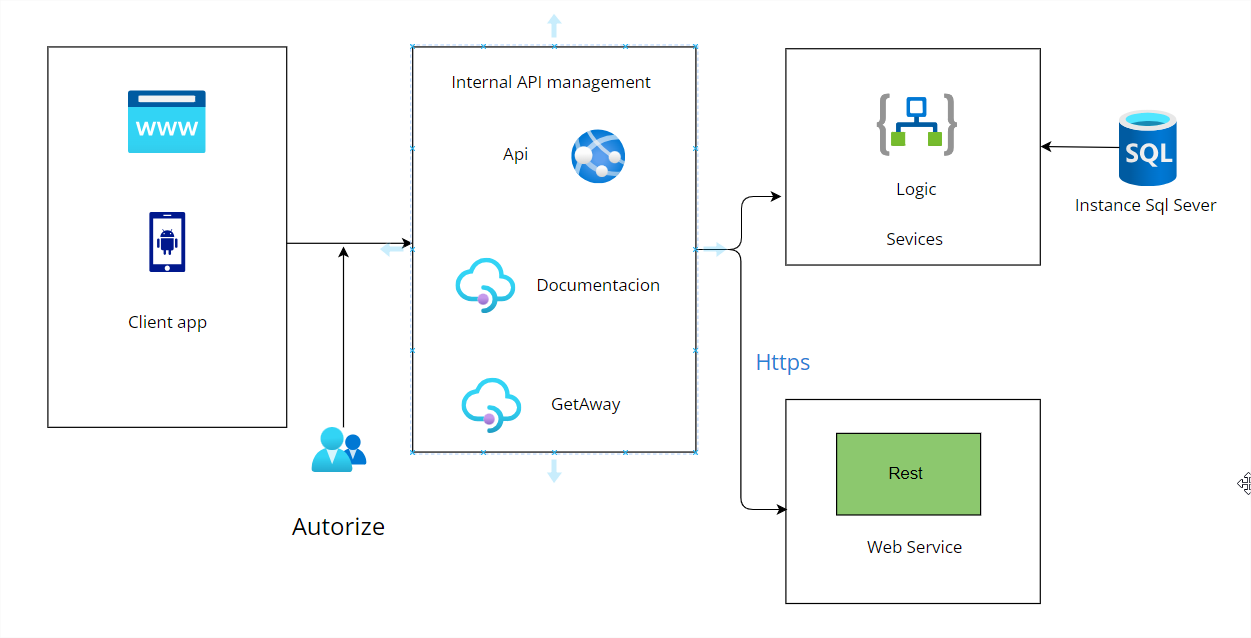
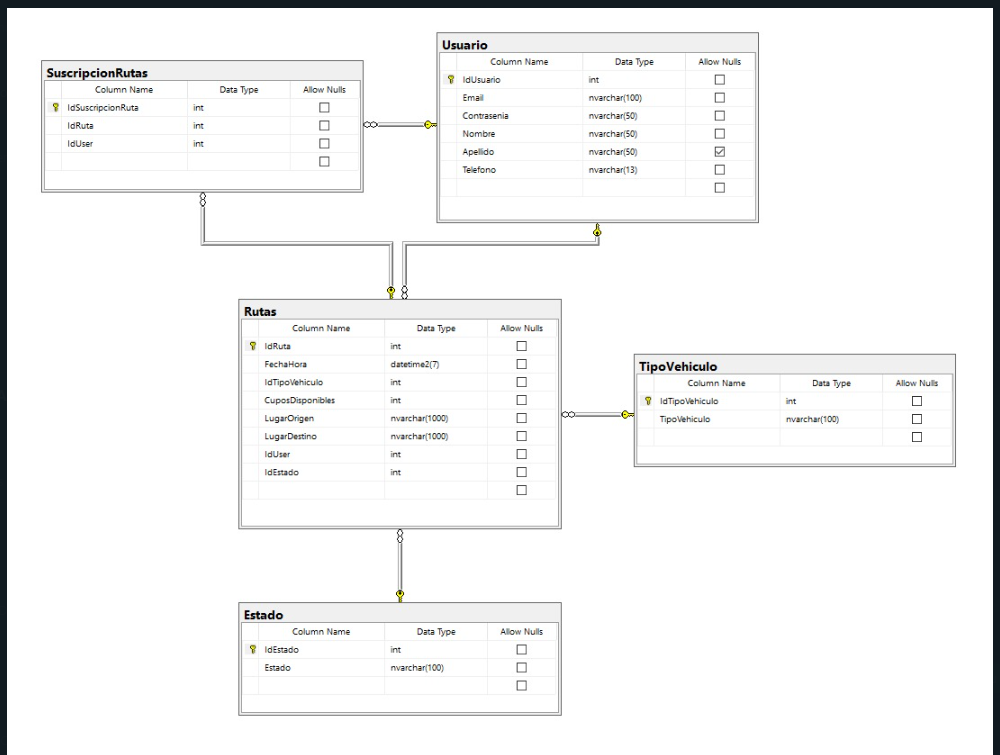
**1. Diagrama de arquitectura de la solución (Basada en la nube de Azure)..**

