1 Documento Inicial – SkyPort v2

1.1 Histórico de cambios

FECHA	DESCRIPCIÓN DE CAMBIO
09/12/2025	Arreglos varios Añadido XP y nivel Arreglo acciones -> bitcoin
07/12/2025	Implementación daltonismo Correcciones pequeñas generales
05/12/2025	Arreglo de rutas Funcionalidad de admin Implementación misiones Corrección flota y mercado
04/12/2025	Implementación pantalla de banco Implementación gestión de misiones
03/12/2025	Implementación código de gestión de flota
02/12/2025	Autenticación implementada para páginas. Incorporada versión con gestión de sesiones y pantallas register y login
29/11/2025	Funcionalidad del register. Rutas de acceso a los recursos
27/11/2025	Creación de la base de datos, documentación asociada y carga de datos inicial
04/11/2025	Cambios pequeños. Mercado. Chat. Flota.
03/11/2025	Unificación de estilos en las páginas. Mejoras de social y otras páginas.
02/11/2025	Cambios en admin. Login explore.
01/11/2025	Página social. Página misiones. Página banco. Página explore Cambios de HTML y CSS en game y redirección
29/10/2025	Modificación del login. Página de recuperación de contraseña. Página de register. Página game.
21/10/2025	Páginas index y login
21/09/2025	Commit inicial. Descripción del proyecto, definición de requisitos funcionales, creación diagrama de casos de uso, tabla de histórico de versiones y creación del documento
09/09/2025	Inicio del proyecto

1.2 Descripción del proyecto

SkyPort es una aplicación web de gestión y simulación aeroportuaria con un enfoque social y colaborativo en tiempo real. La experiencia comienza en el momento en que un jugador accede a la plataforma: puede entrar como invitado, registrarse como usuario o iniciar sesión como administrador. Cada jugador registrado recibe su propio aeropuerto inicial, vacío pero lleno de posibilidades, y con recursos suficientes para empezar a construir su imperio aéreo.

Desde el primer momento, el usuario puede visualizar en un tablero el estado de su infraestructura, su flota y su economía. La progresión es uno de los motores principales del juego: comprar aviones, enviarlos a misiones, calcular recompensas y participar en eventos temporales para optimizar ingresos. Cada avión tiene atributos únicos que afectan directamente al resultado de las misiones, lo que obliga al jugador a tomar decisiones estratégicas sobre qué comprar o vender.

La interacción no termina en el aeropuerto propio. SkyPort fomenta la conexión entre jugadores: permite buscar amigos, enviar solicitudes, visitar aeropuertos ajenos, etc. Además, dispone de un mercado dinámico donde los jugadores pueden comprar o vender, con filtros. Para mantener viva la comunidad, el sistema incorpora un chat global en tiempo real, lo que convierte la plataforma en un espacio social y no solo en un juego de gestión.

SkyPort también busca que los usuarios estén conectados con el mundo real de la aviación. Por eso, incluye un feed de noticias aeronáuticas actualizado en tiempo real, que los jugadores pueden consultar y filtrar por temas de interés, además de la posibilidad de ver acciones de empresas aeronáuticas en directo. Los administradores, por su parte, disponen de herramientas para moderar usuarios y lanzar eventos globales que mantengan el entorno dinámico y atractivo.

Detrás de estas funcionalidades hay un objetivo claro: crear un ecosistema de juego vivo y colaborativo, donde cada acción tenga impacto y cada jugador sienta que su progreso importa. Queremos motivar al usuario a volver cada día, a ver cómo avanza su aeropuerto, a celebrar cuando sus aviones completan misiones y a interactuar con la comunidad. SkyPort busca que la experiencia no sea estática sino en tiempo real.

Nuestra motivación principal es combinar la estrategia y la simulación con la interacción social, de forma que no sea solo un juego de gestión, sino también una plataforma donde los jugadores puedan compartir logros, colaborar y competir de manera sana. Apostamos por un diseño accesible y moderno, que permita jugar desde cualquier dispositivo sin instalaciones, y que sea lo suficientemente escalable para añadir en el futuro nuevos tipos de misiones, eventos, logros y personalizaciones.

En definitiva, SkyPort es una experiencia que mezcla diversión, estrategia y comunidad en un solo lugar, pensada para mantener al jugador enganchado y para ofrecer siempre algo nuevo que hacer, descubrir o mejorar.

