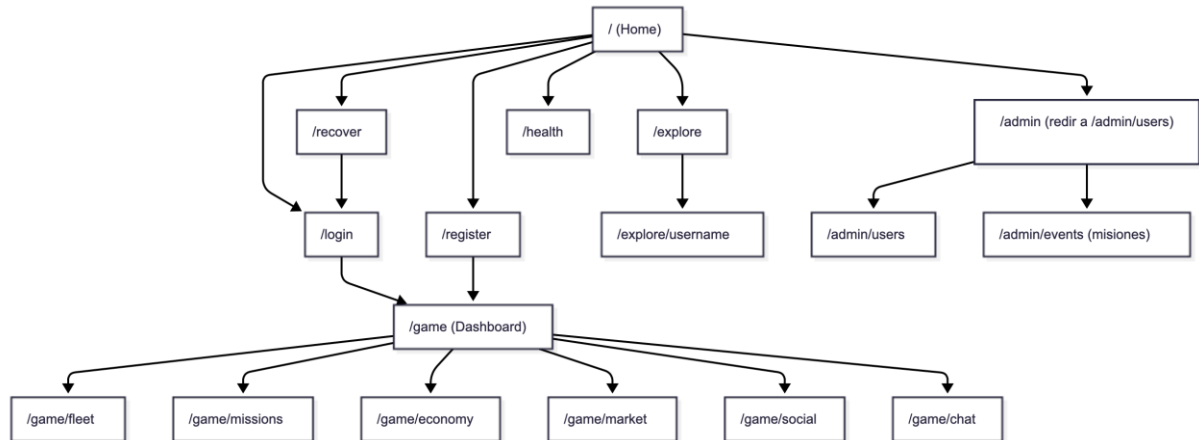


# 1 Documento de Diseño – SkyPort v2

## 1.1 Sitemap

En este apartado se define la estructura general de navegación de SkyPort v2 y las pantallas principales accesibles por el usuario.

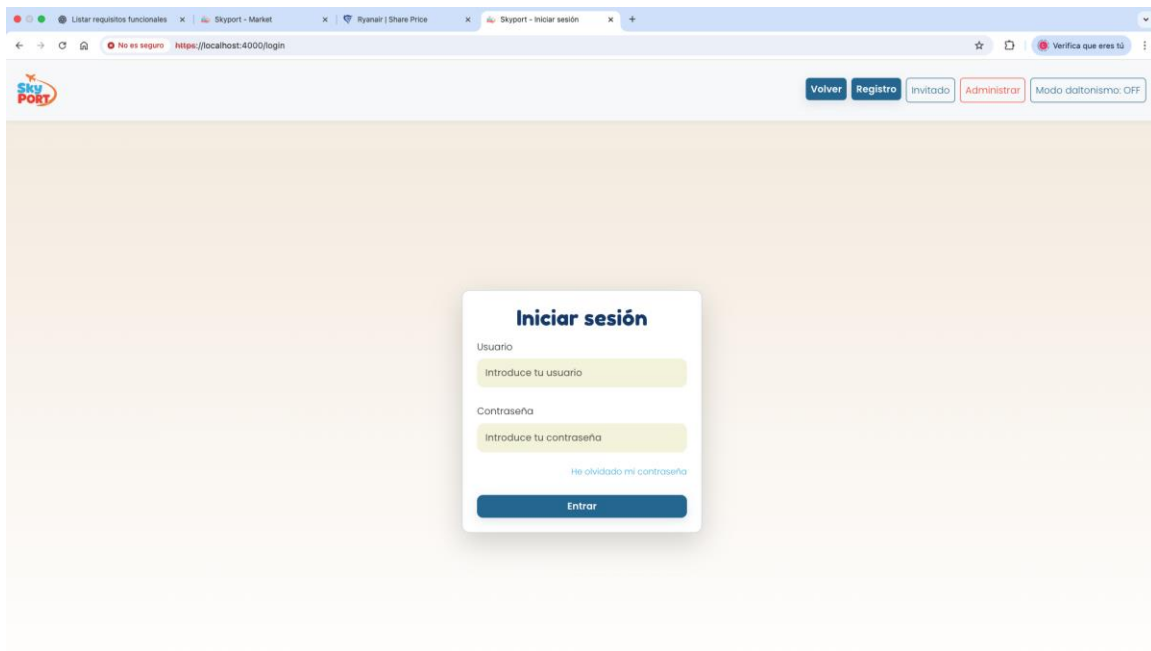


## 1.2 Wireframes

Ahora se describen los wireframes, es decir, la estructura y elementos funcionales de cada pantalla. El objetivo es definir qué aparece en cada vista y cómo interactúa el usuario con el sistema.

