**Sistema de Gestión de Tickets**

**Eventia S.A.S.**

**Gestión de eventos corporativos y académicos**

Preparado por [Ever Alonso Torres Galeano](mailto:eeatg844@hotmail.com)

Contenido

[Sistema de Gestión de Tickets 4](#_Toc211592794)

[Descripción del Proyecto 4](#_Toc211592795)

[Visión General del Negocio 4](#_Toc211592796)

[Identificación del problema 4](#_Toc211592797)

[Objetivos y Alcance 5](#_Toc211592798)

[Objetivos 5](#_Toc211592799)

[1. Soporte Operativo Interno 5](#_Toc211592800)

[2. Seguimiento de Incidentes o Requerimientos 5](#_Toc211592801)

[3. Trazabilidad y Auditoría 5](#_Toc211592802)

[Requisitos 5](#_Toc211592803)

[Requerimientos Funcionales 5](#_Toc211592804)

[1. Gestión de Tickets: 5](#_Toc211592805)

[2. Gestión de Usuarios: 6](#_Toc211592806)

[3. Interfaz de Usuario: Aplicación web de una sola página (SPA) para: 6](#_Toc211592807)

[4. Interoperabilidad: 6](#_Toc211592808)

[Requisitos No Funcionales 6](#_Toc211592809)

[Seguridad 6](#_Toc211592810)

[Rendimiento 6](#_Toc211592811)

[Usabilidad 6](#_Toc211592812)

[Portabilidad 7](#_Toc211592813)

[Disponibilidad 7](#_Toc211592814)

[Solución Propuesta 7](#_Toc211592815)

[Componentes Principales 7](#_Toc211592816)

[Tecnología y Patrones 8](#_Toc211592817)

[Consideraciones de Diseño 9](#_Toc211592818)

[Diagramas 10](#_Toc211592819)

[Diagrama de Arquitectura (C4) 10](#_Toc211592820)

[Diagrama de Despliegue (UML) 13](#_Toc211592821)

[Repositorio 13](#_Toc211592822)

[GitHub 13](#_Toc211592823)

[Documentación Para Uso de Los Servicios API 13](#_Toc211592824)

[Readme.md File 13](#_Toc211592825)

[Swagger 14](#_Toc211592826)

[Visual Studio Endpoint Explorer 14](#_Toc211592827)

[Http File 15](#_Toc211592828)

# Sistema de Gestión de Tickets

# Descripción del Proyecto

## Visión General del Negocio

Eventia S.A.S. es una empresa que se dedica a la organización de eventos presenciales, híbridos y virtuales. En los últimos dos años ha experimentado un crecimiento sostenido que ha generado la necesidad de modernizar su plataforma tecnológica. Actualmente, el equipo operativo de Eventia gestiona la asignación de tickets, seguimiento de usuarios registrados y coordinación logística a través de múltiples hojas de cálculo y herramientas manuales, lo que ha generado errores frecuentes, pérdida de información, retrasos en la atención y falta de trazabilidad.

Con el fin de escalar su operación y garantizar una mejor experiencia para los usuarios internos y externos, la compañía ha decidido iniciar el desarrollo de una nueva plataforma digital que le permita centralizar la gestión de tickets, habilitar un acceso unificado y seguro para usuarios, y facilitar la integración con futuros módulos operativos (ej. manejo de agenda, salas, proveedores, etc.).

El sistema no se limitará a la gestión básica de incidencias, sino que propone una plataforma flexible de control operativo, que centralizará múltiples tipos de solicitudes, automatizará parte del flujo de atención, mejorará la trazabilidad y se adaptará a diferentes actores dentro del ecosistema de gestión de eventos.

## Identificación del problema

El crecimiento de Eventia ha superado la capacidad operativa de sus herramientas actuales. Los flujos manuales, el uso de múltiples plataformas desconectadas y la falta de trazabilidad sobre los tickets generados afectan la eficiencia interna y la calidad del servicio percibida por los clientes. Además, la incorporación de nuevos empleados y equipos ha generado retos en la administración de usuarios y en la escalabilidad técnica del sistema. La nueva plataforma busca resolver estos desafíos ofreciendo una base tecnológica moderna, segura y escalable que siente las bases para la transformación digital de la empresa.

