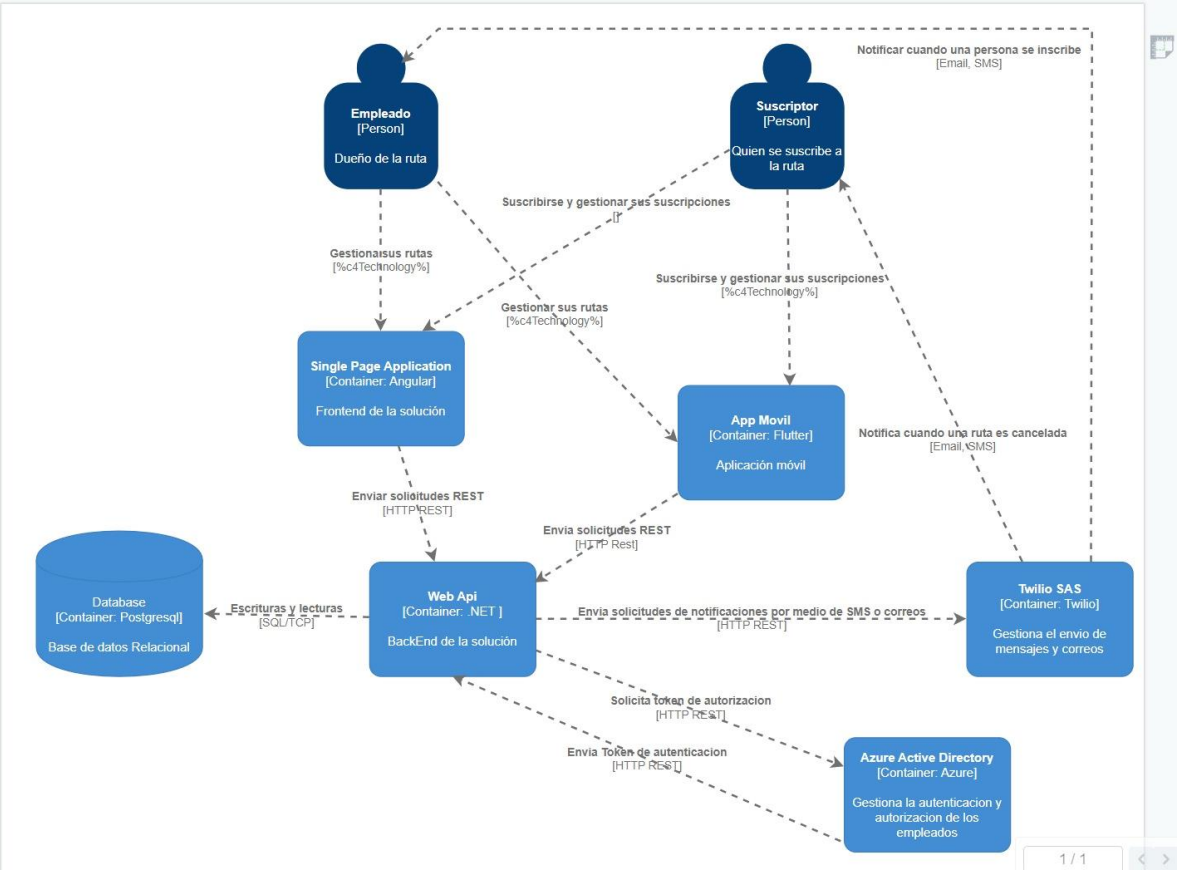
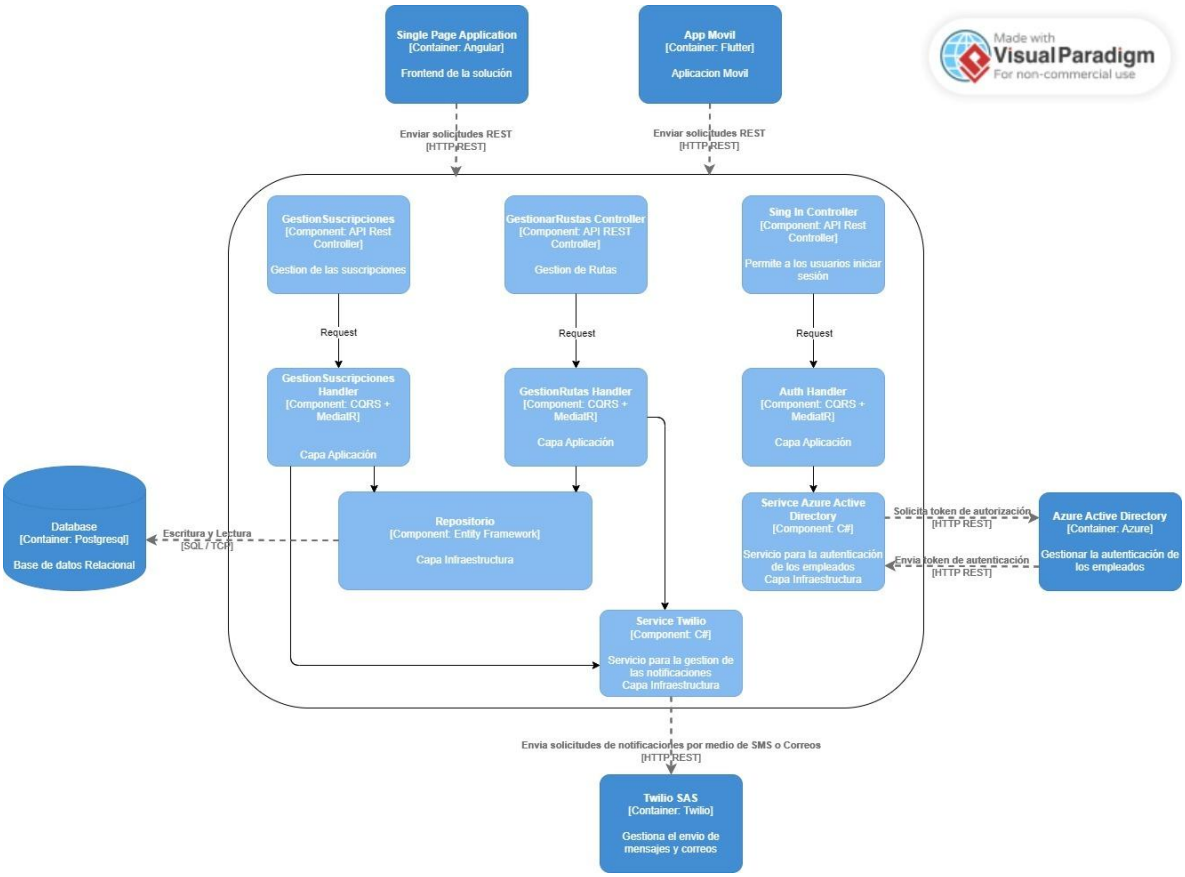
DESCRIPCIÓN DETALLADAMENTE LA SOLUCIÓN PLANTEADA Y SUS COMPONENTES