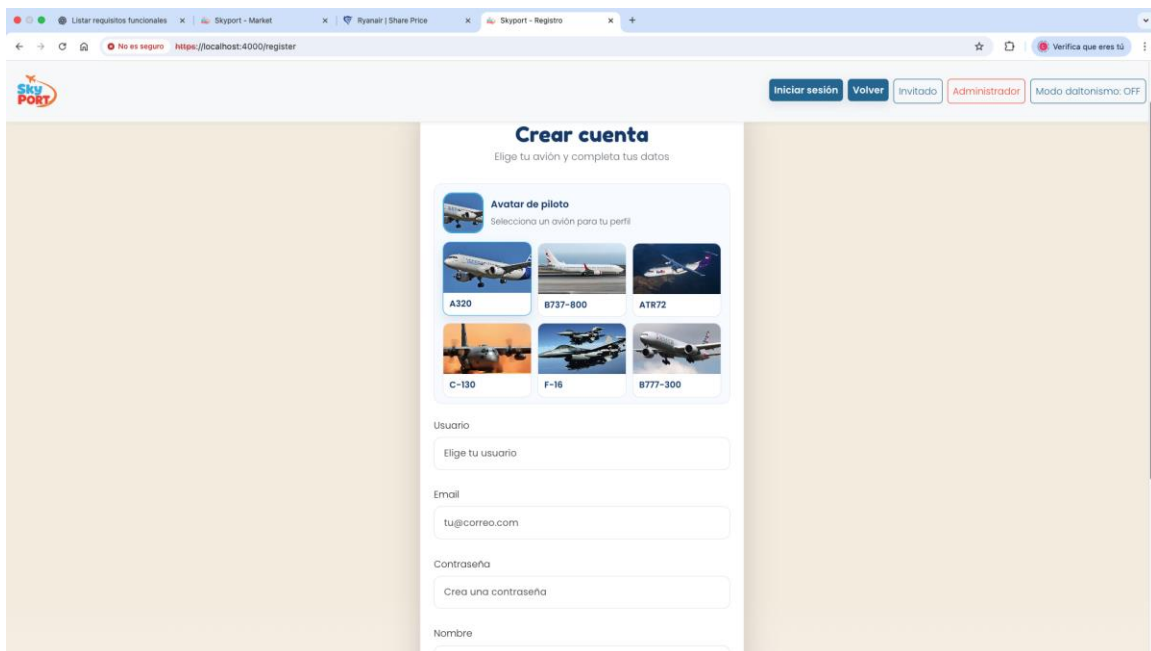
### Login

- Formulario con usuario/email y contraseña.
- Botón “Entrar”.
- Enlaces a registro y recuperar.
- Mensajes de error ante credenciales inválidas.



## Registro

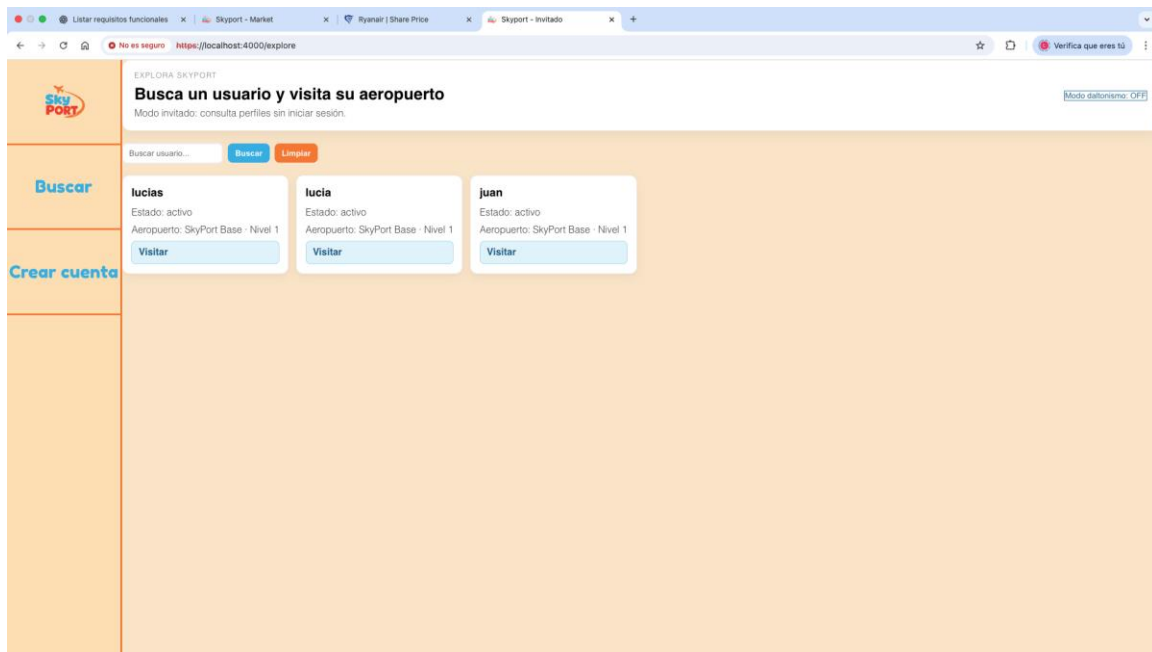
- Formulario username, email y password.
- Botón “Crear cuenta”.
- Al crear: alta con saldo, sesión iniciada y redirección al juego.



## Explorar (público)

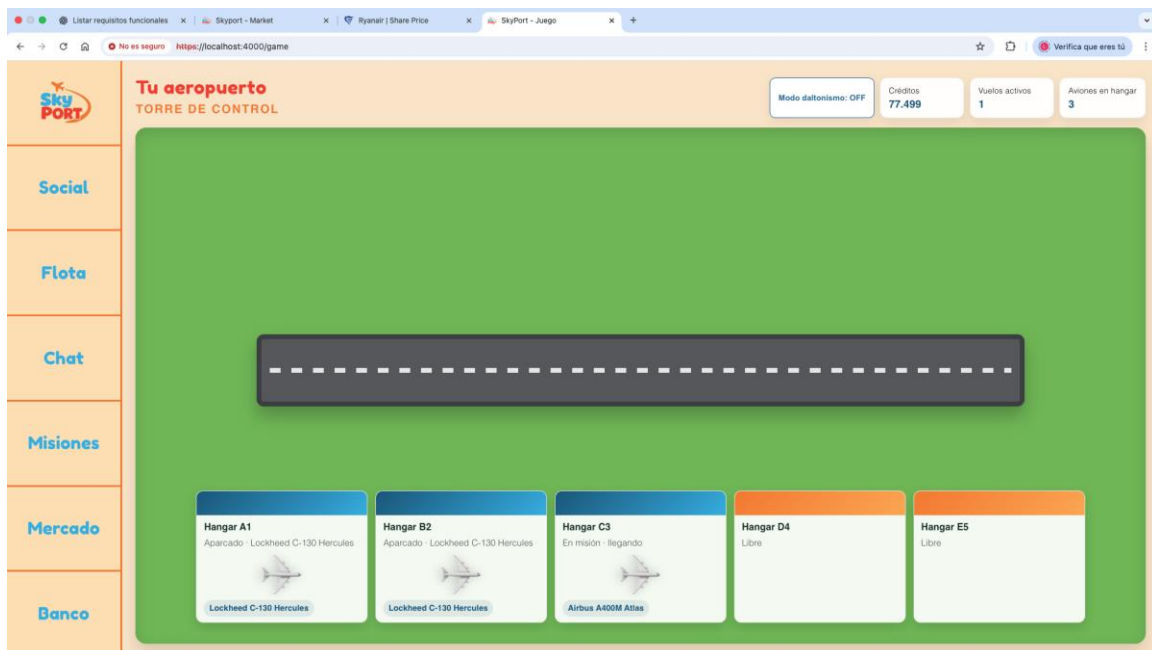
- Buscador de usuarios.
- Lista de resultados (excluye admins).

