

Proyecto Final

Ingeniería de Software I
Departamento de Ingeniería de Sistemas e Industrial
UNAL



1. Levantamiento de requerimientos

La idea de desarrollar ReservaPlus surgió tras identificar una necesidad recurrente en la organización de eventos: la dificultad para encontrar y reservar espacios de manera eficiente. A través de observaciones directas y entrevistas con organizadores de eventos, se evidenció que el proceso actual suele ser lento y confuso, requiriendo múltiples interacciones con los administradores de los espacios y sin garantía de disponibilidad en tiempo real. La problemática se hizo aún más evidente al notar la ausencia de una plataforma centralizada que permitiera la búsqueda, comparación y reserva de espacios de manera ágil y confiable.

Para la elección del proyecto, el equipo llevó a cabo una sesión de lluvia de ideas en la que se consideraron distintas propuestas. A partir de una evaluación basada en viabilidad técnica, impacto y relevancia, se llegó a un consenso mediante votación. Se definieron roles iniciales para organizar el trabajo de investigación y estructurar los primeros lineamientos del sistema, asegurando que cada integrante pudiera contribuir con sus habilidades y conocimientos.

El principal problema identificado es la falta de un sistema eficiente para la reserva de espacios. Actualmente, los usuarios deben contactar directamente a los administradores, lo que genera demoras y falta de transparencia en la disponibilidad. Además, muchos espacios carecen de herramientas digitales para gestionar sus reservas, lo que puede dar lugar a errores y confusiones. La falta de información detallada sobre los espacios también dificulta la toma de decisiones por parte de los clientes. Con ReservaPlus, se busca resolver estos inconvenientes ofreciendo una plataforma intuitiva y accesible.

Los usuarios esperan contar con una herramienta que les permita encontrar espacios disponibles de forma sencilla, con filtros de búsqueda basados en ubicación, precio, capacidad y tipo de evento. También desean acceder a información detallada sobre cada espacio, incluyendo imágenes, características y opiniones de otros usuarios. La posibilidad de reservar y realizar pagos en línea de manera segura es otra de las expectativas clave, junto con la opción de recibir notificaciones sobre sus reservas y recordatorios de eventos próximos. Para los administradores de los espacios, la plataforma debe ofrecer un sistema de gestión eficiente que les permita controlar disponibilidad, automatizar confirmaciones y recibir pagos sin complicaciones.

Para el equipo de desarrollo, este proyecto representa una oportunidad invaluable para fortalecer sus conocimientos en diseño y desarrollo de software, poniendo en práctica habilidades en tecnologías modernas como React.js, Node.js y PostgreSQL. Además, el desarrollo de ReservaPlus permite mejorar la gestión de proyectos mediante metodologías ágiles y aprender a interactuar con usuarios reales para comprender mejor sus necesidades. La construcción de una solución escalable y funcional no solo beneficia a los futuros usuarios, sino que también brinda al equipo experiencia relevante en el ámbito de la ingeniería de software.

2. Análisis de Requerimientos funcionales

La Tabla 1 presenta el análisis de requerimientos del sistema ReservaPlus utilizando el método MoSCoW. Se clasifican las funcionalidades según su prioridad (Must, Should, Could, Won't) y se estima su esfuerzo de implementación mediante la secuencia de Fibonacci. Esta priorización garantiza que el MVP se enfoque en las necesidades críticas del usuario.