En respuesta a la solicitud de la empresa para detallar nuestra solución, empleamos el modelo de arquitectura C4 para proporcionar una visión estructurada y comprensible. Me centre especialmente en las capas de Contenedores y Componentes para ofrecer una descripción exhaustiva.



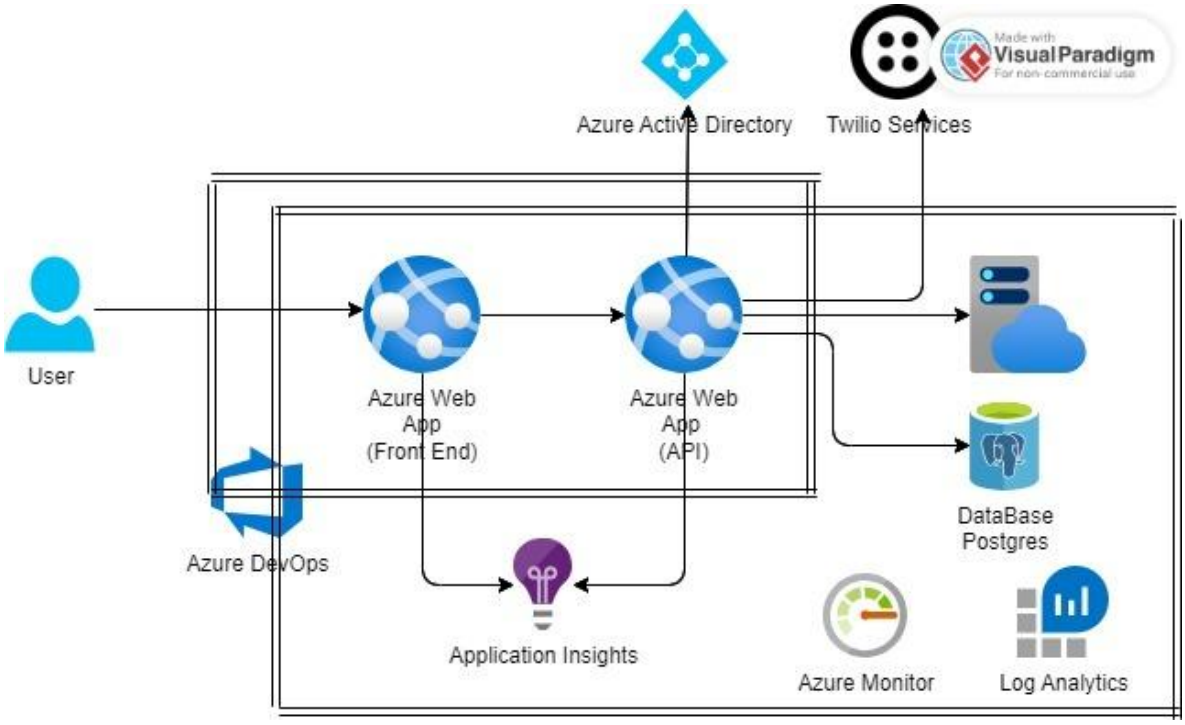
En la capa de Contenedores, delineo las instancias físicas o virtuales que ejecutan nuestros servicios. Detallamos la infraestructura, identificando servicio externos, bases de datos, y otros elementos clave, junto con sus interconexiones. Esto brinda a la empresa una visión transparente de la arquitectura subyacente y cómo los diferentes contenedores interactúan entre sí.



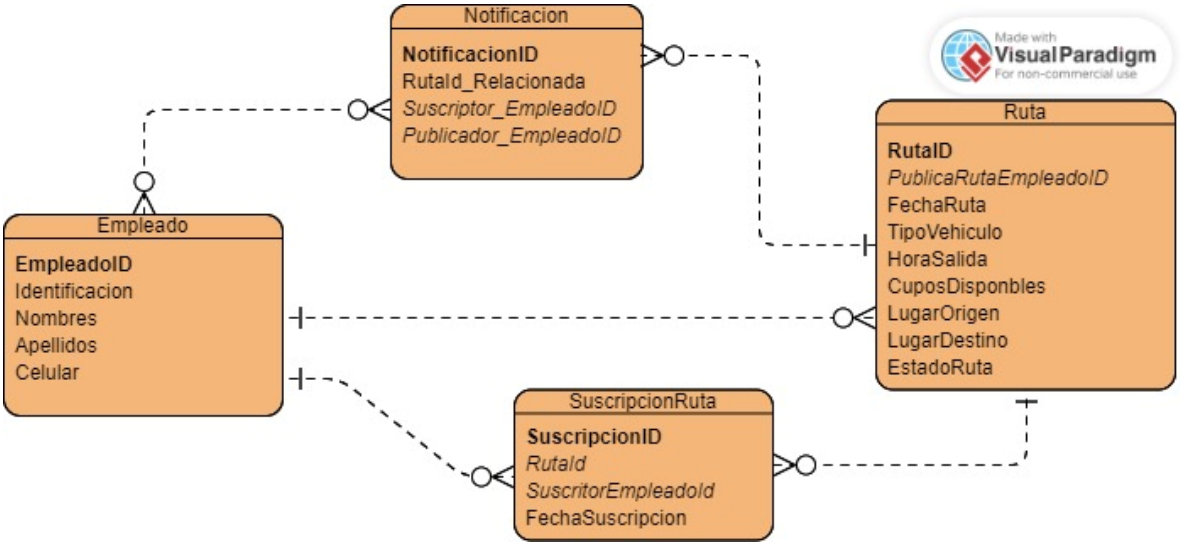
En la capa de Componentes, se puede observar el detalle de los servicios, desglosándolos en módulos y funciones específicas. Identificando los componentes esenciales, describo sus responsabilidades individuales, y esbozamos las relaciones entre ellos. Al hacerlo, proporcionando una comprensión profunda de la lógica interna de nuestra solución, permitiendo a la empresa apreciar la complejidad y la interconexión de cada componente.