- Ficha de usuario: muestra aeropuerto base y flota visible como invitado.
- CTA para ir a login/registro.



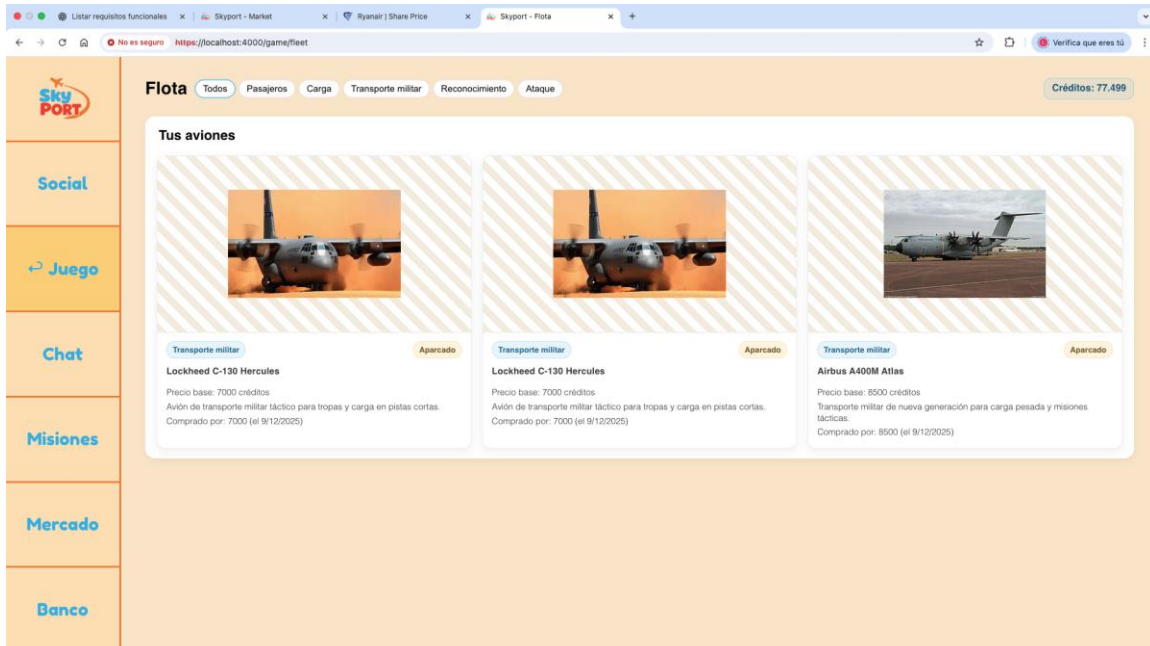
## Dashboard / Game (vista principal)

- Navbar con accesos a Flota, Misiones, Economía, Mercado, Social, Chat.
- KPIs rápidos: saldo, aviones totales, aviones en misión.
- Escena aeropuerto con 5 hangares: muestra estado de cada avión y animaciones de despegue/aterrizaje cuando se activa misión.



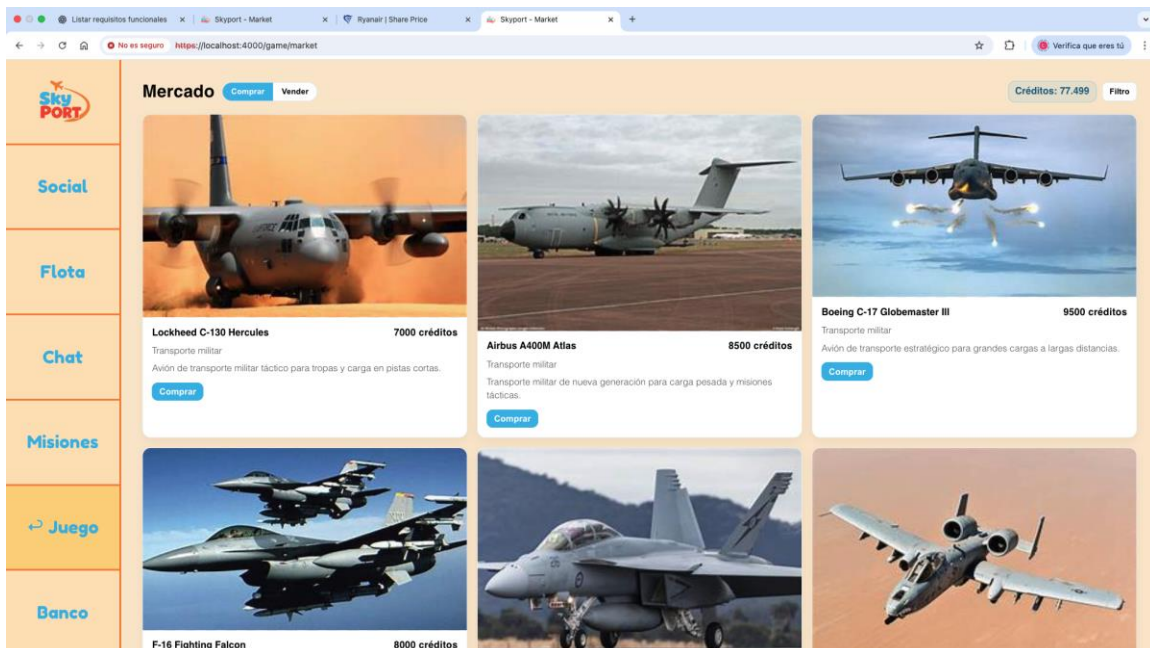
## Flota

- Listado/tarjetas de aviones propios con estado (idle/running/maintenance), modelo y rol.
- Visualización por slots de hangar (no gestión de mejora).
- Acceso al mercado para comprar más aviones.