1.3 Diagrama de casos de uso

A continuación, se incluye una lista de los casos de uso principales (el diagrama visual puede añadirse posteriormente en formato imagen si se desea):

- UC01 Registrar usuario
- UC02 Iniciar sesión
- UC03 Cerrar sesión

- UC04 Recuperar contraseña
- UC05 Explorar jugadores en modo invitado
- UC06 Acceder al panel de juego
- UC07 Consultar flota de aviones
- UC08 Comprar avión en el mercado
- UC09 Vender avión
- UC10 Asignar misión a un avión
- UC11 Consultar estado de misiones
- UC12 Resolver misión completada
- UC13 Abortar misión en curso
- UC14 Consultar economía y movimientos
- UC15 Enviar mensaje en el chat global
- UC16 Ver usuarios conectados
- UC17 Gestionar solicitudes de amistad
- UC18 Consultar eventos activos
- UC19 Consultar cotización externa
- UC20 Administrar usuarios (panel admin)
- UC21 Administrar catálogo de misiones (panel admin)

1.3.1 UC01 – Registrar usuario

Actores:

- Visitante
- Sistema de autenticación

Descripción:

El visitante crea una cuenta nueva en SkyPort v2 para acceder al sistema y obtener su aeropuerto inicial, flota y saldo base

Precondiciones:

- Formulario de registro disponible
- Base de datos accesible

Postcondiciones:

- Se crea un usuario con datos básicos
- Se crea balance inicial y aeropuerto
- Se registra un movimiento de crédito inicial
- Se inicia una sesión automáticamente

Flujo principal:

1. El visitante accede al formulario de registro
2. Introduce username, email, contraseña, nombre y DNI

3. El sistema valida formatos y unicidad
4. El sistema registra usuario, aeropuerto y movimiento económico
5. Se crea la sesión del usuario
6. El sistema redirige al panel de juego

Flujos alternativos / excepciones

- 3a. Username o email ya existen en la base de datos --> error 409
- 3b. Contraseña débil --> error 400
- 5a. Error creando la sesión --> error 500

1.3.2 UC02 – Iniciar sesión

Actores:

- Usuario
- Sistema de autenticación

Descripción:
El usuario introduce sus credenciales para acceder al panel principal del juego.

Precondiciones:

- Usuario registrado y activo.
- Base de datos y servidor en funcionamiento.

Postcondiciones:

- Se crea una sesión válida.
- El usuario es redirigido al dashboard /game.

Flujo principal:

1. El usuario accede a la página de inicio de sesión.
2. Introduce email/usuario y contraseña.
3. El sistema valida credenciales.
4. El sistema regenera sesión y guarda datos del usuario.
5. Redirige al panel de juego.

Flujos alternativos / excepciones:

- 3a. Credenciales incorrectas → 401.
- 3b. Usuario suspendido → 403.
- 3c. Error BD → 500.

1.3.3 UC02 – Cerrar sesión

Actores:

- Usuario

Descripción:
El usuario finaliza su sesión actual y vuelve al login.

Precondiciones:

- Hay una sesión activa.

Postcondiciones:

- La sesión se destruye.
- La cookie deja de ser válida.
- El usuario es redirigido a /login.

Flujo principal:

1. El usuario pulsa “Cerrar sesión”.
2. El servidor invalida la sesión.
3. Se redirige al login.

Flujos alternativos / excepciones:

- 2a. Error invalidando sesión → mostrar mensaje general.

1.3.4 UC04 – Recuperar contraseña

Actores:

- Usuario registrado
- Sistema de correo (futuro)

Descripción:
El usuario solicita restablecer su contraseña mediante correo electrónico.

Precondiciones:

- El email del usuario está registrado.

Postcondiciones:

- (Previsto) Se enviaría un enlace o token de recuperación.

Flujo principal (previsto):

1. El usuario solicita recuperación.
2. El sistema genera token y envía email.
3. Usuario accede al enlace y define nueva contraseña.

Flujos alternativos / excepciones:

- No implementado actualmente.

1.3.5 UC05 – Explorar jugadores en modo invitado

Actores:

- Invitado
- Usuario autenticado

Descripción:
Permite buscar y visualizar aeropuertos y flotas de otros jugadores sin iniciar sesión.

Precondiciones:

- API pública accesible.