Este enfoque basado en C4 no solo cumple con la solicitud de la empresa de una descripción detallada, sino que también ofrece una estructura organizativa que facilita la comunicación y la comprensión tanto a nivel ejecutivo como técnico. La capa de Contenedores muestra la infraestructura de alto nivel, mientras que la capa de Componentes ofrece una vista detallada de la funcionalidad específica de nuestra

DIAGRAMA DE ARQUITECTURA DE LA SOLUCIÓN (BASADA EN LA NUBE DE AZURE).



MODELO ENTIDAD RELACION



TECNOLOGÍAS, LENGUAJES DE PROGRAMACIÓN Y FRAMEWORKS A UTILIZAR Y JUSTIFICACIÓN

Me complace presentar mi propuesta arquitectónica y las tecnologías seleccionadas para abordar el proyecto de rutas de transporte sostenible en el marco de la prueba técnica. La solución propuesta se basará en la utilización de tecnologías modernas y prácticas de desarrollo para garantizar un sistema eficiente y escalable. En este contexto, he elegido Angular como framework para el desarrollo del frontend y una Web API en C# implementando el patrón de diseño CQRS (Command Query Responsibility Segregation) junto con MediatR para el backend.

Frontend con Angular:

Angular será seleccionado como el framework para el desarrollo del frontend debido a su robustez, modularidad y amplia comunidad de desarrollo. Esta elección permitirá la creación de una interfaz de usuario interactiva y dinámica, cumpliendo con los principios de modularización del código, inyección de dependencias y el uso de TypeScript para un desarrollo más seguro y mantenible.

Backend con Web API en C# y CQRS+MediatR:

La implementación del backend se llevará a cabo mediante una Web API desarrollada en C#, aprovechando la potencia de este lenguaje para la creación de servicios web eficientes y seguros. Optaré por adoptar el patrón de diseño CQRS (Command Query Responsibility Segregation) junto con MediatR. Esta elección estratégica ofrecerá beneficios sustanciales en términos de rendimiento, escalabilidad y mantenibilidad del sistema.

Ventajas de CQRS:

- **Rendimiento:** Al separar las operaciones de lectura y escritura, se optimizarán cada una de manera independiente, mejorando significativamente el rendimiento del sistema, especialmente en escenarios de carga elevada.
- **Escalabilidad:** La arquitectura CQRS permitirá escalar de forma independiente las operaciones de lectura y escritura, adaptándose eficientemente a las demandas variables del sistema.
- **Simplicidad:** La separación de responsabilidades simplificará la lógica de negocio, haciendo que el código sea más comprensible y mantenible.
- **Mantenibilidad**: La clara separación de las operaciones de lectura y escritura facilitará la identificación y corrección de problemas, así como la introducción de nuevas características o mejoras sin afectar otras partes del sistema.

- **Flexibilidad para la base de datos:** CQRS permitirá utilizar diferentes modelos de datos para lectura y escritura, adaptándose a las necesidades específicas de cada operación sin comprometer la coherencia del sistema.

MediatR como Implementación de CQRS:

MediatR, una biblioteca en C#, será utilizada para simplificar la implementación de CQRS. Proporcionará un mecanismo eficaz para manejar comandos y consultas de manera separada, promoviendo la organización y mantenimiento eficiente del código del backend.