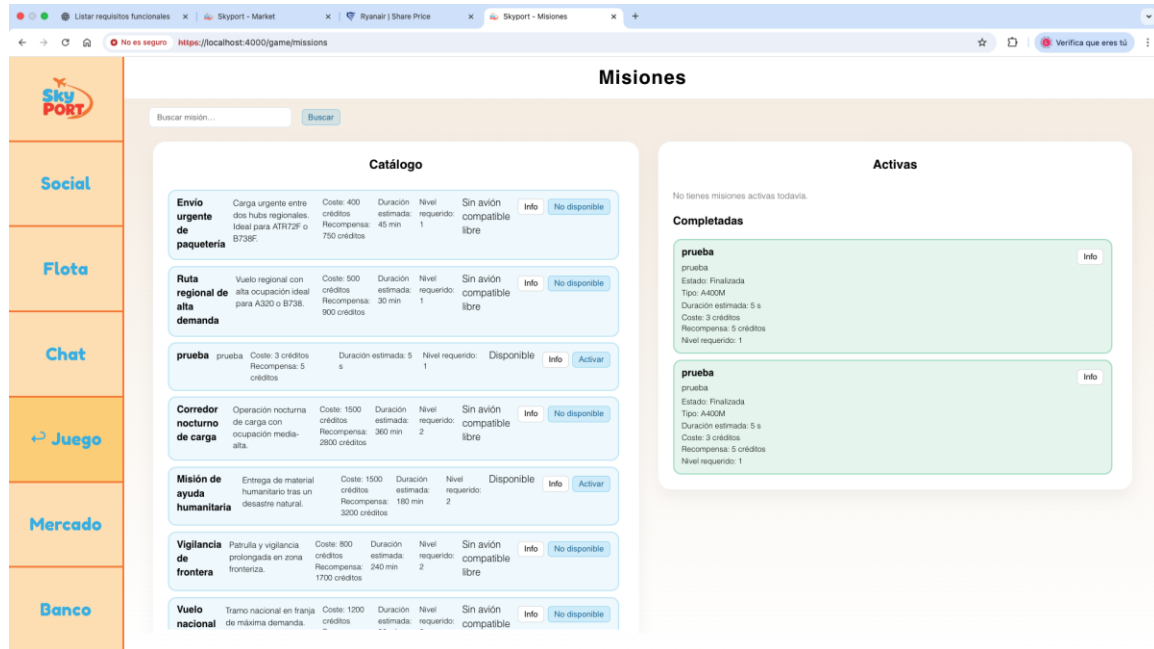
## Mercado

- Catálogo de tipos activos con precio.
- Comprar avión (valida saldo y límite de flota 5) y actualiza saldo/flota.
- Vender avión propio (excepto si está en misión) con precio de reventa.



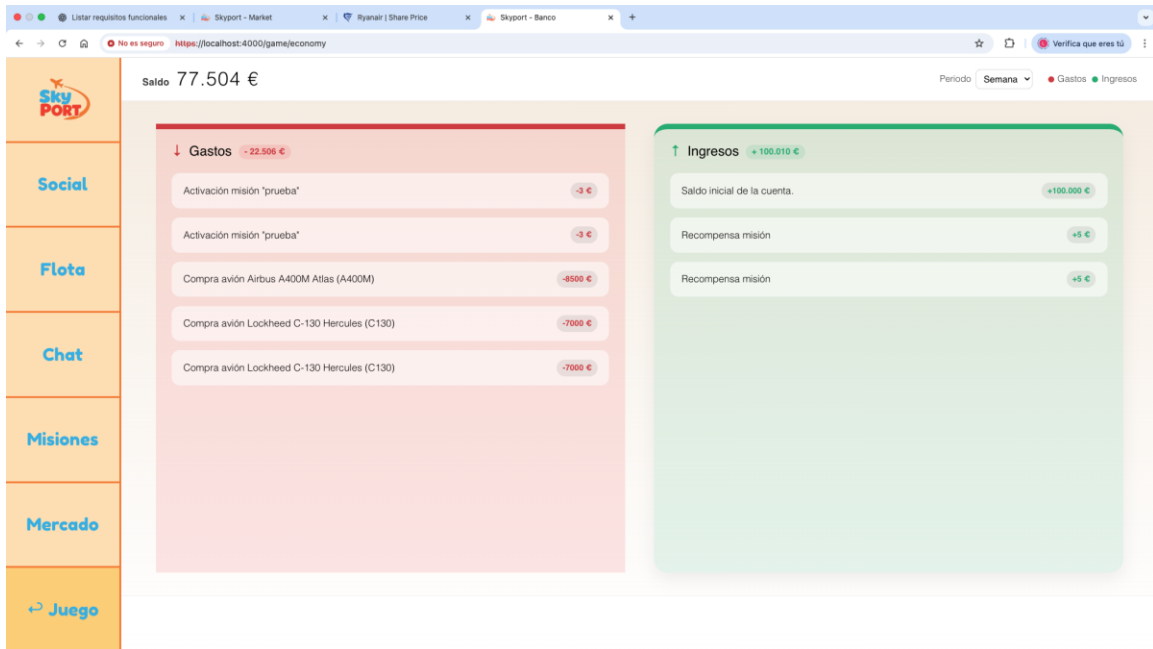
## Misiones

- Columna de catálogo de misiones activas (coste, recompensa, duración).
- Columna de misiones del usuario con estado: en curso, completadas, abortadas.
- Botón “Iniciar/Activar” misión; asigna avión compatible y descuenta coste.
- Acción “Resolver vencidas” para aplicar recompensas y liberar aviones.
- Contadores de tiempo restante en UI principal (basados en finishedAt).