Postcondiciones:

- Se muestra listado de jugadores y su aeropuerto/flota.

Flujo principal:

1. El visitante accede al buscador público.
2. Introduce nombre/criterio.
3. El sistema devuelve lista de jugadores.
4. El visitante elige uno y visualiza aeropuerto y flota.

Flujos alternativos / excepciones:

- 3a. No hay resultados → lista vacía.
- 4a. Jugador no existe → 404.

1.3.6 UC06 – Acceder al panel de juego

Actores:

- Usuario autenticado

Descripción:
El usuario entra al dashboard principal del juego, donde ve KPIs, radar y accesos a módulos.

Precondiciones:

- Sesión activa.

Postcondiciones:

- El usuario visualiza datos principales: saldo, aviones, misiones, eventos, radar.

Flujo principal:

1. Usuario navega a /game.
2. Middleware valida sesión.
3. Se cargan datos básicos del juego.
4. UI muestra el panel.

Flujos alternativos / excepciones:

- 2a. Sesión expirada → redirección a /login.

1.3.7 UC07 – Consultar flota de aviones

Actores:

- Usuario autenticado

Descripción:
El usuario visualiza todos sus aviones, con estado individual.

Precondiciones:

- Existen aviones asociados al usuario.

Postcondiciones:

- Se muestra la flota categorizada por estado.

Flujo principal:

1. Usuario accede a flota (/game/fleet).
2. Sistema consulta user_aircraft.
3. Clasifica según estado (idle / running / mantenimiento).
4. UI presenta tarjetas de aviones.

Flujos alternativos / excepciones:

- 2a. No tiene aviones → mensaje “No tienes aviones”.

1.3.8 UC08 – Comprar avion en el mercado

Actores:

- Usuario autenticado

Descripción:
Compra un avión del catálogo usando su saldo.

Precondiciones:

- Catálogo activo.
- Saldo suficiente.
- Límite de flota no superado (máx. 6).

Postcondiciones:

- Se añade avión a la flota.
- Se descuenta saldo y se registra movimiento.

Flujo principal:

1. Usuario abre mercado.
2. Selecciona tipo de avión.
3. POST /buy → servidor valida.
4. Se inserta avión y movimiento económico.
5. UI actualiza flota y saldo.

Flujos alternativos / excepciones:

- Tipo no existe → 404.
- Saldo insuficiente → 400.
- Flota llena → 400.
- Error BD → 500.

1.3.9 UC09 – Vender avion

Actores:

- Usuario autenticado

Descripción:
El usuario vende un avión que no esté en misión.

Precondiciones:

- Avión pertenece al usuario.
- Estado idle.

Postcondiciones:

- Se elimina avión.
- Se genera ingreso del ~70%.

Flujo principal:

1. Usuario selecciona avión.
2. POST /sell → sistema valida propiedad y estado.
3. Registra movimiento económico.
4. Elimina avión.
5. UI actualiza saldo y flota.

Flujos alternativos / excepciones:

- Avión no encontrado → 404.
- Avión ocupado → 400.
- Error BD → 500.

1.3.10 UC10 – Asignar misión a un avion

Actores:

- Usuario autenticado

Descripción:
Asigna una misión disponible a un avión compatible.

Precondiciones:

- Misión activa.
- Avión compatible y en estado idle.
- Saldo suficiente para el coste.

Postcondiciones:

- Misión creada en user_missions.
- Avión pasa a running.
- Saldo actualizado.

Flujo principal:

1. Usuario abre misiones.
2. Selecciona misión.
3. Selecciona avión.
4. Servidor valida disponibilidad y saldo.
5. Crea misión con hora de fin.
6. Marca avión como ocupado.
7. UI muestra misión en curso.

Flujos alternativos / excepciones:

- Misión no existe → 404.
- Avión incompatible → 400.
- Saldo insuficiente → 400.

1.3.11 UC11 – Consultar estado de misiones

Actores:

- Usuario autenticado

Descripción:
El usuario ve sus misiones activas e históricas.

Precondiciones:

- Existen misiones asociadas.

Postcondiciones:

- Se muestra lista con estado, avión y tiempos.

Flujo principal:

1. Accede a /game/missions.

2. Sistema obtiene misiones.
3. Calcula estados actualizados.
4. UI muestra columnas “en curso” y “completadas”.

Flujos alternativos / excepciones:

- Sin misiones → lista vacía.

