# □ Documento de Requisitos v1.0: Sistema Integral de Reservas

**Proyecto:** Sistema Integral de Reservas de Espacios
**Versión:** 1.0
**Fecha:** 07/11/2025

## 1. Introducción

### 1.1. Propósito
El presente documento detalla los requisitos funcionales, no funcionales y los criterios de aceptación para la construcción de un sistema integral de gestión de reservas de espacios. El objetivo es crear una plataforma robusta, escalable y moderna que permita a los usuarios reservar espacios (como salas de reuniones, puestos de estudio, etc.) mientras proporciona herramientas de administración completas.

### 1.2. Alcance
El sistema cubrirá el ciclo de vida completo de una reserva, desde el registro del usuario y la visualización de la disponibilidad, hasta la creación, gestión y finalización de la reserva mediante una arquitectura orientada a eventos (Kafka). El sistema será accesible a través de una aplicación web (Angular) y será gestionado por una API REST (.NET).

### 1.3. Roles de Usuario
* **Usuario (Registrado):** Puede buscar espacios, ver disponibilidad en el calendario, crear, modificar y cancelar sus propias reservas, y ver su historial.
* **Administrador:** Tiene control total sobre el sistema. Puede gestionar (CRUD) usuarios, gestionar (CRUD) espacios y gestionar (CRUD) todas las reservas del sistema.

---

## 2. □ Requisitos Funcionales (RF)

Los requisitos funcionales se definen a continuación mediante Historias de Usuario (HU) y sus Criterios de Aceptación (CA) asociados.

### RF 1: Gestión de Usuarios y Autenticación
* **HU 1.1:** *Como* usuario nuevo, *quiero* poder registrarme en el sistema (con nombre, email, contraseña) *para* poder acceder a las funciones de reserva.
* **HU 1.2:** *Como* usuario registrado, *quiero* poder iniciar sesión (con email y contraseña) *para* que el sistema me reconozca y me dé acceso.
* **HU 1.3:** *Como* usuario autenticado, *quiero* que mi sesión se mantenga segura (ej. vía JWT).

* **HU 1.4:** *Como* administrador, *quiero* poder ver una lista de todos los usuarios y gestionar sus roles.

#### Criterios de Aceptación (Ejemplos)
* **CA 1.1 (Registro Éxito):** *Dado* un usuario no registrado, *cuando* introduce un email único, un nombre y una contraseña válida, *entonces* el sistema crea su cuenta y le notifica del éxito.
* **CA 1.2 (Registro Fallido):** *Dado* un usuario no registrado, *cuando* introduce un email que ya existe, *entonces* el sistema muestra un error "El email ya está en uso".
* **CA 1.3 (Login Éxito):** *Dado* un usuario registrado, *cuando* introduce sus credenciales correctas, *entonces* el sistema le da acceso y le devuelve un token JWT.

### RF 2: Gestión de Espacios (CRUD)
* **HU 2.1:** *Como* administrador, *quiero* poder crear, leer, actualizar y eliminar (CRUD) los espacios disponibles para reservar (ej. nombre de la sala, capacidad, descripción).

### RF 3: Gestión de Reservas (CRUD)
* **HU 3.1:** *Como* usuario, *quiero* poder crear una reserva para un espacio específico, seleccionando una fecha y un rango horario.
* **HU 3.2:** *Como* usuario, *quiero* poder ver mi historial de reservas (próximas y pasadas).
* **HU 3.3:** *Como* usuario, *quiero* poder modificar o cancelar una reserva próxima.
* **HU 3.4:** *Como* administrador, *quiero* poder ver, modificar o cancelar la reserva de *cualquier* usuario.

#### Criterios de Aceptación (Ejemplos)
* **CA 3.1 (Validación de Superposición):** *Dado* que la "Sala 1" ya está reservada de 10:00 a 11:00, *cuando* otro usuario intenta reservar la "Sala 1" de 10:30 a 11:30, *entonces* el sistema rechaza la reserva y muestra un error "El espacio no está disponible en ese horario".

### RF 4: Visualización y Búsqueda
* **HU 4.1:** *Como* usuario, *quiero* poder ver un calendario visual (diario/semanal/mensual) que muestre la disponibilidad de los espacios.
* **HU 4.2:** *Como* usuario, *quiero* poder buscar y filtrar reservas por usuario (si soy admin), espacio y/o fecha.

### RF 5: Notificaciones y Finalización de Reservas (Kafka)
* **HU 5.1:** *Como* usuario, *quiero* recibir una notificación (in-app o email) cuando mi reserva se crea, modifica, cancela o finaliza.
* **HU 5.2:** *Como* sistema, *quiero* que una reserva se marque como "pendiente de finalización" cuando llegue su fecha de fin (gestionado por eventos).