# Objetivos y Alcance

## Objetivos

Tipo de Gestión de Tickets

### 1. Soporte Operativo Interno

Los tickets funcionarán como unidades de trabajo internas para organizar y rastrear requerimientos específicos dentro de la operación de eventos. Por ejemplo:

• Solicitudes logísticas (montaje, catering, transporte).

• Seguimiento de actividades programadas para el evento.

• Asignación de responsabilidades a agentes o supervisores.

### 2. Seguimiento de Incidentes o Requerimientos

El sistema también gestionará tickets que representen incidentes, reclamos o peticiones específicas de clientes internos o externos, tales como:

• Problemas con el registro de asistentes a un evento.

• Cambios de último momento en agenda o localización.

• Solicitudes de cambio de participante o reenvío de entradas digitales.

### 3. Trazabilidad y Auditoría

Cada ticket permitirá mantener un historial claro de:

• Acciones realizadas (creación, actualización, cambio de estado).

• Usuario responsable en cada etapa.

• Notificaciones generadas hacia otras áreas o módulos del sistema.

Esto garantiza para la operación de Eventia transparencia, control operativo y capacidad de análisis posterior.

# Requisitos

## Requerimientos Funcionales

### 1. Gestión de Tickets:

a) Crear, listar, actualizar y eliminar tickets asociados a eventos.

b) Asignar cada ticket a un usuario específico.

c) Cambiar el estado de los tickets (abierto, en proceso, cerrado).

d) Notificar cambios importantes en los tickets mediante eventos internos del sistema.

### 2. Gestión de Usuarios:

a) Registro y autenticación de usuarios con roles diferenciados (agente, supervisor, administrador). b) Consultar información básica de los usuarios (nombre, correo, rol).

c) Posibilidad de deshabilitar usuarios sin eliminarlos del sistema.

### 3. Interfaz de Usuario: Aplicación web de una sola página (SPA) para:

a) Acceder mediante inicio de sesión seguro.

b) Visualizar tickets según el rol del usuario.

c) Realizar operaciones CRUD sobre los tickets (según permisos).

d) Navegar entre vistas protegidas por autorización.

### 4. Interoperabilidad:

a) Posibilidad de que otros módulos futuros consuman o produzcan datos desde y hacia el sistema de gestión de tickets.

## Requisitos No Funcionales

### Seguridad

* La aplicación contará con roles de usuario donde cada role tendrá unos permisos específicos para acceder a las diferentes funcionalidades.
* Los usuarios deberán estar Autenticados para poder acceder al sistema.
* Las contraseñas debes ser entre 8 -12 caracteres incluyendo letras números y caracteres especiales
* El sistema cerrara la sesión después de un periodo de 30 minutos de inactividad

### Rendimiento

* Las paginas del sistema deber cargarse en menos de 3 segundos hasts con 5000 visitas simultaneas.
* La información que no cambia frecuentemente será almanecada en cache generar respuestas en tiempos mas cortos.

### Usabilidad

* La interfaz debe ser intuitiva y fácil de usar para administradores y usuarios finales.
* La aplicación debe contar con tooltips informativos en cada funcionalidad.

### Portabilidad

* El sistema debe ser compatible múltiples plataformas y dispositivos.

### Disponibilidad

* Debe haber mínimos periodos de tiempo para las ventanas de mantenimiento.

# Solución Propuesta

## Componentes Principales

Dado la problemática actual de la compañía se propone una solución basada en ecosistemas Web Microsoft de alta disponibilidad y concurrencia donde contaremos con dos grandes Módulos:

Una aplicación Web Grafica utilizando Framework de front-end de ultima generación como lo es Reatc para crear una aplicación Responsiva que se adapta fácilmente a los tamaños de pantalla: desde móviles hasta Ordenadores, compatible con entre diferentes navegadores visualmente atractiva e intuitiva para el usuario, que limitara el acceso a solo usuarios autenticados y solo permitirá acceder al usuario a las diferentes funcionalidades para las cuales posea permisos de acceso.

