

# MANUAL DEL COMPRADOR

## Sistema de Gestión de Natatorio

Versión 1.0 - Enero 2025

---

### ÍNDICE

1. [Descripción General del Sistema](#)
  2. [Arquitectura Técnica](#)
  3. [Funcionalidades Principales](#)
  4. [Roles de Usuario](#)
  5. [Módulos del Sistema](#)
  6. [Requisitos de Instalación](#)
  7. [Estructura del Proyecto](#)
  8. [Base de Datos](#)
  9. [Configuración Inicial](#)
  10. [Mantenimiento y Escalabilidad](#)
  11. [Características Avanzadas](#)
  12. [Roadmap de Mejoras Futuras](#)
- 

## 1. DESCRIPCIÓN GENERAL DEL SISTEMA

### 1.1 ¿Qué es este sistema?

Sistema integral de gestión para natatorios/piscinas municipales o privadas que permite:

- Control de acceso mediante códigos QR
- Gestión de abonos y pagos
- Control de aptos médicos
- Administración de usuarios
- Reportes e ingresos

### 1.2 Problema que Resuelve

#### Antes del sistema:

- Control manual de accesos (planillas en papel)
- Verificación manual de abonos vigentes
- No hay historial digitalizado
- Pérdida de información
- Proceso lento en la entrada

#### Con el sistema:

- Acceso en 2 segundos con escaneo QR
- Verificación automática de abono y apto médico
- Historial completo digitalizado

- Reportes automáticos de ingresos
- Gestión profesional y escalable

### 1.3 Beneficios Clave

- ✓ **Eficiencia:** Reduce tiempo de acceso de 2-3 minutos a 2 segundos
  - ✓ **Control:** Trazabilidad completa de todos los accesos
  - ✓ **Ingresos:** Reportes automáticos de facturación
  - ✓ **Flexibilidad:** Tipos de abono configurables dinámicamente
  - ✓ **Profesionalismo:** Interfaz moderna y amigable
  - ✓ **Escalabilidad:** Preparado para crecer con tu negocio
- 

## 2. ARQUITECTURA TÉCNICA

### 2.1 Stack Tecnológico

#### Frontend (Cliente)

- **React 18:** Librería moderna de interfaces
- **React Router:** Navegación entre páginas
- **Tailwind CSS:** Estilos profesionales y responsivos
- **QRCode.js:** Generación de códigos QR
- **html5-qrcode:** Escaneo de códigos QR con cámara

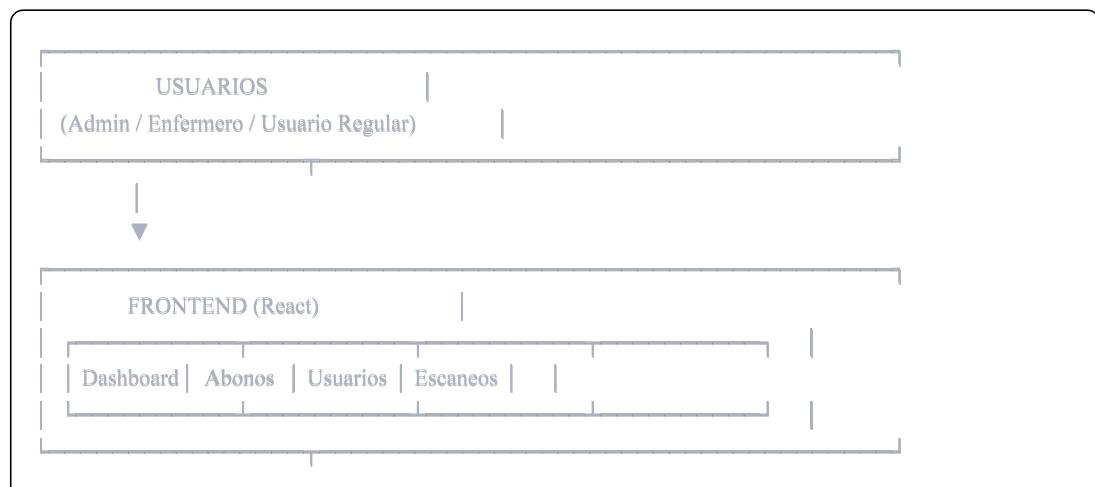
#### Backend (Servidor)