Otros Patrones de Diseño y Buenas Prácticas

Inyección de Dependencias (DI):

La Inyección de Dependencias ha sido un pilar esencial en la arquitectura propuesta. Este patrón no solo posibilita una gestión eficiente de las dependencias, sino que también promueve la modularidad y facilita el mantenimiento del código. Al reducir el acoplamiento entre los componentes, estamos asegurando una arquitectura más flexible y resiliente ante cambios futuros.

Patrón de Repositorio:

La implementación del Patrón de Repositorio es clave para abstraer la lógica de acceso a datos. Esto no solo favorece la separación de las operaciones de persistencia, sino que también simplifica la gestión de entidades. El uso de este patrón proporciona una capa de abstracción que facilita la adaptabilidad a diferentes fuentes de datos y permite un mantenimiento más sencillo.

Patrón Decorador (FluentValidation):

La aplicación del Patrón Decorador con FluentValidation asegura una validación robusta de datos. Este enfoque proporciona una capa consistente y flexible para la validación de formularios y solicitudes. La utilización de FluentValidation me permite definir reglas de validación de manera clara y concisa, mejorando la calidad y consistencia de los datos en todo el sistema.

Patrón DTO (Data Transfer Object):

La adopción del Patrón DTO es esencial para la transferencia eficiente de datos entre capas del sistema. Al garantizar una representación clara y coherente de la información, este patrón facilita la comunicación entre diferentes componentes. Esto no solo mejora la eficiencia en la transferencia de datos, sino que también contribuye a la coherencia en la representación de la información en todo el sistema.

Patrón Factory (AutoMapper):

La adopción del Patrón Factory con AutoMapper se traduce en la simplificación de la asignación de datos entre objetos. Reduciendo la necesidad de código repetitivo, AutoMapper mejora la legibilidad y mantenibilidad del código. Esta elección no solo agiliza el desarrollo, sino que también reduce la probabilidad de errores en la asignación de datos entre entidades.

Logging:

La implementación de registros detallados con Logging es esencial para la monitorización y resolución eficiente de problemas. La inclusión de registros detallados asegura la trazabilidad de las operaciones del sistema, permitiendo una respuesta más rápida y efectiva ante cualquier incidente. El sistema de logging será una herramienta clave para la identificación proactiva de problemas y la mejora continua de la aplicación.

Auditoría:

La incorporación de auditoría es crucial para el seguimiento y la supervisión detallada de las acciones realizadas en el sistema. Además de contribuir a la seguridad, la funcionalidad de auditoría es esencial para el cumplimiento normativo, asegurando que todas las acciones críticas sean registradas y puedan ser revisadas en cualquier momento. Esta capa de auditoría proporcionará una capa adicional de seguridad y transparencia.

Servicios Externos:

La integración de servicios externos, como Twilio para las notificaciones y el servicio Active Directory de Azure para la autenticación y autorización, mejora la seguridad y la experiencia del usuario. Twilio proporcionará una plataforma robusta y confiable para la gestión de notificaciones, mientras que el servicio Active Directory de Azure garantizará un acceso seguro y autorizado a la aplicación. Esta integración fortalecerá la seguridad y usabilidad de la aplicación en su conjunto.

Documentación y Comentarios:

La idea es que también se incluya documentación detallada tanto en el código como en documentos externos para facilitar la comprensión y el mantenimiento futuro. También, destacaré la importancia de comentarios significativos en el código para que otros desarrolladores puedan entender rápidamente la lógica y la estructura.

Pruebas Unitarias:

Enfocaré considerablemente el esfuerzos en la implementación de pruebas unitarias como piedra angular del proceso de desarrollo. La importancia de las pruebas unitarias no solo reside en la validación del código, sino también en la garantía de la calidad y la robustez del sistema en su conjunto. Aquí destaco cómo abordaremos las pruebas unitarias y las herramientas específicas que utilizaremos en Angular y C#:

Las pruebas unitarias no son solo una fase en el ciclo de desarrollo; son un componente esencial para asegurar la confiabilidad del código. Al subrayar la importancia de las pruebas unitarias, no solo estamos validando la funcionalidad esperada, sino que también estamos construyendo una red de seguridad para futuras iteraciones y mejoras del código.

