Documento de Requerimientos

# Prefacio

Este documento está dirigido a desarrolladores de software, arquitectos de sistemas, ingenieros de calidad y personal técnico involucrado en la implementación y mantenimiento de la aplicación de gestión de usuarios. También es de interés para los equipos de seguridad y operaciones, ya que incluye lineamientos para la infraestructura y pautas de rendimiento y seguridad necesarias para el despliegue en producción.

La documentación ha sido creada como una guía de referencia durante todas las etapas del desarrollo, desde la planificación hasta el mantenimiento. Contiene especificaciones detalladas para la implementación de funcionalidades clave, así como para el cumplimiento de estándares de usabilidad, seguridad y compatibilidad. Se recomienda que el equipo técnico consulte este documento durante el desarrollo y lo mantenga actualizado a medida que surjan nuevos requisitos o cambios en la arquitectura o tecnologías.

**Tabla de versions**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Fecha** | **Descripción** | **Autor** |
| 1.0 | 25/10/2024 | Especificación Inicial del documento | Francisco Ram Jiménez Velásquez |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Esta tabla de versiones deberá actualizarse con cada cambio sustancial, reflejando el historial evolutivo del documento y facilitando el control de versiones para el equipo.

## Introducción

Este documento describe de manera estructurada los requisitos funcionales y no funcionales para el desarrollo de una aplicación orientada a la gestión de usuarios, asegurando que sea intuitiva, segura, rápida y compatible con diversos dispositivos y navegadores.

Para garantizar una organización clara y un desarrollo escalable, la aplicación sigue el patrón de arquitectura MVC, permitiendo una separación de responsabilidades y mejorando la mantenibilidad. Además, se implementa modularidad en frontend y backend, empleando React y .NET, lo que facilita la integración y el desarrollo en paralelo de diferentes partes de la aplicación. La aplicación se diseñará con funcionalidades CRUD (Crear, Leer, Actualizar y Eliminar), cada una de ellas con validaciones tanto en el cliente como en el servidor para prevenir errores de datos y mejorar la seguridad de la información.

El enfoque de arquitectura en capas y patrones de diseño como CQRS y la inyección de dependencias proporcionan una base sólida para el desarrollo, promoviendo un software de calidad que es fácilmente testeable y adaptable a futuras necesidades. También se prevén consideraciones para el despliegue en producción, estableciendo los requisitos de hardware, rendimiento y seguridad para garantizar la disponibilidad y la fiabilidad del sistema.

En suma, este documento guía a los desarrolladores en la implementación de una solución robusta que no solo cumple con los requerimientos técnicos actuales, sino que también está diseñada para futuras evoluciones y escalabilidad, permitiendo la integración de funcionalidades avanzadas como autenticación, compatibilidad con aplicaciones móviles y exposición de APIs a terceros.