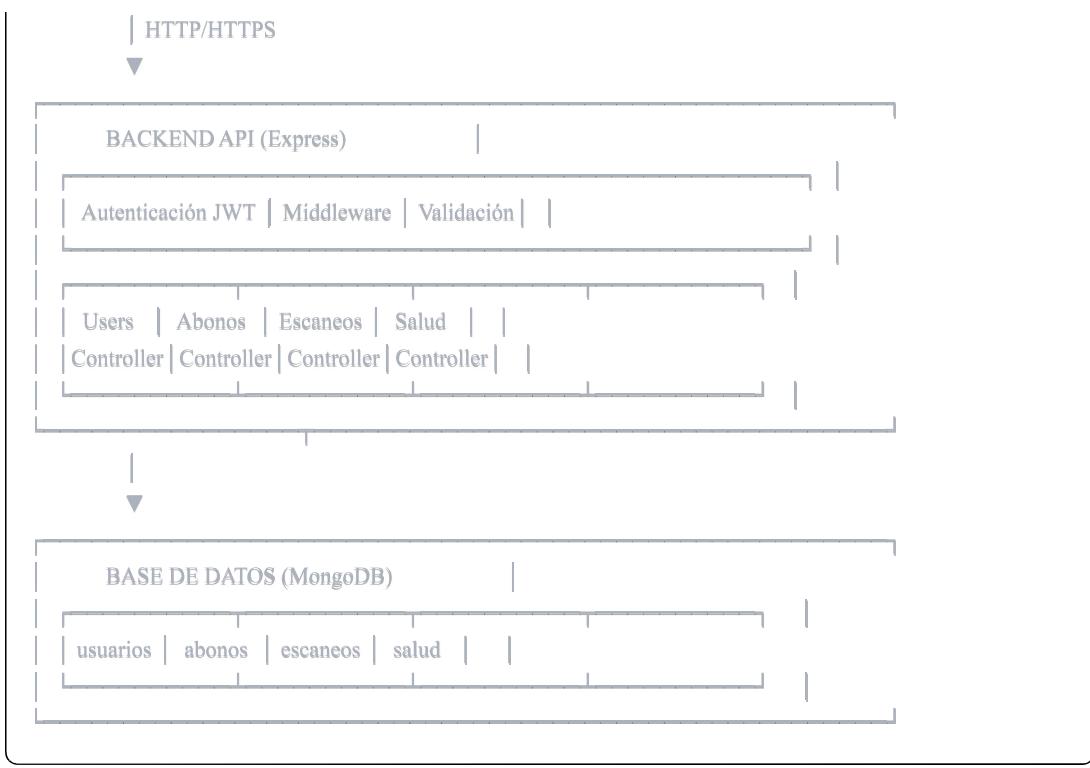
- **Node.js 18+:** Entorno de ejecución JavaScript
- **Express.js:** Framework web minimalista
- **MongoDB:** Base de datos NoSQL
- **Mongoose:** ODM para MongoDB
- **JWT:** Autenticación segura con tokens
- **bcryptjs:** Encriptación de contraseñas

#### Infraestructura

- **MongoDB Atlas:** Base de datos en la nube (recomendado)
- **Vercel/Netlify:** Deploy frontend (opcional)
- **Render/Railway:** Deploy backend (opcional)

### 2.2 Arquitectura del Sistema





## 2.3 Flujo de Datos Típico

### Ejemplo: Escaneo de QR para Acceso

1. Admin escanea código QR del usuario
2. Frontend envía solicitud a `/api/escaneos/escanear`
3. Backend valida token JWT del admin
4. Backend busca usuario por código QR
5. Backend verifica:
  - ¿Usuario activo?
  - ¿Tiene abono vigente?
  - ¿Abono está pagado?
  - ¿Tiene apto médico vigente?
6. Backend crea registro en BD con resultado
7. Backend responde con permitido/denegado + motivos
8. Frontend muestra modal visual con resultado
9. Usuario puede o no acceder según resultado