1.3.12 UC12 – Resolver misión completada

Actores:

- Sistema (principal)
- Usuario autenticado (secundario)

Descripción:
Las misiones finalizadas se resuelven aplicando recompensa y liberando avión.

Precondiciones:

- Hay misiones running con hora de fin vencida.

Postcondiciones:

- Misión pasa a success/failed.
- Avión queda idle.
- Se registran movimientos económicos.

Flujo principal:

1. Cliente o cron llama a /resolve-due.
2. Sistema detecta misiones vencidas.
3. Aplica recompensa o estado final.
4. Libera avión.
5. Devuelve lista actualizada.

Flujos alternativos / excepciones:

- No hay misiones vencidas → respuesta “no_due_missions”.

1.3.13 UC13 – Abortar una misión en curso

Actores:

- Usuario autenticado

Descripción:
El usuario cancela una misión en ejecución.

Precondiciones:

- Misión en running y asociada al usuario.

Postcondiciones:

- Misión pasa a aborted.
- Avión vuelve a idle.

Flujo principal:

1. Usuario pulsa abortar.
2. POST /abort → sistema valida.
3. Actualiza misión y avión.
4. Retorna estado final.

Flujos alternativos / excepciones:

- Misión no encontrada → 404.
- Misión ya finalizada → 400.

1.3.14 UC14 – Consultar economía y movimientos

Actores:

- Usuario autenticado

Descripción:
El usuario consulta movimientos económicos y balance.

Precondiciones:

- Usuario existente.
- Movimientos registrados.

Postcondiciones:

- Se muestran ingresos, gastos y saldo actual.

Flujo principal:

1. Accede a economía.
2. GET /economy → obtiene datos.
3. UI muestra movimientos y totales.

Flujos alternativos / excepciones:

- Usuario no existe → 404.
- Error BD → 500.

1.3.15 UC15 – Enviar mensaje en el chat

Actores:

- Usuario autenticado
- Otros usuarios conectados

Descripción:
Envía un mensaje visible en tiempo real por el chat global.

Precondiciones:

- Sesión activa.
- Socket.IO conectado.

Postcondiciones:

- El mensaje se difunde a todos.

Flujo principal:

1. Usuario abre chat.
2. Escribe mensaje.
3. Cliente emite evento.
4. Servidor agrega metadatos.
5. Broadcast a todos los clientes.
6. UI muestra mensaje.

Flujos alternativos / excepciones:

- Socket desconectado → el mensaje no se envía.

1.3.16 UC16 – Ver usuarios conectados (dentro de social)

Actores:

- Usuario autenticado

Descripción:
El usuario ve qué jugadores están online en tiempo real.

Precondiciones:

- Socket activo.

Postcondiciones:

- La UI muestra usuarios online.

Flujo principal:

1. Cliente recibe identificación inicial.
2. Se suscribe a eventos de presencia.
3. UI actualiza lista de conectados.

Flujos alternativos / excepciones:

- Cliente desconectado → aparece offline.

1.3.17 UC17 – Gestionar soluciones de amistad

Actores:

- Usuario autenticado
- Otros jugadores

Descripción:
Envío, aceptación y rechazo de solicitudes de amistad.

Precondiciones:

- Usuario logueado.

Postcondiciones:

- Se crean amistades o se declinan solicitudes.

Flujo principal:

1. Usuario busca otro jugador.
2. POST /social/request.
3. Receptor recibe evento.
4. Receptor responde accept/decline.
5. Si acepta, se crea friendship.
6. UI actualiza listas.

Flujos alternativos / excepciones:

- Auto-solicitud → 400.
- Ya son amigos → 409.
- Solicitud inexistente → 404.

1.3.18 UC18 – Constultar eventos activos

Actores:

- Usuario autenticado

Descripción:
El usuario ve los eventos activos del juego (boosts, misiones especiales…).

Precondiciones:

- Existen eventos activos en BD.

Postcondiciones:

- UI muestra tarjetas de eventos.

Flujo principal:

1. GET /events.

2. Servidor filtra por fechas y estado.
3. UI muestra eventos activos.

Flujos alternativos / excepciones:

- Sin eventos → lista vacía.

1.3.19 UC19 – Consultar cotización externa (Ryanair)

Actores:

- Visitante o usuario autenticado

Descripción:
Consulta el precio actual de la acción de Ryanair usando una API externa.