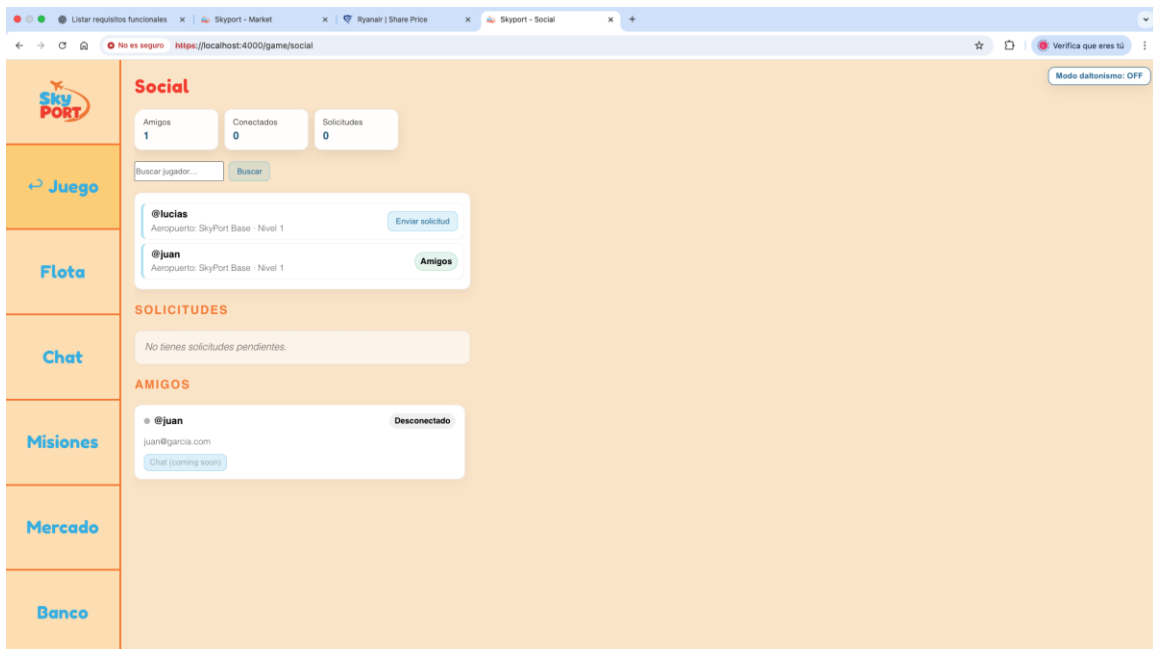
## Economía

- Balance actual.
- Listas de ingresos y gastos a partir de movimientos reales.
- Totales por columna; formato moneda.



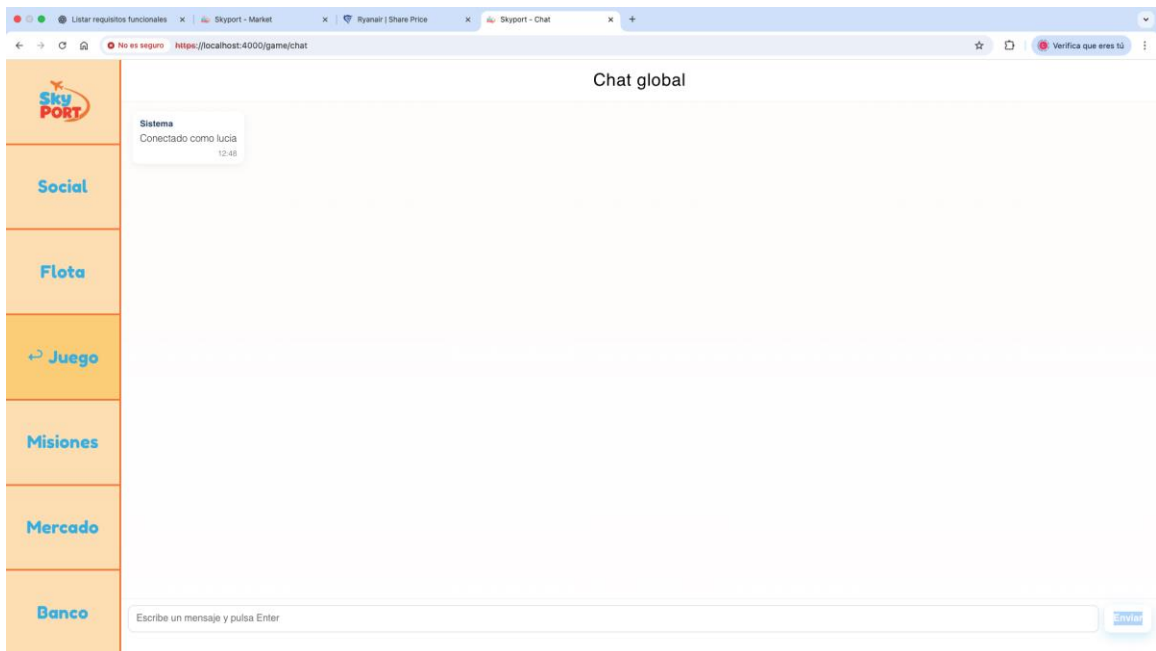
## Social / Amigos

- Buscador de jugadores.
- Botón “Enviar solicitud”.
- Bandeja de solicitudes entrantes con aceptar/declinar.
- Lista de amigos con indicador online.