## 3. FUNCIONALIDADES PRINCIPALES

### 3.1 Control de Acceso con QR

#### Cómo funciona:

1. Cada usuario tiene un código QR único
2. Admin/Enfermero escanea el QR en la entrada
3. Sistema verifica automáticamente:
  - Abono vigente y pagado
  - Apto médico vigente
  - Usuario no baneado

4. Respuesta inmediata:  Permitido o  Denegado

5. Se registra cada intento (exitoso o rechazado)

#### **Ventajas:**

- Velocidad: 2 segundos por usuario
- Sin contacto físico
- Historial completo
- Previene fraudes

### **3.2 Gestión de Abonos**

#### **Tipos de Abono Configurables:**

- Puedes crear cualquier tipo: Mensual, Trimestral, Anual, etc.
- Cada tipo tiene: nombre, precio, duración en días, descripción
- Puedes activar/desactivar tipos sin eliminarlos
- Los tipos descontinuados siguen apareciendo en reportes históricos

#### **Ciclo de Vida de un Abono:**

1. Admin crea abono para un usuario
2. Usuario paga (efectivo, MercadoPago, transferencia)
3. Admin marca como pagado
4. Abono queda vigente por X días
5. Sistema alerta cuando está por vencer
6. Al vencer, se marca automáticamente como vencido

#### **Funcionalidades:**

- Crear abonos con escáner QR
- Marcar pagos con método
- Historial de abonos por usuario
- Filtros por tipo, estado de pago, fechas
- Exportar a CSV
- Renovación de abonos

### **3.3 Gestión de Aptos Médicos**

#### **Cómo funciona:**

1. Enfermero/Admin carga apto médico del usuario
2. Define días de validez (15, 30, 60, etc.)
3. Sistema calcula fecha de vencimiento
4. Alertas automáticas cuando está por vencer
5. En el acceso, se verifica si está vigente

#### **Características:**

- Carga con escáner QR
- Validez configurable
- Notas opcionales del enfermero

- Historial de renovaciones
- Dashboard con alertas de vencimientos

### 3.4 Gestión de Usuarios

#### Tipos de Usuario:

- **Admin:** Control total del sistema
- **Enfermero:** Carga aptos médicos
- **Usuario Regular:** Accede a su dashboard

#### Perfil de Usuario Incluye:

- Datos personales (nombre, DNI, email, teléfono)
- Foto de perfil
- Código QR único
- Fecha de nacimiento
- Dirección completa
- Estado (activo/baneado)

#### Funcionalidades Admin:

- Crear usuarios masivamente
- Editar perfiles
- Banear/Desbanear usuarios
- Buscar por DNI, nombre, email
- Filtros por estado

### 3.5 Reportes e Ingresos

#### Planilla de Ingresos:

- Filtra por rango de fechas
- Filtra por tipo de abono
- Filtra por método de pago
- Muestra monto total
- Exporta a CSV para Excel

#### Estadísticas del Dashboard:

- Total de usuarios activos
- Abonos vigentes
- Accesos de hoy
- Aptos médicos vigentes
- Tasa de éxito de accesos
- Gráficos y métricas

---

## 4. ROLES DE USUARIO

### 4.1 Administrador

#### Permisos Completos:

- Gestionar usuarios (crear, editar, banear)
- Gestionar abonos (crear, marcar pagos, eliminar)
- Gestionar tipos de abono (crear, editar, activar/desactivar)
- Escanear QR para acceso
- Ver todos los reportes e ingresos
- Cargar aptos médicos
- Configurar sistema

#### **Dashboard del Admin:**

- Estadísticas generales
- Accesos de hoy
- Acceso rápido a todas las secciones
- Escaneo QR integrado

#### **4.2 Enfermero**

##### **Permisos Limitados:**

- Cargar aptos médicos
- Escanear QR para verificar aptos
- Ver lista de usuarios
- NO puede gestionar abonos
- NO puede ver ingresos
- NO puede banear usuarios

##### **Dashboard del Enfermero:**

- Lista de usuarios
- Aptos próximos a vencer
- Aptos vencidos
- Carga rápida con QR