* **HU 5.3:** *Como* sistema, *quiero* que un "worker" procese las reservas pendientes, las marque como "Finalizadas" y emita los eventos correspondientes.

#### Criterios de Aceptación (Ejemplos)
* **CA 5.1 (Creación):** *Cuando* un usuario crea una reserva, *entonces* la API publica un evento en el topic `reservas.creadas`.
* **CA 5.2 (Finalización):** *Cuando* un worker consume un evento de `reservas.finalizacion.pend`, *entonces* actualiza el estado de la reserva en la BBDD y publica un evento en `reservas.finalizadas`.

### RF 6: Reportes y Exportación
* **HU 6.1:** *Como* administrador, *quiero* poder exportar un historial de reservas (filtrado por fecha o usuario) en formato CSV o PDF.

---

## 3. □ Requisitos No Funcionales (RNF)

* **Seguridad:**
    * Todas las contraseñas de usuario deben almacenarse hasheadas (ej. BCrypt).
    * La API debe estar protegida y todas las peticiones (excepto login/register) deben estar autenticadas (ej. JWT).
    * El sistema debe implementar control de roles (Usuario vs. Administrador) en los endpoints de la API.
* **Rendimiento:**
    * Los tiempos de respuesta de la API para operaciones críticas (ej. `GET /disponibilidad`) deben ser inferiores a 2 segundos bajo carga normal (50 usuarios concurrentes).
* **Usabilidad:**
    * La interfaz debe ser intuitiva, responsive (adaptable a móvil y escritorio) y seguir los lineamientos del prototipo Figma.
    * Debe cumplir con los estándares de accesibilidad (WCAG 2.1 Nivel AA).
* **Escalabilidad y Mantenibilidad:**
    * Todo el sistema (backend, frontend, BBDD, Kafka) debe estar orquestado con Docker Compose para un despliegue sencillo.
    * El código de backend debe seguir principios de Arquitectura Limpia (separación en `Domain`, `Application`, `Infrastructure`, `API`).
    * El código de frontend debe ser modular (`AuthModule`, `ReservationsModule`, `CoreModule`, `SharedModule`).
* **Fiabilidad:**
    * El sistema debe ser tolerante a fallos en la entrega de eventos Kafka (implementar reintentos y/o patrón Outbox para garantizar la consistencia entre la BBDD y la publicación de eventos).
* **Pruebas:**

    * El backend debe tener una cobertura de pruebas unitarias (xUnit) mínima del 60%.
    * El frontend debe tener una cobertura de pruebas unitarias (Karma/Jasmine) mínima del 50%.

---

## 4. □ Stack Técnico (Definido por el Proyecto)

* **Backend:** .NET 8 (API RESTful)
* **Frontend:** Angular (v17+)
* **Base de datos:** PostgreSQL 14 (o SQL Server)
* **Orquestación:** Docker Compose
* **Eventos:** Apache Kafka (o RabbitMQ)
* **Testing:** xUnit (.NET), Jasmine/Karma (Angular)

—

## 5. □ Modelo de Datos Inicial (Entidades Principales)

Definimos tres entidades centrales para el sistema:

1. **User:** (Gestionada por ASP.NET Identity)
    * `Id` (PK)
    * `Email` (Unique)
    * `PasswordHash`
    * `UserName`
    * `Role` (Relación a tabla de Roles)
2. **Space:**
    * `Id` (PK)
    * `Name` (string)
    * `Description` (string)
    * `Capacity` (int)
3. **Reservation:**
    * `Id` (PK)
    * `UserId` (FK a User)
    * `SpaceId` (FK a Space)
* `StartTime` (datetime)

    * `EndTime` (datetime)
    * `Status` (string: "Confirmada", "Cancelada", "Finalizada")

---

## 6. □ Arquitectura de Eventos (Esquema Kafka)

El sistema utilizará Kafka para la comunicación asíncrona, principalmente para la finalización de reservas y notificaciones.

| Topic | Emisor | Consumidor | Descripción |
| :--- | :--- | :--- | :--- |
| `reservas.creadas` | API | Worker, Notificador| Publica evento al crear reserva. |
| `reservas.finalizacion.pend` | API/Scheduler| Worker Finalizador | Reserva pendiente a finalizar. |
| `reservas.finalizadas` | Worker Finalizador | Notificaciones | Reserva finalizada o cancelada. |
| `reservas.notificaciones` | Worker/API | Servicio Email/App | Notificaciones en la app/email. |

*(Fin del Documento v1.0)*