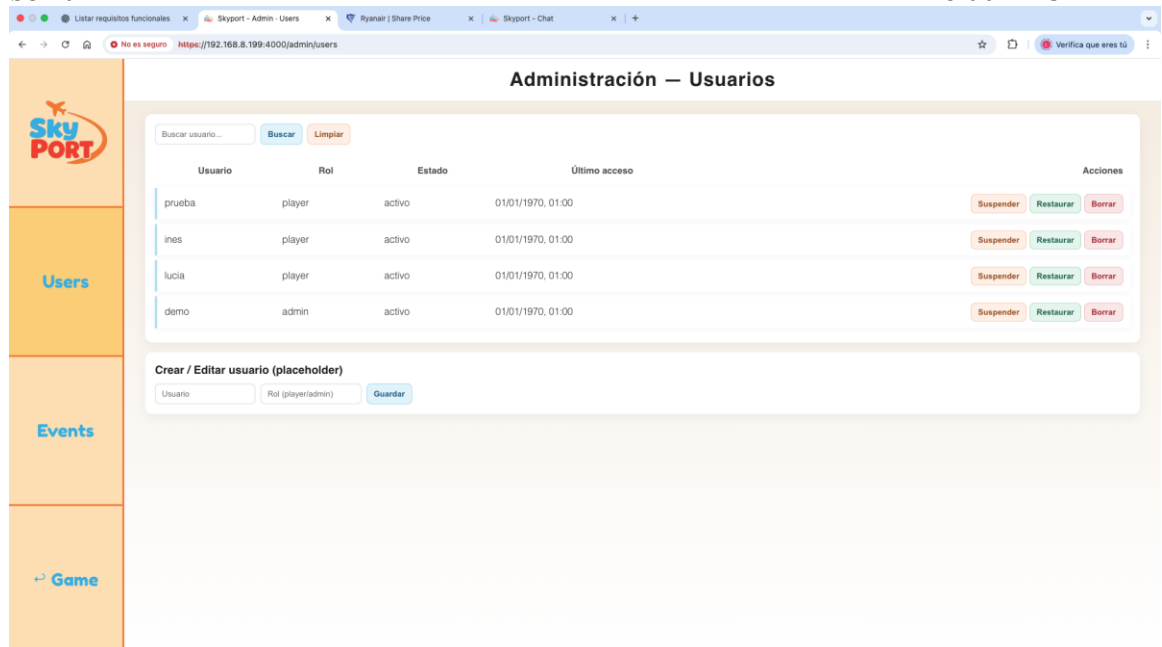
## Chat global

- Historial de mensajes en tiempo real (Socket.IO).
- Campo de texto y botón/enviar con Enter.
- Muestra autor y hora de cada mensaje.



## Admin – Usuarios

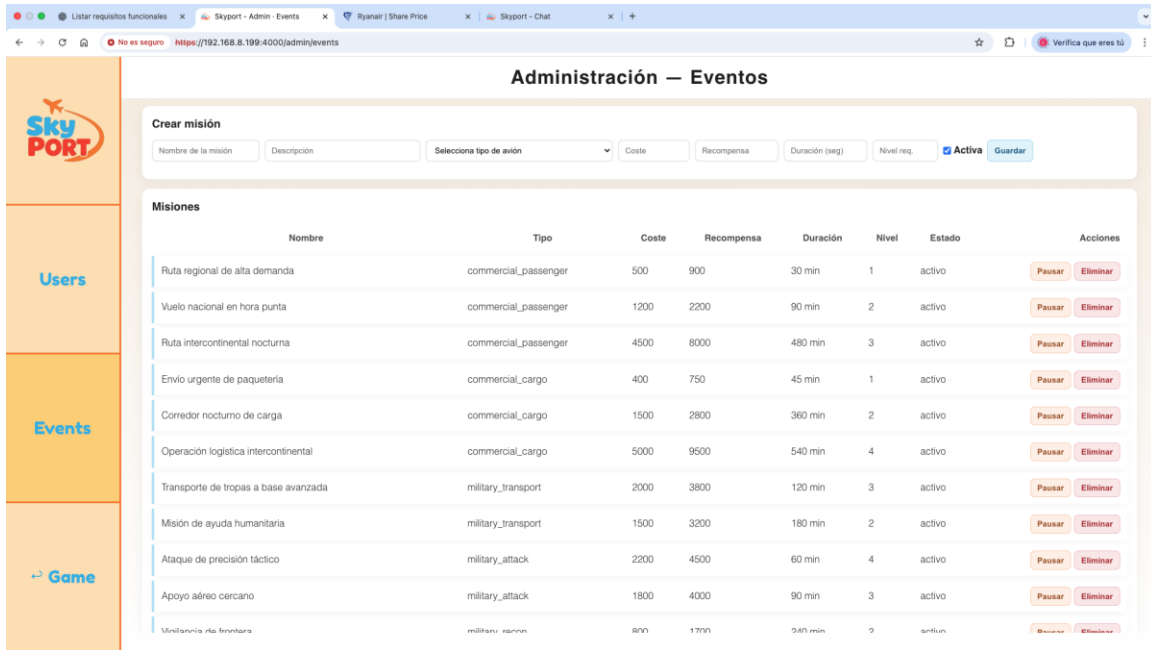
- Listado de usuarios con roles y estado.
- Crear usuario (no admin), activar/desactivar, editar roles (sin poder dar admin) y borrar no-admins.



## Admin – Misiones

- Listado completo de misiones.

- Crear, editar (coste, recompensa, duración, tipo, nivel, activo) y borrar misiones.



## 1.3 Arquitectura

Este apartado describe la arquitectura técnica de SkyPort v2, incluidas sus capas lógicas, componentes, flujos de datos, integración de servicios y decisiones tecnológicas. El objetivo es proporcionar una visión clara y realista del diseño del sistema desde una perspectiva técnica, alineada con el código actualmente implementado.

SkyPort v2 sigue una arquitectura cliente-servidor basada en Node.js y Express, organizada en cuatro capas principales:

- Capa de Presentación (Frontend)
- Capa de Lógica de Negocio (Backend / Express)
- Capa de Persistencia (SQLite + Knex)
- Capa de Comunicación en Tiempo Real (Socket.IO)

### 1.3.1 Capas y componentes

#### Capa de Presentación (Frontend)

El frontend está compuesto por vistas HTML servidas por Express desde el directorio /views y por recursos estáticos ubicados en /public (CSS, imágenes y JavaScript). La lógica en el navegador se encarga de:

- Obtener datos mediante llamadas fetch a las APIs REST del backend.
- Inyectar dinámicamente la información en el DOM (tablas, tarjetas de flota, misiones, economía, etc.).
- Gestionar animaciones visuales (hangares, aviones, timers).
- Establecer la comunicación en tiempo real mediante Socket.IO para chat y presencia.

