Índice

[Requerimientos Funcionales 4](#_Toc138335474)

[RF01 – Registro de Usuario 4](#_Toc138335475)

[RF02 – Login (Usuario sin identificar). 4](#_Toc138335476)

[RF03 – Logout (usuario autenticado activo) 4](#_Toc138335477)

[RF04 – Alta tipo (usuario autenticado activo) 5](#_Toc138335478)

[RF05 – Listar todos los tipos disponibles (usuario autenticado activo) 5](#_Toc138335479)

[RF06 – Búsqueda por nombre (usuario autenticado activo) 6](#_Toc138335480)

[RF07 – Eliminar tipo (usuario autenticado activo) 6](#_Toc138335481)

[RF08 – Modificación de descripción y costo por persona. (usuario autenticado activo) 7](#_Toc138335482)

[RF09 – Registro de cabaña (usuario autenticado) 7](#_Toc138335483)

[RF10 – Buscar cabaña (usuario autenticado) 7](#_Toc138335484)

[RF11 – Registrar mantenimiento de cabaña (usuario autenticado) 8](#_Toc138335485)

[RF12 – Listar Mantenimientos (usuario autenticado) 8](#_Toc138335486)

[RF13 – Listar Mantenimientos Por Fecha(usuario autenticado) 9](#_Toc138335487)

[RF14 – Obtener Cabañas por valor (usuario autenticado) 9](#_Toc138335489)

[RF15 – Obtener Cabañas por valor entre valores (usuario autenticado) 9](#_Toc138335490)

[Script de creación de tablas 10](#_Toc138335491)

[Script datos de prueba 12](#_Toc138335492)

[Diagrama Casos de Uso 14](#_Toc138335493)

[Diagrama Lógica Negocio 15](#_Toc138335494)

[Diagrama Cliente MVC 16](#_Toc138335495)

[Diagrama Lógica De Aplicación 17](#_Toc138335496)

[Diagrama de los servicios Web Api 18](#_Toc138335497)

[Diagrama DTOs 19](#_Toc138335498)

[Diagrama Excepciones Dominio 20](#_Toc138335499)

[20](#_Toc138335500)

[Diagrama Lógica Acceso Datos 21](#_Toc138335501)

[Diagrama Macro 22](#_Toc138335502)

[22](#_Toc138335503)

# Requerimientos Funcionales

A continuación, se presenta la sección con los requerimientos funcionales detectados para esta etapa:

## RF01 – Registro de Usuario

El sistema debe permitir la creación de usuarios utilizando una mail y una contraseña válidos. La dirección de correo electrónico debe tener un formato adecuado y la contraseña debe tener al menos 6 caracteres, incluyendo letras mayúsculas, minúsculas y dígitos. Además, el sistema debe implementar las reglas de negocio necesarias para validar la información proporcionada por el usuario. Es importante destacar que el sistema no debe permitir la creación de usuarios con mail duplicado, garantizando así la unicidad de cada usuario registrado.

## RF02 – Login (Usuario sin identificar).

El sistema debe ser capaz de verificar las credenciales proporcionadas por un visitante sin identificar y autorizar el acceso en caso de ser válidas. Si las credenciales no son correctas, el sistema debe redirigir al usuario nuevamente a la página de inicio de sesión o a la página principal si ya ha iniciado sesión previamente. Además, se debe implementar un filtro de seguridad que evite el acceso directo a ciertos requerimientos mediante URL, obligando al usuario a utilizar el proceso de inicio de sesión para acceder a ellos.

## RF03 – Logout (usuario autenticado activo)

El sistema debe permitir al usuario autenticado activo cerrar sesión de manera segura. Al hacerlo, se deben borrar todas las sesiones y datos de autenticación asociados al usuario actual. Una vez que la sesión se ha cerrado correctamente, el sistema debe redirigir al usuario a la página de inicio de sesión para que pueda autenticarse nuevamente y acceder a las funcionalidades protegidas.

## RF04 – Alta tipo (usuario autenticado activo)

La funcionalidad permite a los usuarios autenticados registrar nuevos tipos de cabaña en el sistema. Se deben cumplir requisitos como tener un nombre único, descripción detallada, y un costo por huésped positivo. Los datos se guardarán en la base de datos mediante Entity Framework. Esta funcionalidad se implementa en un entorno web utilizando el patrón MVC. Su objetivo es ampliar la variedad de tipos de cabaña disponibles y mejorar la experiencia de reserva para los usuarios.