#### **4.3 Usuario Regular**

##### **Permisos Mínimos:**

- Ver su propio dashboard
- Ver su código QR
- Ver estado de su abono
- Ver estado de su apto médico
- Ver su historial de accesos
- Ver su historial de abonos
- Editar su perfil (email, teléfono, foto)
- NO puede ver otros usuarios
- NO puede gestionar nada

##### **Dashboard del Usuario:**

- Su código QR para escanear
- Estado actual del abono

- Estado actual del apto médico
  - Últimos 10 accesos
  - Historial completo de abonos
- 

## 5. MÓDULOS DEL SISTEMA

### 5.1 Módulo de Autenticación

#### Características:

- Login con email y contraseña
- Tokens JWT con expiración
- Middleware de protección de rutas
- Verificación de roles
- Sesiones seguras

#### Archivos Clave:

```
backend/middleware/authMiddleware.js  
backend/controllers/authController.js  
frontend/src/context/AuthContext.jsx
```

### 5.2 Módulo de Usuarios

#### Características:

- CRUD completo de usuarios
- Generación automática de QR único
- Subida de foto de perfil en base64
- Búsqueda y filtros
- Paginación

#### Archivos Clave:

```
backend/models/Usuario.js  
backend/controllers/userController.js  
backend/routes/users.js  
frontend/src/pages/admin/Usuarios.jsx
```

### 5.3 Módulo de Abonos

#### Características:

- Tipos de abono dinámicos y configurables
- Creación con asignación de usuario
- Estados: pendiente de pago, pagado, vigente, vencido
- Métodos de pago: efectivo, MercadoPago, transferencia
- Historial completo por usuario

#### Archivos Clave:

```
backend/models/Abono.js  
backend/models/Configuracion.js  
backend/controllers/abonoController.js  
backend/controllers/configuracionController.js  
backend/routes/abonos.js  
frontend/src/pages/admin/Abonos.jsx  
frontend/src/pages/admin/ConfiguracionTarifas.jsx
```

#### **5.4 Módulo de Escaneos (Control de Acceso)**

##### **Características:**

- Escaneo con cámara del dispositivo
- Validación de múltiples condiciones
- Registro de cada intento
- Motivos de rechazo detallados
- Historial por usuario y por día

##### **Archivos Clave:**

```
backend/models/Escaneo.js  
backend/controllers/escaneoController.js  
backend/routes/escaneos.js  
frontend/src/pages/admin/EscanearQR.jsx  
frontend/src/pages/admin/Dashboard.jsx (escaneo integrado)
```

#### **5.5 Módulo de Salud (Aptos Médicos)**

##### **Características:**

- Carga de apto con días de validez
- Cálculo automático de vencimiento
- Renovaciones
- Alertas de vencimiento
- Notas del enfermero

##### **Archivos Clave:**

```
backend/models/PruebaSalud.js  
backend/controllers/escaneoController.js (incluye funciones de salud)  
backend/routes/salud.js  
frontend/src/pages/enfermero/CargarApto.jsx  
frontend/src/pages/enfermero/Dashboard.jsx
```

#### **5.6 Módulo de Reportes**

##### **Características:**

- Planilla de ingresos con filtros
- Exportación a CSV
- Estadísticas del dashboard
- Reportes por fecha, tipo, método de pago

##### **Archivos Clave:**

```
backend/routes/reportes.js  
backend/controllers/reporteController.js (si existe)  
frontend/src/pages/admin/PlanillaIngresos.jsx
```

## 6. REQUISITOS DE INSTALACIÓN

### 6.1 Requisitos del Sistema

#### Software Necesario:

- Node.js 18+ (recomendado: versión LTS)
- npm 9+ o yarn 1.22+
- MongoDB 6+ (local o Atlas)
- Git (para control de versiones)

#### Hardware Mínimo (Servidor):

- CPU: 2 cores
- RAM: 2GB
- Disco: 10GB
- Conexión a Internet estable

#### Hardware Mínimo (Clientes):

- Navegador moderno (Chrome, Firefox, Edge, Safari)
- Cámara web o de dispositivo móvil (para escanear QR)

