ARQ1 | TP-Arquitectura de software 1

Etapa I

Identificar los casos de uso en base a los requerimientos funcionales y describirlos.

Casos de uso

1. Conductor registra ubicación y disponibilidad:
Un conductor que está dispuesto a ofrecer sus servicios registra su ubicación actual y su disponibilidad en Huberto. Esto permite que la plataforma identifique su posición y le asigne posibles solicitudes de viaje cercanas. El conductor puede actualizar su disponibilidad en cualquier momento para indicar si está disponible para realizar viajes o no.

2. Pasajero busca conductores cercanos:
Un pasajero que necesita un viaje puede usar la aplicación para buscar conductores cercanos a su ubicación actual. La aplicación muestra una lista de conductores disponibles ordenados por cercanía al pasajero, lo que le permite evaluar las opciones y seleccionar el conductor de su elección.

3. Pasajero solicita un viaje:
Cuando el pasajero decide realizar un viaje, la aplicación envía una notificación a los conductores cercanos con los detalles de la ubicación del pasajero y la solicitud de viaje. Esta acción inicia el proceso de emparejamiento entre el pasajero y los conductores disponibles.

4. Conductor acepta un viaje:
Un conductor que ha recibido una notificación sobre una solicitud de viaje puede aceptarla. Al hacerlo, el conductor confirma su disponibilidad para llevar al pasajero y se inicia la comunicación entre ambos. Esta acción también permite al pasajero conocer qué conductor ha aceptado su solicitud.

5. Pasajero y conductor rastrean ubicaciones en tiempo real:
Una vez que un conductor ha aceptado un viaje y se ha iniciado el emparejamiento,

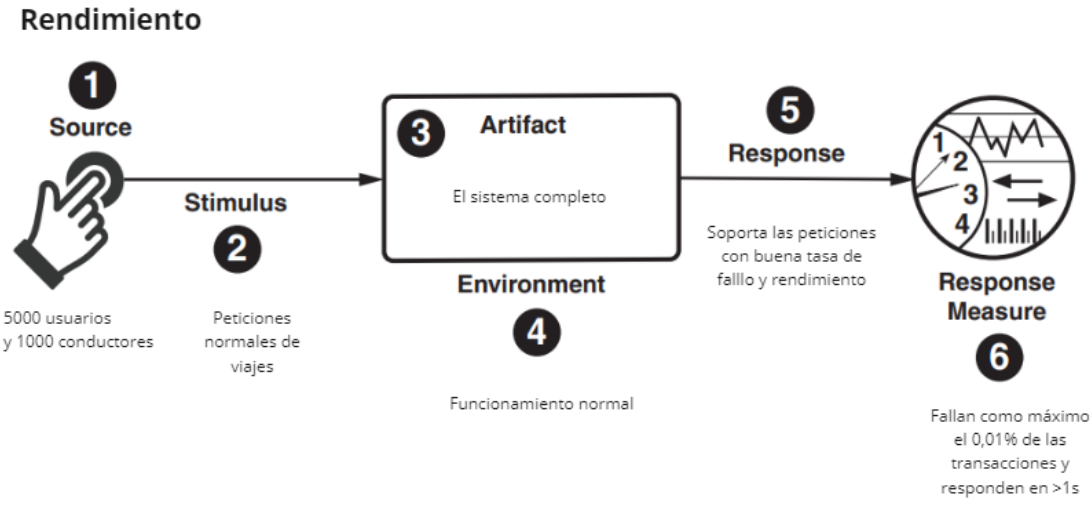
tanto el pasajero como el conductor pueden rastrear sus ubicaciones en tiempo real. Esto proporciona visibilidad sobre la llegada del conductor y permite al pasajero estar preparado para el encuentro.

6. Conductor finaliza el viaje:
Después de completar el viaje y llegar a la ubicación de destino del pasajero, el conductor presiona un botón para finalizar el viaje actual. Esta acción marca el final del trayecto y permite que el conductor esté disponible para recibir nuevas solicitudes de viaje.