****

**2.Modelo entidad relación.**

****

**3. Tecnologías, lenguajes de programación y frameworks a utilizar y justificación de la selección de cada uno.**

Para este proyecto trabajaría con estas herramientas.

* Git
* Framework .Net core, Angular, xamarin(Net MAUI), entity , xUnit
* Lenguajes c#, javascript , typescript

Git : Para control de versiones

**Frameworks**

* **.NET Core :** se integra con tecnologías modernas y populares , como ASP.NET Core para el desarrollo web.
* **Entity Framework Core** : para el acceso a bases de datos.
* **Xamarin o net MAUI** : para el desarrollo de aplicaciones móviles multiplataforma con el lenguaje c#.
* **Angular** : le permite comunicarse con los componentes, alimentarlos con datos y aceptar eventos de los componentes; por lo tanto, hace que los componentes sean reutilizables y más aislados. Angular permite la creación rápida de vistas de interfaz de usuarios con una sintaxis de plantilla muy sencilla y eficaz. Además, con sus herramientas de líneas de comandos puedes comenzar a construir en menor tiempo y agregar componentes, pruebas e implementaciones al instante. Con esta framework se trabajaría la app web
* **xUnit**: para la realización de pruebas unitarias en tecnología .net

**Lenguajes**

Javascript y typescript : son uno de los lenguajes más populares,y en este caso se utilizará para trabajar de lado de las aplicaciones del lado cliente. son lenguajes predilectos para el trabajo web, con buena documentación , una buena comunidad , flexibles , escalables y de bajo nivel.

* **c#:** Con este se desarrollaría toda parte del backend en .net core por el motivo que es un lenguaje que está introduciendo mucho desarrollo funcional y no se mantiene únicamente como un lenguaje de programación orientado a objetos, por lo tanto es un lenguaje que va a estar siempre en la última tendencia y va a estar siempre a la con actualizaciones en su framework y su core, otra ventaja es que es un lenguaje muy sencillo de usar y muy fácil de aprender, debido a que tiene una sintaxis muy similar a C y a Java además de contar con documentación y una comunidad a nivel mundial que genera información sobre el mismo.

**3. Metodología de desarrollo.**

Metodología Scrum.

4.**Descripción de las buenas prácticas metodológicas que pueden agilizar el proceso**

