Contenido

MANUAL TÉCNICO

COMPONENTES

ParkinGO está desarrollado como una aplicación móvil híbrida que opera tanto en dispositivos Android como iOS, utilizando tecnologías web modernas para ofrecer una experiencia nativa. El sistema se compone de los siguientes elementos:

Aplicación cliente (Ionic + Angular)

- Framework principal: Ionic 8.0 con Angular 20.0
- Interfaz de usuario responsiva y componentes nativos
- Soporte multiplataforma mediante Capacitor 7.4.4
- Sistema de navegación basado en rutas de Angular Router

Backend serverless (Firebase)

- Base de datos: Cloud Firestore para almacenamiento en tiempo real
- Autenticación: Firebase Authentication con correo y contraseña
- Hosting: Firebase Hosting para despliegue web
- Sincronización automática de datos entre clientes conectados

Módulos nativos

- Escaneo de códigos QR: Capacitor ML Kit Barcode Scanning v7.4.0
- Generación de códigos QR: Librería qrcode v1.5.4
- Escáner web alternativo: ZXing Browser v0.1.5 para entornos sin capacidades nativas

Herramientas de desarrollo

- TypeScript 5.8 para tipado estático
- Angular CLI 20.0 para gestión del proyecto
- ESLint con configuración Angular para calidad de código

ALCANCE

Este manual técnico documenta la arquitectura, procesos internos y estructura de datos del sistema ParkinGO. Está dirigido a desarrolladores y personal técnico encargado del mantenimiento, actualización o extensión de la plataforma.

Funcionalidades cubiertas:

- Registro e inicio de sesión de usuarios y administradores
- Creación y gestión de reservas de espacios
- Visualización interactiva del plano del parqueadero

- Generación y validación de códigos QR de acceso
- Registro de ingresos mediante escaneo QR o entrada manual
- Procesamiento de salidas con cálculo automático de tarifas
- Cancelación de reservas con liberación de espacios
- Gestión administrativa de espacios y clientes
- Sistema de reportes operativos

Límites del sistema:

- No incluye procesamiento de pagos en línea
- No gestiona múltiples sedes simultáneamente
- Requiere conexión a internet constante para operar

DESCRIPCIÓN DE LOS PROCESOS

Proceso de registro de usuario

1. El usuario accede a la pantalla de registro (`/pages-users/auth/register`)
2. Completa el formulario con correo electrónico, contraseña, nombre y teléfono
3. El sistema valida que el correo no esté registrado previamente
4. Firebase Authentication crea la cuenta con las credenciales proporcionadas
5. Se almacena la información adicional del usuario en Firestore (colección `users`)
6. El sistema genera automáticamente un rol de usuario estándar
7. Se redirige al usuario a la pantalla de inicio de sesión

Validaciones aplicadas:

- Correo electrónico en formato válido
- Contraseña mínima de 6 caracteres
- Campos obligatorios completos
- Unicidad del correo electrónico

Proceso de autenticación

1. El usuario ingresa credenciales en la pantalla de login
2. Firebase Authentication valida las credenciales contra su base de datos
3. Si la autenticación es exitosa, se obtiene el token de sesión

4. El sistema recupera la información del usuario desde Firestore

5. Se almacena el estado de sesión localmente

6. El usuario es redirigido según su rol:

   - Usuarios estándar → `/pages-users/home`

   - Administradores → `/admin-pages/admin`

Manejo de errores:

- Credenciales incorrectas: Mensaje "Correo o contraseña incorrectos"

- Usuario no existe: Mensaje "Esta cuenta no existe"

- Error de red: Mensaje de conexión fallida

Proceso de creación de reserva

Ubicación: `src/app/pages/pages-users/reservations/reservations.page.ts`

1. El usuario selecciona el tipo de vehículo (Automóvil, Motocicleta o Bicicleta)

2. El sistema consulta espacios disponibles en Firestore:

```typescript
query(
  collection(firestore, 'spaces'),
  where('vehicleType', '==', type),
  where('status', '==', 'Available')
)
```

3. Se valida que la placa no tenga reservas activas:

```typescript
query(
  collection(firestore, 'reservations'),
  where('plate', '==', plate),
  where('status', 'in', ['pending', 'active'])
)
```

4. El usuario completa los datos del vehículo y selecciona un espacio

5. Se crea el documento de reserva con los siguientes campos:

- `vehicleType`: Tipo de vehículo
- `plate`: Placa del vehículo (opcional para bicicletas)
- `model`: Modelo del vehículo
- `space`: Código del espacio asignado
- `spaceId`: ID del documento del espacio
- `userId`: ID del usuario que reserva
- `email`: Correo del usuario
- `startDate`: Fecha y hora de creación
- `endDate`: Fecha de vencimiento (24 horas después)
- `pricePerHour`: Tarifa por hora del espacio
- `status`: Estado inicial `'pending'`

6. Se genera un código QR con los datos de la reserva:

```typescript
const qrData = JSON.stringify({ id: docRef.id, plate: reservationData.plate });
const qrCode = await QRCode.toDataURL(qrData);
```

7. Se actualiza el estado del espacio a `'Occupied'`
8. Se muestra confirmación al usuario

Reglas de negocio:

- Máximo una reserva activa por placa
- Las bicicletas no requieren placa
- Las reservas vencen automáticamente después de 24 horas
- Los espacios quedan bloqueados inmediatamente

Proceso de ingreso de vehículo

Ubicación: `src/app/pages/admin-pages/ingreso/ingreso.page.ts`

El sistema ofrece dos métodos de ingreso:

A. Ingreso mediante escaneo QR

1. El administrador accede a la pantalla de ingreso
2. Navega al escáner QR (`/admin-pages/scanner/scan-qr`)
3. El escáner activa la cámara del dispositivo:

```typescript
await BarcodeScanner.checkPermissions();
await BarcodeScanner.startScan();
```

4. Al detectar un código QR, se extrae el ID de la reserva
5. Se busca la reserva en Firestore y se valida su estado (`pending`)
6. Se registra el horario de ingreso y se cambia el estado a `active`:

```typescript
await updateDoc(ref, {
  status: 'active',
  entryTime: new Date()
});
```

7. El espacio permanece como `'Occupied'`
8. Se muestra confirmación del ingreso

B. Ingreso manual

1. El administrador busca la reserva por placa en la lista
2. Selecciona la reserva correspondiente
3. Confirma los datos del vehículo
4. El sistema ejecuta el mismo proceso de activación que el escaneo QR

C. Ingreso sin reserva previa

1. El administrador presiona "Ingresar vehículo manualmente"
2. Completa el formulario con datos del vehículo
3. Selecciona un espacio disponible
4. Se crea directamente una reserva con estado `active`:

```typescript
const reservationData = {
  vehicleType: manual.type,
  plate: manual.plate,
  model: manual.model,
```

```
  space: manual.space,
  startDate: new Date(),
  pricePerHour: selectedSpace.pricePerHour,
  status: 'active',
  entryTime: new Date()
};
```

5. Se marca el espacio como ocupado

Proceso de salida de vehículo

Ubicación: `src/app/pages/admin-pages/scanner/scan-qr-salida/scan-qr-salida.page.ts`

1. El administrador accede al escáner de salida
2. Escanea el código QR del vehículo que está saliendo
3. El sistema valida que la reserva esté en estado `active`
4. Se calcula el tiempo de permanencia:

```typescript
const entryTime = vehicle.entryTime.toDate();
const exitTime = new Date();
const diffMs = exitTime - entryTime;
const hours = Math.ceil(diffMs / (1000 * 60 * 60));
```

5. Se calcula el costo total:

```typescript
const total = vehicle.pricePerHour * hours;
```

6. Se registra la salida en la colección `salidas`:

```typescript
await addDoc(collection(firestore, 'salidas'), {
  reservationId,
  plate: vehicle.plate,
  model: vehicle.model,
```

```
  space: vehicle.space,
  vehicleType: vehicle.vehicleType,
  entryTime,
  exitTime,
  hours,
  total,
  userId: vehicle.userId
});
```

7. Se actualiza la reserva como `finalizado`:

```typescript
await updateDoc(ref, {
  status: 'finalizado',
  exitTime,
  hours,
  total
});
```

8. Se libera el espacio:

```typescript
await updateDoc(doc(firestore, 'spaces', vehicle.space), {
  status: 'Available'
});
```

9. Se muestra el total a cobrar al cliente

Casos especiales:

- Si la reserva no está activa, se rechaza la salida
- Si el QR es inválido, se muestra error
- El cálculo siempre redondea hacia arriba (hora completa mínima)

Proceso de cancelación de reserva

Ubicación: `src/app/pages/pages-users/index/index.page.ts`}

1. El usuario accede a "Mis reservas"

2. Selecciona la reserva que desea cancelar

3. Confirma la acción de cancelación

4. El sistema verifica el estado actual (solo se pueden cancelar reservas `pending`)

5. Se cambia el estado de la reserva a `cancelled`:

```typescript
await updateDoc(ref, { status: 'cancelled' });
```

6. Se busca el espacio asociado por código:

```typescript
const spaceQuery = query(
  collection(firestore, 'spaces'),
  where('code', '==', res.space)
);
```

7. Se libera el espacio cambiando su estado a `Available`

8. Se notifica al usuario la cancelación exitosa

Reglas:

- Solo se pueden cancelar reservas en estado `pending`

- Las reservas `active` (vehículo ya ingresado) no pueden cancelarse por esta vía

- Las reservas `finalizado` o `cancelled` permanecen en el historial

DIAGRAMAS UML

DIAGRAMA DE CONTEXTO:

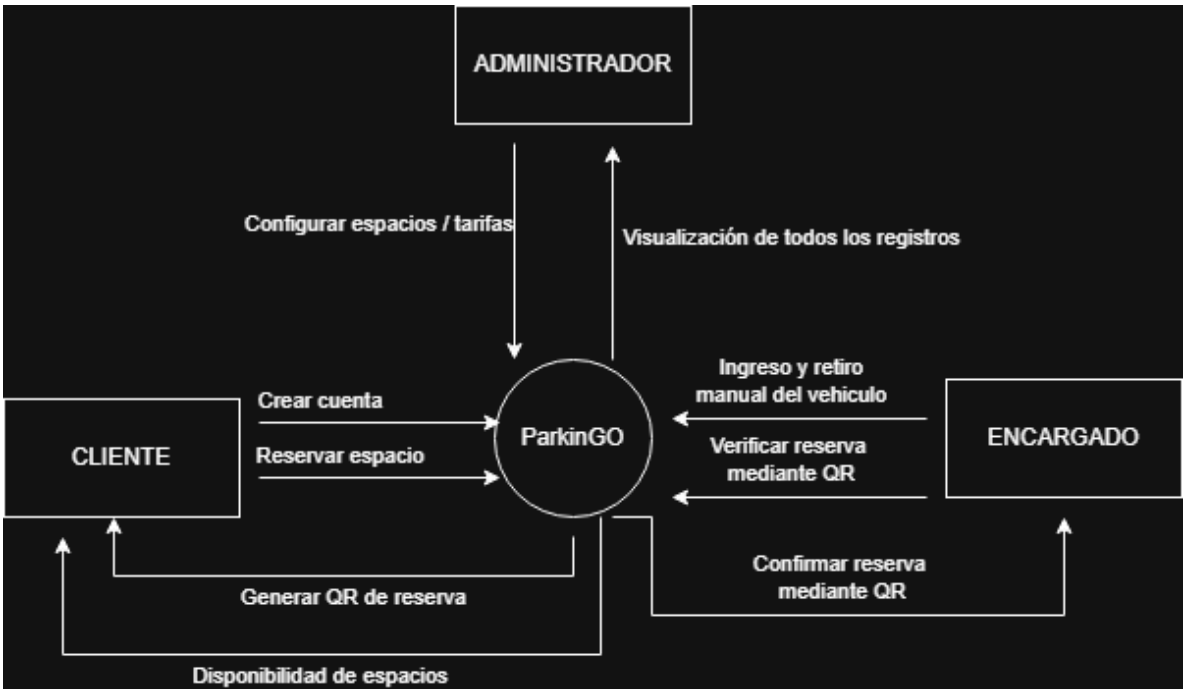
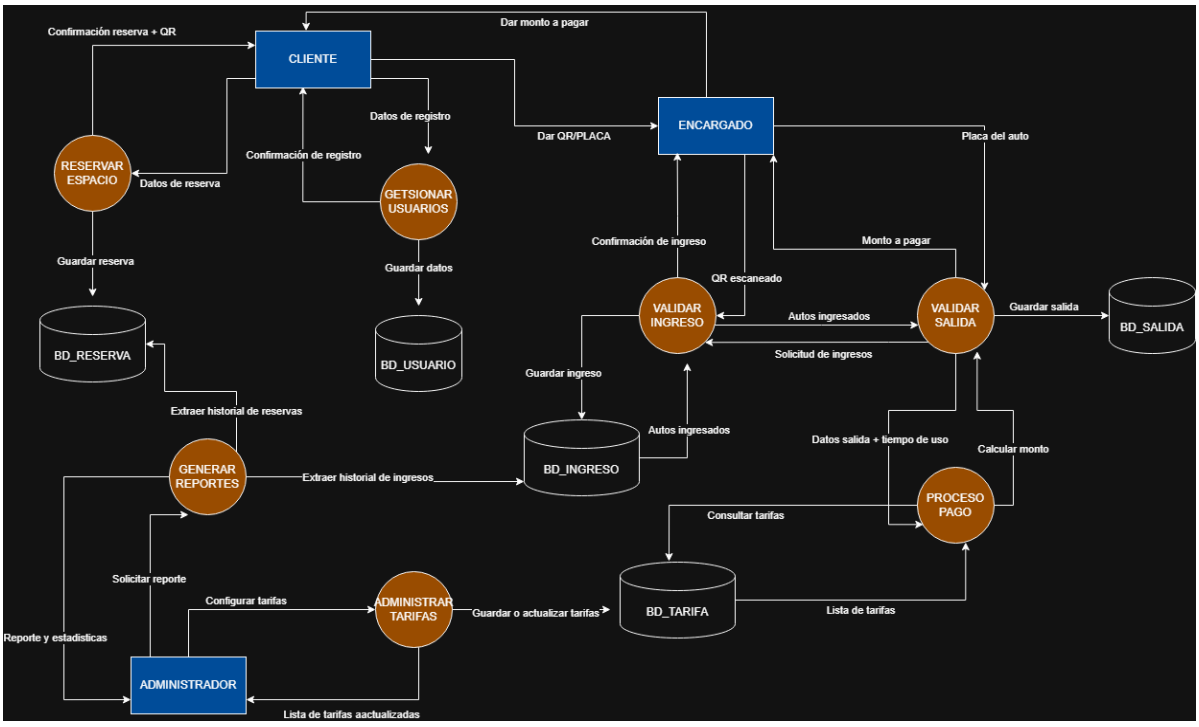
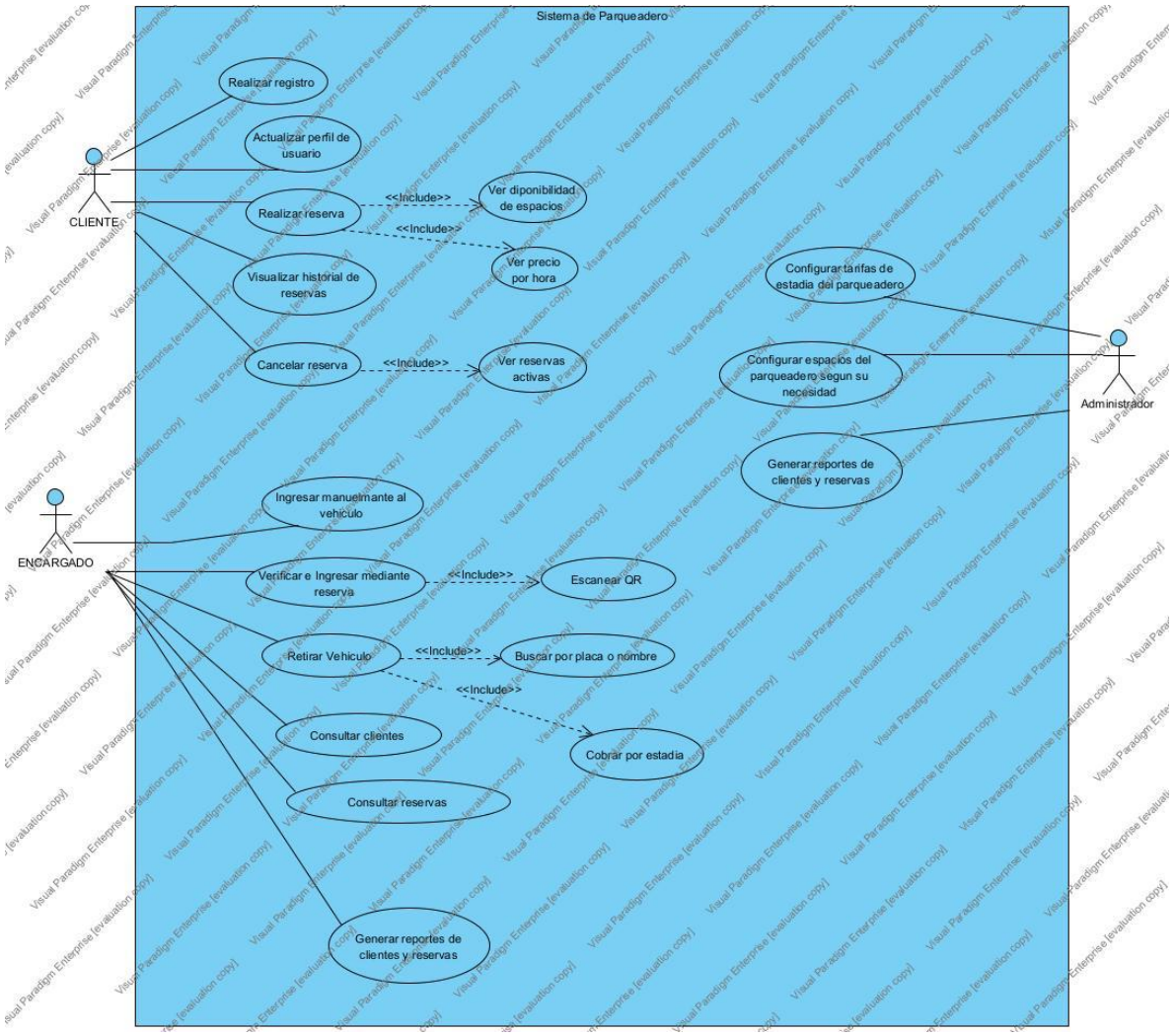
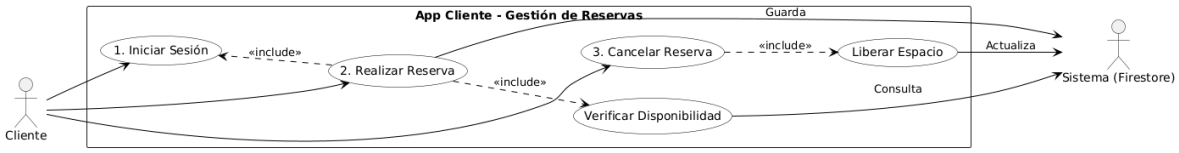