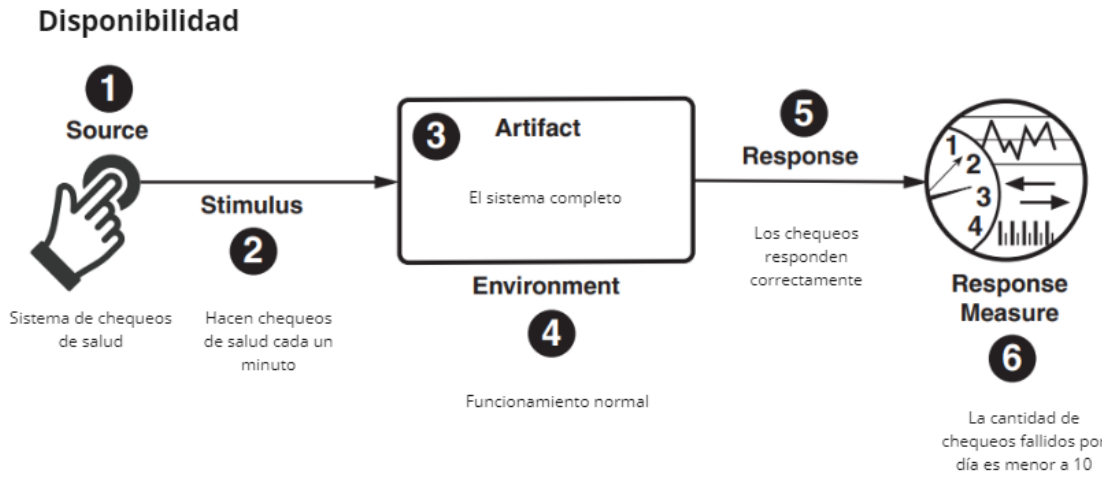
7. Pasajero realiza el pago del viaje:
Una vez que el viaje ha finalizado, el pasajero realiza el pago correspondiente. El precio del viaje se calcula en función de la distancia recorrida y del horario en que se realizó. El pasajero puede revisar los detalles del pago y confirmar la transacción dentro de la aplicación.

Atributos de calidad y escenarios:

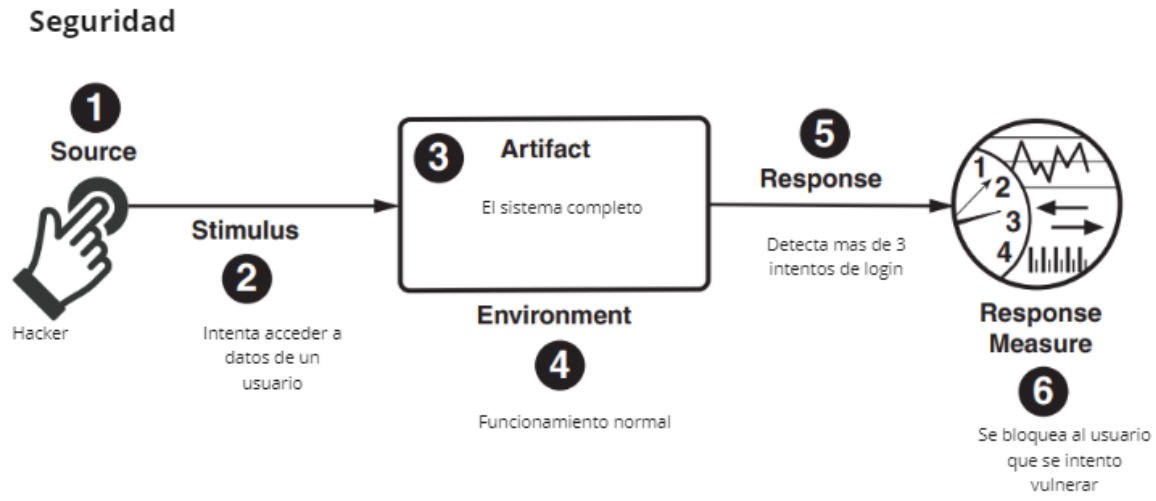
- **Rendimiento**: Garantizar un rendimiento fluido es crucial para brindar una **experiencia de usuario** satisfactoria y **retener a los usuarios,** dado que la competencia posee transacciones rápidas y baja tasa de fallo. El sistema debe manejar simultáneamente al menos 1000 conductores y 5000 pasajeros buscando y solicitando viajes.