Componentes clave del frontend:



- Vistas HTML servidas por Express para las secciones principales (login, registro, juego/dashboard, flota, misiones, economía, admin, etc.).
- Hojas de estilo CSS específicas por sección (flota, misiones, economía, autenticación, administración), ubicadas en /public/css.
- Scripts JavaScript modulares en /public/js/\*, responsables de:
- Invocar a las APIs REST (/api/auth, /api/game, /api/explore, /api/admin, /api/stocks).
- Gestionar timers y actualizaciones periódicas de datos (por ejemplo, misiones o economía).
- Controlar animaciones del tablero del aeropuerto y del estado de los aviones.
- Inicializar y manejar la conexión Socket.IO para chat global y radar de usuarios conectados.

### Capa de Lógica de Negocio (Backend / Express)

El backend utiliza Express como framework principal y está organizado en middlewares, rutas HTML y rutas API. Esta capa se encarga de:

- Gestionar la lógica de negocio (flota, misiones, economía, social, administración).
- Validar y procesar las peticiones entrantes.
- Controlar la autenticación, las sesiones y la autorización por roles.
- Coordinar el acceso a la base de datos a través de las capas de persistencia.

Componentes clave del backend:

- App.js: Actúa como punto de entrada del servidor.
  - o Configura middlewares globales:
    - helmet con políticas de Content Security Policy (CSP).
    - compression para comprimir respuestas.
    - morgan en desarrollo para registro de peticiones.
    - Parsers JSON y URL-encoded.
    - Servir estáticos desde /public.
  - o Inicializa el sistema de sesiones y lo comparte con Socket.IO.
  - o Monta las rutas HTML y las APIs REST.
- Gestión de sesiones
  - o Middleware central de sesión definido en middleware/session.
  - o Configurable con expiración por inactividad (idle) y expiración absoluta.
  - o Reutilizado tanto en peticiones HTTP como en conexiones Socket.IO, asegurando que el usuario es el mismo en ambas capas.
- Rutas HTML
  - o Definidas principalmente en routes/index.js.
  - o Incluyen rutas públicas (por ejemplo, páginas de login, register, explore) y rutas protegidas.
  - o Uso de middlewares como requireAuthPage y requireRolePage para controlar el acceso según el rol del usuario (jugador, admin).
    - APIs REST (rutas en /routes/api/\*)
    - /api/auth: login, logout, registro y recuperación de cuenta.
    - /api/explore: búsqueda y ficha pública de aeropuertos visibles.
    - /api/game: núcleo del juego (economía, flota, mercado, misiones, social básico).

- /api/admin: gestión de usuarios y catálogo de misiones desde el panel de administración.
- /api/stocks: consulta de cotizaciones (por ejemplo, Ryanair) mediante integración externa.

### Capa de Persistencia (SQLite + Knex)

La persistencia de datos se implementa con SQLite, gestionado mediante Knex.js como *query builder*. La base de datos se almacena en el fichero local db/skyport.sqlite3.

Configuración de persistencia:

- Knex (db/knex.js)
  - Centraliza la configuración de acceso a la base de datos.
  - Gestiona migraciones para crear y actualizar las tablas de forma ordenada.
  - Gestiona seeds (datos iniciales) para poblar la base con información base.
- Tablas principales (ejemplos relevantes):
  - users: credenciales y datos básicos de los usuarios.
  - user\_airports: información del aeropuerto asociado a cada usuario.
  - aircraft\_types: catálogo de tipos de aeronaves disponibles.
  - user\_aircraft: aviones propiedad de cada usuario.
  - missions: catálogo de misiones base.
  - user\_missions: misiones activas e históricas por usuario y avión.
  - account\_movements: movimientos económicos (ingresos y gastos).
  - friend\_requests: solicitudes de amistad entre jugadores.
  - friendships: relaciones de amistad consolidadas.
  - events: eventos del juego (por ejemplo, boosts o misiones temáticas, según el alcance implementado).
- Tiendas de datos (Data Stores) Implementadas en: /data/\*Store/db.js (por ejemplo, usersStore, fleetStore, missionsStore, economyStore, socialStore, eventsStore).
  - Encapsulan las consultas SQL y exponen métodos de alto nivel (como getUserFleet, createMission, getAccountMovements, etc.).
  - Favorecen una arquitectura modular y facilitan el mantenimiento y la evolución del modelo de datos.
- Seeds iniciales
  - Crean un usuario de demostración.
  - Rellenan el catálogo de tipos de avión.
  - Insertan misiones base.
  - Inician saldos económicos mínimos para comenzar a jugar.

### Capa de Comunicación en Tiempo Real (Socket.IO)

La capa de tiempo real se basa en Socket.IO, acoplado al mismo servidor HTTP/HTTPS donde se ejecuta Express. Esta capa permite funcionalidades como:

- Chat global en tiempo real.
- Presencia de usuarios conectados (radar).
- Algunas notificaciones sociales básicas (por ejemplo, solicitudes de amistad).

Componentes principales:

- Servidor Socket.IO

- Se monta sobre el servidor HTTP/HTTPS creado en app.js.
- Reutiliza el middleware de sesión mediante un wrapper (wrap(sessionMiddleware)), de forma que cada conexión WebSocket está asociada a la misma sesión que las peticiones HTTP.
- Módulo de tiempo real (utils/realtime.js)
  - o Mantiene un mapa de usuarios online y las salas asociadas a cada usuario.
  - o Expone funciones auxiliares para emitir eventos a todos los usuarios o a usuarios individuales.
  - o Eventos actualmente implementados:
    - chat:message: envío y recepción de mensajes en el chat global.
    - Eventos de presencia (chat:ready, join/leave): actualizan el radar de usuarios conectados.
    - Eventos sociales (social:request, social:accepted): notificaciones relacionadas con solicitudes de amistad y su aceptación.