**de desarrollo, para la entrega de software de calidad en los diferentes ambientes.**

1. **Desarrollo Ágil:**
   * Utilizar metodologías ágiles como Scrum o Kanban para permitir una respuesta rápida a los cambios en los requisitos del proyecto. Dividir el desarrollo en iteraciones cortas (sprints) para entregar funcionalidades de manera incremental.
2. **Integración Continua:**
   * Implementar integración continua para asegurar que el código se integre de manera regular y se realicen pruebas automáticamente.
3. **Entrega Continua:**
   * Implementar prácticas de entrega continua para desplegar automáticamente versiones del software en entornos de prueba y producción. Automatizar la construcción, prueba y despliegue para acelerar el ciclo de vida del software.
4. **Automatización de Pruebas:**
   * Crear suites de pruebas automáticas para verificar la funcionalidad y detectar regresiones. Incluir pruebas unitarias, de integración y pruebas de extremo a extremo en la estrategia de pruebas.
5. **Control de Versiones:**
   * Utilizar sistemas de control de versiones como Git para gestionar el código fuente y facilitar la colaboración entre los miembros del equipo. Etiquetar versiones para una fácil identificación y despliegue.
6. **Gestión de Configuración:**
   * Implementar una sólida gestión de configuración para controlar cambios en la configuración del entorno y en la infraestructura.
7. **Monitoreo y Registro (Logging):**
   * Implementar soluciones de monitoreo y registro para identificar y corregir problemas rápidamente en entornos de producción.
8. **Refactorización Continua:**
   * Priorizar la refactorización continua para mantener un código limpio y fácil de mantener. Utilizaré análisis estático de código para identificar posibles mejoras.
9. **Gestión de Dependencias:**
   * Utilizar herramientas de gestión de dependencias como npm, o NuGet para gestionar las bibliotecas y dependencias de mi proyecto.
10. **Comunicación y Colaboración:**
    * Fomentar la comunicación regular entre los miembros del equipo, los interesados. Utilizar herramientas de colaboración como Slack, Microsoft Teams o Jira para mejorar la comunicación.
11. **Documentación Efectiva:**
    * Mantendré una documentación clara y actualizada que describe la arquitectura, el código, los procedimientos de despliegue y otros aspectos importantes.

5. **Infraestructura y plataformas necesarias para soportar el desarrollo.**

* Entorno de Desarrollo:
  + -Visual Studio 2022 Visual Studio Code
  + -Windows server o linux
* Gestión de Versiones:
  + Git: Utiliza Git como sistema de control
* -Infraestructura de Servidores:
  + Servidores para Implementación:en Azure, para implementar aplicaciones .NET Core en la nube.
  + SQL Server,
  + Azure DevOps Configura pipelines de CI/CD (Integración Continua/Despliegue Continuo) para automatizar pruebas y despliegues.
* -Documentación:
  + Swagger para documentar la API de manera automática y facilitar la interacción con otros desarrolladores.
* -Autenticación y Autorización:
  + Implementar la autenticación y autorización utilizando Identity Server, JWT
* Despliegue Local:
  + Configurar un entorno de desarrollo,implementaciones locales de bases de datos y servicios.

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

1. **Riesgo: Cambios en los Requisitos:**
   * **Mitigación:** Establecer un proceso de gestión de cambios sólido y comunicar claramente los impactos de los cambios en términos de tiempo y recursos.
2. **Riesgo: Problemas de Integración:**
   * **Mitigación:** Realizar pruebas de integración frecuentes, implementar integración continua y utilizar entornos de prueba que reflejen la configuración de producción.
3. **Riesgo: Fallos en la Seguridad:**
   * **Mitigación:** Implementar prácticas de desarrollo seguro, realizar auditorías de seguridad, y utilizar herramientas de análisis de seguridad estática y dinámica.
4. **Riesgo: Retrasos en el Desarrollo:**
   * **Mitigación:** Desarrollar un plan de proyecto realista, utilizar metodologías ágiles para adaptarse a los cambios, y asignar recursos adecuadamente.
5. **Riesgo: Problemas de Rendimiento:**
   * **Mitigación:** Realizar pruebas de rendimiento durante el desarrollo, utilizar herramientas de perfilado de código, y diseñar la arquitectura considerando la escalabilidad.
6. **Riesgo: Problemas de Comunicación:**
   * **Mitigación:** Establecer canales de comunicación claros, realizar reuniones regulares de seguimiento, y utilizar herramientas de gestión de proyectos para mantener a todos informados.
7. **Riesgo: Falta de Expertise en Tecnologías Utilizadas:**
   * **Mitigación:** Capacitar al equipo en las tecnologías necesarias antes del inicio del proyecto, contratar expertos externos si es necesario, y fomentar el aprendizaje continuo.
8. **Riesgo: Cambios en el Equipo:**
   * **Mitigación:** Mantener un plan de contingencia para cambios en el personal, documentar procesos y decisiones, y fomentar la colaboración y el conocimiento compartido dentro del equipo.
9. **Riesgo: Problemas de Cumplimiento Normativo:**
   * **Mitigación:** Realizar análisis de impacto en términos de cumplimiento normativo, colaborar con expertos legales, y mantener una documentación clara y precisa.
10. **Riesgo: Problemas de Infraestructura:**
    * **Mitigación:** Utilizar proveedores de servicios en la nube confiables, implementar copias de seguridad regulares, y tener un plan de recuperación ante desastres.
11. **Riesgo: Resistencia al Cambio por Parte de los Usuarios:**
    * **Mitigación:** Involucrar a los usuarios desde el inicio del proyecto, proporcionar capacitación adecuada, y tener un plan de comunicación para explicar los beneficios del cambio.
12. **Riesgo: Problemas de Escalabilidad:**
    * **Mitigación:** Diseñar la arquitectura para escalar horizontalmente, realizar pruebas de carga y rendimiento, y monitorear continuamente el rendimiento.