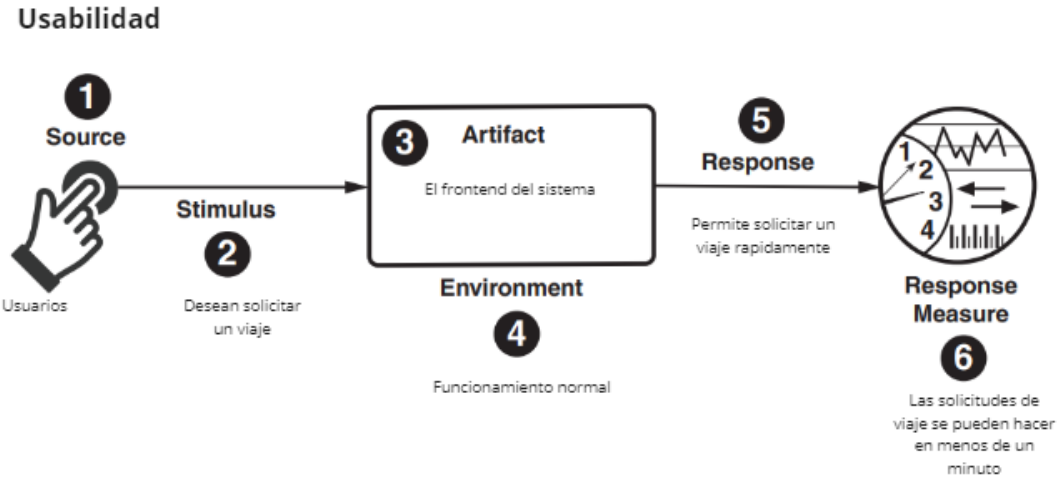
Disponibilidad: La disponibilidad alta es esencial para cumplir con las **expectativas** de los usuarios y mantener la **confianza** en la plataforma. El sistema debe tener una disponibilidad del 99.9%, garantizando que los usuarios puedan realizar y aceptar viajes en cualquier momento.



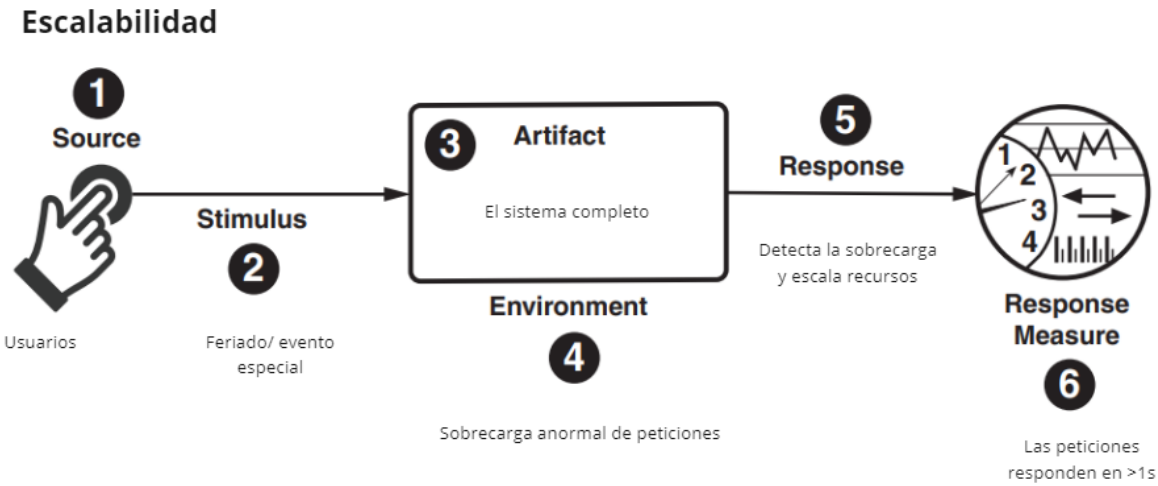
Seguridad: La seguridad de los datos es fundamental para **proteger la información sensible** de los usuarios y cumplir con las **regulaciones de privacidad**. Los datos de ubicación, los pagos y los datos personales de los usuarios deben estar encriptados (De forma tal que nadie fuera de la organización que lo desarrolla o el usuario pueda acceder) para garantizar su privacidad.



- **Usabilidad**: Una interfaz intuitiva reduce la **fricción del usuario**, lo que aumenta la probabilidad de que los usuarios **utilicen la plataforma con frecuencia**. La interfaz de usuario debe ser intuitiva, permitiendo a los usuarios solicitar un viaje en menos de 1 minuto.



Escalabilidad: La escalabilidad asegura que la plataforma pueda manejar **aumentos en la demanda sin degradar el rendimiento**. El sistema debe ser escalable para acomodar un aumento repentino en la demanda, como durante eventos especiales(i.e. paros de colectivo, día de elecciones).