Resultado Esperado:

- Identificación temprana de errores y malentendidos en la lógica de negocio.
- Garantía de la estabilidad del código a medida que evoluciona.

Herramientas Específicas:

En el contexto de Angular, utilizaremos Jasmine como nuestro principal framework de pruebas unitarias. Jasmine proporciona una sintaxis clara y expresiva que facilita la escritura y ejecución de pruebas, y se integra perfectamente con el entorno de prueba proporcionado por Karma.

Para las pruebas unitarias en el backend en C#, optaremos por Xunit, una potente herramienta que ofrece un marco de pruebas sólido y flexible. Xunit es conocido por su capacidad para ejecutar pruebas de manera paralela, lo que acelera significativamente el proceso de ejecución de pruebas.

METODOLOGÍA DE DESARROLLO.

Para implementar una metodología robusta que garantice eficiencia, agilidad y calidad en todo el ciclo de vida del software. Para lograr esto, he optado por combinar lo mejor de Scrum y DevOps en la metodología de desarrollo.

En primer lugar, empleare Scrum como nuestro marco de trabajo principal. Organizando las tareas en sprints, permitiéndonos abordar de manera iterativa y focalizada los objetivos clave del proyecto. Las reuniones regulares de planificación de sprint y revisión facilitarán la transparencia y la colaboración dentro del equipo, promoviendo un enfoque ágil y adaptable.

Además, para asegurar una entrega continua y eficiente, integrare principios de DevOps en nuestro proceso. Automatizando los procesos de construcción, prueba y despliegue para lograr una entrega constante y confiable del software. Esta integración estrecha entre los equipos de desarrollo y operaciones acelerará la capacidad para llevar nuevas funcionalidades a producción de manera rápida y estable.

En conjunto, esta estrategia combina lo mejor de ambos mundos: la agilidad de Scrum y la eficiencia de DevOps. Con esta metodología, no solo aspiro a mejorar la velocidad de desarrollo, sino también a fortalecer la colaboración interdepartamental y a garantizar una experiencia del usuario excepcional. Esta propuesta sienta las bases para un desarrollo futuro exitoso y sostenible del proyecto, alineado con las mejores prácticas de la industria.

Descripción de las buenas prácticas metodológicas que pueden agilizar el proceso de desarrollo, para la entrega de software de calidad en los diferentes ambientes.

En el marco de nuestra metodología de desarrollo, hemos identificado varias buenas prácticas metodológicas que agilizarán significativamente nuestro proceso de desarrollo y contribuirán a la entrega de software de calidad en diversos entornos.

En primer lugar, implementaremos la Integración Continua (CI) y el Despliegue Continuo (CD). La CI asegurará que las modificaciones de código se integren de manera regular, permitiendo una detección temprana de posibles conflictos y reduciendo el riesgo de errores durante el desarrollo. Por su parte, el CD automatizará el proceso de despliegue, facilitando la entrega continua y confiable de nuestras aplicaciones en diferentes entornos, desde pruebas hasta producción.

Otra práctica fundamental será la adopción de pruebas automatizadas en todas las etapas del desarrollo. La automatización de pruebas no solo acelera el proceso de verificación de código, sino que también mejora la consistencia y la exhaustividad de las pruebas. Esto resulta esencial para garantizar la estabilidad y la calidad del software en entornos variables.

En cuanto al análisis estático de código, integraremos herramientas como SonarQube para realizar evaluaciones continuas de la calidad del código. Este análisis proporcionará una visión detallada de posibles problemas, vulnerabilidades y mejoras en el código fuente, permitiendo correcciones proactivas y manteniendo altos estándares de calidad a lo largo del desarrollo.

INFRAESTRUCTURA Y PLATAFORMAS NECESARIAS PARA EL DESARROLLO DE LA SOLUCIÓN

Azure App Services Web (Frontend y API):

Optare por Azure App Services Web para implementar tanto el frontend como la API. Esta elección asegurará una escalabilidad automática y un entorno de ejecución gestionado. La flexibilidad y escalabilidad proporcionadas por esta plataforma en la nube nos permitirán adaptar fácilmente a las cambiantes demandas del sistema. Además, la administración centralizada simplificará las tareas de mantenimiento y garantizará una alta disponibilidad.