### 6.2 Instalación Paso a Paso

#### Paso 1: Clonar el Repositorio

```
bash  
  
git clone <URL_DEL_REPOITORIO>  
cd natatorio-sistema
```

#### Paso 2: Configurar Backend

```
bash  
  
cd backend  
npm install
```

#### Crear archivo `.env`:

```
env  
  
PORT=5000  
MONGODB_URI=mongodb://localhost:27017/natatorio  
# O MongoDB Atlas:  
# MONGODB_URI=mongodb+srv://usuario:password@cluster.mongodb.net/natatorio  
  
JWT_SECRET=tu_clave_secreta_muy_segura_aqui  
NODE_ENV=production
```

### Paso 3: Configurar Frontend

```
bash  
  
cd frontend  
npm install
```

Crear archivo `.env`:

```
env  
  
VITE_API_URL=http://localhost:5000/api  
# O en producción:  
# VITE_API_URL=https://tu-backend.com/api
```

### Paso 4: Inicializar Base de Datos

Opción A - MongoDB Local:

```
bash  
  
# Instalar MongoDB  
# Windows: https://www.mongodb.com/try/download/community  
# Linux: sudo apt install mongodb  
# Mac: brew install mongodb-community  
  
# Iniciar MongoDB  
mongod
```

Opción B - MongoDB Atlas (Recomendado):

1. Crear cuenta en <https://www.mongodb.com/cloud/atlas>
2. Crear cluster gratuito
3. Crear usuario de base de datos
4. Whitelist IP (0.0.0.0/0 para permitir todas)
5. Copiar connection string al `.env`

### Paso 5: Crear Usuario Admin Inicial

Ejecutar script de inicialización:

```
bash  
  
cd backend  
node scripts/createAdmin.js
```

O crear manualmente en MongoDB:

```
javascript
```

```
{  
  nombre: "Admin",  
  apellido: "Principal",  
  email: "admin@natatorio.com",  
  password: "$2a$10$...", // Hash de "admin123"  
  rol: "admin",  
  activo: true,  
  qrCode: "ADMIN-UNIQUE-CODE"  
}
```

## Paso 6: Iniciar el Sistema

Terminal 1 - Backend:

```
bash  
  
cd backend  
npm run dev  
# Server running on http://localhost:5000
```

Terminal 2 - Frontend:

```
bash  
  
cd frontend  
npm run dev  
# Client running on http://localhost:5173
```

## Paso 7: Acceder al Sistema

Abrir navegador en: <http://localhost:5173>

Login inicial:

- Email: [admin@natatorio.com](mailto:admin@natatorio.com)
- Password: [admin123](#)

**⚠ IMPORTANTE:** Cambiar la contraseña del admin inmediatamente.

---

## 7. ESTRUCTURA DEL PROYECTO

### 7.1 Estructura del Backend

```
backend/  
  config/  
    db.js          # Configuración MongoDB  
  controllers/  
    authController.js  # Login, registro  
    userController.js   # CRUD usuarios  
    abonoController.js  # CRUD abonos  
    configuracionController.js # Tipos de abono  
    escaneoController.js # Escaneos y salud  
    reporteController.js  # Reportes  
  middleware/  
    authMiddleware.js   # JWT, verificación roles  
  models/  
    Usuario.js        # Schema usuarios
```

```

    └── Abono.js      # Schema abonos
    └── Configuracion.js   # Schema tipos de abono
    └── Escaneo.js     # Schema escaneos
    └── PruebaSalud.js  # Schema aptos médicos
    └── routes/
        └── auth.js      # Rutas autenticación
        └── users.js      # Rutas usuarios
        └── abonos.js     # Rutas abonos
        └── configuracion.js  # Rutas configuración
        └── escaneos.js    # Rutas escaneos
        └── salud.js       # Rutas salud
        └── reportes.js   # Rutas reportes
    └── scripts/
        └── createAdmin.js # Script crear admin
    └── .env           # Variables de entorno
    └── .env.example   # Ejemplo de .env
    └── server.js      # Punto de entrada
    └── package.json