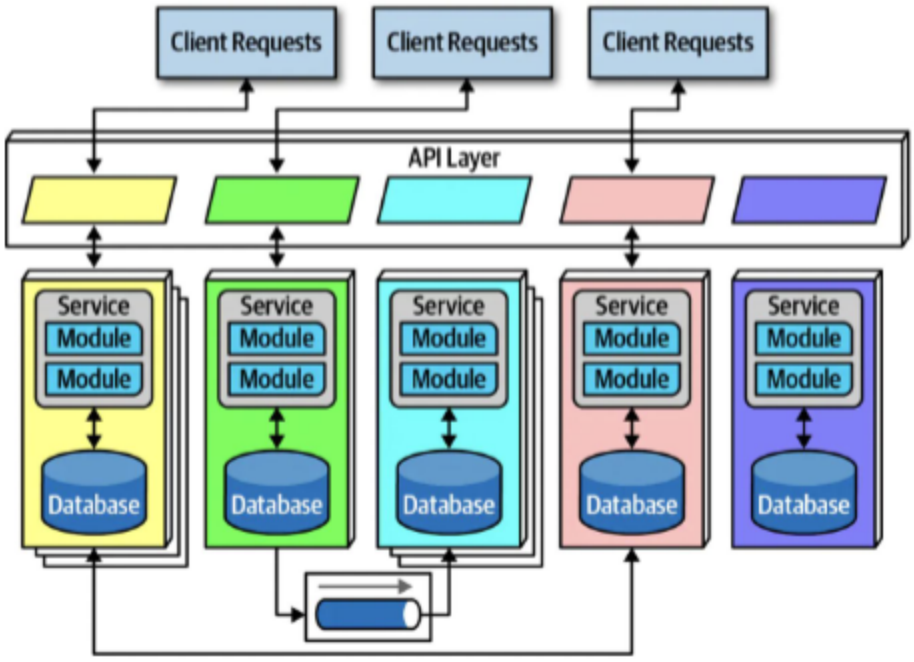


Etapa II

Arquitectura

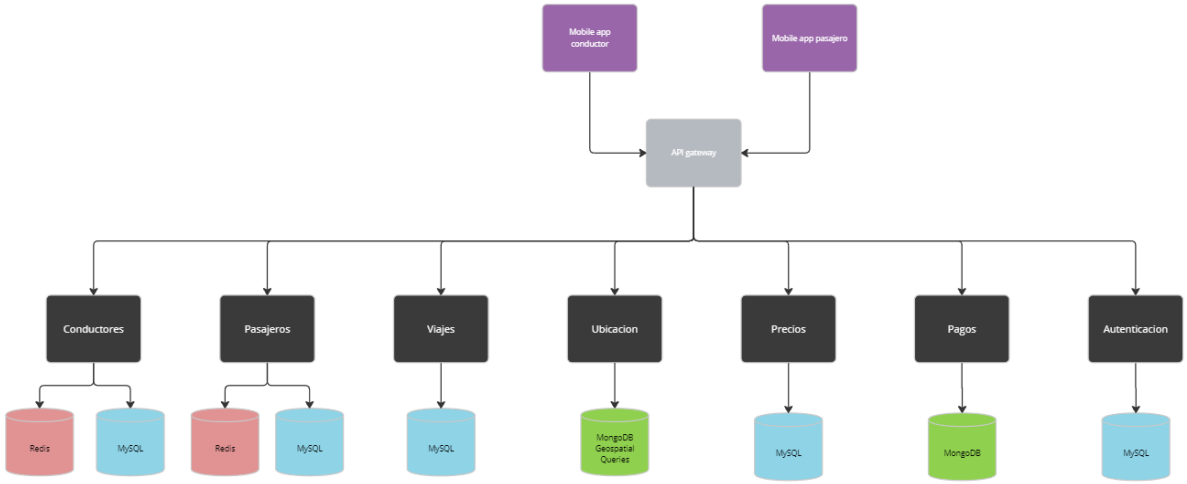
Elegí una arquitectura orientada a micro servicios para poder obtener mayor escalabilidad en caso de que la aplicación tenga un aumento considerable de usuarios. En caso de aumento, priorizamos las siguientes ventajas:

- **Desacoplamiento al realizar nuevas features o fixes**: Como los microservicios estan divididos según ámbitos del negocio, al momento de modificar o agregar algo a la codebase, solo debemos modificar uno o dos microservicios.
- **Deployment mas rapido**: A pesar de que la configuración de un pipeline de varios microservicios puede ser tedioso de un principio, una vez establecido podremos deployar atómicamente, de forma que solo corramos los tests necesarios para un microservicio y esto nos permitirá deployment continuo y rollback mucho mas fácilmente.
- **Divisiones de equipos**: Al tener un microservicio por cada área del negocio, podremos tener células pequeñas y dinámicas que se especialicen en un área del negocio y el código que le corresponde, esto acelera la resolución de bugs y las nuevas implementaciones.