Precondiciones:

- API key configurada.
- Internet o proveedor accesible.

Postcondiciones:

- Se muestra precio, fuente y timestamp.

Flujo principal:

1. Cliente llama a /api/stocks/ryanair.
2. Sistema usa cache o solicita a TwelveData.
3. Devuelve precio y metadatos.
4. UI actualiza gráfico.

Flujos alternativos / excepciones:

- API key ausente → 500.
- Fallo proveedor externo → usa cache o 502.

1.3.20 UC20 – Administrar usuarios (administrador)

Actores:

- Administrador

Descripción:
Gestión completa de usuarios desde el panel administrativo.

Precondiciones:

- Sesión con rol admin.

Postcondiciones:

- Se crean, modifican o eliminan jugadores (no admins).

Flujo principal:

1. Admin accede a /admin/users.
2. GET lista de usuarios.
3. Puede crear, editar roles, activar/desactivar o borrar.

Flujos alternativos / excepciones:

- Intento de editar admin → 403.
- Duplicados → 409.
- Usuario no existe → 404.

1.3.21 UC21 – Administrar catálogo de misiones (administrador)

Actores:

- Administrador

Descripción:
CRUD completo del catálogo de misiones del juego.

Precondiciones:

- Rol admin.

Postcondiciones:

- Misiones actualizadas o creadas/eliminadas correctamente.

Flujo principal:

1. Admin accede a /admin/missions.
2. GET lista de misiones.
3. Puede crear, editar (coste, recompensa, duración, tipo), activar/desactivar, eliminar.

Flujos alternativos / excepciones:

- Campos obligatorios faltan → 400.
- Misión inexistente → 404.
- Error BD → 500.

1.4 Requisitos funcionales

1. Acceso y cuentas
    a. RF1: El sistema permitirá iniciar sesión como invitado, usuario o administrador.
    b. RF2: El sistema permitirá registrar un nuevo usuario con nombre único.
    c. RF3: El sistema permitirá iniciar/cerrar sesión de un usuario registrado.
    d. RF4: El sistema restringirá el acceso a funciones avanzadas a usuarios

    e. RF5: El sistema permitirá recuperación de cuenta (p. ej., restablecer credenciales).
    f. RF6: El sistema permitirá verificación de nombre de usuario disponible en tiempo real.
2. Aeropuerto del jugador
    a. RF7: Tras registrarse, el usuario dispondrá de un aeropuerto inicial vacío con recursos de arranque.
    b. RF9: El usuario dispondrá de huecos de hangar limitados para albergar aviones.
    c. RF11: El sistema mostrará un tablero del aeropuerto con estado de pistas, huecos, aviones y economía.
3. Aviones y economía
    a. RF12: El usuario podrá comprar aviones de distintos tipos y costes.
    b. RF13: El usuario podrá vender aviones propios a precio fijo.
    c. RF14: Cada avión tendrá atributos de misión (capacidad, alcance, tiempo, recompensa base).
    d. RF15: El usuario podrá asignar aviones a misiones; al completarse, recibirá dinero.
    e. RF16: El sistema calculará la recompensa de misión según tipo de misión, avión y mejoras.
    f. RF17: El sistema aplicará tiempos de misión (cooldown y duración) visibles para el usuario.
    g. RF19: El sistema permitirá bonificaciones temporales (eventos, rachas, logros).
    h. RF20: El sistema ofrecerá la posibilidad de ver la situación económica (ingresos/gastos).
4. Misiones
    a. RF21: El sistema ofrecerá un catálogo de misiones disponible.
    b. RF22: El sistema validará requisitos de misión (pista mínima, tipo de avión, slots libres).
    c. RF23: El sistema mostrará en tiempo real el progreso de cada misión activa.
    d. RF24: El sistema generará misiones especiales por tiempo limitado.
5. Social (amigos y visitas)
    a. RF25: El usuario podrá buscar otros usuarios por nombre y enviar solicitud de amistad.
    b. RF26: El receptor podrá aceptar o rechazar solicitudes de amistad.
    c. RF27: El usuario podrá visitar aeropuertos de otros usuarios (si son públicos o amigos).
    d. RF29: El sistema mostrará el estado básico de los aviones del aeropuerto visitado (en vuelo, en mantenimiento, libres).
6. Mercado e intercambio
    a. RF30: El sistema dispondrá de un mercado para comprar/vender aviones con el sistema.