Azure PostgreSQL Database:

Selecciono Azure PostgreSQL Database como sistema de gestión de bases de datos. Esta elección brinda un almacenamiento eficiente, seguro y escalable para la aplicación. La base de datos en la nube proporcionará redundancia, copias de seguridad automatizadas y la capacidad de escalar según las necesidades del sistema. La integración con otros servicios de Azure facilitará la gestión y monitorización continua de la base de datos.

Azure DevOps:

Azure DevOps se erigirá como la herramienta central para la integración continua y la implementación continua (CI/CD). La integración de Azure DevOps asegurará un flujo de trabajo automatizado y eficiente, desde el desarrollo hasta la implementación. Esta estrategia garantizará la coherencia y calidad en todas las etapas del ciclo de vida del desarrollo.

Application Insights, Azure Monitor y Log Analytics:

Para el monitoreo integral del rendimiento, la disponibilidad y la seguridad de la aplicación, confio en Application Insights, Azure Monitor y Log Analytics. Esta combinación de servicios permitirá la identificación proactiva de problemas y una optimización continua. Estas herramientas analíticas asegurarán un monitoreo en tiempo real y proporcionarán información valiosa para la mejora continua y la toma de decisiones informadas.

POSIBLES RIESGOS QUE PUEDEN MATERIALIZARSE EN LA EJECUCIÓN DEL PROYECTO Y CÓMO MITIGARLOS.

Riesgo: Cambios en los Requisitos del Usuario

Mitigación: Establecer una comunicación continua con los stakeholders y definir requisitos de manera clara y detallada desde el inicio del proyecto. Implementar procesos de gestión de cambios para evaluar y adaptarse a modificaciones.

Riesgo: Fallos en la Planificación y Estimación de Tiempos

Mitigación: Utilizar metodologías de desarrollo ágiles para adaptarse a cambios y realizar estimaciones realistas. Realizar revisiones periódicas del progreso y ajustar la planificación según sea necesario.

Riesgo: Problemas de Calidad del Código

Mitigación: Implementar prácticas de desarrollo de código limpio y realizar revisiones regulares de código. Integrar pruebas automatizadas y análisis estático para identificar y corregir problemas tempranamente.

Riesgo: Escasez de Habilidades y Competencias

Mitigación: Realizar una evaluación exhaustiva de las habilidades del equipo y proporcionar capacitación cuando sea necesario. Considerar la contratación de expertos externos para complementar las habilidades internas.

Riesgo: Problemas de Compatibilidad y Migración

Mitigación: Realizar pruebas exhaustivas de compatibilidad con sistemas existentes. Implementar estrategias graduales de migración y contar con planes de contingencia para minimizar impactos en caso de problemas.

Riesgo: Cambios en la Tecnología o Herramientas Utilizadas

Mitigación: Mantenerse actualizado sobre las tendencias tecnológicas y evaluar regularmente la estabilidad y soporte de las herramientas utilizadas. Establecer procesos para migraciones controladas y realizar pruebas de compatibilidad.

Riesgo: Fallos en la Seguridad de la Información

Mitigación: Incorporar medidas de seguridad desde el diseño. Realizar auditorías y pruebas de seguridad regularmente. Mantenerse informado sobre nuevas amenazas y vulnerabilidades.

Riesgo: Problemas de Comunicación entre Equipos

Mitigación: Establecer canales de comunicación claros y herramientas colaborativas. Celebrar reuniones regulares para alinear a los equipos y garantizar una comprensión mutua de los objetivos del proyecto.

Riesgo: Cambios en la Dirección o en la Prioridad del Proyecto

Mitigación: Establecer una comunicación abierta con la alta dirección. Documentar claramente los objetivos y alcances del proyecto. Realizar revisiones periódicas para asegurar la alineación con las metas organizativas.

Riesgo: Dependencia de Proveedores Externos

Mitigación: Diversificar proveedores siempre que sea posible. Establecer acuerdos contractuales claros, incluyendo niveles de servicio y planes de contingencia para situaciones imprevistas.