## RF05 – Listar todos los tipos disponibles (usuario autenticado activo)

La funcionalidad permite mostrar en el sistema todos los datos de los tipos de cabaña existentes. Utilizando Entity Framework, se accede a la base de datos para obtener de manera eficiente la información completa de cada tipo de cabaña, como nombre, descripción y costo por huésped. En caso de no haber tipos de cabaña registrados, se muestra un mensaje indicando la falta de resultados. Esta funcionalidad brinda a los usuarios una visión completa de las opciones disponibles para facilitar la toma de decisiones en las reservas de alojamiento.

## RF06 – Búsqueda por nombre (usuario autenticado activo)

La funcionalidad permite mostrar tipos de cabaña ingresando un nombre específico. Los resultados mostrarán los tipos de cabaña que contengan o coincidan con el nombre buscado. Si no se encuentran resultados, se mostrará un mensaje indicando la falta de coincidencias. Esta función facilita a los usuarios encontrar tipos de cabaña según sus preferencias.

## RF07 – Eliminar tipo (usuario autenticado activo)

La funcionalidad permite a los usuarios ingresar el nombre de un tipo de cabaña y mostrar sus datos correspondientes. Se realiza una verificación para asegurarse de que el tipo de cabaña no esté siendo utilizado en ninguna cabaña actualmente. En caso de que el tipo de cabaña no esté en uso, se procede a eliminarlo utilizando repositorio y Entity Framework. Si el tipo de cabaña no existe en el sistema o no se puede eliminar debido a su uso en cabañas, se mostrará un mensaje indicando la situación. Esta funcionalidad garantiza un control adecuado sobre la eliminación de tipos de cabaña y proporciona una retroalimentación clara al usuario.

## RF08 – Modificación de descripción y costo por persona. (usuario autenticado activo)

La funcionalidad permite ingresar al detalle viendo la descripción y costo en un formato editable. Se realizan validaciones para asegurar que la descripción y el costo sean válidos. Una vez validados, se actualizan los valores en el sistema utilizando repositorio y Entity Framework. En caso de que el tipo de no exista en el sistema o no sea posible modificarlo por alguna razón, se mostrará un mensaje indicando la situación.

## RF09 – Registro de cabaña (usuario autenticado)

La funcionalidad permite agregar nuevas cabañas al sistema. Los requisitos incluyen proporcionar un nombre único y alfabético, una descripción de 10 a 500 caracteres, y especificar si la cabaña tiene jacuzzi y está disponible para reservas. Además, se asigna automáticamente un número de habitación único y se indica la capacidad de huéspedes. Durante el registro, se puede subir una foto de la cabaña, la cual se guarda con un nombre basado en el nombre de la cabaña y se permite la posibilidad de agregar múltiples fotos. Todos los datos se almacenan correctamente en la base de datos utilizando Entity Framework. Se emiten mensajes para informar si el registro se realizó exitosamente o si se produjo algún error durante el proceso de guardado.

## RF10 – Buscar cabaña (usuario autenticado)

La funcionalidad permite buscar cabañas en el sistema utilizando diferentes criterios, como el nombre, tipo, capacidad y habilitación para reservas. Los resultados de la búsqueda incluyen información detallada sobre las cabañas, como número, nombre, descripción, jacuzzi, tipo y habilitación para reservas. Se muestra una miniatura de la foto y se proporcionan enlaces para registrar y mostrar mantenimientos. En caso de no encontrar resultados, se muestra un mensaje informativo.

## RF11 – Registrar mantenimiento de cabaña (usuario autenticado)

La funcionalidad permite agregar registros de mantenimiento para una cabaña específica. Se validan los campos fecha, descripción, costo y nombre del trabajador. Los mantenimientos se guardan utilizando Entity Framework, y se verifica que no se exceda el límite diario de 3 mantenimientos. Se muestran mensajes indicando si el mantenimiento pudo ser guardado o si hubo algún problema.

## RF12 – Listar Mantenimientos (usuario autenticado)

La funcionalidad permite visualizar los registros de mantenimiento realizados en una cabaña dentro de un rango de fechas específico. Los resultados se ordenan por costo de mantenimiento y se muestran todos los datos relevantes. En caso de no encontrar mantenimientos en ese rango, se muestra un mensaje informativo. La función utiliza filtros mediante Linq para obtener los resultados deseados. Esta característica brinda a los usuarios una forma conveniente de revisar y analizar el historial de mantenimientos de una cabaña en un período de tiempo determinado.