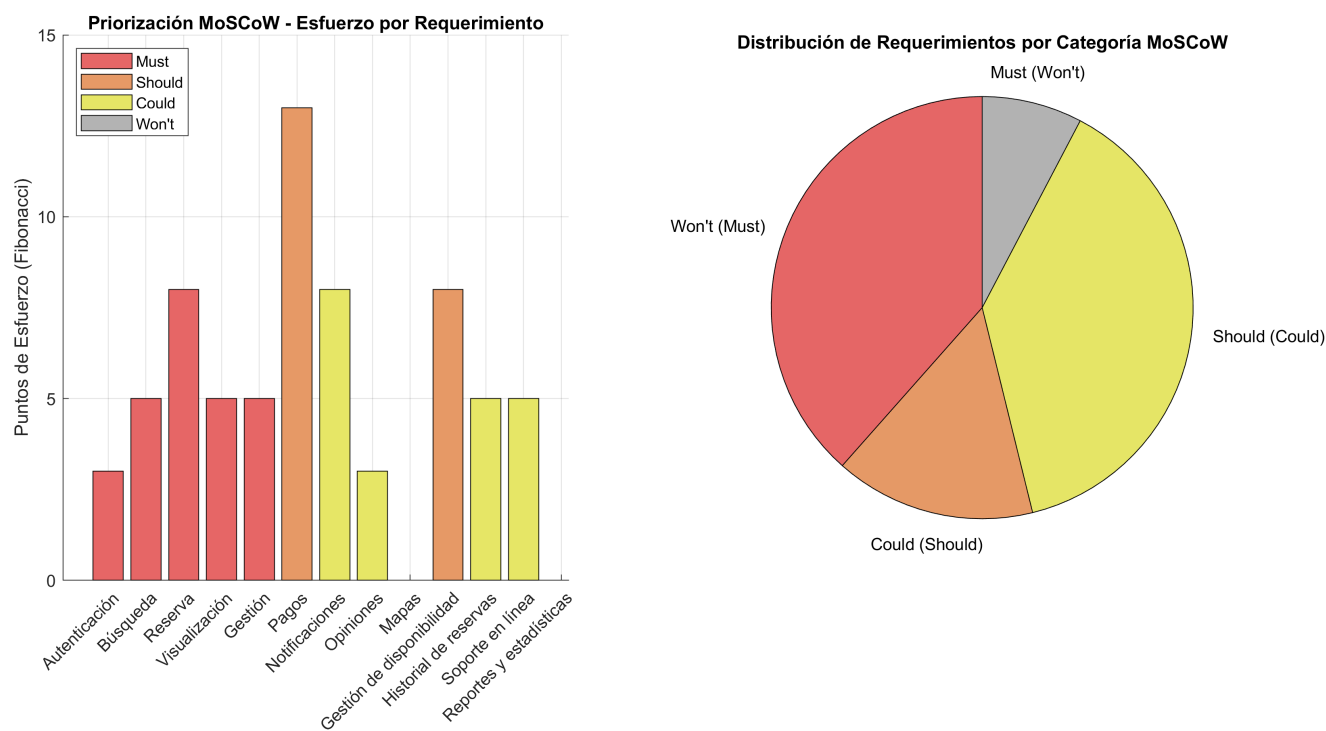
Como se observa, las funcionalidades Must concentran el 62.5 % de los requerimientos, con un esfuerzo total de 26 puntos (ej: reserva en línea y gestión de reglas). El sistema de pagos (Should), aunque complejo (13 puntos), se pospone para iteraciones posteriores. Esta distribución asegura una entrega inicial viable con capacidad de escalabilidad.

En la Figura 1 se complementa el análisis mediante dos perspectivas: (a) el esfuerzo técnico por funcionalidad y (b) la distribución porcentual de requerimientos por categoría. Esta dualidad permite evaluar tanto la complejidad individual como el equilibrio general del proyecto. El gráfico de barras revela que la reserva en línea (8 puntos) y el sistema de pagos (13 puntos) son los componentes más demandantes. En paralelo, el diagrama circular confirma que el 81.25 % de los esfuerzos se destinan a requerimientos Must y Should, alineándose con los objetivos del MVP.