## Glosario

1. **Requerimiento funcional**: Características específicas y funciones que el sistema debe realizar para cumplir con las necesidades del usuario, como creación, consulta, actualización y eliminación de usuarios.
2. **Requerimiento no funcional**: Atributos de calidad del sistema que no están directamente relacionados con las funciones principales, como seguridad, usabilidad, rendimiento, y compatibilidad.
3. **CRUD**: Acrónimo de **Create, Read, Update, Delete**. Operaciones básicas de creación, consulta, actualización y eliminación de datos en una aplicación.
4. **Validación**: Proceso de comprobación para asegurar que los datos ingresados cumplen con los requisitos especificados, como el formato correcto de un email o un número de teléfono.
5. **Error 400 (Bad Request)**: Código de respuesta HTTP que indica que la solicitud al servidor contiene algún error en los datos enviados.
6. **Error 404 (Not Found)**: Código de respuesta HTTP que indica que el recurso solicitado no se encuentra en la base de datos o servidor.
7. **Error 409 (Conflict)**: Código de respuesta HTTP que indica un conflicto en la solicitud, como cuando un dato, por ejemplo, un email, ya existe en el sistema.
8. **Error 500 (Server Error)**: Código de respuesta HTTP que indica un error interno en el servidor que impide procesar la solicitud correctamente.
9. **MVC (Model-View-Controller)**: Patrón de diseño arquitectónico que divide una aplicación en tres componentes: **Modelo** (gestión de datos y lógica de negocio), **Vista** (interfaz de usuario), y **Controlador** (intermediario entre la vista y el modelo).
10. **Modularidad**: Organización de una aplicación en componentes independientes y reutilizables, facilitando el mantenimiento y desarrollo del sistema.
11. **Inyección de Dependencias (DI)**: Patrón de diseño en el cual las dependencias de una clase son inyectadas desde el exterior, lo que permite desacoplar las clases y mejorar la testabilidad y mantenibilidad del código.
12. **CQRS (Command Query Responsibility Segregation)**: Patrón de diseño que separa las operaciones de lectura y escritura en una aplicación, permitiendo optimizar el rendimiento y escalabilidad al tratarlas por separado.
13. **Sanitización**: Proceso de limpiar y validar los datos de entrada para evitar inyecciones de código y ataques maliciosos.
14. **Cifrado**: Técnica de seguridad para proteger la información de los usuarios almacenada en la base de datos, de modo que solo usuarios autorizados puedan acceder a ella.
15. **OpenAPI**: Especificación para describir y documentar APIs de forma estandarizada, permitiendo la auto-generación de documentación y la interoperabilidad entre sistemas.
16. **Material Design**: Sistema de diseño visual desarrollado por Google, que establece principios y guías para el diseño de interfaces de usuario intuitivas y atractivas.
17. **Monolith First**: Estrategia de desarrollo en la que se comienza con una arquitectura monolítica para simplificar el inicio del proyecto, permitiendo su migración a microservicios cuando sea necesario.
18. **Tiempo de Respuesta**: Medida del tiempo que tarda el sistema en completar una operación de usuario, como una consulta o actualización de datos. En este caso, el objetivo es mantener las operaciones de CRUD bajo 3 segundos.
19. **Compatibilidad**: Capacidad de la aplicación para funcionar correctamente en diferentes entornos, como navegadores modernos y dispositivos de varios tamaños de pantalla.
20. **.NET y ASP.NET**: Plataformas de desarrollo de aplicaciones de Microsoft que permiten construir aplicaciones web, servicios API y otras soluciones utilizando C# y otros lenguajes compatibles.
21. **Entity Framework**: Framework de mapeo objeto-relacional (ORM) para .NET que facilita la interacción entre la aplicación y la base de datos.
22. **Dapper**: Micro ORM para .NET que permite acceder y manipular datos de bases de datos de manera eficiente y optimizada en comparación con ORM completos como Entity Framework.
23. **API Gateway**: Servicio de gestión de APIs que facilita la publicación, seguridad, monitoreo y administración de APIs en aplicaciones basadas en microservicios.
24. **Alta Disponibilidad (HA)**: Diseño de sistemas y bases de datos para asegurar su funcionamiento continuo y minimizar el tiempo de inactividad en caso de fallos.
25. **MUIReact**: Biblioteca de componentes para React basada en los principios de Material Design, usada para crear interfaces de usuario consistentes y atractivas.

## Definición de Requerimientos de Usuario

### Requerimientos funcionales

* Crear usuario:
  + El sistema debe permitir que el usuario cree nuevos registros con los atributos:
    - Dirección
    - Teléfono
    - Email
  + Flujos de error:
    - Si alguno de los campos obligatorios está vacío, el sistema debe mostrar un mensaje de error indicando que todos los campos deben ser completados.
    - Si el formato del teléfono o el email es incorrecto, el sistema debe mostrar un mensaje específico de error.
    - Si se intenta crear un usuario con un email que ya existe en la base de datos, el sistema debe impedir la creación y notificar al usuario que el email ya está registrado.
* Leer usuario:
  + El sistema debe permitir que el usuario consulte y visualice los registros existentes, mostrando los tres atributos mencionados.
  + Flujos de error:
    - Si no se encuentra el registro especificado, el sistema debe mostrar un mensaje indicando que el usuario no existe en la base de datos.
    - En caso de que haya un error de conexión al servidor o base de datos, el sistema debe mostrar un mensaje de error y sugerir intentar más tarde.
* Actualizar usuario:
  + El sistema debe permitir que el usuario actualice la información de los registros existentes, pudiendo modificar:
    - Dirección
    - Teléfono
    - Email
  + Flujos de error:
    - Si el usuario deja algún campo vacío durante la actualización, el sistema debe mostrar un mensaje indicando que todos los campos deben ser completados.
    - Si se ingresa un formato inválido en el teléfono o email, el sistema debe mostrar un mensaje de error específico.
    - Si el email actualizado ya existe en otro registro, el sistema debe notificar que el email no puede duplicarse.
    - Si el registro a actualizar no se encuentra, el sistema debe notificar que el usuario no existe.
* Eliminar usuario:
  + El sistema debe permitir que el usuario elimine registros existentes.
  + Flujos de error:
    - Si el registro a eliminar no existe, el sistema debe mostrar un mensaje indicando que no se encuentra el usuario.
    - En caso de que haya un error de conexión al servidor o base de datos, el sistema debe mostrar un mensaje de error y sugerir intentar más tarde.