Utilizo una arquitectura de DB por servicio, para reducir la complejidad de las bases de datos, mejorar la escalabilidad y el desacoplamiento.

Diagrama general de arquitectura



De arriba hacia abajo, tenemos:

Mobile app conductor: Es la aplicación Mobile por la cual interactuara el conductor al momento de ofrecer viajes y recibir pagos.

Mobile app pasajero: Es la aplicación Mobile por la cual interactuara el pasajero al momento de ordenar viajes y enviar pagos.

API gateway: Es una aplicación que centraliza las peticiones a Huberto, todas las requests deberán pasar por el y así podremos controlar logs, métricas y poner medidas de seguridad fácilmente.

MS-Conductores: Es un microservicio que se encargara de:

- CRUD de conductores.
- Estado de disponibilidad, ver si el conductor esta activo y disponible para realizar viajes

MS-Pasajeros: Es un microservicio que se encargara de:

- CRUD de pasajeros.

MS-Viajes: Es un microservicio que se encargara de:

- Solicitud y cancelación de viajes.
- Seguimiento del estado de los viajes, incluyendo la hora de inicio y finalización.
- Generación de resúmenes de viajes y recibos para pasajeros y conductores.

MS-Ubicación: Es un microservicio que se encargara de:

- Rastreo y actualización de la ubicación en tiempo real de conductores y pasajeros.
- Gestión de servicios de mapas y navegación para proporcionar rutas óptimas.
- Cálculo de la distancia entre conductores y pasajeros.

MS-Precios: Es un microservicio que se encargara de:

- Cálculo de precios de viaje basados en factores como distancia, tiempo y tarifas de servicio.
- Oferta de estimaciones de precios a pasajeros antes de confirmar un viaje.
- Administración de promociones y códigos de descuento.

MS-Pagos: Es un microservicio que se encargara de:

- Procesamiento de pagos de pasajeros por los viajes realizados.
- Gestión de métodos de pago y tarjetas de crédito.
- Registro de transacciones y generación de facturas.
- Distribución de ingresos a conductores.

MS-Autenticación: Es un microservicio que se encargara de:

- Autenticación y autorización de usuarios y conductores.
- Gestión de tokens de acceso y sesiones.
- Control de roles y permisos de usuario.
- Seguridad de la plataforma.

Bases de datos

> 📢 Quizá me adelante un poco a la etapa III en esta parte, pero quise aclarar un poco lo que aparecía en el diagrama de componentes.

Elegí mayormente bases de datos relacionales **MYSQL** priorizando ciertas ventajas:

- Las estructuras de datos no van a cambiar mucho porque son mayormente CRUDs
- Es mas fácil conseguir desarrolladores/equipo que las maneje
- La performance de MySQL esta dentro de lo esperado por los criterios de performance establecidos como atributos de calidad

Para algunos casos particulares agregue o cambie el tipo de DB, por ejemplo en **ms-pasajeros** y **ms-conductores** agregue una base de datos redis a cada uno para obtener datos que se consultan recurrentemente (Datos del usuario/config) de forma mas eficiente por medio de consultas a cache.

En el caso de **ms-ubicacion** decidi utilizar MongoDB, pues tiene una funcionalidad optimizada para almacenar ubicaciones en tiempo real, esta feature se llama Geospatial Queries, estas queries te permiten obtener usuarios cercanos por medio de esferas y verificar si estas se intersectan, entre otras funcionalidades.

En **ms-pagos** supuse que tendríamos muchas integraciones con otras plataformas de pago, MercadoPago, Visa, Stripe, etc. Para poder soportar diferentes estructuras con respecto a los datos de las integraciones, sugiero utilizar una base de datos MongoDB. De esta forma podríamos tener una entidad “Pago” que represente un pago pero podría estar estructurada de forma diferente según la integración de donde se emita.