DIAGRAMA DE PROCESO:



CASOS DE USO:





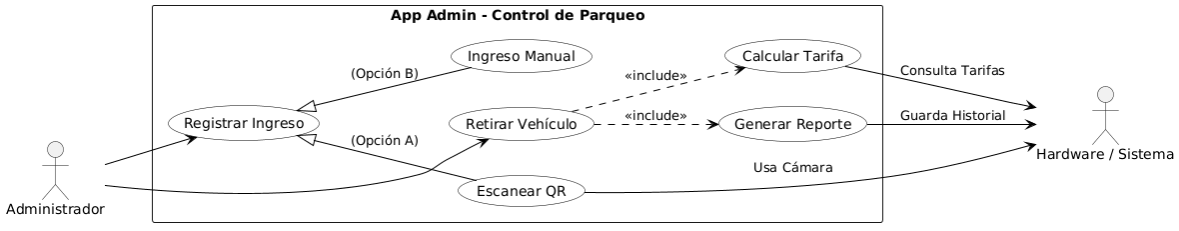


DIAGRAMA DE CLASES:

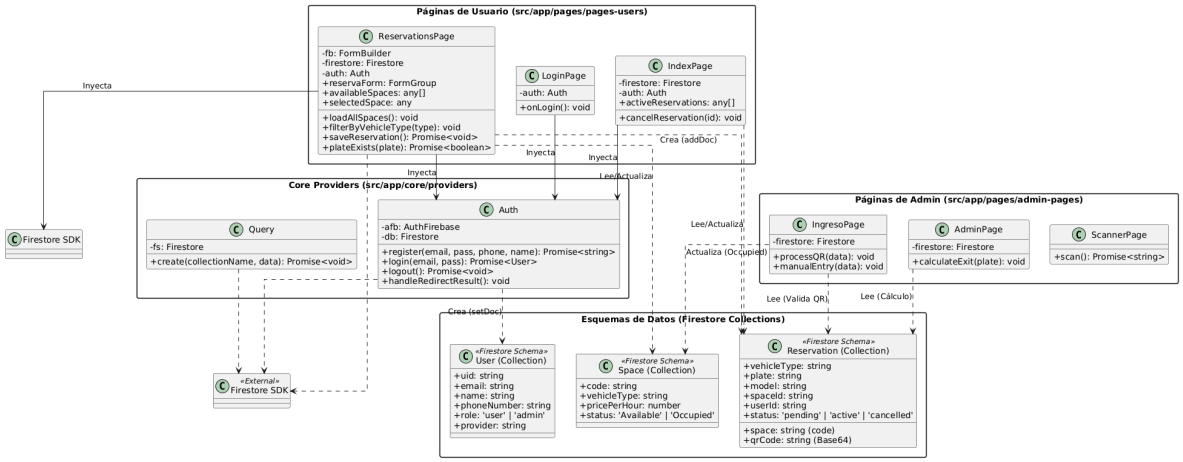
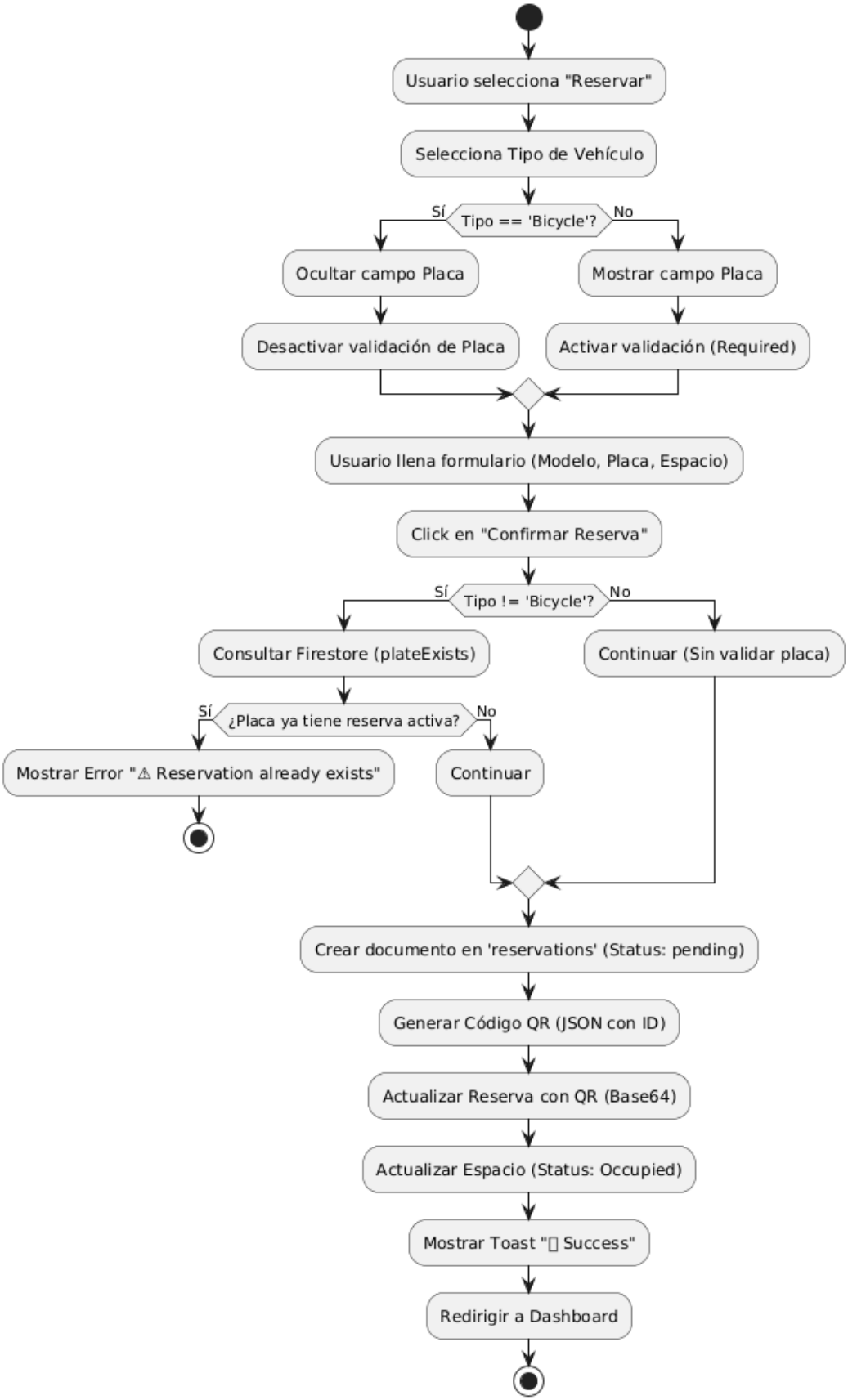
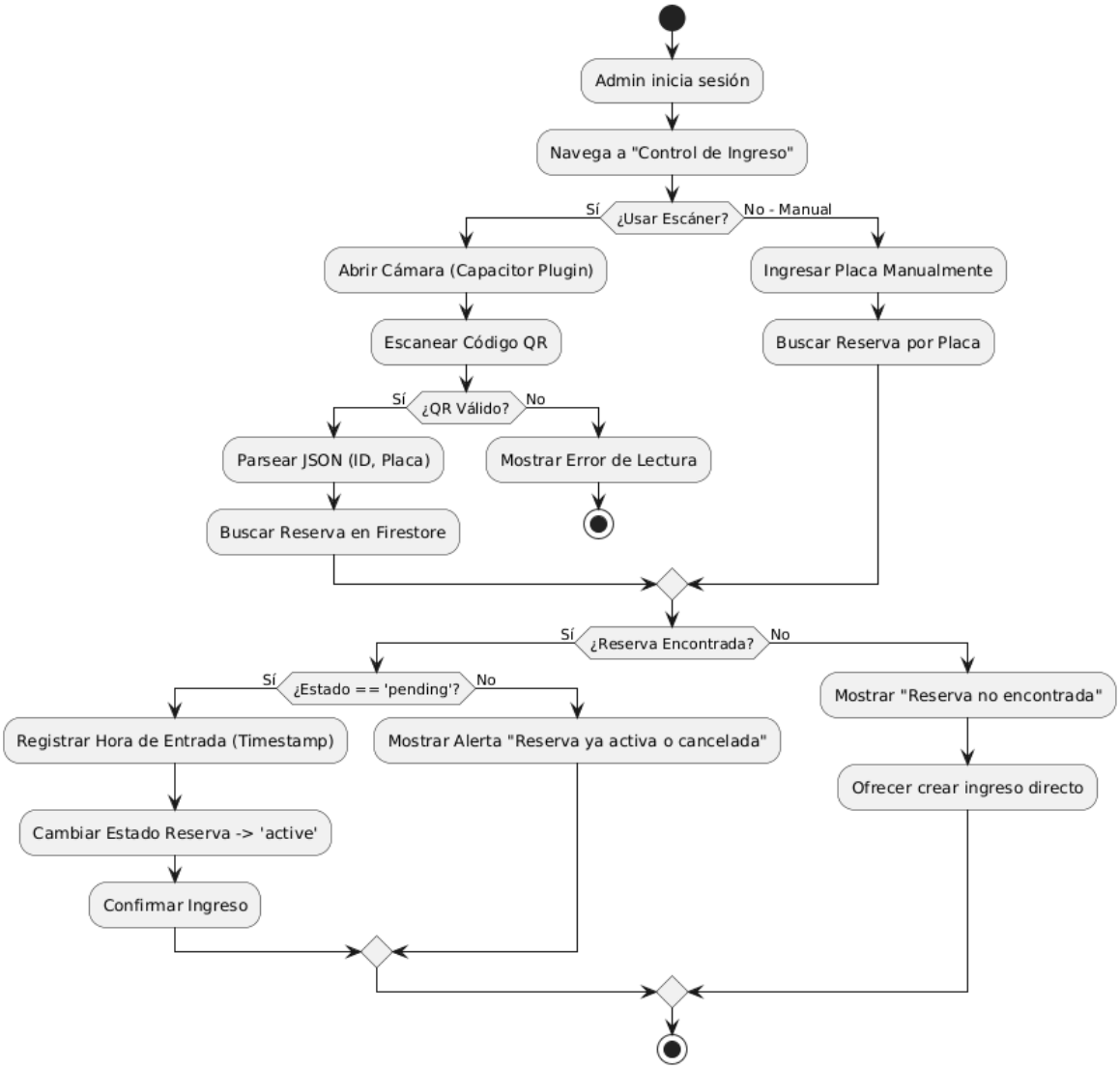
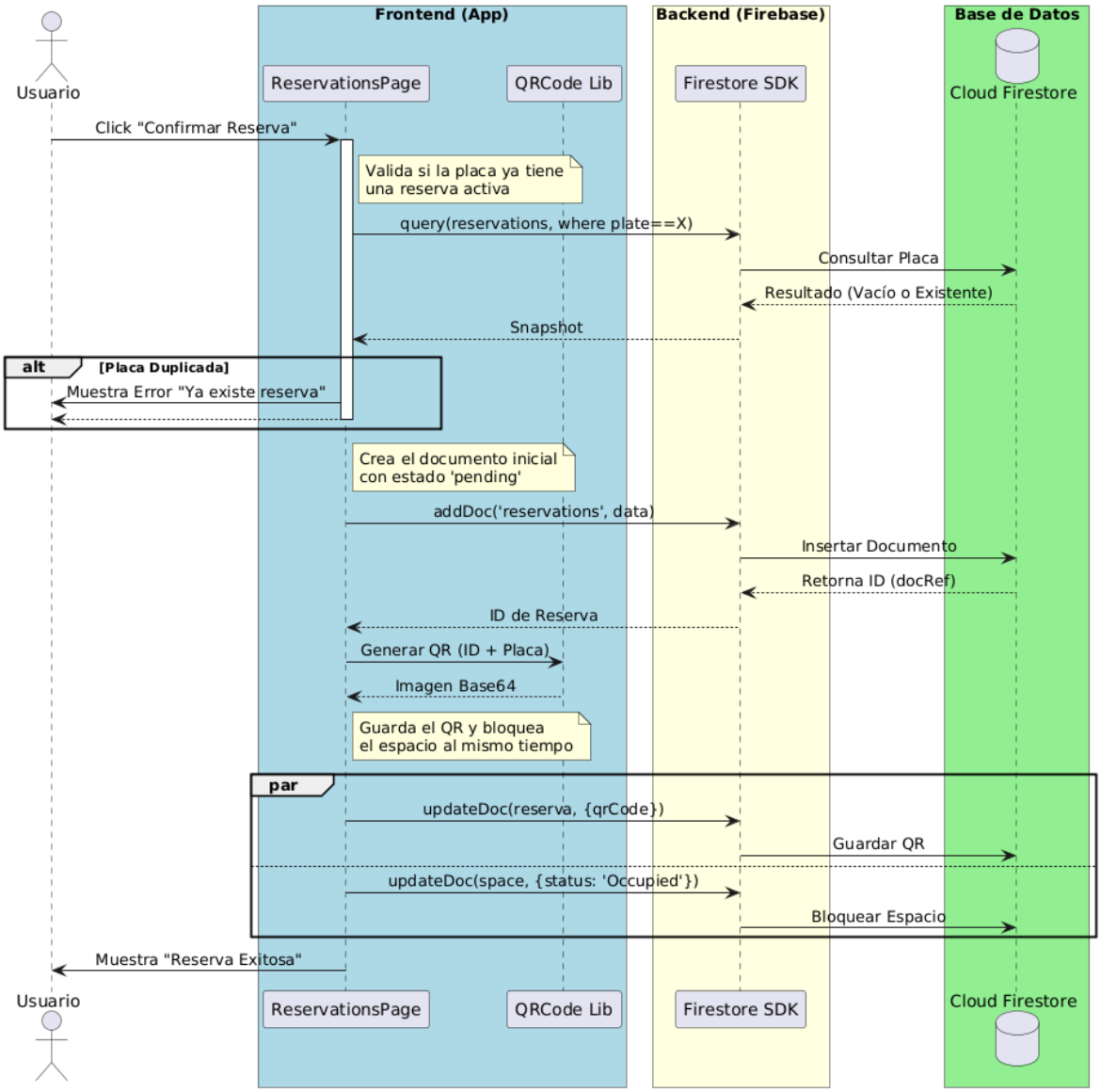