Para el back-end se utilizara una aplicación Asp.Net que ofrece una aplicación Web de Api ligera (Minimal API), ideal para microservicios , donde cada punto de conexión (endpoint) estará bajo un protocolo seguro Https y cada petición contara con una validación de Autenticación , para lo cual se contara con un Servidor Ligero de Identidad creado con Json Web Token, El usuario inicia ingresando el login, password y el endpoint de Login generara un token Bearer de Autenticación con una vigencia de 30 minutos(Configurable),el token que deberá ser enviado en la cabecera (header) de cada solicitud , de lo contario se obtendrá un condigo de no Autorizado para acceder la información (401 Unauthorized), Finalmente se contara con una Base de Datos Relacional Cloud como lo es Azure SQL Server y un sistema de gestión de Cache de cache Distribuida: Azure Redix para la información que no cambia frecuentemente y de esta manera ofrecer un mejor rendimiento en los tiempos de respuesta.

La gestión de correos se realizará enviando cada mensaje a una Cola de Servicio (Azure Service queue) donde se almacenarán temporalmente y se complementará con un Azure Function que al extraer cada mensaje de la cola disparará una lógica de envío de correo para evitar un cuello de botella con los envíos masivos de correos .

Para finalmente utilizar Azure Portal como Iaas (Infrastruture as a Service) para hospedar Tanto el sitio Web de back end con la Api, Los despliegues se realizaran de manera automática (Aunque con un workflow de autorización) mediante Azure Pipeline tanto para el CI como para CD en los diferentes ambientes tanto inferiores como los son Dev/QA, así como los ambientes superiores como UAT y PROD.

## Tecnología y Patrones

|  |  |  |  |
| --- | --- | --- | --- |
| **Tipo** | **Nombre** | **Descripción** | **Versión/Proveedor** |
| Patrón | Dependency Injection | es un patrón de diseño de software donde un objeto no crea sus propias dependencias, sino que estas se le proporcionan desde una fuente externa | - |
| Patrón | Microservicios | estilo de arquitectura para desarrollar aplicaciones que divide un gran sistema en servicios pequeños, independientes y especializados | - |
| Patrón | Arquitectura Limpia | patrón de diseño de software que organiza el código en capas para desacoplar la lógica de negocio de los detalles técnicos y de infraestructura | - |
| Patrón | Minimal API | un enfoque de programación en ASP.NET Core que permite crear servicios web de forma rápida y sencilla, utilizando código y dependencias mínimas. | - |
| Tecnología | . Net | Framework | 8 LTS |
| Tecnología | Swagger Doc | Api Documentation | 8.1.4 |
| Tecnología | JWToken | Json Web Token | 8.0.21 |
| Tecnología | Azure SQL Server | Paas | 20 |
| Tecnología | Azure DevOps | Paas | - |
| Tecnología | Entity Framework | ORM | 8.0.21 |
| Tecnología | CPM | Central Package Manager | - |
| Patrón | Option Pattern | Patron de diseño que utiliazd clases fuertemente tipadas para representar grupos de configuraciones | - |
| Tecnología | Redis Cache | Cache Dsitribuida | - |
| Tecnología | Azure Message Queue | Cola de Mensajes | - |
| Tecnología | Azure Function | servicio ligero orientado a eventos que auto-gestiona la infraestructura | - |

## Consideraciones de Diseño

Luego de Para La situación planteada se opta por el uso de un patrón de diseño de Arquitectura Hexagonal para mantener un mínimo acoplamiento entre la capas del la aplicación Combinado con el Patrón de Inyección de Dependencias para controlar la inversión de control enfocado a microservicios para descomponer una gran aplicación en una colección de pequeños y ligeramente acoplados servicios que pueden ser desarrollados desplegados y escalados de manera independiente brindando un aislamiento de las funcionalidades permitiendo que una acción (Falla/ Ajuste/despliegue ) en un componente especifico no afecte todo el sistema.

Utilizaremos en la Capa de Presentación: Minimal API para obtener un alto rendimiento por ser un componente ligero y de baja latencia que es muy eficiente con la gestión de middleware para ser utilizado en microservicios , Gestionando las Configuraciones con Option Pattern para mejor verificación dado que nos permite inyectar configuraciones y centralizar configuraciones fuertemente tipadas así como los orígenes de cada configuración, Como repositorio de datos persistente utilizaremos el confiable Azure SQL Server que soportara una Base de datos Cloud Transaccional y robusta y eficiente para grandes volúmenes de Datos, que nos permite un Sistema de Tablas Versionado para una optima gestión de Datos Históricos que brinda un acceso eficiente a los datos antiguos y asi obtener una ordenada trazabilidad de los cambios realizado en cada registro.