```

## 7.2 Estructura del Frontend

```

frontend/
    └── public/
        └── index.html
    └── src/
        └── assets/          # Imágenes, logos
        └── components/      # Componentes reutilizables
            └── Navbar.jsx
            └── Modal.jsx
            └── ...
        └── context/
            └── AuthContext.jsx # Context de autenticación
        └── pages/
            └── admin/         # Páginas del admin
                └── Dashboard.jsx
                └── Usuarios.jsx
                └── Abonos.jsx
                └── ConfiguracionTarifas.jsx
                └── EscanearQR.jsx
                └── PlanillaIngresos.jsx
            └── enfermero/      # Páginas del enfermero
                └── Dashboard.jsx
                └── CargarApto.jsx
            └── user/           # Páginas del usuario
                └── Dashboard.jsx
                └── Login.jsx      # Página de login
        └── services/        # Servicios API
            └── api.js         # Cliente Axios configurado
            └── authService.js
            └── userService.js
            └── abonoService.js
            └── escaneoService.js
            └── statsService.js
        └── App.jsx          # Componente principal
        └── main.jsx         # Punto de entrada
        └── index.css        # Estilos Tailwind
    └── .env             # Variables de entorno
    └── .env.example

```

```
|--- vite.config.js      # Configuración Vite  
|--- tailwind.config.js # Configuración Tailwind  
└--- package.json
```

## 8. BASE DE DATOS

### 8.1 Modelos de Datos

#### Usuario

```
javascript  
  
{  
  _id: ObjectId,  
  nombre: String,  
  apellido: String,  
  email: String (único),  
  password: String (hash brypt),  
  dni: String (único),  
  telefono: String,  
  fechaNacimiento: Date,  
  direccion: {  
    calle: String,  
    numero: String,  
    ciudad: String,  
    provincia: String,  
    codigoPostal: String  
  },  
  fotoPerfil: String (base64),  
  qrCode: String (único),  
  rol: String (enum: admin, enfermero, usuario),  
  activo: Boolean,  
  baneado: Boolean,  
  motivoBaneo: String,  
  fechaBaneo: Date,  
  abonoActual: ObjectId → Abono,  
  pruebaSalud: ObjectId → PruebaSalud,  
  createdAt: Date,  
  updatedAt: Date  
}
```

#### Abono

```
javascript
```

```

{
  _id: ObjectId,
  usuario: ObjectId → Usuario,
  tipoAbono: String (ID del tipo desde Configuracion),
  precio: Number,
  fechaInicio: Date,
  fechaFin: Date,
  pagado: Boolean,
  metodoPago: String (enum: efectivo, mercadopago, transferencia),
  fechaPago: Date,
  notas: String,
  createdAt: Date,
  updatedAt: Date
}

```

### Configuracion (Singleton)

```

javascript

{
  _id: ObjectId,
  tiposAbono: [
    {
      id: String (único, ej: "mensual"),
      nombre: String,
      precio: Number,
      duracionDias: Number,
      descripcion: String,
      activo: Boolean
    }
  ],
  createdAt: Date,
  updatedAt: Date
}

```

### Escaneo

```

javascript

{
  _id: ObjectId,
  usuario: ObjectId → Usuario,
  abono: ObjectId → Abono,
  fechaHora: Date,
  exitoso: Boolean,
  motivoRechazo: String (enum: abono_vencido, sin_abono, etc.),
  escaneadoPor: ObjectId → Usuario (admin que escaneó),
  notas: String,
  ipAddress: String,
  userAgent: String,
  createdAt: Date
}

```

### PruebaSalud

```

javascript

```

```
{
  _id: ObjectId,
  usuario: ObjectId → Usuario,
  fechaPrueba: Date,
  fechaVencimiento: Date,
  vigente: Boolean (virtual),
  diasValidez: Number,
  notas: String,
  cargadoPor: ObjectId → Usuario (enfermero/admin),
  createdAt: Date,
  updatedAt: Date
}
```

## 8.2 Índices Importantes

Para optimizar el rendimiento, crear estos índices:

```
javascript

// Usuarios
db.usuarios.createIndex({ email: 1 }, { unique: true });
db.usuarios.createIndex({ dni: 1 }, { unique: true });
db.usuarios.createIndex({ qrCode: 1 }, { unique: true });

// Abonos
db.abonos.createIndex({ usuario: 1, fechaFin: -1 });
db.abonos.createIndex({ pagado: 1, fechaFin: -1 });

// Escaneos
db.escaneos.createIndex({ usuario: 1, fechaHora: -1 });
db.escaneos.createIndex({ fechaHora: -1 });

// PruebaSalud
db.pruebassalud.createIndex({ usuario: 1 }, { unique: true });
```