### Requerimientos no funcionales

* **Usabilidad**:
  + **Interfaz de usuario:** La aplicación debe ser intuitiva y fácil de navegar, con una interfaz limpia y accesible que permita al usuario encontrar las funciones de creación, consulta, actualización y eliminación de manera rápida y clara.
  + **Ayuda:** La aplicación debe contar con mensajes informativos para guiar al usuario en caso de errores o dudas.
* **Seguridad**:
  + **Sanitización:** Todos los datos ingresados deben ser sanitizados para evitar inyecciones de código o datos maliciosos, especialmente en los campos de email y teléfono.
  + **Protección de datos**: La información de los usuarios, incluyendo dirección, teléfono y email, debe estar encriptada en la base de datos para proteger la privacidad de los usuarios.
* **Rendimiento**:
  + **Tiempo de respuesta:** Las operaciones de CRUD deben ejecutarse en menos de 3 segundos en condiciones normales de carga.
* **Compatibilidad**:
  + **Navegadores y dispositivos:** La aplicación debe ser compatible con los navegadores web modernos (Chrome, Firefox, Safari, Edge) y debe adaptarse a diferentes tamaños de pantalla, incluidos dispositivos móviles y de escritorio.

## Arquitectura de Sistema

* **Modelos de Arquitectura**:
  + **Arquitectura MVC**:
    - Descripcion: El patrón Model-View-Controller (MVC) se utiliza para separar las responsabilidades dentro de la aplicación.
      * **Modelo**: Maneja la lógica de negocio y la manipulación de datos.
      * **Vista**: Se encarga de la presentación de la interfaz de usuario.
      * **Controlador**: Actúa como intermediario entre el Modelo y la Vista, gestionando las interacciones del usuario.
    - Justificación: Este enfoque permite una organización clara del código, facilitando el mantenimiento y la escalabilidad de la aplicación. Al separar las preocupaciones, los desarrolladores pueden trabajar en paralelo en distintas partes del sistema, lo que acelera el tiempo de desarrollo y mejora la calidad del software y ya hay componentes pre existentes disponibles para implementar.
  + **Modularidad en Frontend y Backend**:
    - Descripción: La arquitectura modular permite la creación de componentes independientes en el Frontend y el Backend, promoviendo la reutilización y la facilidad de mantenimiento**.**
    - Justificación: Al utilizar React, se pueden construir interfaces de usuario dinámicas y reactivas mediante componentes reutilizables. Esto no solo mejora la experiencia del usuario, sino que también simplifica el proceso de desarrollo al permitir la implementación de cambios en componentes individuales sin afectar al resto de la aplicación. La modularidad en el Backend facilita la implementación de servicios y funciones específicas, permitiendo que los desarrolladores se concentren en áreas individuales del sistema. Esto reduce el acoplamiento y mejora la capacidad de escalar la aplicación en el futuro. La combinación con la plataforma .NET ofrece herramientas y librerías que optimizan la implementación de esta arquitectura.
  + **Arquitectura en Capas**:
    - Descripción: La arquitectura en capas organiza el sistema en diferentes niveles, donde cada capa tiene una responsabilidad específica y se comunica únicamente con las capas adyacentes. Generalmente, se pueden identificar las siguientes capas:
      * **Capa de Presentación**: Interactúa con el usuario y muestra los datos.
      * **Capa de Lógica de Negocio**: Contiene las reglas y procesos del negocio.
      * **Capa de Acceso a Datos**: Se encarga de la interacción con la base de datos y la persistencia de la información.