### 1.3.2 Infraestructura y despliegue local

En el entorno actual, SkyPort v2 se ejecuta en un único servidor Node.js en local, con la siguiente configuración:

- **Modos de arranque:**
  - o SKYPORT\_MODE=dev: ejecución en HTTP para desarrollo local (<http://localhost:PORT>).
  - o SKYPORT\_MODE=lan-https: ejecución en HTTPS con certificados autofirmados ubicados en /certs, permitiendo pruebas seguras en red local (LAN).
- **Configuración mediante .env:**
  - o Puerto del servidor (PORT).
  - o Secretos de sesión (SESSION\_SECRET).
  - o Parámetros de expiración de sesión (idle y absoluta).
  - o URL de Redis si se utiliza como almacén de sesión.
  - o Claves de APIs externas, como TWELVE\_KEY para la integración de cotizaciones.
- **Componentes físicos:**
  - o Servidor único Node.js (versión 18+).
  - o Base de datos SQLite en disco local (db/skyport.sqlite3).
  - o Redis opcional para almacenamiento de sesiones (con *fallback* a MemoryStore si no se configura).
  - o Navegador del usuario (en el mismo equipo o en la LAN) conectándose al servidor.

### 1.3.3 Flujos de datos principales

A continuación se resumen los flujos de datos más relevantes en el sistema, de extremo a extremo:

- **Autenticación (login / register):**
  - o POST /api/auth/login valida las credenciales del usuario comparando el *hash* bcrypt almacenado.

- En caso de éxito, se regenera la sesión y se guardan en ella el userId y los roles asociados.
- El registro de usuario crea la cuenta, el aeropuerto inicial, un avión de arranque y un primer movimiento contable.
- **Recuperación de cuenta:**
  - Existe flujo de solicitud y cambio de contraseña con pantalla específica y endpoints correspondientes, según el alcance actual.
- **Juego (flota, mercado, misiones):**
  - /api/game/fleet devuelve la flota del usuario junto con información del catálogo de aviones.
  - Endpoints de mercado permiten comprar y vender aviones con el sistema, actualizando tanto la flota como los movimientos económicos.
  - Endpoints de misiones permiten activar, abortar y resolver misiones, gestionando estados y tiempos en la tabla user\_missions y registrando los ingresos/gastos asociados.
- **Economía:**
  - /api/game/economy devuelve el saldo actual del usuario y la lista de movimientos económicos ordenados.
  - Las operaciones de compra, venta y misiones actualizan esta información de manera consistente.
- **Social (amistades):**
  - Búsqueda de usuarios por nombre.
  - Envío, aceptación y rechazo de solicitudes de amistad a través de endpoints REST y eventos en tiempo real.
  - Listado de amigos con indicación de si se encuentran online o no (integrado con la capa Socket.IO).
- **Chat:**
  - La conexión Socket.IO se establece desde el cliente al cargar la pantalla de juego o chat.
  - Los mensajes enviados se emiten mediante chat:message y se difunden en tiempo real a todos los clientes conectados.
- **Administración (panel admin):**
  - Endpoints de /api/admin permiten el CRUD básico de usuarios (por ejemplo, suspender o restaurar), sin capacidad para otorgar privilegios de administrador desde la interfaz.
  - También se incluye gestión del catálogo de misiones (crear, modificar, desactivar), según el alcance implementado.
- **Cotizaciones bursátiles (stocks):**
  - El endpoint /api/stocks integra con TwelveData para obtener el precio de acciones como Ryanair.
  - El sistema aplica una caché de corta duración para evitar llamadas excesivas y utiliza el valor cacheado si el proveedor externo falla.

#### 1.3.4 Seguridad y hardening

SkyPort v2 incorpora varias medidas de seguridad a nivel de servidor:

- **Cabeceras y CSP:**
  - Uso de helmet para establecer una Content Security Policy (CSP) restrictiva, limitando los orígenes permitidos para scripts, estilos, fuentes y conexiones.

- Desactivación del encabezado X-Powered-By para no exponer detalles del framework.
- **Gestión de sesiones:**
  - Cookies de sesión firmadas, con las banderas adecuadas (httpOnly, etc.).
  - Expiración configurable por inactividad y tope máximo de vida (expiración absoluta).
  - Posibilidad de rotación de sesiones y almacenamiento en Redis para entornos más exigentes.
- **Protección de rutas:**
  - Middlewares requireAuth y requireRole (y versiones específicas para vistas) protegen rutas sensibles.
  - Solo los usuarios autenticados pueden acceder a las funcionalidades de juego, y solo los administradores a la sección de administración.

### 1.3.5 Limitaciones actuales

Pese a contar con una arquitectura sólida, existen algunas limitaciones relevantes en el estado actual del proyecto:

- **Notificaciones de misiones completadas**
  - La actualización del estado de las misiones y la visualización de misiones completadas requieren refresco manual o nuevas peticiones al backend.
  - No existe todavía un sistema de *push* en tiempo real que notifique automáticamente al usuario la finalización de una misión.
- **Gestión avanzada de infraestructuras aeroportuarias:**
  - No se han implementado aún mejoras de pistas ni ampliaciones de hangar completas (más allá de la estructura básica planificada).
  - Tampoco existe todavía un marketplace completo entre jugadores (las operaciones se realizan principalmente contra el sistema).
- **Chat:**
  - El chat global funciona en tiempo real, pero es no persistente: los mensajes no se almacenan de forma histórica en la base de datos.
  - Esto simplifica la arquitectura, pero limita la consulta de conversaciones anteriores.
  - Estas limitaciones se corresponden con decisiones conscientes de alcance para esta versión, priorizando un MVP funcional, estable y evaluable, sobre funcionalidades avanzadas que implicarían una mayor complejidad técnica y de mantenimiento.