---

## 9. CONFIGURACIÓN INICIAL

### 9.1 Configuración de Tipos de Abono

Al iniciar el sistema por primera vez, debes configurar los tipos de abono disponibles:

1. Login como admin
2. Ir a "Configuración de Tarifas"
3. Crear tipos base:
  - **Mensual:** \$5000, 30 días
  - **Trimestral:** \$13500, 90 días
  - **Semestral:** \$24000, 180 días
  - **Anual:** \$45000, 365 días
4. Puedes agregar tipos personalizados:
  - **Mensual Adultos Mayores:** \$3000, 30 días
  - **Mensual Estudiantes:** \$4000, 30 días
  - **Mensual Niños:** \$3500, 30 días

## 9.2 Carga Inicial de Usuarios

### Opción A: Carga Manual (Pocos Usuarios)

1. Ir a "Usuarios"
2. Clic en "Crear Usuario"
3. Completar formulario
4. El código QR se genera automáticamente

### Opción B: Carga Masiva (Muchos Usuarios)

Crear script de importación desde CSV:

```
javascript

// backend/scripts/importUsers.js
const fs = require('fs');
const csv = require('csv-parser');
const Usuario = require('../models/Usuario');

async function importUsers() {
  const users = [];

  fs.createReadStream('usuarios.csv')
    .pipe(csv())
    .on('data', (row) => {
      users.push({
        nombre: row.nombre,
        apellido: row.apellido,
        email: row.email,
        dni: row.dni,
        telefono: row.telefono,
        password: 'cambiar123', // Password temporal
        qrCode: `USER-${row.dni}` // QR basado en DNI
      });
    })
    .on('end', async () => {
      await Usuario.insertMany(users);
      console.log(`${users.length} usuarios importados`);
    });
}

importUsers();
```

## 9.3 Configuración de Correo (Opcional)

Para enviar notificaciones por email:

```
bash

npm install nodemailer
```

```
javascript
```

```
// backend/config/email.js
const nodemailer = require('nodemailer');

const transporter = nodemailer.createTransport({
  service: 'gmail',
  auth: {
    user: process.env.EMAIL_USER,
    pass: process.env.EMAIL_PASSWORD
  }
});

module.exports = transporter;
```

Agregar a `.env`:

```
env
EMAIL_USER=natatorio@gmail.com
EMAIL_PASSWORD=tu_password_de_app
```

## 10. MANTENIMIENTO Y ESCALABILIDAD

### 10.1 Tareas de Mantenimiento Rutinarias

#### Diarias:

- Verificar que el servidor esté activo
- Revisar logs de errores
- Backup automático (si está configurado)

#### Semanales:

- Revisar usuarios con abonos próximos a vencer
- Revisar aptos médicos próximos a vencer
- Limpiar logs antiguos (si es necesario)

#### Mensuales:

- Backup manual de la base de datos
- Revisar métricas de uso
- Actualizar dependencias de Node.js
- Verificar espacio en disco

### 10.2 Backups de Base de Datos

#### Backup Manual (MongoDB):

```
bash
# Backup completo
mongodump --uri="mongodb://localhost:27017/natatorio" --out=/backups/$(date +%Y%m%d)

# Restaurar backup
mongorestore --uri="mongodb://localhost:27017/natatorio" /backups/20250120
```

## **Backup Automático (Script):**

```
bash

#!/bin/bash
# backup.sh
DATE=$(date +%Y%m%d_%H%M%S)
mongodump --uri="$MONGODB_URI" --out="/backups/natatorio_$DATE"

# Eliminar backups de más de 30 días
find /backups -type d -mtime +30 -exec rm -rf {} \;
```

Configurar en crontab:

```
bash

# Ejecutar todos los días a las 2 AM
0 2 * * * /path/to/backup.sh
```