    - Justificación: Este enfoque modulariza el desarrollo, permitiendo a los equipos trabajar de manera independiente en cada capa. Esto facilita el mantenimiento y las pruebas, ya que las modificaciones en una capa pueden realizarse sin impactar directamente en las demás. Además, la arquitectura en capas mejora la seguridad al encapsular la lógica de negocio y el acceso a datos, permitiendo implementar medidas de seguridad de manera más efectiva. Al combinar esta arquitectura con .NET, se aprovechan las herramientas y librerías disponibles para facilitar el desarrollo y la integración de cada capa de manera eficiente.
  + **Monolith First:**
    - Descripción: La estrategia Monolith first nos permite crear un monolito en las primeras etapas de la aplicación a pesar de pensar que se podría ver beneficiada por un enfoque de micro servicios evitando el trabajo extra de configuración entre ellos. Sin embargo, con las segregaciones adecuadas, permite separarlo fácilmente cuando se requiera una implementación de micro servicios.
    - Justificación: La aplicación a desarrollar es muy sencilla, por lo que el trabajar en micro servicios representaría un esfuerzo mayor al beneficio que se puede obtener de ello en etapas tempranas del desarrollo. Sin embargo, ante el posible crecimiento, nos permite separarlo tomando en cuenta tiempos de configuración.
* **Patrones de diseño**
  + **Patron CQRS (Command Query Responsability Seggregation)**:
    - Descripción: El patrón CQRS separa las operaciones de lectura (queries) de las de escritura (commands) en una aplicación. En lugar de utilizar una única modelo para manejar tanto la lectura como la escritura, CQRS crea dos modelos distintos:
      * **Comandos**: Son responsables de cambiar el estado del sistema. Estos son operaciones que modifican datos.
      * **Consultas**: Se encargan de recuperar información del sistema sin modificar su estado.
    - Justificación: Esta separación permite optimizar el rendimiento y la escalabilidad de la aplicación. Al tratar las consultas y los comandos por separado, se pueden adaptar y escalar de manera independiente, lo que mejora la eficiencia en la gestión de recursos. Además, CQRS puede facilitar la implementación de patrones como el Event Sourcing, donde los cambios en el estado del sistema se almacenan como una serie de eventos. Esto no solo proporciona una mejor trazabilidad de los cambios, sino que también permite recuperar el estado del sistema en cualquier momento. Integrar CQRS con .NET ofrece herramientas que simplifican su implementación y permiten aprovechar los beneficios de esta arquitectura.
  + **Inyección de Dependencias:** 
    - Descripción: La inyección de dependencias es un patrón de diseño que permite a un objeto recibir sus dependencias de otros objetos en lugar de crearlas por sí mismo. Este enfoque promueve una arquitectura más limpia y desacoplada al permitir que las clases sean menos dependientes de implementaciones concretas.
    - Justificación: La inyección de dependencias mejora la mantenibilidad y la testabilidad del código. Al desacoplar las clases, se facilita la sustitución de implementaciones sin necesidad de modificar el código que las utiliza. Esto resulta especialmente útil en pruebas unitarias, donde se pueden inyectar dependencias simuladas (mocks) para verificar el comportamiento de una clase sin depender de sus implementaciones reales. Además, la inyección de dependencias fomenta la adherencia al principio de inversión de dependencias, que es uno de los principios SOLID, lo que conduce a un diseño de software más robusto y flexible. En entornos .NET, frameworks como ASP.NET Core incluyen soporte nativo para la inyección de dependencias, lo que simplifica su implementación y uso en aplicaciones.