## RF13 – Listar Mantenimientos Por Fecha(usuario autenticado)

## La función permite a los usuarios autenticados ver los registros de mantenimiento de una cabaña en un rango de fechas específico. Los resultados se ordenan por costo y se muestran todos los datos relevantes. En caso de no encontrar registros en ese rango, se muestra un mensaje informativo. Esta función ofrece a los usuarios una forma sencilla de revisar y analizar el historial de mantenimientos de una cabaña en un período de tiempo determinado.

## RF14 – Obtener Cabañas por valor (usuario autenticado)

La función permite obtener las cabañas con un costo menor al valor especificado. Se devuelve una lista de cabañas que cumplen con este criterio, o se muestra un mensaje informativo si no se encuentran cabañas.

## RF15 – Obtener Cabañas por valor entre valores (usuario autenticado)

La función permite obtener los registros de mantenimiento dentro de un rango de valores específico. Se devuelve una lista de registros que cumplen con este criterio, mostrando el nombre de la persona y el monto total del mantenimiento. En caso de no encontrar registros en ese rango, se muestra un mensaje informativo.

# Script de creación de tablas

CREATE DATABASE HotelObligatorio

GO

USE HotelObligatorio

GO

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[Usuarios] (

[Id] INT IDENTITY (1, 1) NOT NULL,

[Mail] NVARCHAR (MAX) NOT NULL,

[Password] NVARCHAR (MAX) NOT NULL

);

GO

CREATE TABLE [dbo].[Tipos] (

[Id] INT IDENTITY (1, 1) NOT NULL,

[Nombre] NVARCHAR (MAX) NOT NULL,

[Descripcion] NVARCHAR (MAX) NOT NULL,

[CostoPorHuesped] FLOAT (53) NOT NULL,

CONSTRAINT [PK\_Tipos] PRIMARY KEY CLUSTERED ([Id] ASC)

);

CREATE TABLE [dbo].[Cabanias] (

[numHabitacion] INT NOT NULL,

[Nombre] NVARCHAR (MAX) NOT NULL,

[Descripcion] NVARCHAR (MAX) NOT NULL,

[TieneJacuzzi] BIT NOT NULL,

[Estado] BIT NOT NULL,

[MaxHuespedes] INT NOT NULL,

[Foto] NVARCHAR (MAX) NOT NULL,

[TipoId] INT NOT NULL

);

GO

CREATE NONCLUSTERED INDEX [IX\_Cabanias\_TipoId]

ON [dbo].[Cabanias]([TipoId] ASC);

GO

ALTER TABLE [dbo].[Cabanias]

ADD CONSTRAINT [PK\_Cabanias] PRIMARY KEY CLUSTERED ([numHabitacion] ASC);

GO

ALTER TABLE [dbo].[Cabanias]

ADD CONSTRAINT [FK\_Cabanias\_Tipos\_TipoId] FOREIGN KEY ([TipoId]) REFERENCES [dbo].[Tipos] ([Id]) ON DELETE CASCADE;

GO

CREATE TABLE [dbo].[Mantenimientos] (

[Id] INT IDENTITY (1, 1) NOT NULL,

[Fecha] DATETIME2 (7) NOT NULL,

[Descripcion] NVARCHAR (MAX) NOT NULL,

[Costo] FLOAT (53) NOT NULL,

[NombreFuncionario] NVARCHAR (MAX) NOT NULL,

[CabaniaRealizadanumHabitacion] INT NOT NULL

);

GO

CREATE NONCLUSTERED INDEX [IX\_Mantenimientos\_CabaniaRealizadanumHabitacion]

ON [dbo].[Mantenimientos]([CabaniaRealizadanumHabitacion] ASC);

GO

ALTER TABLE [dbo].[Mantenimientos]

ADD CONSTRAINT [PK\_Mantenimientos] PRIMARY KEY CLUSTERED ([Id] ASC);

GO

ALTER TABLE [dbo].[Mantenimientos]

ADD CONSTRAINT [FK\_Mantenimientos\_Cabanias\_CabaniaRealizadanumHabitacion] FOREIGN KEY ([CabaniaRealizadanumHabitacion]) REFERENCES [dbo].[Cabanias] ([numHabitacion]) ON DELETE CASCADE;

GO

# Script datos de prueba

protected override void Up(MigrationBuilder migrationBuilder)