Es importante revisar y actualizar regularmente la lista de riesgos a lo largo del proyecto, ajustando las estrategias de mitigación según sea necesario. La gestión proactiva de riesgos contribuirá significativamente al éxito del proyecto.

# OTROS ELEMENTOS QUE CONSIDERE NECESARIOS PARA ESTE PROYECTO.

Además de la infraestructura, las metodologías de desarrollo y la gestión de riesgos mencionadas anteriormente, hay varios elementos adicionales que son esenciales para el éxito de este proyecto:

1. **Cumplimiento Legal y Normativo:**
   * Asegurarse de que la aplicación cumpla con todas las leyes y regulaciones relevantes relacionadas con la privacidad de los datos, la seguridad y otros aspectos legales.
2. **Documentación Completa:**
   * Mantener una documentación completa y actualizada que incluya el diseño de la arquitectura, diagramas de flujo, manuales de usuario, y cualquier otro documento relevante para el desarrollo y la operación de la aplicación.
3. **Pruebas de Seguridad:**
   * Realizar pruebas de seguridad regulares para identificar posibles vulnerabilidades y asegurar que la aplicación cumpla con los estándares de seguridad.
4. **Monitoreo y Análisis de Desempeño Continuo:**
   * Establecer un sistema de monitoreo continuo para supervisar el rendimiento de la aplicación en producción y detectar posibles problemas antes de que afecten a los usuarios.
5. **Gestión de Versiones y Despliegue Continuo:**
   * Implementar un sistema robusto de gestión de versiones para rastrear los cambios en el código fuente y facilitar despliegues consistentes y seguros.
6. **Soporte y Mantenimiento Post-lanzamiento:**
   * Planificar estrategias para el soporte y mantenimiento continuo de la aplicación después del lanzamiento, incluyendo la corrección de errores, actualizaciones de seguridad y mejoras de funcionalidad.
7. **Capacitación de Usuarios:**
   * Proporcionar capacitación a los usuarios finales para garantizar que comprendan cómo utilizar la aplicación de manera eficiente y efectiva.
8. **Escalabilidad Futura:**
   * Diseñar la arquitectura de la aplicación con la escalabilidad futura en mente, considerando posibles incrementos en el número de usuarios y la carga de trabajo.
9. **Gestión Eficiente de Proyectos:**
   * Utilizar herramientas de gestión de proyectos y técnicas de seguimiento para garantizar que el equipo cumpla con los plazos y objetivos establecidos.
10. **Retroalimentación y Mejora Continua:**
    * Establecer un proceso para recopilar retroalimentación de los usuarios y el equipo de desarrollo, y utilizar esa retroalimentación para realizar mejoras continuas en la aplicación.
11. **Contingencia y Plan de Recuperación ante Desastres:**
    * Desarrollar planes de contingencia y de recuperación ante desastres para mitigar posibles interrupciones del servicio y garantizar la disponibilidad continua de la aplicación.
12. **Consideraciones de Accesibilidad:**
    * Asegurarse de que la aplicación sea accesible para personas con discapacidades, cumpliendo con los estándares de accesibilidad web.
13. **Implementación de Pruebas de Usuario:**
    * Realizar pruebas de usuario con un grupo representativo para asegurarse de que la aplicación cumple con las expectativas y necesidades de los usuarios finales.

Estos elementos adicionales contribuirán a la implementación exitosa del proyecto y al mantenimiento eficiente de la aplicación a lo largo del tiempo. Es crucial tener en cuenta estos aspectos desde las etapas iniciales del desarrollo hasta la operación continua del sistema.