## Definición de Requerimientos de Sistema

### Requerimientos funcionales

* Crear usuario:
  + Entrada de datos:
    - El sistema debe proporcionar un formulario para la entrada de datos con los campos: dirección, teléfono y email.
  + Validaciones:
    - Validaciones de lado del cliente (React) y Servidor para asegurar que los campos obligatorios estén completos y en el formato correcto.
    - Al enviar la solicitud al servidor (ASP.NET):
      * **400 Bad Request:** Si alguno de los campos obligatorios está vacío o si el formato del teléfono o email es incorrecto.
      * **409 Conflict:** Si el email ya existe en la base de datos, indicando un conflicto de duplicación.
      * **500 Server Error:** En caso de errores genéricos no controlados por el sistema, se devolverá este código con un mensaje de intentar mas tarde.
* Consultar usuario:
  + Consulta de Registros:
    - Proporcionar una interfaz para listar y consultar los datos de cada usuario registrado.
    - Al encontrar el usuario, mostrar botones de edición y eliminación para flujos posteriores.
  + Validaciones:
    - Validaciones de lado del cliente (React) y Servidor para asegurar que el campo de consulta sea un Id numerico.
    - Al enviar la solicitud al servidor (ASP.NET):
      * **404 Not Found:** Si el usuario solicitado no se encuentra en la base de datos.
      * **409 Conflict:** Si el email ya existe en la base de datos, indicando un conflicto de duplicación.
      * **500 Server Error:** En caso de errores genéricos no controlados por el sistema, se devolverá este código con un mensaje de intentar mas tarde.
* Actualizar usuario:
  + Entrada de datos:
    - Proporcionar un formulario prellenado con la información del usuario para facilitar su modificación.
  + Validaciones:
    - Validaciones de lado del cliente (React) y Servidor para asegurar que los campos obligatorios estén completos y en el formato correcto.
    - Al enviar la solicitud al servidor (ASP.NET):
      * **400 Bad Request:** Si alguno de los campos obligatorios está vacío o si el formato del teléfono o email es incorrecto.
      * **404 Not Found:** Si el usuario que se quiere actualizar no se encuentra en la base de datos.
      * **409 Conflict:** Si el email ya existe en la base de datos, indicando un conflicto de duplicación.
      * **500 Server Error:** En caso de errores genéricos no controlados por el sistema, se devolverá este código con un mensaje de intentar mas tarde.
* Eliminar usuario:
  + Eliminación por Id:
    - Permitir que el usuario seleccione y elimine un registro específico.
  + Validaciones:
    - Validaciones de lado del cliente (React) y Servidor para asegurar que el campo de consulta sea un Id numerico.
    - Al enviar la solicitud al servidor (ASP.NET):
      * **404 Not Found:** Si el usuario que se quiere actualizar no se encuentra en la base de datos.
      * **500 Server Error:** En caso de errores genéricos no controlados por el sistema, se devolverá este código con un mensaje de intentar mas tarde.

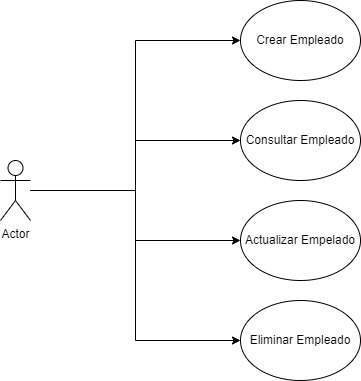
### Requerimientos no funcionales

* **Usabilidad**:
  + **Interfaz de usuario:** La aplicación debe cumplir con los siguientes criterios de usabilidad:
    - No tener mas de tres clicks para ejecutar una acción desde su inicio hasta su completez, exceptuando flujos excepcionales.
  + **Ayuda:** Los siguientes elementos deben tener información de su uso:
    - **Inputs:** Deben tener outlines que describan que debe ingresarse. Así mismo, mostrar mensajes de validación cuando no se cumplan los criterios.
    - **Botones:** Deben tener tooltips que describan brevemente la acción a realizar.
* **Diseño Gráfico**:
  + **Estándares de Diseño:** La intefaz debe estar estandarizada bajo los conceptos y diseño de componentes de Material Design.
* **Seguridad**:
  + **Sanitización:** Se deben utilizar métodos de sanitización de entradas a la base de datos y de logs de registros.
  + **Protección de datos**: La información personal debe permanecer cifrada en la base de datos, con un método síncrono o asíncrono. La llave para el cifrado no debe ser *hardodeada* en el código o en archivos de configuración. Debe ser obtenida por variables de entorno o un manejador de secretos.
* **Rendimiento**:
  + **Tiempo de respuesta:** Las operaciones de CRUD deben ejecutarse en menos de 3 segundos en condiciones normales de carga. Para esto se deben ocupar técnicas de Queries precompilados de Entity Framework, uso de micro-ORM como Dapper o llamadas con el cliente puro a la base de datos para optimizar el tiempo de respuesta de la base de datos según sea necesario el caso y las métricas necesarias para cada proceso.
* **Compatibilidad**:
  + **Navegadores y dispositivos:** Compatible con navegadores modernos y adaptado a pantallas móviles y de escritorio usando diseño responsivo.
* **Plataforma:**
  + **Lenguaje de Desarrollo:** Se utilizará el lenguaje de OOP C# para el desarrollo del aplicativo en servidor y el lenguaje Typescript con el framework NextJS y el framework de componentes MUIReact ya que se cuenta con libertad de elección de tecnologías y se tiene tanto experiencia como componentes previamente desarrollados en ambas tecnologías.

