

Arquitecturas Web

Agenda de Viajes

Informe



Facultad de Ciencias Exactas, Universidad Nacional del Centro de la Provincia de Buenos Aires. Tandil, 24/11/2020.

Integrantes:

Belen Enemark belenenmark@gmail.com
Juan Cruz Deccechis juandeccechis@gmail.com
Mateo Zarrabeitia mateo.zarrabeitia96@gmail.com

Índice de contenido

1. Contexto	3
2. Decisiones	4
2.1 Arquitectura	4
3. Resultado	8

1. Contexto

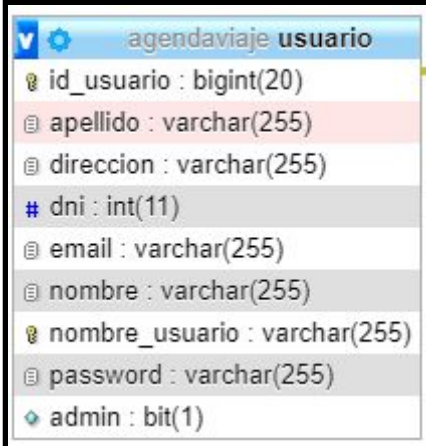
En el presente informe se detalla el trabajo realizado por el grupo n°2 como entrega final a la cátedra Arquitecturas web. Entre los puntos a destacar se encuentran las decisiones arquitectónicas para la solución, los framework y tecnologías utilizados, y las distintas vistas de la solución final.

El objetivo del trabajo es desarrollar una agenda de viajes, la cual permita observar los viajes planeados a futuro y los ya realizados por un usuario. Se requiere acceder a cierta información de los viajes del usuario, junto con información de los planes incluidos (ya sea un medio de transporte o un tipo de alojamiento). Además, la información puede ser ingresada completando un formulario, o por medio de un archivo de texto. Por último, se solicitan distintos tipos de búsquedas o filtros de la información.

2. Decisiones

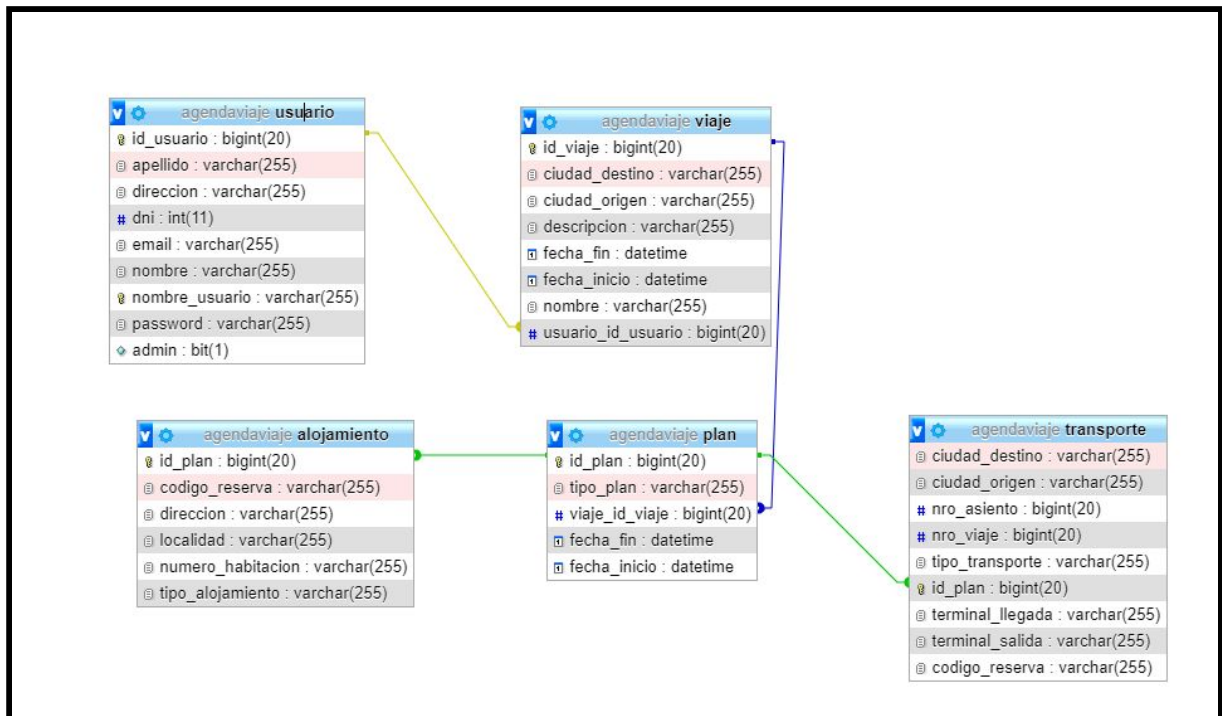
Para resolver los requerimientos del enunciado se plantea una solución con micro-servicios, en la que se separe la arquitectura del usuario de la de los viajes y planes realizados por cada usuario (sección que planteamos resolver con una estructura similar a un “ToDoList”).

