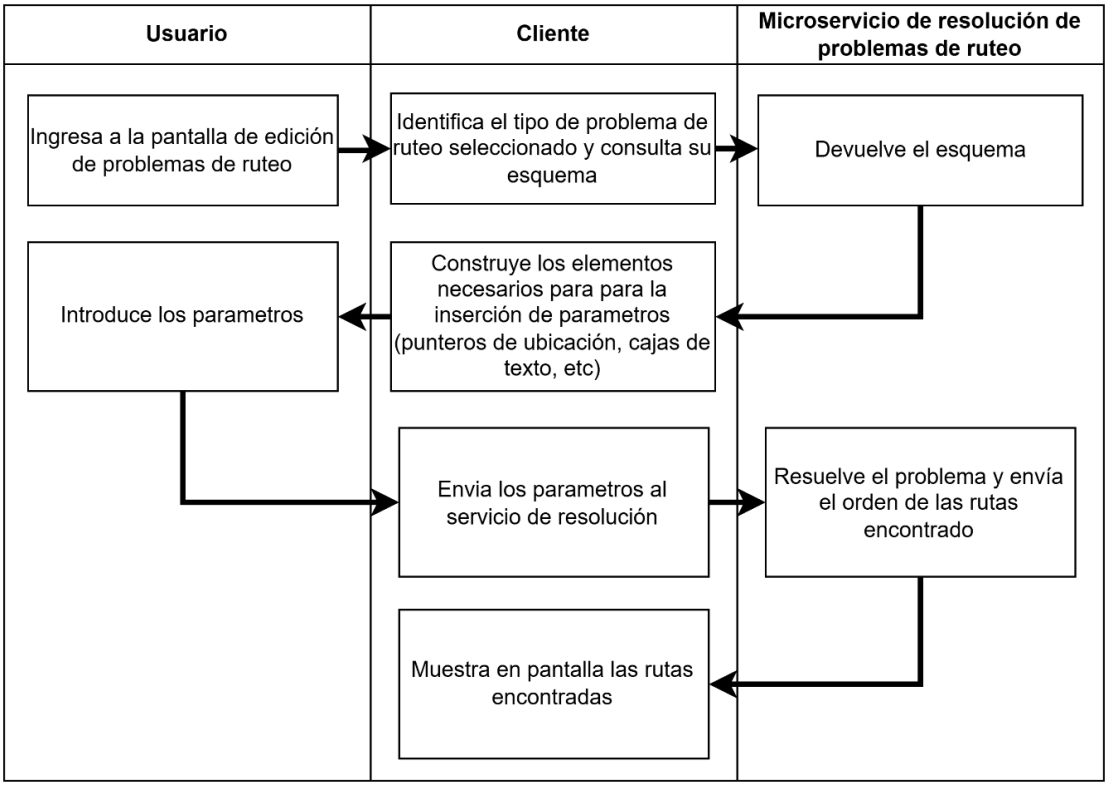

Figura 7. Diagrama de flujo del proceso de resolución un problema de ruteo mediante la aplicación.

6. CONCLUSIONES

En este artículo se ha propuesto una arquitectura de microservicios para una aplicación de ruteo de vehículos. Esta arquitectura sirve como base para la implementación de aplicaciones web que tengan una alta demanda y requieran soportar una gran cantidad de usuarios. El diseño basado en microservicios mejora la escalabilidad y mantenibilidad del producto. Además, el marco de trabajo propuesto para la creación de nuevos microservicios de resolución de problemas de ruteo de vehículos es una nueva estrategia para manejar el crecimiento de componentes de la interfaz de usuario, la cual facilita la implementación de nuevos microservicios de resolución de problemas de ruteo de vehículos mediante la construcción de los elementos de la interfaz mediante esquemas proporcionados por el servidor.

7. LITERATURA CITADA

Alves F., Pacheco F., Rocha A. M., Pereira A. I. y Leitão P. 2021. Solving a Logistics System for Vehicle Routing Problem Using an Open-Source Tool. *Computational Science and Its Applications*, 12953: 397-412.

Barrios D. A. 2018. Arquitectura de Microservicios. *Revista Tecnología Investigación y Academia. 6*(1): 37-46.

Docker, Inc. (9 de abril de 2025). *Docker: Empowering developers to build and run applications anywhere*. Obtenido de https://www.docker.com/

Errami N., Queiroga E., Sadykov R. y Uchoa E. 2023. *VRPSolverEasy: a Python library for the exact solution of a rich vehicle routing problem.* INFORMS Journal on Computing, 36(4): 956-965.

FastAPI (9 de abril de 2025). Obtenido de https://fastapi.tiangolo.com/

Li R., Qi M., Cheng C. y Lai W. 2016. Design of Dynamic Vehicle Routing System Based on Online Map Service. *13th International Conference on Service Systems and Service Management (ICSSSM)*. 1: 1-5.

Nugroho A., Izzah A. y Eliyen K. 2021. Mobile Application Development to Solve Vehicle Routing Problems in Marketing or Tour Trip Planning. *Jurnal RESTI Rekayasa Sistem dan Teknologi Informasi, 5*(2): 27-33.

Openrouteservice. (2025). *OpenRouteService* [API de código abierto]. Heidelberg Institute for Geoinformation Technology. https://openrouteservice.org/

Peng L., y Murray C. 2020. VeRoViz: A Vehicle Routing Visualization Toolkit. *INFORMS, 34*(4): 1841-2382.

Pessoa A., Sadykov R., Uchoa E. y Vanderbeck F. 2020. A generic exact solver for vehicle routing and related problems. Mathematical Programming, 183: 483-523.

React. (9 de mayo de 2024). Obtenido de https://react.dev

Redis. (9 de abril de 2025). *Redis: A powerful in-memory data structure store*. Obtenido de https://redis.io/

Surwase V. (2016). REST API Modeling Languages - A Developer's Perspective. IJSTE - International Journal of Science Technology & Engineering. 2: 634-63.