## Modelos de Sistema

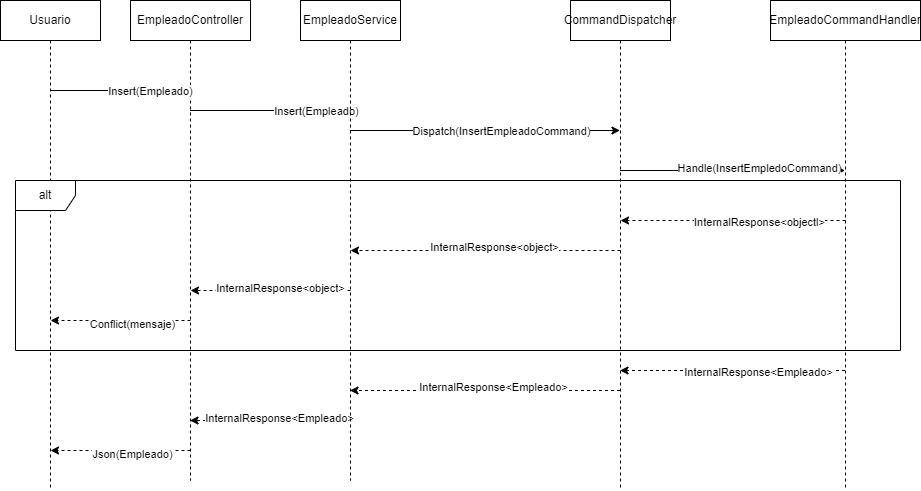
Para ejemplificar los flujos, interacciones y estructura se realizaron los siguientes diagramas:

### Diagrama de casos de uso:

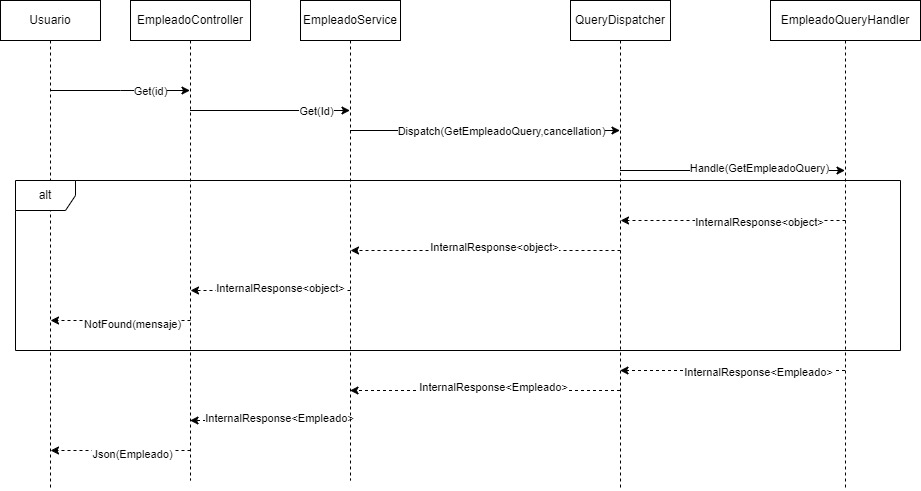


### Diagramas de Secuencia

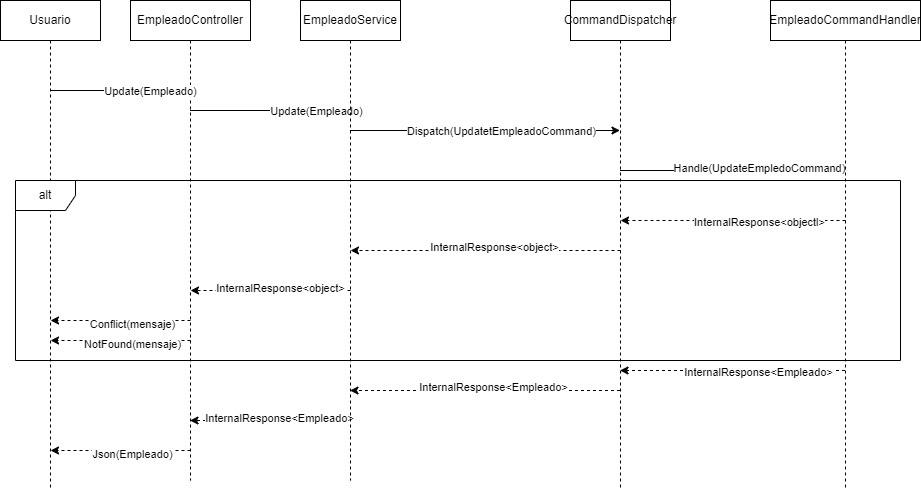
#### Create



#### Read



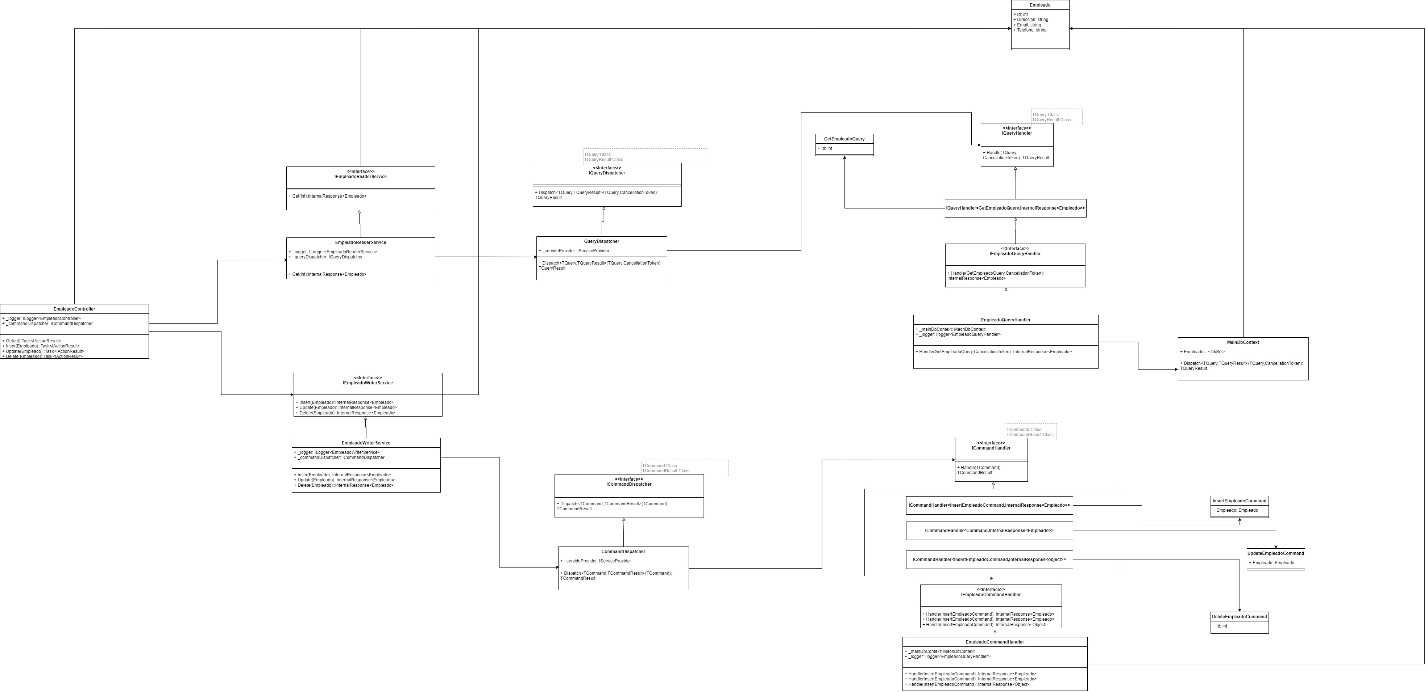
#### Update



#### Delete

### 

### Diagrama de Clases



Nota: Todos los diagramas aquí expuestos pueden consultarse como imágenes, en la misma carpeta de éste archivo, así como en su formato fuente para el programa de modelado Draw.io