{

migrationBuilder.InsertData("Usuarios", new[] { "Id", "Email", "Password" }, new object[] { 1, "diego@ort.com", "diego123" });

migrationBuilder.InsertData("Usuarios", new[] { "Id", "Email", "Password" }, new object[] { 2, "mateo@ort.com", "mateo123" });

migrationBuilder.InsertData("Tipos", new[] { "Id", "Nombre", "Descripcion", "CostoPorHuesped" }, new object[] { 1, "Deluxe", "Cabanias mas costosas", 200 });

migrationBuilder.InsertData("Tipos", new[] { "Id", "Nombre", "Descripcion", "CostoPorHuesped" }, new object[] { 2, "Clasica", "Cabanias clasicas", 150 });

migrationBuilder.InsertData("Tipos", new[] { "Id", "Nombre", "Descripcion", "CostoPorHuesped" }, new object[] { 3, "Economica", "Cabanias mas economicas", 50 });

migrationBuilder.InsertData("Tipos", new[] { "Id", "Nombre", "Descripcion", "CostoPorHuesped" }, new object[] { 4, "Renovada", "Cabanias hechas a nuevas", 120 });

migrationBuilder.InsertData("Cabanias", new[] { "numHabitacion", "Nombre", "Descripcion", "TieneJacuzzi", "Estado", "MaxHuespedes", "Foto", "TipoId" }, new object[] { 1, "Cabania extra deluxe ", "Cabania de muy alta gama 1", true, true, 8, "foto1", 1 });

migrationBuilder.InsertData("Cabanias", new[] { "numHabitacion", "Nombre", "Descripcion", "TieneJacuzzi", "Estado", "MaxHuespedes", "Foto", "TipoId" }, new object[] { 2, "Cabania deluxe ", "Cabania alta gama ", true, true, 6, "foto1", 1 });

migrationBuilder.InsertData("Cabanias", new[] { "numHabitacion", "Nombre", "Descripcion", "TieneJacuzzi", "Estado", "MaxHuespedes", "Foto", "TipoId" }, new object[] { 3, "Cabania clasica ", "Cabania clasica ", false, false, 5, "foto1", 2 });

migrationBuilder.InsertData("Cabanias", new[] { "numHabitacion", "Nombre", "Descripcion", "TieneJacuzzi", "Estado", "MaxHuespedes", "Foto", "TipoId" }, new object[] { 4, "Cabania eco ", "Cabania economica pero bonita", true, false, 2, "foto1", 3 });

migrationBuilder.InsertData("Cabanias", new[] { "numHabitacion", "Nombre", "Descripcion", "TieneJacuzzi", "Estado", "MaxHuespedes", "Foto", "TipoId" }, new object[] { 5, "Cabania super eco ", "Cabania economica extrema", false, false, 2, "foto1", 3 });

migrationBuilder.InsertData("Cabanias", new[] { "numHabitacion", "Nombre", "Descripcion", "TieneJacuzzi", "Estado", "MaxHuespedes", "Foto", "TipoId" }, new object[] { 6, "Cabania clase media ", "Cabania intermedia", true, true, 5, "foto1", 2 });

migrationBuilder.InsertData("Mantenimientos", new[] { "Id", "Fecha", "Descripcion", "Costo", "NombreFuncionario", "CabaniaRealizadanumHabitacion" }, new object[] { 1, DateTime.UtcNow, "Reparacion de tuberias", 500, "Messi", 1 });

migrationBuilder.InsertData("Mantenimientos", new[] { "Id", "Fecha", "Descripcion", "Costo", "NombreFuncionario", "CabaniaRealizadanumHabitacion" }, new object[] { 2, DateTime.UtcNow, "Reparacion de puertas", 200, "Messi", 1 });

migrationBuilder.InsertData("Mantenimientos", new[] { "Id", "Fecha", "Descripcion", "Costo", "NombreFuncionario", "CabaniaRealizadanumHabitacion" }, new object[] { 3, DateTime.UtcNow, "Reparacion de electricidad", 500, "Messi", 1 });

migrationBuilder.InsertData("Mantenimientos", new[] { "Id", "Fecha", "Descripcion", "Costo", "NombreFuncionario", "CabaniaRealizadanumHabitacion" }, new object[] { 4, DateTime.UtcNow.AddDays(-5), "Reparacion de techo", 800, "Messi", 2 });

migrationBuilder.InsertData("Mantenimientos", new[] { "Id", "Fecha", "Descripcion", "Costo", "NombreFuncionario", "CabaniaRealizadanumHabitacion" }, new object[] { 5, DateTime.UtcNow.AddDays(-5), "Reparacion de electricidad", 200, "Messi", 2 });