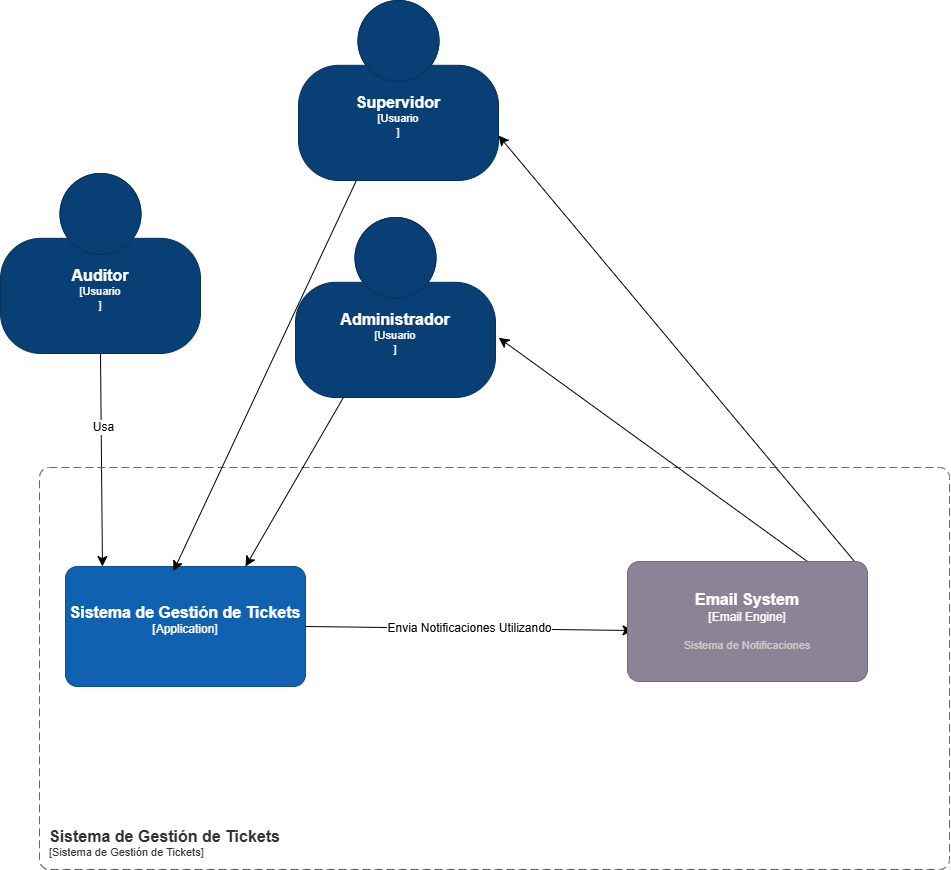
Para La capa de infraestructura Utilizaremos como ORM (Object Relational Mapping o Mapeo de objetos de datos relacionales) Entity Framework ya que ofrece un proveedor dedicado para SQL Server en ecosistemas Dot Net que combinados ofrecen una robusta escalable y segura solución de base de datos en la nube que finalmente sera soportado por el marco de trabajo Dot Net Framework 8 LTS (Long Term Support) que nos ofrece un gran performance una gestión de Dependency Injection de primer Nivel enfocado en la estabilidad que permite una mejor planeación a largo plazo para los equipos de tecnología y product owners al ofrecer un entorno consistente por significativos periodos de tiempo, con un ecosistema robusto y soporto técnico empresarial que brinda actualización para temas críticos de seguridad generando confiabilidad y protección contra vulnerabilidades, Para el Manejo de Paquetes (NuGet) se utiliza CPM (Central Package Manager) para gestionar de manera centralizada las versiones de los paquetes utilizados en cada proyecto de la solución.

La combinación de una arquitectura limpia, DI, Minimal API con Option Pattern y microservicios en una plataforma .NET LTS moderna ofrece una arquitectura práctica y útil, comprobable, escalable que reduce el costo de mantenimiento a largo plazo, a su vez que permite aumentar la velocidad del equipo generando una eficiencia operativa.