Tabla 1: Análisis de Requerimientos MoSCoW

Requerimiento	Categoría	Esfuerzo	Justificación
Autenticación de usuarios y administradores	Must	3	Complejidad técnica baja (Firebase Authentication simplifica el proceso). Impacto alto en seguridad y acceso diferenciado. Recursos: experiencia en integración de APIs.
Búsqueda de espacios con filtros básicos	Must	5	Complejidad media (integración frontend/backend). Impacto crítico (core del proyecto). Recursos: conocimientos en React y Node.js.
Reserva de espacios en línea	Must	8	Alta complejidad (gestión de concurrencia y validación en tiempo real). Impacto esencial (MVP). Recursos: PostgreSQL para transacciones ACID.
Visualización en tiempo real de ocupación	Must	5	Complejidad media-alta (WebSockets o polling). Impacto alto en prevención de conflictos. Recursos: bibliotecas React disponibles.
Gestión administrativa de reglas	Must	5	Complejidad media (CRUD con validaciones). Impacto diferenciador. Recursos: experiencia en Express.js.
Sistema de pagos integrado	Should	13	Alta complejidad (APIs de pago). Impacto importante pero posponible para MVP. Requiere investigación en Stripe/- PayPal.
Notificaciones (email/SMS)	Could	8	Complejidad media (Twilio/SendGrid). Impacto medio en UX. Dependencia de servicios externos.
Opiniones de usuarios	Could	3	Baja complejidad (CRUD simple). Valor agregado no esencial para MVP.
Integración con mapas	Won't	–	Complejidad alta (API de Google Maps). Excede alcance del MVP.
Gestión de disponibilidad avanzada	Should	8	Complejidad media (manejo de reglas horarias, cancelaciones y bloqueos). Impacto alto en eficiencia de reservas. Recursos: validaciones en backend.
Historial de reservas	Could	5	Complejidad media (asociación de usuarios con reservas previas). Impacto medio en experiencia del usuario. Recursos: base de datos optimizada.
Soporte en línea (chatbot o formulario)	Could	5	Complejidad media (uso de chatbots o sistema de tickets). Impacto medio en atención al cliente. Recursos: integración con herramientas externas.
Reportes y estadísticas para administradores	Could	8	Complejidad media-alta (análisis de datos y visualización de tendencias). Impacto alto en la toma de decisiones. Recursos: librerías de gráficos y dashboard.

Figura 1: Análisis de priorización y esfuerzo de requerimientos utilizando el método MoSCoW. (a) Gráfico de barras que muestra la estimación de esfuerzo para cada requerimiento. (b) Diagrama de pastel que representa la distribución porcentual de requerimientos según su prioridad.



3. Análisis gestión de software

3.1. Tiempo

Tabla 2: Estimación temporal del desarrollo (Incluye nuevos Must)

Requerimiento	Puntos Fibonacci	Tiempo (días)	Distribución de etapas
Autenticación de usuarios y administradores	3	4	Diseño (1d), Desarrollo (2d), Pruebas (1d)
Búsqueda de espacios con filtros básicos	5	6	Diseño (2d), Desarrollo (3d), Pruebas (1d)
Reserva de espacios en línea	8	10	Diseño (3d), Desarrollo (5d), Pruebas (2d)
Visualización en tiempo real de ocupación	5	6	Diseño (2d), Desarrollo (3d), Pruebas (1d)
Gestión administrativa de reglas	5	6	Diseño (2d), Desarrollo (3d), Pruebas (1d)
Total MVP	26	32	6.4 semanas (5 días laborales/semana)

Justificación: La estimación considera 4 desarrolladores estudiantes trabajando en paralelo con dedicación parcial. Cada punto Fibonacci se ha ajustado considerando la curva de aprendizaje y las responsabilidades académicas. Las etapas incluyen diseño arquitectónico, desarrollo de funcionalidades y pruebas de integración.