migrationBuilder.InsertData("Mantenimientos", new[] { "Id", "Fecha", "Descripcion", "Costo", "NombreFuncionario", "CabaniaRealizadanumHabitacion" }, new object[] { 6, DateTime.UtcNow.AddDays(-5), "Reparacion de pisos", 300, "Messi", 3 });

migrationBuilder.InsertData("Mantenimientos", new[] { "Id", "Fecha", "Descripcion", "Costo", "NombreFuncionario", "CabaniaRealizadanumHabitacion" }, new object[] { 7, DateTime.UtcNow.AddDays(-10), "Reparacion de pisos", 300, "Don Ramon", 4 });

migrationBuilder.InsertData("Mantenimientos", new[] { "Id", "Fecha", "Descripcion", "Costo", "NombreFuncionario", "CabaniaRealizadanumHabitacion" }, new object[] { 8, DateTime.UtcNow.AddDays(-10), "Reparacion de techo", 340, "Don Ramon", 4 });

migrationBuilder.InsertData("Mantenimientos", new[] { "Id", "Fecha", "Descripcion", "Costo", "NombreFuncionario", "CabaniaRealizadanumHabitacion" }, new object[] { 9, DateTime.UtcNow.AddDays(-15), "Reparacion de electricdad", 400, "Zombo", 4 });

migrationBuilder.InsertData("Mantenimientos", new[] { "Id", "Fecha", "Descripcion", "Costo", "NombreFuncionario", "CabaniaRealizadanumHabitacion" }, new object[] { 10, DateTime.UtcNow.AddDays(-15), "Reparacion de mesas", 300, "Zombo", 4 });

migrationBuilder.InsertData("Mantenimientos", new[] { "Id", "Fecha", "Descripcion", "Costo", "NombreFuncionario", "CabaniaRealizadanumHabitacion" }, new object[] { 11, DateTime.UtcNow.AddDays(-20), "Reparacion de sillas", 340, "Zombo", 4 });

migrationBuilder.InsertData("Mantenimientos", new[] { "Id", "Fecha", "Descripcion", "Costo", "NombreFuncionario", "CabaniaRealizadanumHabitacion" }, new object[] { 12, DateTime.UtcNow.AddDays(-25), "Reparacion de internet", 400, "Zombo", 4 });

migrationBuilder.InsertData("Mantenimientos", new[] { "Id", "Fecha", "Descripcion", "Costo", "NombreFuncionario", "CabaniaRealizadanumHabitacion" }, new object[] { 13, DateTime.UtcNow.AddDays(-25), "Reparacion de tuberias", 500, "Ibra", 1 });

migrationBuilder.InsertData("Mantenimientos", new[] { "Id", "Fecha", "Descripcion", "Costo", "NombreFuncionario", "CabaniaRealizadanumHabitacion" }, new object[] { 14, DateTime.UtcNow.AddDays(-25), "Reparacion de puertas", 200, "Ibra", 1 });

migrationBuilder.InsertData("Mantenimientos", new[] { "Id", "Fecha", "Descripcion", "Costo", "NombreFuncionario", "CabaniaRealizadanumHabitacion" }, new object[] { 15, DateTime.UtcNow.AddDays(-30), "Reparacion de tuberias", 500, "Ibra", 1 });

migrationBuilder.InsertData("Mantenimientos", new[] { "Id", "Fecha", "Descripcion", "Costo", "NombreFuncionario", "CabaniaRealizadanumHabitacion" }, new object[] { 16, DateTime.UtcNow.AddDays(-30), "Reparacion de tuberias", 700, "Forlan", 5 });

migrationBuilder.InsertData("Mantenimientos", new[] { "Id", "Fecha", "Descripcion", "Costo", "NombreFuncionario", "CabaniaRealizadanumHabitacion" }, new object[] { 17, DateTime.UtcNow.AddDays(-30), "Reparacion de puertas", 400, "Forlan", 6});

# Diagrama Casos de Uso

Diagrama

Descripción generada automáticamente

# Diagrama, Esquemático Descripción generada automáticamenteDiagrama Lógica Negocio

# Diagrama Cliente MVC

# Diagrama, Escala de tiempo Descripción generada automáticamenteDiagrama Lógica De Aplicación

# Diagrama Descripción generada automáticamenteDiagrama de los servicios Web Api

# Diagrama DTOsImagen que contiene Escala de tiempo Descripción generada automáticamente

# Diagrama Excepciones Dominio

# Diagrama Descripción generada automáticamente

# Diagrama Descripción generada automáticamenteDiagrama Lógica Acceso Datos

# Diagrama Macro

# 