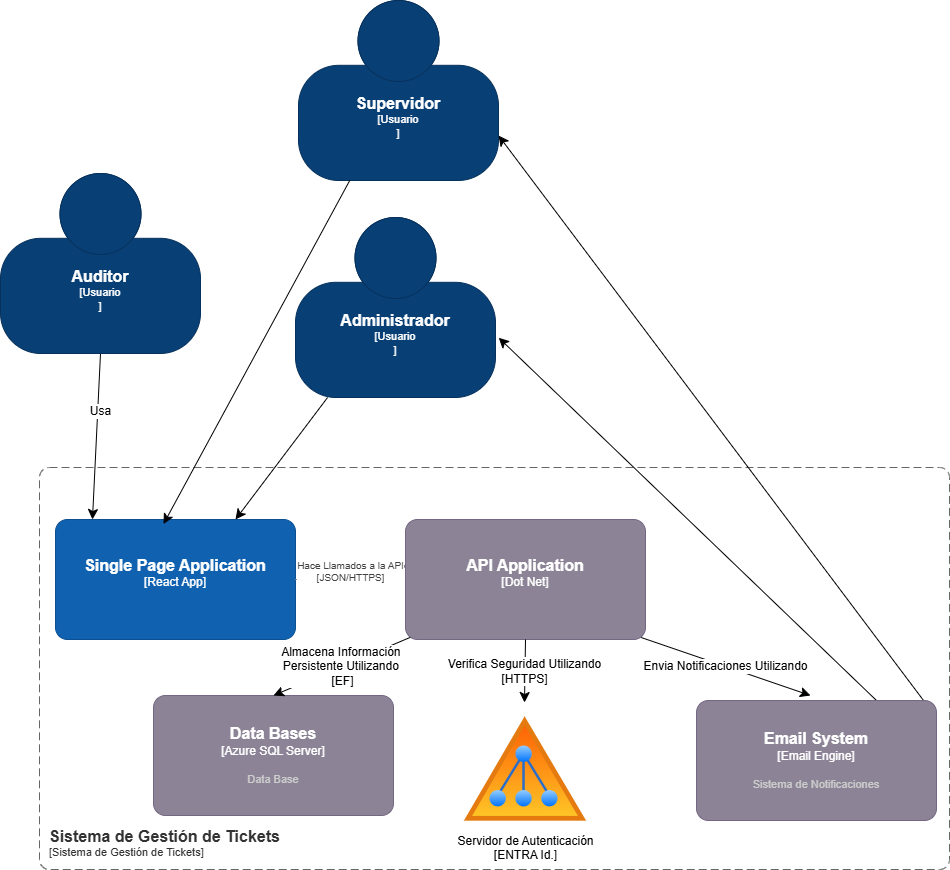
## Diagramas

### Diagrama de Arquitectura (C4)

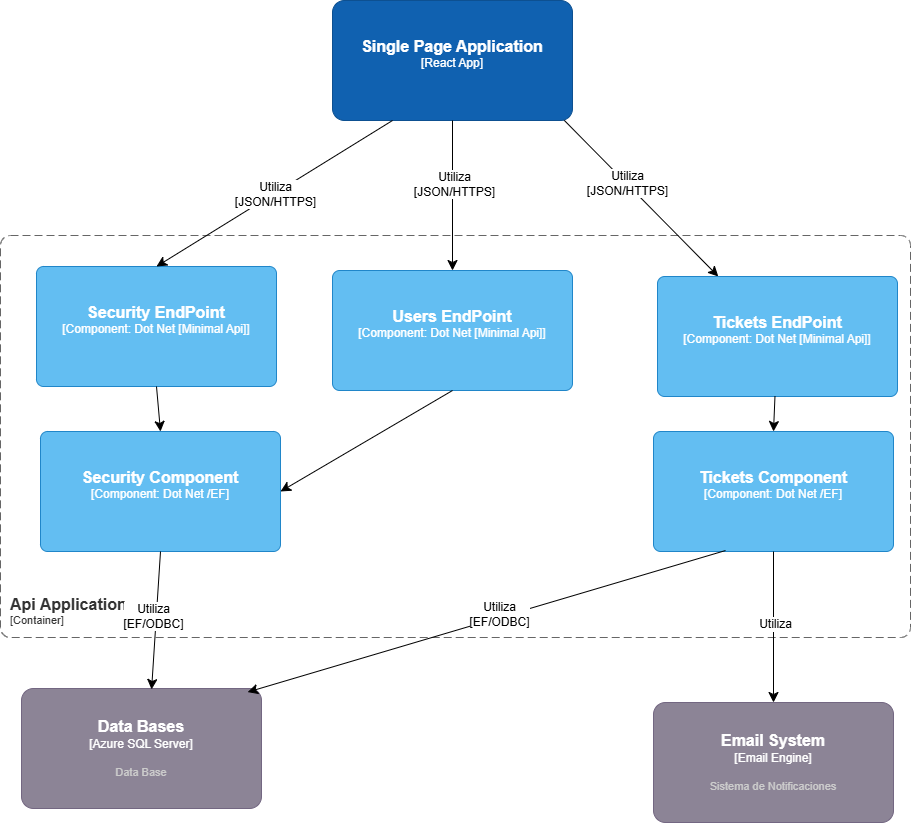
#### Contexto



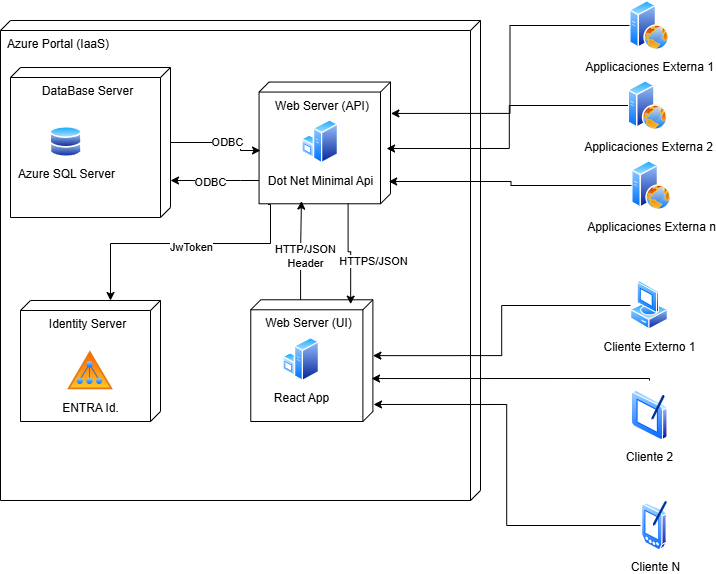
#### Contenedor



#### Componente



