Documentación Técnica – Gestión de Reservas

1. Introducción

Esta documentación describe la implementación técnica del sistema de gestión de reservas, incluyendo los servicios backend, base de datos, pruebas unitarias y la integración con Kafka para eventos.
El objetivo es proporcionar un manual completo para desarrolladores que necesiten entender o ampliar el sistema.

---

2. Arquitectura General

- **Backend:** ASP.NET Core Web API
- **Base de datos:** SQL Server (Docker)
- **Mensajería:** Kafka para publicación de eventos
- **Pruebas:** xUnit para pruebas unitarias básicas

Diagrama de arquitectura:

[Cliente/Swagger UI] ---> [ASP.NET Core API] ---> [SQL Server Docker]

\

---> [Kafka Producer] ---> [Topic reserva-creada]

---

3. API – Endpoints CRUD

3.1 Reservas

Método	Ruta	Descripción
GET	/Reserva	Listar todas las reservas
GET	/Reserva/{id}	Obtener reserva por ID
POST	/Reserva	Crear nueva reserva
PUT	/Reserva/{id}	Actualizar reserva existente
DELETE	/Reserva/{id}	Eliminar reserva

3.1.1 POST /Reserva

Request Body Ejemplo:

{

"idUsuario": 1,

"idEspacio": 1,

"fecha": "2025-11-12T08:44:16.420Z",

```
  "horaInicio": "14:00:00",
  "duracionMinutos": 60,
  "estado": "Confirmada",
  "usuario": null,
  "espacio": null
}
```


Response 201 Created:


```
{
  "id": 1,
  "idUsuario": 1,
  "idEspacio": 1,
  "fecha": "2025-11-12T08:44:16.420Z",
  "horaInicio": "14:00:00",
  "duracionMinutos": 60,
  "estado": "Confirmada",
  "usuario": null,
  "espacio": null
}
```


Notas:

- Valida que no existan solapamientos de horario.
- Publica evento Kafka: reserva-creada.

3.1.2 PUT /Reserva/{id}

Request Body Ejemplo:


```
{
  "fecha": "2025-11-12T09:00:00",
  "duracionMinutos": 50,
  "estado": "Pendiente",
  "idUsuario": 1,
  "idEspacio": 1
}
```


Response 200 OK: Actualiza la reserva y publica evento Kafka: reserva-actualizada.

3.1.3 DELETE /Reserva/{id}

Elimina la reserva y publica evento Kafka: reserva-eliminada.

3.2 Usuarios

Método	Ruta	Descripción
GET	/Usuario	Listar todos los usuarios
GET	/Usuario/{id}	Obtener usuario por ID
POST	/Usuario	Crear nuevo usuario
PUT	/Usuario/{id}	Actualizar usuario
DELETE	/Usuario/{id}	Eliminar usuario

Eventos Kafka: usuario-creado, usuario-actualizado, usuario-eliminado.

3.3 Espacios

Método	Ruta	Descripción
GET	/Espacio	Listar todos los espacios
GET	/Espacio/{id}	Obtener espacio por ID
POST	/Espacio	Crear nuevo espacio
PUT	/Espacio/{id}	Actualizar espacio
DELETE	/Espacio/{id}	Eliminar espacio

Eventos Kafka: espacio-creado, espacio-actualizado, espacio-eliminado.

---

4. Control de Disponibilidad y Validaciones

- Antes de crear o actualizar una reserva, el backend comprueba:
    - Que el **espacio no esté ocupado** en la misma fecha y horario.
    - Que la **duración y horario** sean válidos.
- Si hay conflicto, retorna **400 Bad Request** con mensaje descriptivo.

---

5. Pruebas Unitarias con xUnit

- Pruebas básicas para servicios backend:

- Crear reserva
- Validar solapamiento
- Actualizar reserva
- Eliminar reserva

- Ejemplo de test:

```
[Fact]
public async Task CrearReserva_ValidaDisponibilidad()
{
   var service = new ReservaService(_context);
   var result = await service.CrearReserva(nuevaReserva);
   Assert.NotNull(result);
}
```

---

6. Base de Datos (Docker)

6.1 Tablas principales

Usuarios

Campo	Tipo	Notas
Id	int	PK
Nombre	string	
NumeroDoc	string	
Contrasena	string	
Rol	string	

Espacios

Campo	Tipo	Notas
Id	int	PK
Nombre	string	
Tipo	string	
Capacidad	int	
Ubicacion	string	

Reservas

Campo	Tipo	Notas
Id	int	PK
IdUsuario	int	FK Usuarios
IdEspacio	int	FK Espacios
Fecha	datetime	
HoraInicio	time	
DuracionMinutos	int	
Estado	string	

6.2 Relaciones

- Reservas.IdUsuario → Usuarios.Id
- Reservas.IdEspacio → Espacios.Id

---

7. Kafka – Publicación de Eventos

- **Topics utilizados:**
    - reserva-creada, reserva-actualizada, reserva-eliminado

Reservas:

docker exec -it kafka kafka-console-consumer --bootstrap-server localhost:9092 --topic reserva-creada --from-beginning

docker exec -it kafka kafka-console-consumer --bootstrap-server localhost:9092 --topic reserva-actualizada --from-beginning

docker exec -it kafka kafka-console-consumer --bootstrap-server localhost:9092 --topic reserva-eliminada --from-beginning

    - usuario-creado, usuario-actualizado, usuario-eliminado

Usuarios:

docker exec -it kafka kafka-console-consumer --bootstrap-server localhost:9092 --topic usuario-creado --from-beginning

docker exec -it kafka kafka-console-consumer --bootstrap-server localhost:9092 --topic usuario-actualizado --from-beginning

docker exec -it kafka kafka-console-consumer --bootstrap-server localhost:9092 --topic usuario-eliminado --from-beginning

- espacio-creado, espacio-actualizado, espacio-eliminado

Espacios:

docker exec -it kafka kafka-console-consumer --bootstrap-server localhost:9092 --topic espacio-creado --from-beginning

docker exec -it kafka kafka-console-consumer --bootstrap-server localhost:9092 --topic espacio-actualizado --from-beginning

docker exec -it kafka kafka-console-consumer --bootstrap-server localhost:9092 --topic espacio-eliminado --from-beginning

- **Formato de mensaje:** JSON serializado de la entidad modificada.

Ejemplo de evento reserva-creada:


```
{
  "Id":1,
  "IdUsuario":1,
  "IdEspacio":1,
  "Fecha":"2025-11-12T08:44:16.420Z",
  "HoraInicio":"14:00:00",
  "DuracionMinutos":60,
  "Estado":"Confirmada",
  "Usuario":null,
  "Espacio":null
}
```


---

8. Consumo básico de la base de datos en Docker

- Levantar contenedor SQL Server:

docker run -e "ACCEPT_EULA=Y" -e "SA_PASSWORD=Abc123.!" -p 1433:1433 -d mcr.microsoft.com/mssql/server:2022-latest

- Conectar desde la API: Server=localhost,1433;Database=Gestion;User Id=sa;Password=Abc123.!
- Kafka: levantar contenedor zookeeper y broker con Docker Compose.

---

9. Notas Finales

- Toda operación CRUD se refleja en Kafka.
- Validaciones críticas se realizan en backend para evitar conflictos.
- Pruebas unitarias iniciales permiten validar la lógica antes de integración completa.
- Se puede extender la documentación con diagramas ERD y diagramas de flujo de eventos.