    b. RF32: El sistema permitirá listados en un mercado público con filtros por tipo/atributos.
7. Chat y comunicación
    a. RF33: El sistema incluirá chat global no persistente.
8. Invitado (modo explore)
    a. RF35: El invitado podrá listar aeropuertos públicos y ver su estado general.
    b. RF36: El invitado podrá visualizar actividad básica (aviones en misión).
    c. RF37: El invitado verá llamados a registrarse para jugar o interactuar.
9. Noticias aeronáuticas en tiempo real
    a. RF38: El sistema mostrará un feed de noticias de aeronáutica en una sección dedicada.
10. Panel del administrador
    a. RF40: El administrador podrá gestionar usuarios (buscar, suspender, restaurar).
    b. RF42: El administrador podrá lanzar eventos globales (boosts, misiones temáticas).
11. Notificaciones y tiempo real
    a. RF43: El sistema enviará notificaciones en tiempo real por eventos clave (misión completada, solicitud de amistad, chat)
12. Seguridad y fair play
    a. RF44: El sistema prevendrá acciones imposibles (compras sin saldo, misiones sin requisitos).

1.5 Justificación de requisitos cumplidos

La mayoría de los requisitos funcionales esenciales han sido implementados con éxito porque se priorizó aquello que permitía asegurar un ciclo completo de juego: registro, progreso económico, gestión de aviones y realización de misiones. Desde el inicio del desarrollo se definió como objetivo principal disponer de una base sólida y estable sobre la cual el jugador pudiera interactuar con su aeropuerto, generar ingresos y avanzar en la experiencia del juego. Por ello se dedicaron recursos a construir la infraestructura central del sistema: autenticación segura, gestión de usuarios, economía interna, operaciones de compra/venta y un motor básico de misiones totalmente funcional.

Asimismo, se completaron funcionalidades sociales críticas, como el sistema de amistades y la exploración de aeropuertos públicos, ya que aportan interacción entre jugadores sin requerir una gran complejidad técnica adicional. También se consiguieron implementar elementos clave de tiempo real como el chat global y las actualizaciones de temporizadores de misión, garantizando una experiencia dinámica y conectada.

En conjunto, los requisitos cumplidos representan todos aquellos que eran imprescindibles para que el juego fuese jugable de principio a fin y reflejara correctamente su concepto

central. Se priorizó aquello que desbloqueaba el uso cotidiano de la plataforma, mantenía la coherencia del sistema económico y garantizaba la consistencia del mundo compartido entre jugadores, dejando para fases posteriores las funcionalidades más avanzadas o complementarias.

1.6 Requisitos no cumplidos y justificación

RF8 – Mejorar pistas (tierra → asfalto → oro): No se implementó porque requiere un sistema de niveles y efectos sobre misiones que habría cambiado el balance económico. Se priorizó mantener la estabilidad del núcleo del juego.

RF10 – Ampliar hangar para aumentar número de aviones: La ampliación implicaba rediseñar límites internos y una interfaz específica. Se descartó por falta de tiempo y para no afectar la lógica de compra de aviones ya funcional.

RF18 – Costes de mantenimiento periódicos: Exige tareas programadas que afectan automáticamente al saldo del jugador. Se pospuso por su complejidad y riesgo de generar inconsistencias económicas:

RF28 – Enviar temporalmente un avión a un amigo: Requiere sincronizar estados entre cuentas y validar disponibilidad en ambos jugadores. Se dejó fuera por la complejidad de coordinación y lógica compartida.

RF31 – Intercambios entre jugadores: Implica un sistema de negociación y validaciones dobles que podría desequilibrar la economía. Por su alcance y riesgo de abuso, se pospuso.

RF34 – Chat privado persistente entre usuarios: Necesitaba almacenamiento de mensajes y canales privados con control de acceso. Se descartó por falta de tiempo y para evitar sobrecargar el backend.

RF39 – Feed actualizado en tiempo real con filtros: Dependía de integrar fuentes externas y un sistema de actualización continua. Se omitió al no contar con un proveedor estable durante el desarrollo.

RF41 – Configurar parámetros del juego desde admin: Requería un panel completo con impacto directo en precios y timers del juego. Se pospuso para evitar desbalancear el sistema sin pruebas exhaustivas.