### Diagrama de Despliegue (UML)



# Repositorio

## GitHub

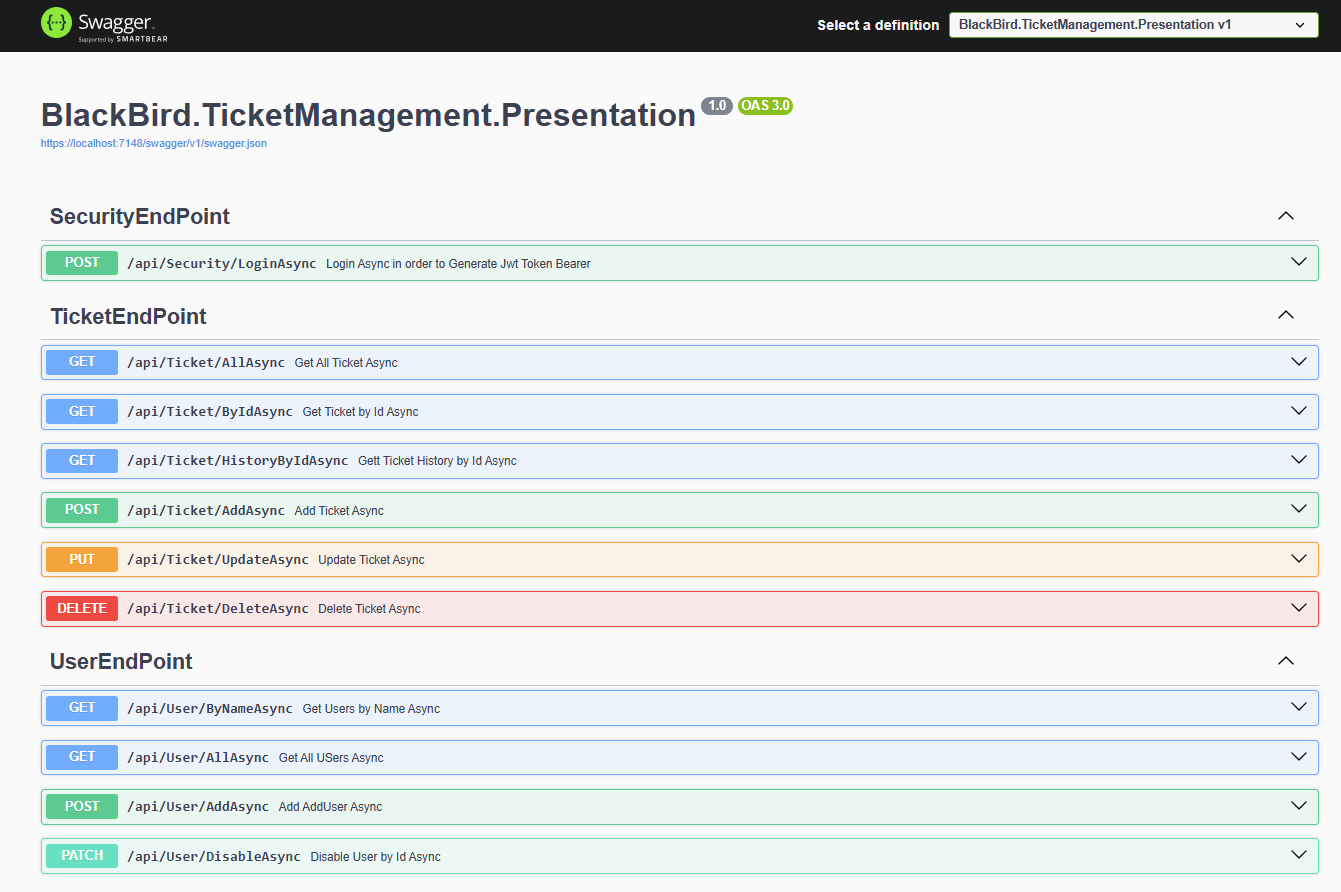
https://github.com/EverAlonsoTorresGaleano

# Documentación Para Uso de Los Servicios API

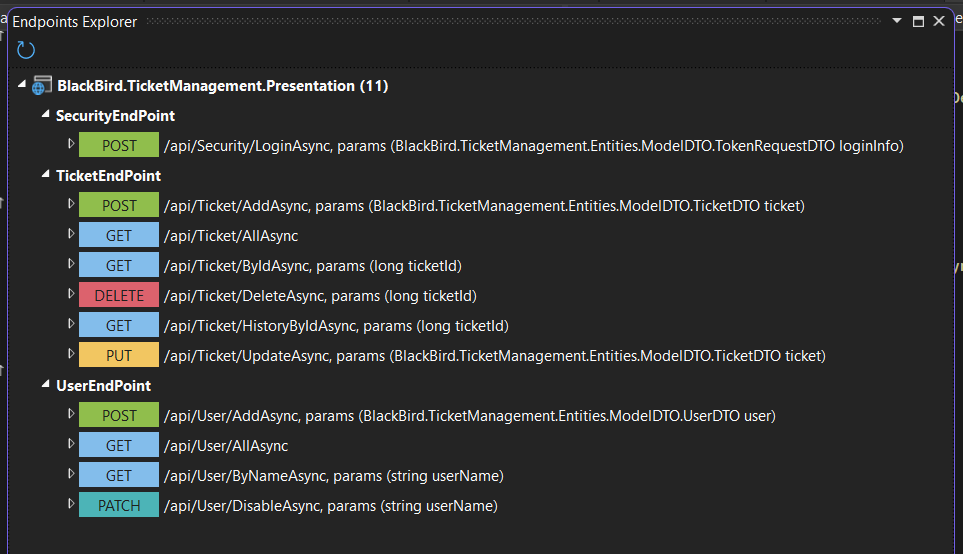
## Readme.md File

Ubicado en BlackBird\BlackBird.TicketManagement\Readme.md

## Swagger



## Visual Studio Endpoint Explorer



## Http File

Ubicado en BlackBird\BlackBird.TicketManagement\BlackBird.TicketManagement.Presentation