Dentro de la funcionalidad de los usuarios se realiza una distinción de roles entre los perfiles de usuario y admin (para de esta manera poder cumplir con los requerimientos para los usuarios).



agendaviaje usuario	
id_usuario	bigint(20)
apellido	varchar(255)
direccion	varchar(255)
dni	int(11)
email	varchar(255)
nombre	varchar(255)
nombre_usuario	varchar(255)
password	varchar(255)
admin	bit(1)

Dentro de la funcionalidad de viajes se resuelve implementar la lógica para que cada usuario pueda ingresar sus viajes (pasados o futuros indistintamente), y en la que cada viaje tenga la información característica solicitada en el enunciado y la lista de planes que incluye. A su vez, cada plan puede ser de alojamiento o de transporte (cada tipo de plan almacena distinta información, siguiendo las necesidades plasmadas en el enunciado). Finalmente, se decidió determinar la zona geográfica como la ciudad destino del viaje (requisito solicitado en el enunciado), para obtener información sobre los sitios más populares como destinos turísticos, donde focalizar la venta de productos de software orientados al turismo.

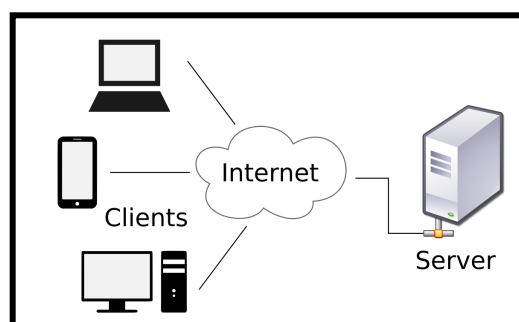


2.1 Arquitectura

El diseño arquitectónico de la solución resulta una arquitectura cliente-servidor, con un tercer nivel para la BD, dividida en micro-servicios para mayor portabilidad.

El patrón arquitectónico cliente-servidor genera las bases para gestionar el almacenamiento de datos, el estilo de la comunicación, y la distribución de responsabilidades entre los demandantes de recursos o servicios (clientes), y sus proveedores (servidores).

En estas arquitecturas la capacidad de procesamiento está repartida entre los clientes y los servidores, aunque son más importantes las ventajas de tipo organizativo debidas a la centralización de la gestión de la información y la separación de responsabilidades, lo que facilita y clarifica el diseño del sistema. Este estilo arquitectónico beneficia por sobre todo a los atributos de calidad: Disponibilidad, Escalabilidad y Portabilidad.