## Evolución de Sistema

En base a las funciones solicitadas, se prevee que el podría verse en la necesidad de implementar lo siguiente:

* Autenticación: Para asegurar que el aplicativo y el api a consumir sean seguros, es probable que se requiera un sistema de autenticación. Para ello, ASP ya tiene implementadas extensiones que permiten agregar distintos autenticadores. Tanto los nativos de .NET como Identity, como externos como Okta y Cognito. Por lo que se debe tomar en consideración esto como primer punto de partida para su implementación.
* Compatibilidad a Aplicaciones Moviles: En caso de que se desee extender el uso del aplicativo a otros entornos como el desarrollo móvil nativo, éste debe ser implementado mediante el consumo del API ya construido.
* Exposición de API a terceros: En caso de necesitar publicar el API para el consumo de externos al aplicativo, .NET permite la auto generación de documentación mediante la especificación OpenAPI y métodos para complementarla con ejemplos de solicitudes y posibles códigos de respuesta.
* Migración a plataformas de Nube: En caso de querer escalar la aplicación a entornos de nube, la misma implementación de .NET de OpenApi permite agregar metada necesaria para hacer importaciones a servicios de exposición de API como AWS APIGateway. Esto ayudaría considerablemente a todos los procesos de entrega continua.

## Apéndice

**Requerimientos recomendados para producción**

1. **Sistema operativo**:
   * **Windows Server 2019 o 2022** (recomendado para entornos empresariales) o una distribución Linux optimizada para servidores como **Ubuntu 20.04 LTS** o superior.
2. **Procesador**:
   * **CPU de 2.5 GHz** o superior, idealmente con **4 núcleos físicos o más**.
   * Procesadores Xeon o AMD EPYC en servidores físicos o un equivalente de alto rendimiento en entornos de nube.
3. **Memoria RAM**:
   * Mínimo **16 GB de RAM** para aplicaciones de carga media, pero **32 GB o más** si esperas un alto volumen de transacciones.
4. **Almacenamiento**:
   * **Unidad SSD NVMe** (ideal para un rápido acceso a datos).
   * Espacio mínimo de **20 GB** para .NET y SQL Server, pero es preferible contar con al menos **100 GB** para escalabilidad y manejo de backups.
5. **SQL Server**:
   * **SQL Server 2019 o 2022 Standard o Enterprise Edition**, según las necesidades.
   * Configurar almacenamiento en disco adicional para registros de transacciones, copias de seguridad y archivos de datos.
6. **Respaldo y Alta disponibilidad**:
   * **Replica de base de datos o cluster** (por ejemplo, Always On Availability Groups en SQL Server) para mayor resiliencia.
   * Implementar **backups automatizados** y sistemas de recuperación de desastres.
7. **Seguridad y Red**:
   * **Cifrado SSL/TLS** para conexiones a SQL Server.
   * **Firewall y segmentación de red** para proteger el acceso.
   * **VPN o redes privadas virtuales** para acceso seguro.
8. **Consideraciones de Red**:
   * Ancho de banda de **100 Mbps** o superior para conexiones de alto rendimiento entre aplicaciones y bases de datos remotas.

Index

[Prefacio 1](#_Toc180775759)

[Introducción 1](#_Toc180775760)

[Glosario 2](#_Toc180775761)

[Definición de Requerimientos de Usuario 4](#_Toc180775762)

[*Requerimientos funcionales* 4](#_Toc180775763)

[*Requerimientos no funcionales* 5](#_Toc180775764)

[Arquitectura de Sistema 5](#_Toc180775765)

[Definición de Requerimientos de Sistema 8](#_Toc180775766)

[*Requerimientos funcionales* 8](#_Toc180775767)

[*Requerimientos no funcionales* 9](#_Toc180775768)

[Modelos de Sistema 10](#_Toc180775769)

[Evolución de Sistema 13](#_Toc180775770)

[Apéndice 14](#_Toc180775771)