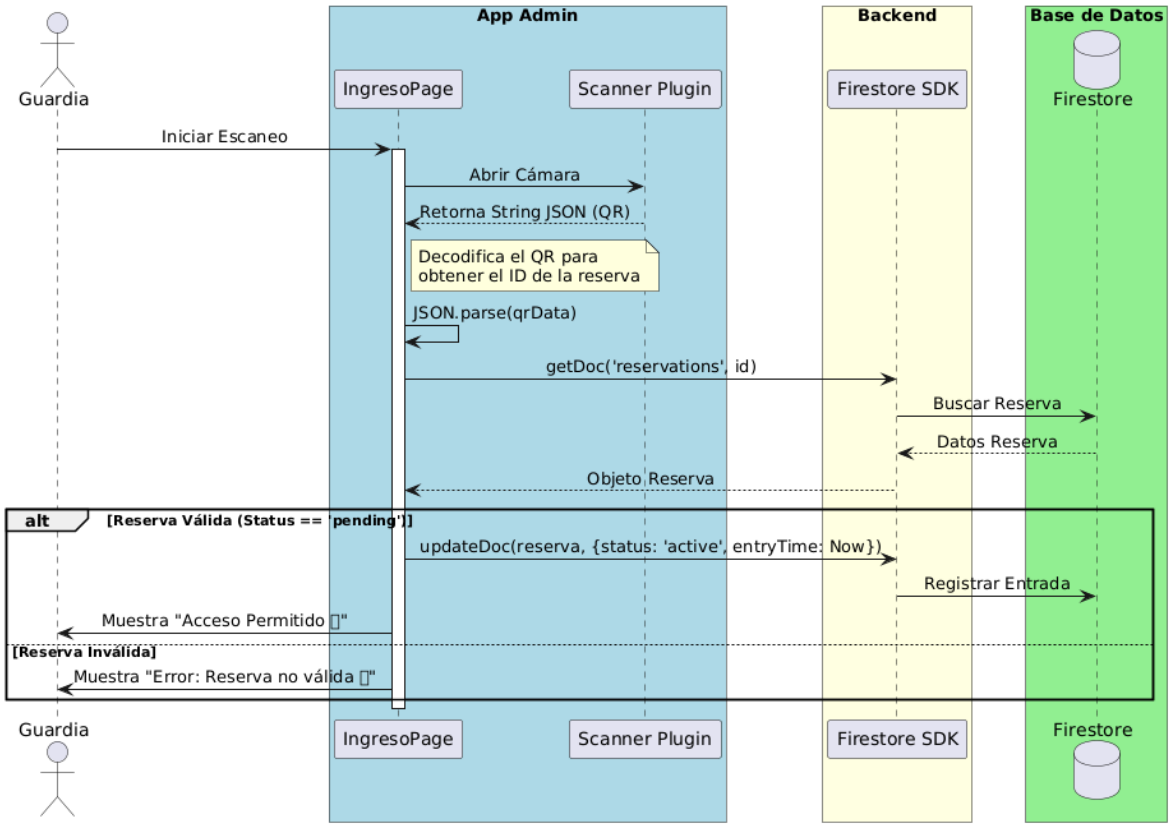
DIAGRAMA DE ACTIVIDADES:

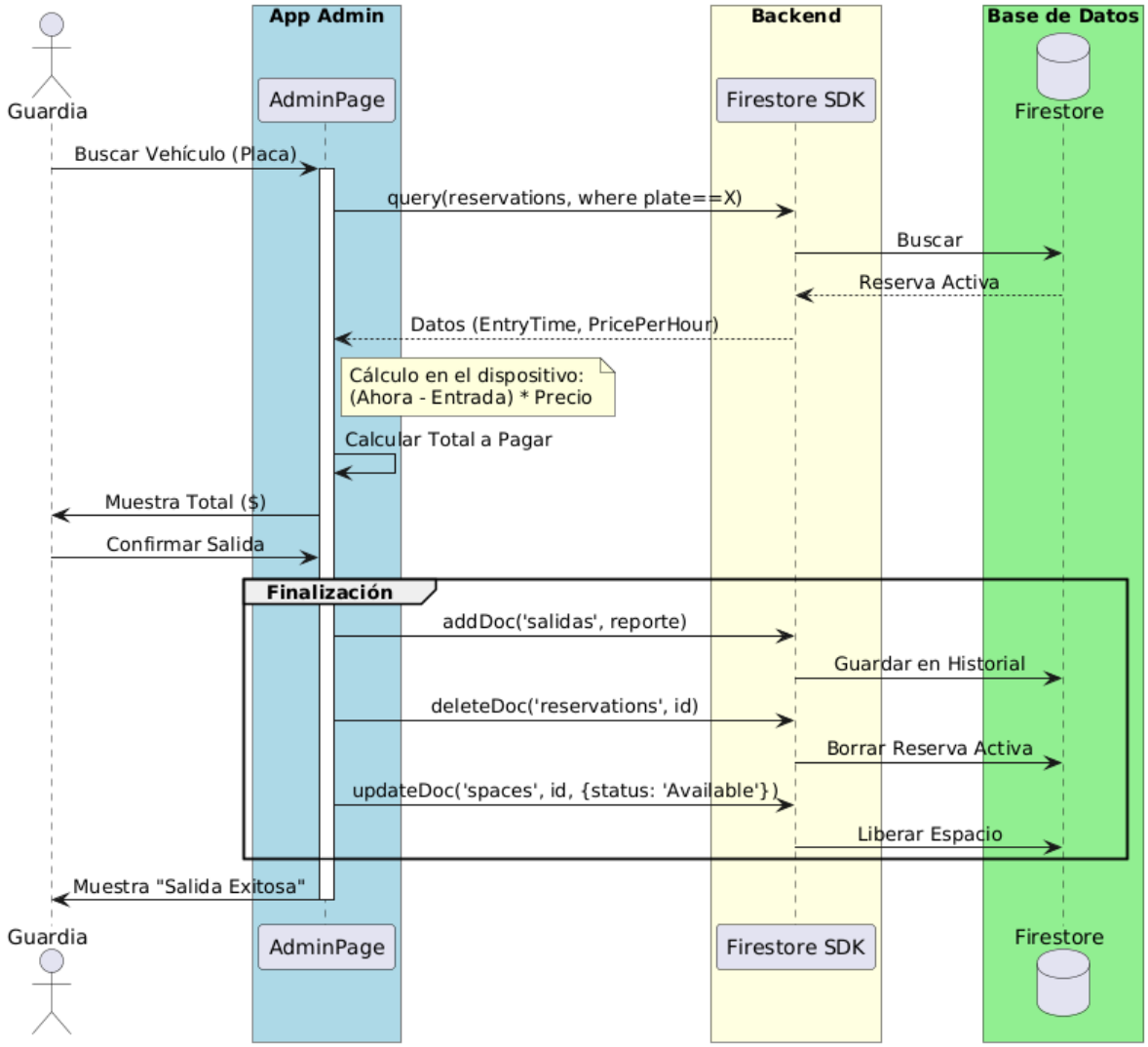

Usuario selecciona "Reservar"
Selecciona Tipo de Vehículo
Tipo == 'Bicycle'?
Sí
No
Ocultar campo Placa
Mostrar campo Placa
Desactivar validación de Placa
Activar validación (Required)
Usuario llena formulario (Modelo, Placa, Espacio)
Click en "Confirmar Reserva"
Tipo != 'Bicycle'?
Sí
No
Consultar Firestore (plateExists)
Continuar (Sin validar placa)
¿Placa ya tiene reserva activa?
Sí
No
Mostrar Error "⚠ Reservation already exists"
Continuar
Crear documento en 'reservations' (Status: pending)
Generar Código QR (JSON con ID)
Actualizar Reserva con QR (Base64)
Actualizar Espacio (Status: Occupied)
Mostrar Toast "□ Success"
Redirigir a Dashboard

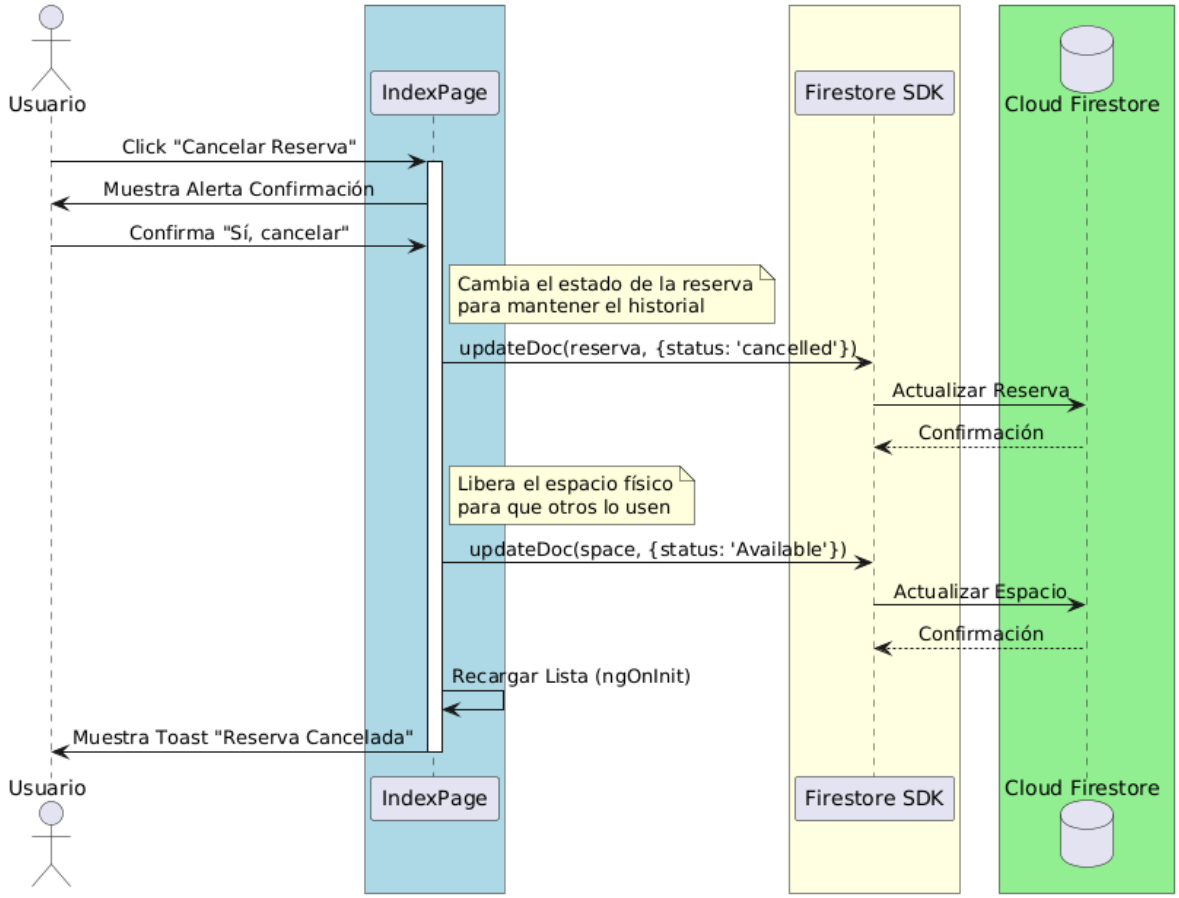

Admin inicia sesión
Navega a "Control de Ingreso"
¿Usar Escáner?
Sí
No - Manual
Abrir Cámara (Capacitor Plugin)
Escanear Código QR
¿QR Válido?
Sí
No
Parsear JSON (ID, Placa)
Buscar Reserva en Firestore
Mostrar Error de Lectura
Ingresar Placa Manualmente
Buscar Reserva por Placa
¿Reserva Encontrada?
Sí
No
¿Estado == 'pending'?
Sí
No
Registrar Hora de Entrada (Timestamp)
Cambiar Estado Reserva -> 'active'
Confirmar Ingreso
Mostrar Alerta "Reserva ya activa o cancelada"
Mostrar "Reserva no encontrada"
Ofrecer crear ingreso directo

DIAGRAMAS DE SECUENCIA:

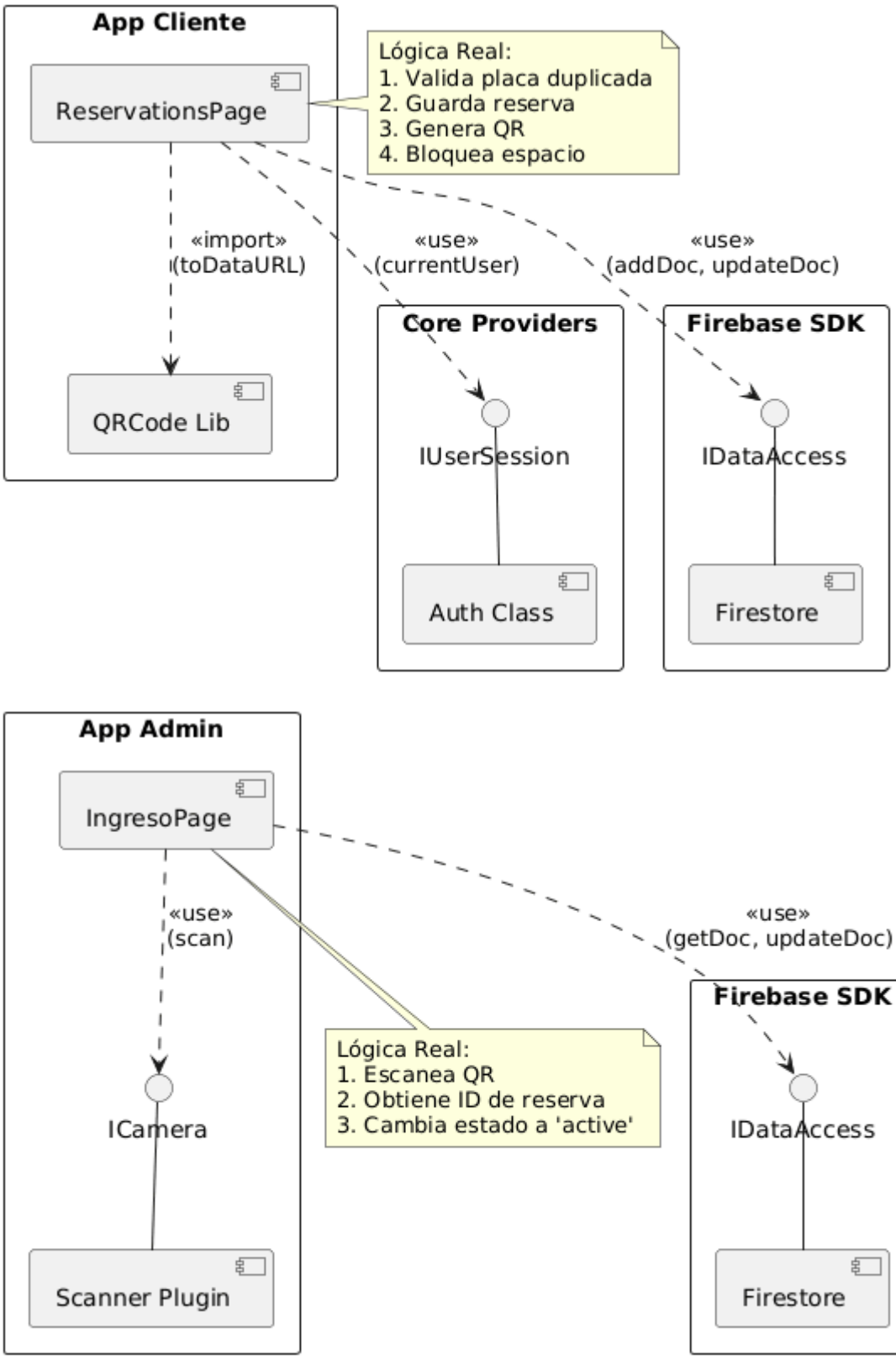

Frontend (App)
Backend (Firebase)
Base de Datos
Usuario
ReservationsPage
QRCode Lib
Firestore SDK
Cloud Firestore
Click "Confirmar Reserva"
Valida si la placa ya tiene una reserva activa
query(reservations, where plate==X)
Consultar Placa
Resultado (Vacío o Existente)
Snapshot
alt
[Placa Duplicada]
Muestra Error "Ya existe reserva"
Crea el documento inicial con estado 'pending'
addDoc('reservations', data)
Insertar Documento
Retorna ID (docRef)
ID de Reserva
Generar QR (ID + Placa)
Imagen Base64
Guarda el QR y bloquea el espacio al mismo tiempo
par
updateDoc(reserva, {qrCode})
Guardar QR
updateDoc(space, {status: 'Occupied'})
Bloquear Espacio
Muestra "Reserva Exitosa"

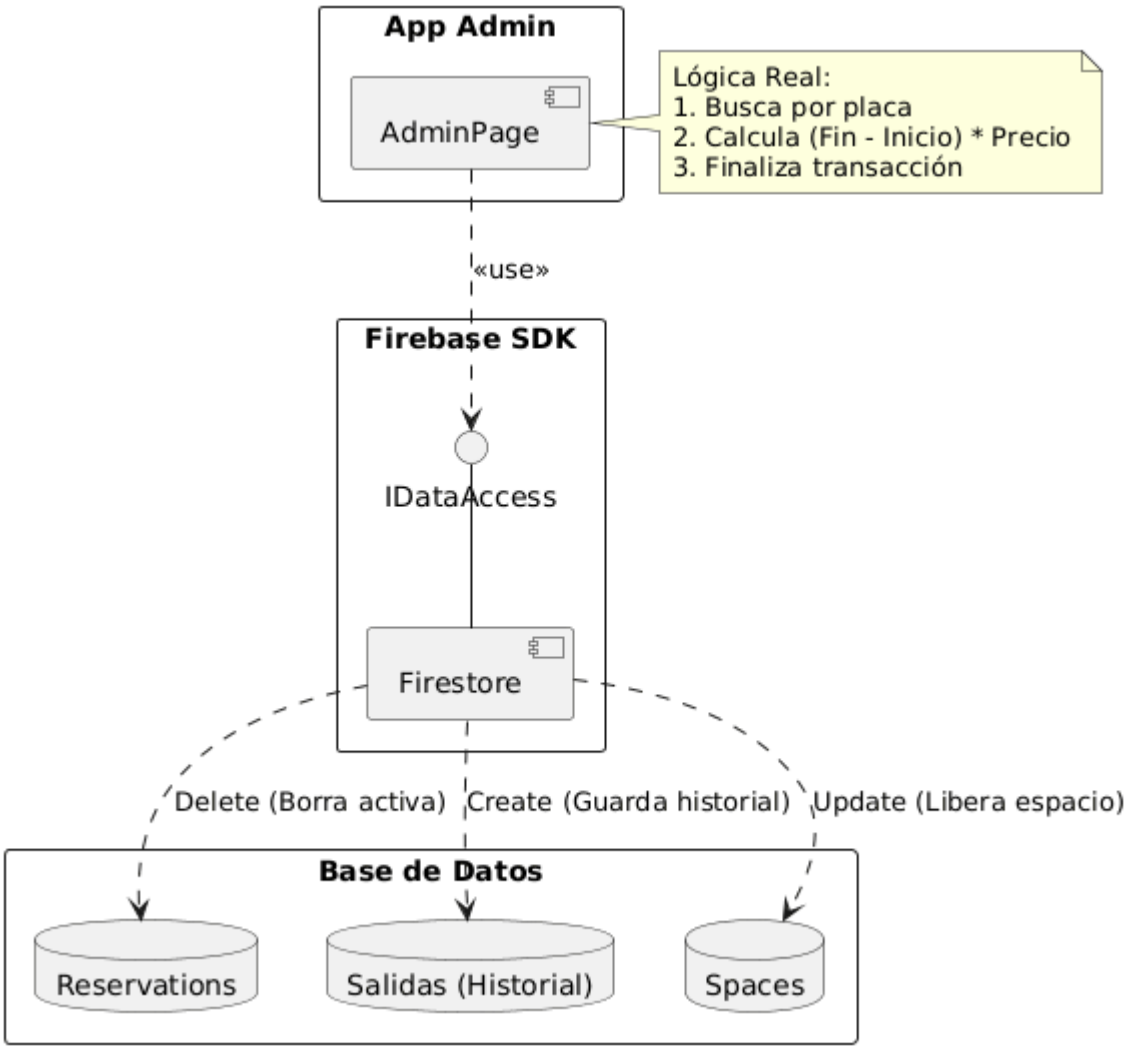

App Admin
Backend
Base de Datos
Guardia
IngresoPage
Scanner Plugin
Firestore SDK
Firestore
Iniciar Escaneo
Abrir Cámara
Retorna String JSON (QR)
Decodifica el QR para obtener el ID de la reserva
JSON.parse(qrData)
getDoc('reservations', id)
Buscar Reserva
Datos Reserva
Objeto Reserva
alt
[Reserva Válida (Status == 'pending')]
updateDoc(reserva, {status: 'active', entryTime: Now})
Registrar Entrada
Muestra "Acceso Permitido "
[Reserva Inválida]
Muestra "Error: Reserva no válida "
Guardia
IngresoPage
Scanner Plugin
Firestore SDK
Firestore


App Admin
Backend
Base de Datos
Guardia
AdminPage
Firestore SDK
Firestore
Buscar Vehículo (Placa)
query(reservations, where plate==X)
Buscar
Reserva Activa
Datos (EntryTime, PricePerHour)
Cálculo en el dispositivo:
(Ahora - Entrada) * Precio
Calcular Total a Pagar
Muestra Total ($)
Confirmar Salida
Finalización
addDoc('salidas', reporte)
Guardar en Historial
deleteDoc('reservations', id)
Borrar Reserva Activa
updateDoc('spaces', id, {status: 'Available'})
Liberar Espacio
Muestra "Salida Exitosa"
Guardia
AdminPage
Firestore SDK
Firestore



DIAGRAMAS DE COMPONENTES:


App Cliente
ReservationsPage
Lógica Real:
1. Valida placa duplicada
2. Guarda reserva
3. Genera QR
4. Bloquea espacio
«import»
(toDataURL)
«use»
(currentUser)
«use»
(addDoc, updateDoc)
QRCode Lib
Core Providers
IUserSession
Auth Class
Firebase SDK
IDataAccess
Firestore
App Admin
IngresoPage
«use»
(scan)
«use»
(getDoc, updateDoc)
Lógica Real:
1. Escanea QR
2. Obtiene ID de reserva
3. Cambia estado a 'active'
ICamera
Scanner Plugin
Firebase SDK
IDataAccess
Firestore


App Admin
AdminPage
Lógica Real:
1. Busca por placa
2. Calcula (Fin - Inicio) * Precio
3. Finaliza transacción
«use»
Firebase SDK
IDataAccess
Firestore
Delete (Borra activa)
Create (Guarda historial)
Update (Libera espacio)
Base de Datos
Reservations
Salidas (Historial)
Spaces