3.2. Costo

Notas:

- Costos de desarrollo inicial: \$2,375.55 (sin cambios por ser costo fijo)

Tabla 3: Desglose de costos del MVP (Actualizado)

Concepto	Detalle	Costo (USD)
Recursos humanos	4 desarrolladores junior (2.5M COP c/u)	2375.55
Heroku	Plan Hobby para despliegue	7
PostgreSQL	Add-on básico en Heroku	9
Firebase	Plan Spark (gratuito)	0
Stripe	Modo pruebas para pagos	0
Total mensual		2,391.55

- Buffer presupuestal actualizado: 20 % = \$478.31 (nuevo total con buffer: \$2,869.86)

3.3. Alcance

- **Incluye en MVP:**
 - Requerimientos "Must" priorizados con esfuerzo ajustado
 - Optimización del stack técnico (Firebase Auth, PostgreSQL transaccional)
 - Validación técnica de componentes complejos (WebSockets para visualización)
- **Excluye del MVP:**
 - Sistema de pagos integrado (Should - Complejidad alta)
 - Notificaciones automáticas (Could - Dependencia de Twilio/SendGrid)
 - Gestión de disponibilidad avanzada (Should - Reglas horarias complejas)
 - 5 nuevos requerimientos "Could" (ver Tabla 1)

Gestión de riesgos (actualizada):

- *Nuevo riesgo técnico:* Dependencia de servicios externos (Firebase/Heroku)
- *Buffer ajustado:* 20 % aplicado sobre nuevo presupuesto base
- *Comunicación:* Reuniones diarias de sincronización (15 min)

8. Diseño y arquitectura

8.1. Arquitectura usada

El sistema ReservaPlus adopta una arquitectura basada en cliente-servidor con separación entre frontend y backend. El backend sigue un modelo RESTful API utilizando Node.js y Express.js, mientras que el frontend se desarrolla con React.js. La base de datos utilizada es PostgreSQL, alojada en Heroku.

8.1.1. Justificación de la elección

- **Escalabilidad:** La arquitectura cliente-servidor permite escalabilidad horizontal del backend y optimización del frontend según la carga.
- **Flexibilidad:** Separar la lógica de negocio del frontend permite actualizaciones y mejoras sin afectar la experiencia del usuario.
- **Seguridad:** Uso de Firebase para la autenticación de usuarios y PostgreSQL para garantizar integridad de datos.
- **Eficiencia:** PostgreSQL maneja transacciones ACID necesarias para reservas seguras.

8.2. Diseño de base de datos

ReservaPlus emplea un modelo relacional con las siguientes entidades principales:

- Usuarios (ID, Nombre, Correo, Rol)
- Espacios (ID, Nombre, Ubicación, Capacidad, Precio, PropietarioID)
- Reservas (ID, UsuarioID, EspacioID, FechaHora, Estado)
- Pagos (ID, ReservaID, Monto, Estado, Metodo)
- Opiniones (ID, UsuarioID, EspacioID, Calificacion, Comentario)

8.2.1. Justificación de Elección SQL vs. NoSQL

Se eligió PostgreSQL por:

- Consistencia y transacciones seguras: Garantiza que una reserva no se duplique o genere conflictos de disponibilidad.
- Consultas complejas: Relacionar datos como reservas por usuario y espacios disponibles en tiempo real.
- Escalabilidad controlada: Aunque No SQL ofrece flexibilidad, PostgreSQL con índices y particionamiento permite manejar grandes volúmenes de datos eficientemente.

9. Patrones de diseño

9.1 Patron Usado : Observer Pattern

Para este proyecto elegimos el observer pattern debido a que la lógica del sistema requiere notificar automáticamente a los usuarios (o módulos) cuando se realizan cambios en el estado de un espacio. Por ejemplo, si un espacio se reserva, se elimina o cambia su disponibilidad, los usuarios interesados deben ser informados en tiempo real.

9.2 Qué problema resuelve

Mantiene sincronización automática entre los usuarios y los estados de los espacios sin necesidad de realizar consultas constantes a la base de datos.

Reduce el acoplamiento, permitiendo que el sistema se mantenga modular: los espacios no necesitan saber específicamente quiénes los observan.

9.3: Por qué fue necesario en el proyecto:

Los usuarios deben recibir notificaciones automáticas. También es clave para actualizar interfaces en tiempo real y evitar inconsistencias.