Archivo BlackBird.TicketManagement.Presentation.http

@BlackBird.TicketManagement.Presentation\_HostAddress = https://localhost:7148

######################

###SECURITY END POINT

######################

### Login Async

POST {{BlackBird.TicketManagement.Presentation\_HostAddress}}/api/Security/LoginAsync

Content-Type: application/json

{

"grant\_type": "ClientCredentials",

"scope": "",

"application\_id": "BlackBird.TicketManagement.UI",

"user\_id": "ever.torresg",

"user\_secret": "123"

}

######################

### USER ENDPOINT

######################

@tokenBearer= eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJVc2VySWQiOiIyIiwiR2l2ZW5OYW1lIjoiRXZlciBBbG9uc28gVG9ycmVzIEdhbGVhbm8iLCJFbWFpbCI6ImVlYXRnODQ0QGhvdG1haWwuY29tIiwibmJmIjoxNzYwNjY5OTM1LCJleHAiOjE3NjA2ODg1MzUsImlhdCI6MTc2MDY4NzkzNSwiaXNzIjoiYmxhY2tiaXJkLmlkZW50aXR5Lmlzc3Vlci5jb20iLCJhdWQiOiIvL2F1ZGllbmNlLmJsYWNrYmlyZC5pZGVudGl0eS5jb20vYXV0aGVudGljYWQtdXNlcnMifQ.99BKzTZp-kyZP1ifhbBsDPsDzZaUA\_0Psl2f14\_304g

### Get All User Async

GET {{BlackBird.TicketManagement.Presentation\_HostAddress}}/api/User/AllAsync

Authorization: Bearer {{tokenBearer}}

###DisableAsync

@userName= ever.torresg

PATCH {{BlackBird.TicketManagement.Presentation\_HostAddress}}/api/User/DisableAsync?userId={{userName}}

Authorization: Bearer {{tokenBearer}}

### Get User ByNameAsync

GET {{BlackBird.TicketManagement.Presentation\_HostAddress}}/api/User/ByNameAsync?userName={{userName}}

######################

###TICKET END POINT

######################

### Get All Tickets Async

GET {{BlackBird.TicketManagement.Presentation\_HostAddress}}/api/Ticket/AllAsync

Authorization: Bearer {{tokenBearer}}

### Get Ticket By Id

@ticketId=7

GET {{BlackBird.TicketManagement.Presentation\_HostAddress}}/api/Ticket/ByIdAsync?ticketId={{ticketId}}

Authorization: Bearer {{tokenBearer}}

### Get Ticket History ById Async

GET {{BlackBird.TicketManagement.Presentation\_HostAddress}}/api/Ticket/HistoryByIdAsync?ticketId={{ticketId}}

Authorization: Bearer {{tokenBearer}}

### Add New Ticket Async

POST {{BlackBird.TicketManagement.Presentation\_HostAddress}}/api/Ticket/AddAsync

Authorization: Bearer {{tokenBearer}}

Content-Type: application/json

{

"ticketTypeFk": 5,

"ticketStateFk": 8,

"createdByUserFk": 2,

"createdDate": "2025-10-17T05:00:53.879Z",

"details": null,

"audience": null,

"localization": null,

"eventDate": null,

"updatedByUserFk": null,

"updatedDate": null,

"asignedToUserFk": null

}

### Update Ticket Async

PUT {{BlackBird.TicketManagement.Presentation\_HostAddress}}/api/Ticket/UpdateAsync

Authorization: Bearer {{tokenBearer}}

Content-Type: application/json

{

"ticketId": 0,

"ticketTypeFk": 0,

"ticketStateFk": 0,

"createdByUserFk": 0,

"createdDate": "2025-10-17T05:00:53.879Z",

"details": "string",

"audience": "string",

"localization": "string",

"eventDate": "2025-10-17T05:00:53.879Z",

"updatedByUserFk": 0,

"updatedDate": "2025-10-17T05:00:53.879Z",

"asignedToUserFk": 0,

}

### Delete Ticket Async

DELETE {{BlackBird.TicketManagement.Presentation\_HostAddress}}/api/Ticket/DeleteAsync?ticketId={{ticketId}}

Authorization: Bearer {{tokenBearer}}