JUAN PABLO MONTOYA BEDOYA

Sistema de gestión de flotas inteligente

Estructura y diseño

1.Análisis del problema.

Un mecanismo inteligente para monitorear numerosos automóviles es sencillo de construir, funciona eficazmente y es rentable. Crear un sistema versátil que ayude a diversos sectores, con un plan adaptable para aumentar el rendimiento mediante un enfoque bien organizado basado en pequeños servicios. Esta opción ofrece una oportunidad más óptima para que el sistema se expanda y evolucione cuando sea necesario para alinearse con las preferencias del usuario.

2.Requisitos funcionales.

Seguimiento en tiempo real.

Este sistema permite el seguimiento continuo de la velocidad y la ubicación del automóvil.

La información se archivará en un servicio de AWS con limitaciones, y superando estas limitaciones se empleará almacenamiento en la nube para economizar.

Optimización de rutas.

"Cálculo del análisis de rutas eficientes bajo influencias ambientales."

Aquí está la versión simplificada.

Mantenimiento predictivo.

- Generar alertas automáticas basadas en datos recopilados por sensores.

Mantener los datos "en las instalaciones", pero sincronizarlos siempre que sea posible con la nube, reducir costos y mejorar la velocidad.

Gestión de conductores.

La evaluación del conductor depende de la información de mantenimiento y del circuito.

Integración con terceros.

El uso de programas informáticos que interactúan con otras aplicaciones ayudará a facilitar las conexiones en el futuro.

3.Requisitos técnicos.

Escalabilidad.

- Infraestructura basada en la nube con capacidad de autoescalado para ajustarse a la demanda.

Alta disponibilidad.

¿Puedes proporcionar un ejemplo y un análisis? - ¡Claro que sí! La computación en la nube tiene.

Modularidad.

Se puede simplificar de la siguiente manera: Usamos pequeñas partes (microservicios) para que los sitios web y las aplicaciones funcionen mejor al encontrar rutas más rápidas y notificarnos cuando necesitan reparaciones.

Seguridad.

Seguir normas como el GDPR y asegurar que los datos estén protegidos con encriptación.

4.Restricciones.

Presupuesto limitado.

Estamos dedicados a emplear infraestructura en la nube y metodologías de código abierto.

Implementación gradual.

Simplifica lo siguiente. Comienza con una versión básica de tu proyecto que priorice.

Compatibilidad global.

5.Desafíos adicionales.

Gestión de picos de tráfico.

- Implementar equilibradores de carga, como el Elastic Load Balancer de AWS.

Sincronización en áreas de baja conectividad.

Crear un sistema que almacene información localmente en el dispositivo y se sincronice cuando haya una conexión a internet disponible.

Diseño de la interfaz.

- Crear plataformas específicas para diferentes usuarios.

- Administradores: aplicación web.

- Controladores: interfaz móvil.

- Integraciones externas: APIs dedicadas.

6.Selección de arquitectura.

Microservicios.

Integración de una vía interna de diálogo y un mediador unificado que amalgama las consultas, dando como resultado la accesibilidad del servicio.

7. Ventajas y desventajas (Compensaciones).

Beneficios.

- Modularidad que facilita las actualizaciones y el mantenimiento.

- Escalabilidad para adaptarse a las crecientes demandas.

- Aislamiento de fallos entre servicios.

Debilidades.

- Mayor complejidad en la implementación y operación inicial.

8.Ventajas y desventajas (Compensaciones).

Patrones de diseño.

Arquitectura en capas.

"La división organizada de tareas mejora la capacidad del sistema para crecer y ser amigable con el medio ambiente."

9.Aplicación de los 12 factores.

1.Base de código: Un solo repositorio para todos los servicios.

2.Dependencias: Uso de contenedores Docker para aislar componentes.

3.Configuración: Variables de entorno administradas.

4.Servicios de soporte: Bases de datos como PostgreSQL para cumplir con los requisitos.

5.Compilación e implementación: Implementaciones automatizadas usando CI/CD.

6.Procesos: Servicios sin estado con almacenamiento externo.

7.Enlace de puertos: APIs expuestas a través de una puerta de enlace de API.

8.Concurrencia: escalado usando Kubernetes.

9.Disponibilidad: Servicios que arrancan y terminan rápidamente.

10.Paridad entre entornos: Uso de Docker para asegurar la consistencia.

11.Registros: Centralización y gestión de registros.

12.Procesos de gestión: Tareas administradas con contenedores efímeros para optimizar recursos.