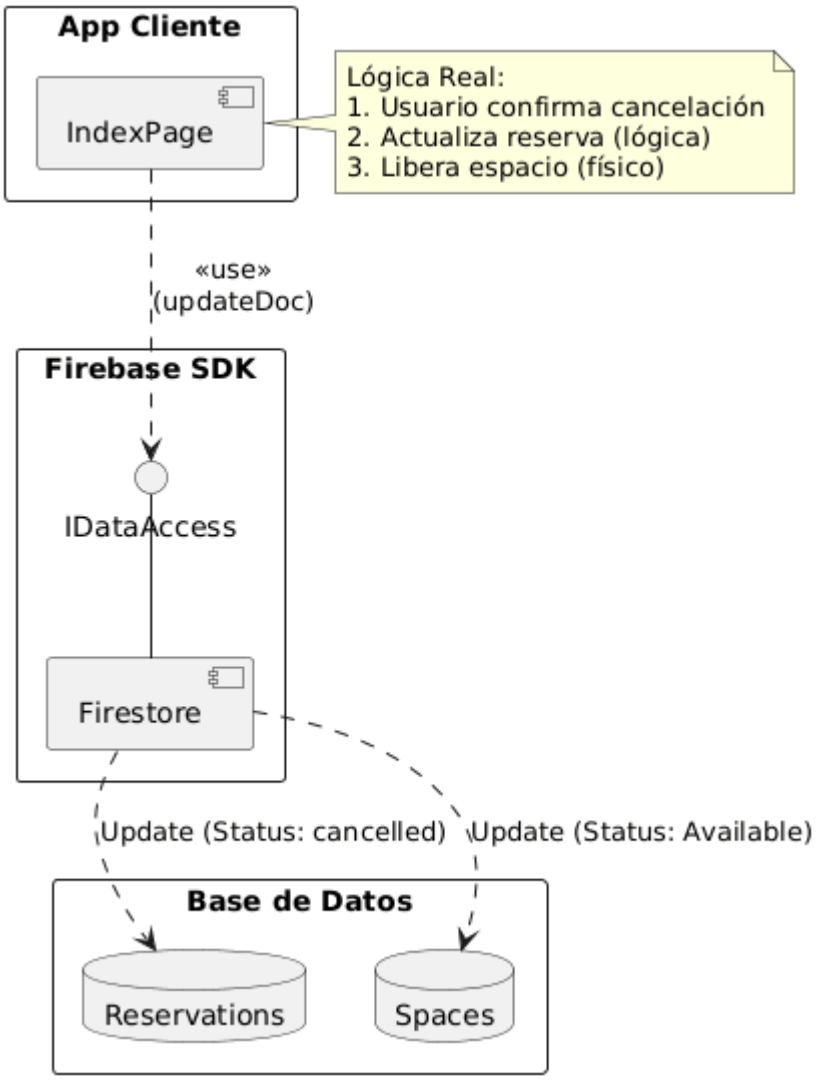
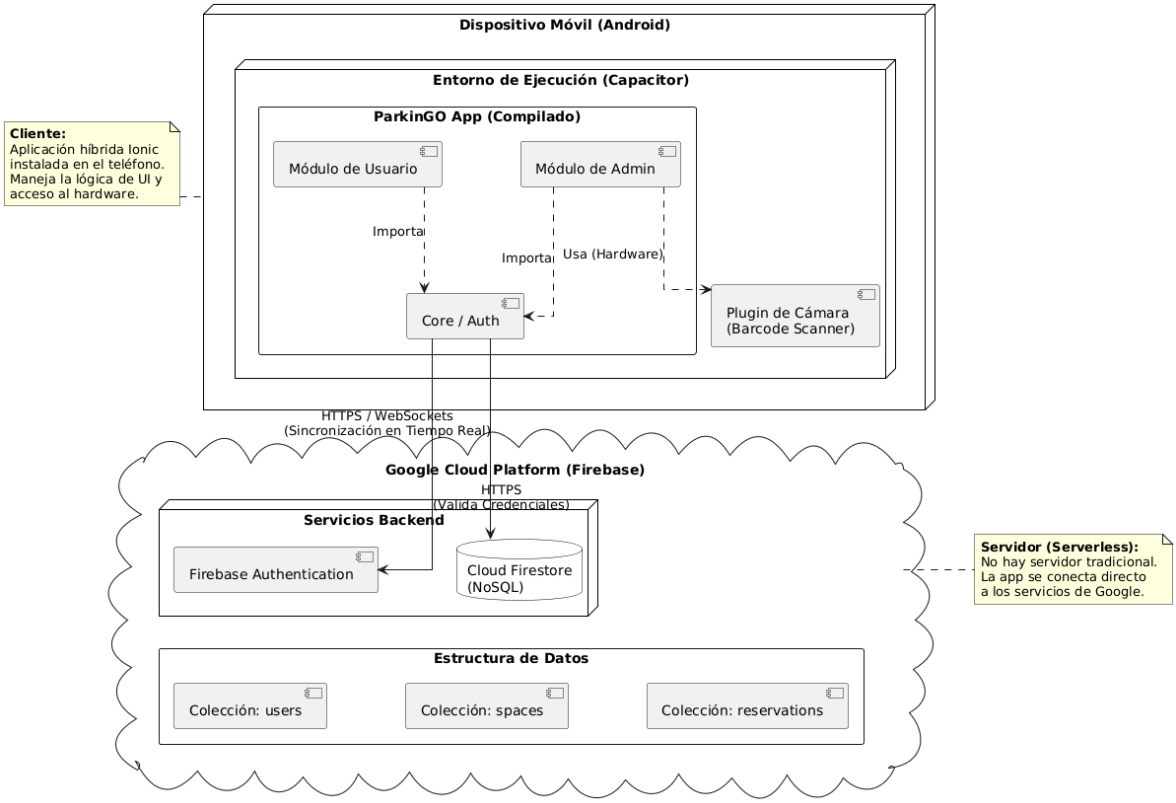


DIAGRAMA DE DESPLIEGUE:



ADMINISTRACIÓN DE USUARIOS

El sistema maneja dos tipos de usuarios con permisos diferenciados:

Usuarios estándar

- Acceso: Aplicación móvil (vistas de cliente)
- Permisos:
  - Crear reservas de espacios
  - Ver historial personal de reservas
  - Cancelar reservas propias en estado `pending`
  - Visualizar plano del parqueadero
  - Acceder a códigos QR de sus reservas activas
  - Modificar datos de perfil

Usuarios administradores

- Acceso: Panel administrativo
- Permisos:
  - Visualizar todas las reservas del sistema
  - Registrar ingresos de vehículos (escaneo QR o manual)

- Procesar salidas con cálculo de tarifas
- Gestionar espacios del parqueadero
- Consultar reportes operativos
- Ver lista completa de clientes
- Configurar tarifas por tipo de vehículo
- Cancelar cualquier reserva

Gestión de sesiones:

- Las sesiones se mantienen mediante Firebase Authentication
- Los tokens de sesión se renuevan automáticamente
- El cierre de sesión limpia completamente el estado local
- No hay límite de tiempo para sesiones activas

Creación de administradores:

Actualmente, la creación de cuentas de administrador debe realizarse manualmente mediante:

1. Creación de usuario estándar
2. Modificación del documento en Firestore agregando el campo `role: 'admin'`
3. El usuario podrá acceder al panel administrativo en su próximo inicio de sesión

MODELO RELACIONAL DE LA BASE DE DATOS

Firebase Firestore es una base de datos NoSQL orientada a documentos. A continuación se describe la estructura de las colecciones principales:

Colección: `users`

```
{
  id: string (auto-generado),
  email: string,
  name: string,
  phone: string,
  role: string ('user' | 'admin'),
  createdAt: timestamp
```

}
```

Colección: `spaces`
```
{
  id: string (auto-generado),
  code: string (Ej: "A-01", "B-05"),
  vehicleType: string ('Car' | 'Motorcycle' | 'Bicycle'),
  status: string ('Available' | 'Occupied'),
  pricePerHour: number,
  position: {
    top: string,
    left: string
  } (para renderizado en plano)
}
```
Colección: `reservations`
```
{
  id: string (auto-generado),
  vehicleType: string,
  plate: string | 'N/A',
  model: string,
  space: string (código del espacio),
  spaceId: string (ID del documento space),
  userId: string,
  email: string,
  startDate: timestamp (fecha de creación),
  endDate: timestamp (vencimiento - 24h después),

```
  entryTime: timestamp | null (cuando ingresa al parqueadero),
  exitTime: timestamp | null (cuando sale),
  pricePerHour: number,
  hours: number | null (calculado en salida),
  total: number | null (calculado en salida),
  status: string ('pending' | 'active' | 'finalizado' | 'cancelled'),
  qrCode: string (data URL del QR generado)
}
```

Colección: `salidas`

```
{
  id: string (auto-generado),
  reservationId: string,
  plate: string | null,
  model: string | null,
  space: string | null,
  vehicleType: string | null,
  entryTime: timestamp,
  exitTime: timestamp,
  hours: number,
  total: number,
  userId: string | null
}
```

Relaciones entre colecciones:

1. `reservations.userId` → `users.id` (un usuario puede tener múltiples reservas)
2. `reservations.spaceId` → `spaces.id` (una reserva ocupa un espacio)
3. `reservations.space` → `spaces.code` (referencia por código legible)
4. `salidas.reservationId` → `reservations.id` (una salida corresponde a una reserva)

Índices recomendados:

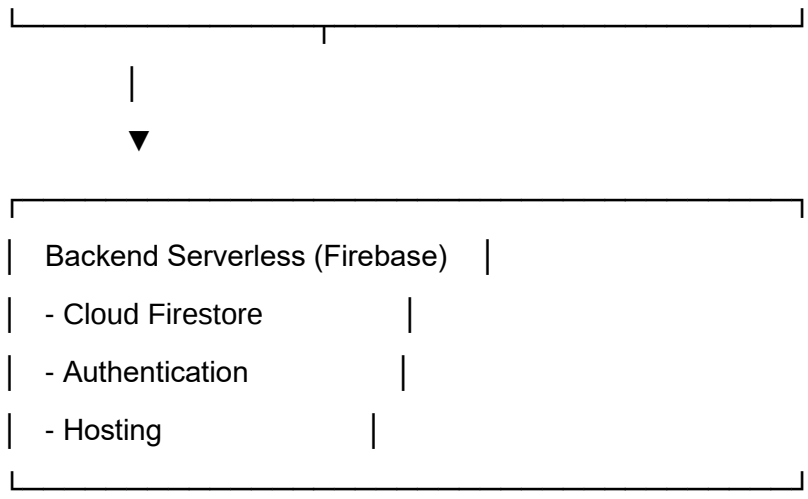
- `reservations`: Índice compuesto en `(userId, status)`
- `reservations`: Índice en `status` para consultas administrativas
- `spaces`: Índice compuesto en `(vehicleType, status)`
- `salidas`: Índice en `userId` para reportes de usuario