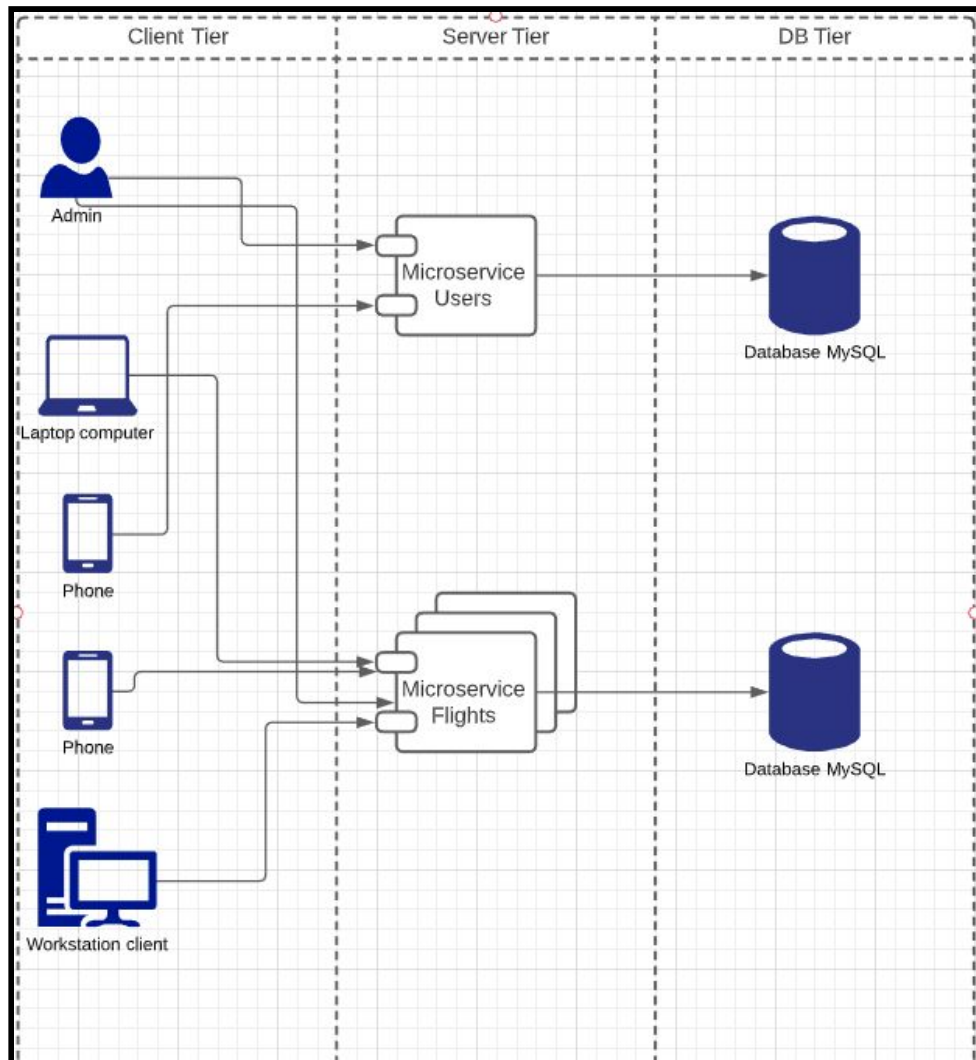


Diagrama de Componentes y Conectores

La comunicación entre clientes y servidores se realiza por REST, bajo el protocolo HTTP, a los servicios expuestos por el servidor por medio de Jersey. Y la comunicación entre servidores y las BD se realiza por Hibernate.

Los detalles del nivel del servidor se pueden observar en la siguiente imagen provista por la cátedra:

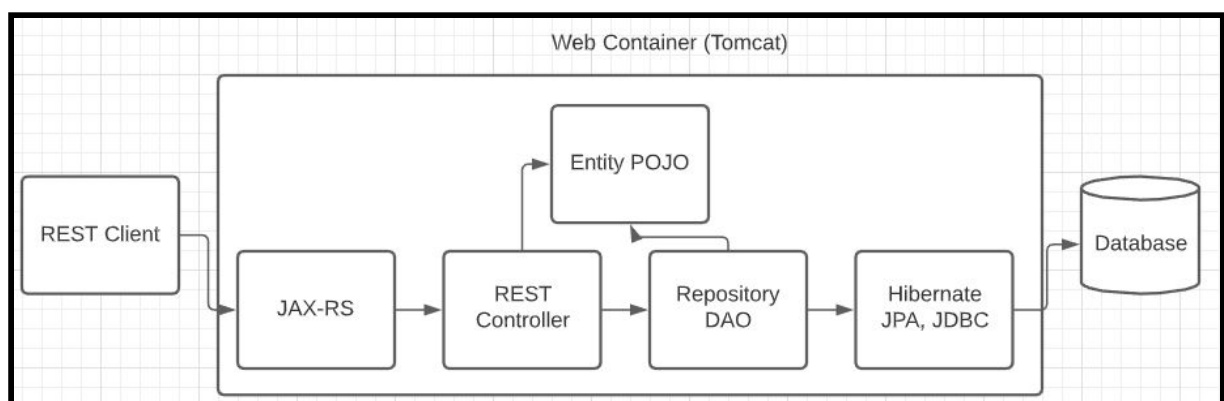


Diagrama de módulos

Dentro de los framework a utilizar para cada micro-servicio, el enunciado provisto por la cátedra especifica que la BD sea MySQL, y el backend se desarrolle sobre SpringBoot. Continuando, la comunicación entre cliente y servidor debe realizarse mediante servicios REST (comunicación sobre protocolo HTTP), deben realizarse test unitarios implementados en JUnit, y documentar los servicios REST con Swagger.

Para el desarrollo del backend del sistema se utilizó un framework compatible con la tecnología JPA, de manera que facilite llevar a cabo la lógica del negocio, manteniendo la relación con el modelo de almacenamiento de datos. Se decidió utilizar el framework Spring por diversos motivos, dentro de los que se destacan: es muy utilizado en la construcción de aplicaciones, es de código abierto, permite el completo desarrollo de herramientas, desde el frontend hasta el backend, y utiliza la plataforma Java.

Para el manejo de dependencias que brinden soporte a las distintas necesidades, y sean compatibles, se utilizó el gestor de dependencias Maven. Maven utiliza un Project Object Model (POM) para describir el proyecto de software a construir, sus dependencias de otros módulos y componentes externos, junto con el control de las versiones y el orden de construcción de los elementos. El uso de un gestor de dependencias mejora la flexibilidad y portabilidad de la aplicación.

A la hora de llevar a cabo la interacción entre la lógica del negocio y el modelo de datos existen principalmente dos tecnologías de software basadas en la persistencia de datos, estas son JPA (Java Persistence API) y JDBC (Java Database Connectivity). La diferencia principal de estas tecnologías es que JPA ofrece un mapeo objeto-relacional (ORM - Object Relational Mapping), mientras que JDBC ofrece un canal directo para realizar operaciones en la base de datos a través del dialecto SQL. Para el desarrollo se decidió utilizar JPA debido a su característica principal nombrada que nos ofrece no perder la ventaja de un diseño orientado a objetos al interactuar con una base de datos. Este mapeo objeto-relacional permite abstraer los beans de las entidades en la lógica del negocio, de las tablas en el modelo de datos. De esta forma, las clases de java se mapean con tablas en la base de datos, permitiendo así trabajar siempre a nivel de objetos.

Dentro de las implementaciones de JPA, se utilizó Hibernate. Hibernate es una herramienta de mapeo objeto-relacional (ORM) para la plataforma Java que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML) o anotaciones en los beans de las entidades que permiten establecer estas relaciones. Esta herramienta, como se nombró anteriormente, implementa como parte de su código la especificación JPA.

El controlador es un componente desarrollado en el lenguaje Java, por lo que para permitir al servidor recibir peticiones web se requiere hacer uso de los servlets. Un servlet es un objeto de java que pertenece a una clase que extiende de `javax.servlet.http.HttpServlet`, y que permite desarrollar páginas web dinámicas al admitir peticiones a través del protocolo HTTP. De forma muy resumida, los servlets reciben peticiones desde un navegador web, las procesan y devuelven una respuesta al navegador; para nuestro caso en formato JSON.

Interfaz de usuario

HTML



CSS



JS



Angular



Comunicación



http://



Lógica de negocio



maven



HIBERNATE

Persistencia de los datos



Control de versiones



Testing

JUnit



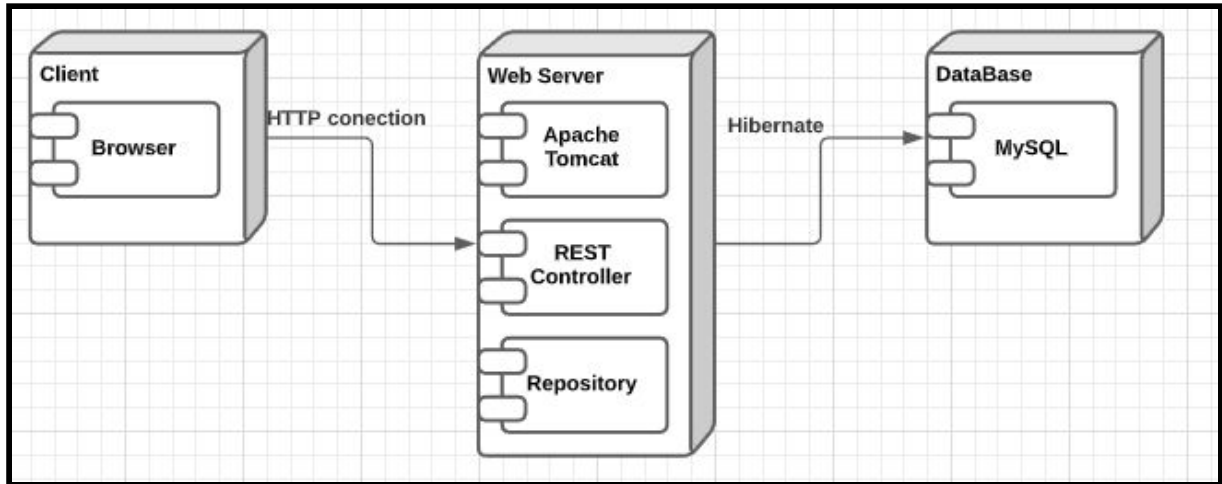
Swagger



POSTMAN

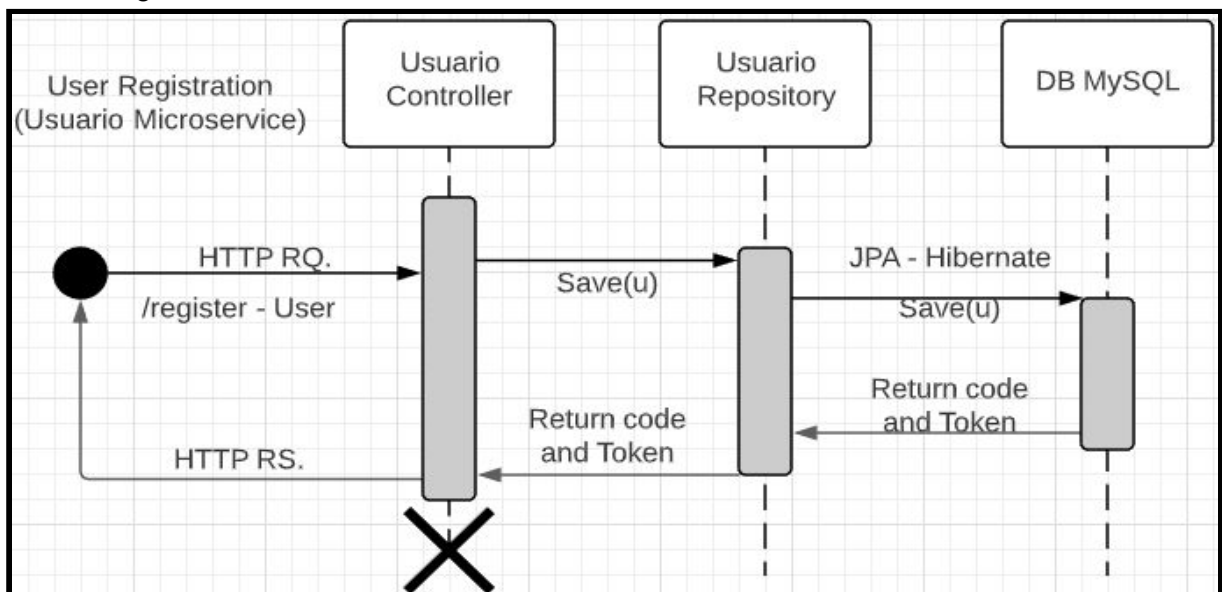
3. Resultado

El resultado final obtenido se puede analizar a partir de los siguiente diagrama de despliegue:

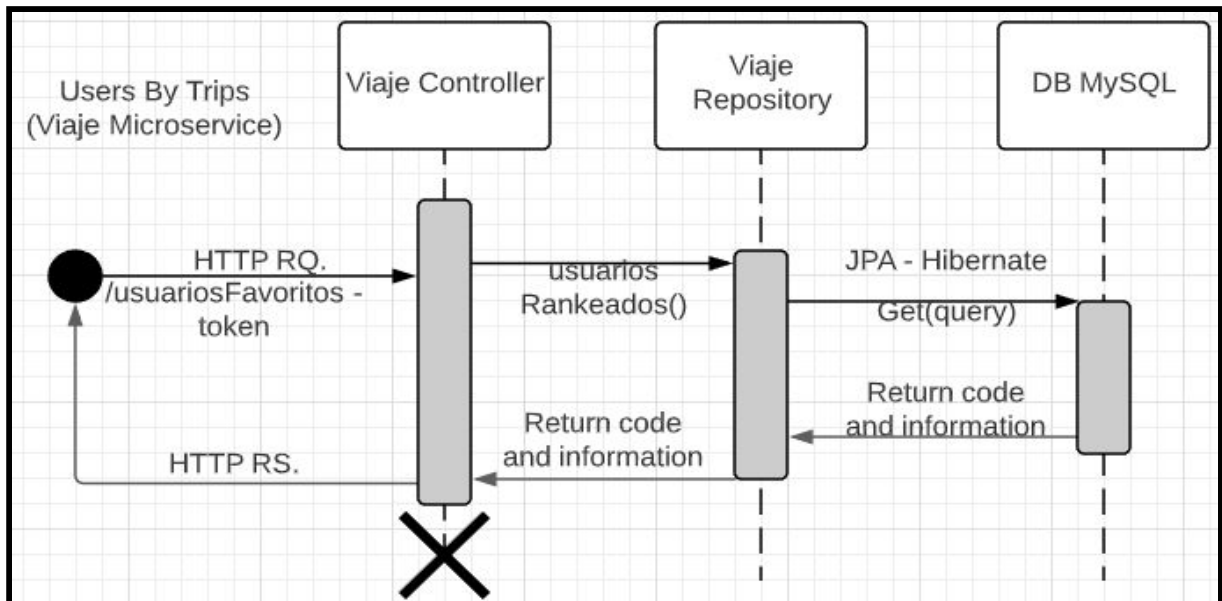


Como ejemplos ilustrativos de la funcionalidad, se muestran los diagramas de secuencia de los casos:

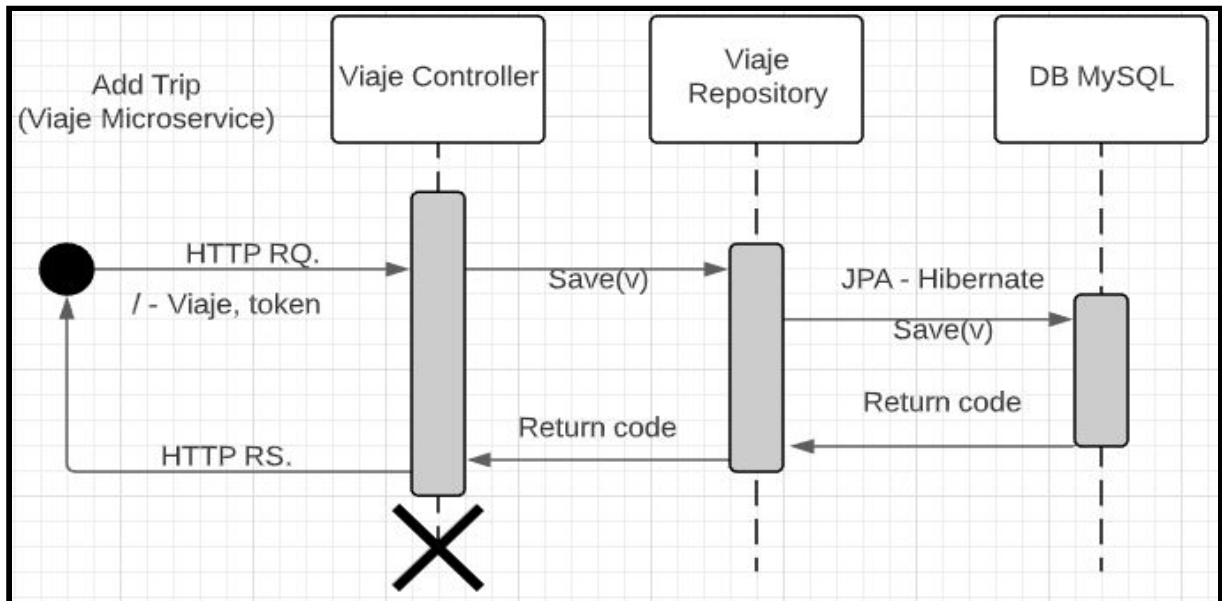
- Registrar usuario:



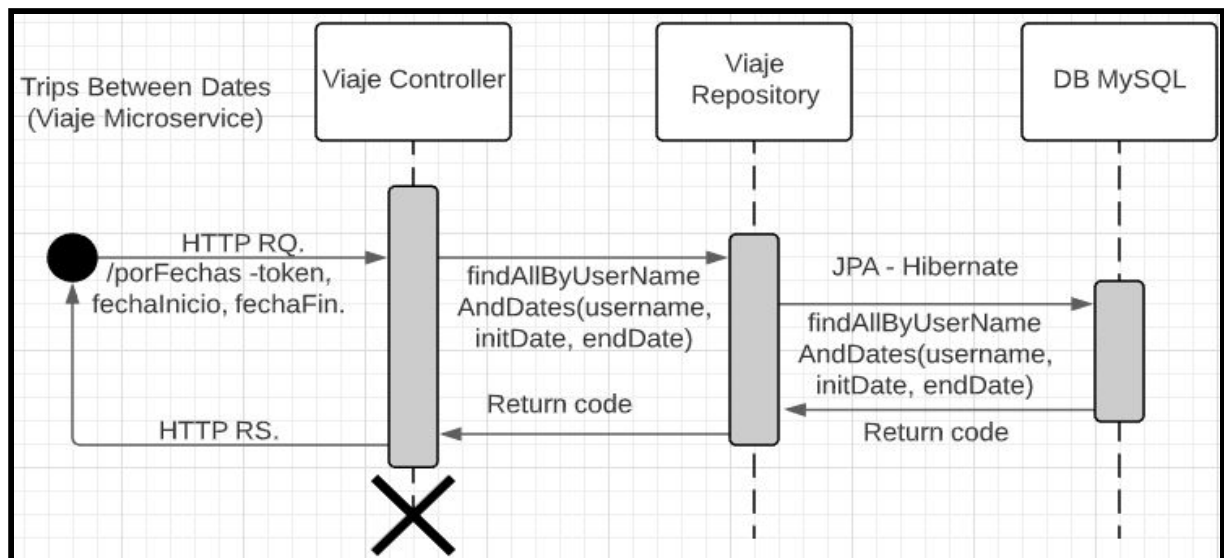
- Reporte de usuarios:



- Agregar viaje:



- Reporte de viajes:



Como se puede observar, la lógica de los microservicios es similar y con las abstracciones generadas por las tecnologías escogidas, las funcionalidades resultan simples de realizar.

3.1. Test

Los test realizados con swagger se encuentran en la sección "Informe Test".