DESCRIPCIÓN DE LA PLATAFORMA

Arquitectura general:

ParkinGO utiliza una arquitectura de aplicación híbrida con backend serverless:

```
┌─────────────────────────────────────────────┐
│  Capa de Presentación (Ionic)               │
│  - Componentes Angular                      │
│  - Páginas de usuario                       │
│  - Páginas de administración                │
└──────────────────┬──────────────────────────┘
                   │
                   ▼
┌─────────────────────────────────────────────┐
│  Capa de Lógica de Negocio                  │
│  - Servicios Angular                        │
│  - Validaciones                             │
│  - Transformaciones de datos                │
└──────────────────┬──────────────────────────┘
                   │
                   ▼
┌─────────────────────────────────────────────┐
│  Capa de Integración                        │
│  - Angular Fire (SDK Firebase)              │
│  - Capacitor Plugins                        │
```



Tecnologías principales:

Componente	Tecnología	Versión
Framework frontend	Angular	20.0.0
UI Framework	Ionic	8.0.0
Lenguaje	TypeScript	5.8.0
Plataforma nativa	Capacitor	7.4.4
Backend	Firebase	-
Base de datos	Firestore	-
Autenticación	Firebase Auth	-
Escaneo QR nativo	ML Kit Barcode	7.4.0
Escaneo QR web	ZXing Browser	0.1.5
Generación QR	qrcode	1.5.4

Configuración de Firebase:

La aplicación se conecta a Firebase mediante la siguiente configuración (archivo `environment.ts`):

```typescript
export const environment = {
  production: false,
  FIREBASE_APP: {
```

```
    apiKey: "AIzaSyAFJ00NzTPhCBUSClg-0YbT0WldzZ_Zki4",
    authDomain: "parkingo-4ce1a.firebaseapp.com",
    projectId: "parkingo-4ce1a",
    storageBucket: "parkingo-4ce1a.firebasestorage.app",
    messagingSenderId: "771656130059",
    appId: "1:771656130059:web:dbd2a7650a36dfb501b10e"
  }
};
```

Estructura de directorios del proyecto:

```
src/
├── app/
│   ├── core/
│   │   └── providers/
│   │       └── auth/          # Servicio de autenticación
│   ├── pages/
│   │   ├── pages-users/       # Páginas de usuario final
│   │   │   ├── auth/          # Login y registro
│   │   │   ├── home/          # Dashboard usuario
│   │   │   ├── reservations/  # Nueva reserva
│   │   │   ├── index/         # Mis reservas
│   │   │   └── config/        # Configuración perfil
│   │   └── admin-pages/       # Páginas administrativas
│   │       ├── admin/         # Dashboard admin
│   │       ├── ingreso/       # Gestión de ingresos
│   │       ├── scanner/       # Escaneo QR ingreso/salida
│   │       ├── clients/       # Lista de clientes
│   │       ├── reports/       # Reportes operativos
```

```
│   │       └── config-admin/   # Configuración admin
│   └── app-routing.module.ts
├── assets/                  # Recursos estáticos
├── environments/            # Configuraciones por entorno
└── theme/                   # Estilos globales
```

Flujo de navegación:

Usuario estándar:

```
/pages-users/auth/login → /pages-users/home
                        → /pages-users/reservations
                        → /pages-users/index
                        → /pages-users/config
```

Administrador:

```
/admin-pages/auth/login-admin → /admin-pages/admin
                              → /admin-pages/ingreso
                              → /admin-pages/scanner/scan-qr
                              → /admin-pages/scanner/scan-qr-salida
                              → /admin-pages/clients
                              → /admin-pages/reports
                              → /admin-pages/config-admin
```

DOCUMENTACIÓN DEL CÓDIGO FUENTE

Convenciones de nomenclatura:

- Componentes: PascalCase terminado en `Page` o `Component`
  - Ejemplo: `ReservationsPage`, `AdminSubmenuComponent`

- Servicios: PascalCase con sufijo `Service`
  - Ejemplo: `AuthService`

- Variables y funciones: camelCase
  - Ejemplo: `availableSpaces`, `loadReservations()`

- Constantes: UPPER_SNAKE_CASE
  - Ejemplo: `FIREBASE_APP`

- Archivos: kebab-case
  - Ejemplo: `scan-qr.page.ts`, `ingreso.page.html`

Estructura de un componente típico:

```typescript
import { Component, OnInit } from '@angular/core';
import { Firestore, collection, getDocs } from '@angular/fire/firestore';

@Component({
  selector: 'app-example',
  templateUrl: './example.page.html',
  styleUrls: ['./example.page.scss'],
  standalone: false
})
export class ExamplePage implements OnInit {

  // Variables de estado
  data: any[] = [];
  loading = false;
```

```
  // Constructor con inyección de dependencias
  constructor(
    private firestore: Firestore
  ) { }

  // Método de inicialización
  async ngOnInit() {
    await this.loadData();
  }

  // Métodos privados
  private async loadData() {
    this.loading = true;
    const snapshot = await getDocs(collection(this.firestore, 'collection'));
    this.data = snapshot.docs.map(doc => ({
      id: doc.id,
      ...doc.data()
    }));
    this.loading = false;
  }
}
```

Patrones de código utilizados:

1. Inyección de dependencias: Todos los servicios y utilidades se inyectan mediante el constructor

2. Async/Await: Se utiliza para operaciones asíncronas en lugar de callbacks o promesas encadenadas

3. Reactive Forms: Para formularios con validaciones complejas (ejemplo: formulario de reservas)

4. Real-time listeners: Para datos que deben actualizarse automáticamente:

```typescript
onSnapshot(query(collection(firestore, 'reservations')), (snap) => {
  this.reservations = snap.docs.map(d => ({ id: d.id, ...d.data() }));
});
```

5. Template-driven forms: Para formularios simples (ejemplo: búsqueda por placa)

Manejo de errores:

Los errores se capturan en bloques try-catch y se muestran mediante:

```typescript
try {
  // Operación
} catch (error) {
  console.error('Error:', error);
  const toast = document.createElement('ion-toast');
  toast.message = 'Ocurrió un error: ' + error.message;
  toast.duration = 3000;
  toast.color = 'danger';
  document.body.appendChild(toast);
  await toast.present();
}
```

Comentarios en el código:

El código incluye comentarios descriptivos en secciones clave:

```typescript
// ----------------------------------------
// LÓGICA DE INGRESO DE VEHÍCULO
// ----------------------------------------
```

// 1. Validar permisos de cámara

// 2. Iniciar escaneo QR

// 3. Procesar código escaneado

```

## DESCRIPCIÓN DE LOS ACUERDOS DE NIVELES DE SERVICIOS (ANS)

El sistema ParkinGO opera bajo los siguientes parámetros de servicio:

Disponibilidad del sistema:

- Objetivo: 99.5% de disponibilidad mensual

- Basado en la disponibilidad de Firebase (SLA de Google Cloud)

- Mantenimientos programados fuera de horarios pico

Tiempo de respuesta:

- Consultas a base de datos: < 2 segundos

- Creación de reservas: < 3 segundos

- Procesamiento de salidas: < 3 segundos

- Generación de códigos QR: < 1 segundo

Rendimiento del escaneo QR:

- Tiempo de detección de código: < 1 segundo

- Procesamiento post-escaneo: < 2 segundos

Respaldo de datos:

- Firebase realiza respaldos automáticos diarios

- Retención de datos históricos: ilimitada

- Recuperación ante desastres: gestiona por Firebase

Soporte y mantenimiento:

- Actualizaciones de seguridad: según liberación de Firebase y Angular

- Corrección de bugs críticos: 24-48 horas

- Mejoras y nuevas funcionalidades: según roadmap de desarrollo

Limitaciones conocidas:

- Dependencia total de conexión a internet

- No funciona en modo offline
```

- Capacidad máxima: definida por límites de plan de Firebase
- Escalabilidad horizontal limitada a capacidades de Firestore

Monitoreo:

- Se recomienda implementar Firebase Analytics para seguimiento de uso
- Monitoreo de errores mediante console logs y Firebase Crashlytics
- Revisión periódica de límites de Firestore (lecturas/escrituras)

ANEXOS

A. Comandos de desarrollo

Instalación de dependencias:

```bash
npm install
```

Ejecutar en desarrollo (navegador):

```bash
ionic serve
```

Compilar para producción:

```bash
ionic build --prod
```

Sincronizar con Android:

```bash
npx cap sync android
```

Abrir proyecto en Android Studio:

```bash
npx cap open android
```

B. Variables de entorno

El proyecto requiere configurar las credenciales de Firebase en:

- `src/environments/environment.ts` (desarrollo)

- `src/environments/environment.prod.ts` (producción)

C. Dependencias principales

Ver archivo `package.json` para lista completa. Principales dependencias:

- `@angular/core`: ^20.0.0

- `@ionic/angular`: ^8.0.0

- `@angular/fire`: ^20.0.1

- `@capacitor/core`: 7.4.4

- `@capacitor-mlkit/barcode-scanning`: ^7.4.0

- `qrcode`: ^1.5.4

D. Configuración de permisos (Android)

El archivo `android/app/src/main/AndroidManifest.xml` debe incluir:

```xml
<uses-permission android:name="android.permission.CAMERA" />
<uses-feature android:name="android.hardware.camera" android:required="false" />
```

E. Recursos adicionales

- Documentación oficial de Ionic: https://ionicframework.com/docs

- Documentación de Firebase: https://firebase.google.com/docs

- Documentación de Angular: https://angular.dev

- Guía de Capacitor: https://capacitorjs.com/docs

Versión del documento: 1.0

Fecha: Noviembre 2025

Autores: Equipo de desarrollo ParkinGO