## **10.3 Escalabilidad**

### **Cuando el Sistema Crece:**

#### **50-100 usuarios:**

- MongoDB local está bien
- Un solo servidor

#### **100-500 usuarios:**

- Migrar a MongoDB Atlas
- Considerar CDN para archivos estáticos
- Optimizar imágenes (comprimir fotos de perfil)

#### **500-1000+ usuarios:**

- Load balancer
- Múltiples instancias del backend
- Redis para caché de sesiones
- CDN para todo el frontend

### **Mejoras de Performance:**

1. Implementar caché con Redis
2. Paginación en todas las listas
3. Lazy loading de imágenes
4. Comprimir respuestas del API (gzip)
5. Índices en BD para queries frecuentes

---

## **11. CARACTERÍSTICAS AVANZADAS**

### **11.1 Código QR Único por Usuario**

Cada usuario tiene un QR único que incluye:

- ID único generado automáticamente

- Formato: `USER-{timestamp}-{random}`
- Almacenado en campo `qrCode`
- Generado automáticamente al crear usuario

#### **Mostrar QR al Usuario:**

```
javascript

// En el dashboard del usuario
import QRCode from 'qrcode';

QRCode.toCanvas(canvasRef.current, user.qrCode, {
  width: 200,
  margin: 2
});
```

#### **11.2 Validaciones en Tiempo Real**

##### **Al escanear QR, el sistema verifica:**

1. ¿Código QR válido?
2. ¿Usuario existe?
3. ¿Usuario activo?
4. ¿Usuario no baneado?
5. ¿Tiene abono asignado?
6. ¿Abono vigente (no vencido)?
7. ¿Abono pagado?
8. ¿Tiene apto médico?
9. ¿Apto médico vigente?

##### **Cualquier falla = Acceso Denegado**

#### **11.3 Historial Completo**

##### **El sistema registra:**

- Todos los escaneos (exitosos y rechazados)
- Motivo exacto de cada rechazo
- Quién escaneó el QR
- Fecha y hora exacta
- IP del dispositivo
- User agent del navegador

##### **Esto permite:**

- Auditoría completa
- Detectar patrones de fraude
- Estadísticas de uso
- Resolver disputas

#### **11.4 Modal de Confirmación con Foto**

##### **Funcionalidad de Seguridad:**

Al escanear un QR, antes de crear el abono o cargar apto:

1. Se muestra foto del usuario
2. Se muestra nombre completo y DNI
3. Admin/Enfermero confirma que es la persona correcta
4. Previene errores y suplantación de identidad

## 11.5 Tipos de Abono Dinámicos

### Ventaja Clave:

No estás limitado a tipos predefinidos. Puedes:

- Crear nuevos tipos en cualquier momento
- Cambiar precios sin afectar abonos existentes
- Desactivar tipos temporalmente
- Los tipos históricos persisten en reportes

### Ejemplo de Uso:

Promoción especial:

1. Crear tipo "Promoción Verano"
  2. Precio: \$8000, 45 días
  3. Vender durante enero-febrero
  4. Desactivar tipo en marzo
  5. Los que lo compraron siguen teniéndolo
  6. Aparece en reportes históricos
- 

## 12. ROADMAP DE MEJORAS FUTURAS

### 12.1 Próximas Funcionalidades (Corto Plazo)

#### Notificaciones por Email:

- Enviar email cuando abono está por vencer (7 días antes)
- Enviar email cuando apto médico vence (15 días antes)
- Confirmación de pago por email

#### Notificaciones Push:

- Notificaciones en navegador
- Recordatorios personalizados

#### Integración MercadoPago:

- Pago online de abonos
- Link de pago generado automáticamente
- Webhook para marcar como pagado

#### Reportes Avanzados:

- Gráficos de ingresos mensuales
- Estadísticas de uso por horario
- Análisis de tipos de abono más vendidos

- Proyecciones de ingresos

## 12.2 Mejoras de Mediano Plazo

### App Móvil Nativa:

- React Native para iOS y Android
- Escaneo de QR más rápido
- Funciona offline (sincroniza después)
- Notificaciones push nativas

### Sistema de Turnos:

- Reserva de carriles
- Límite de capacidad
- Turnos para clases grupales

### Gestión de Clases:

- Crear clases grupales
- Inscripción de alumnos
- Control de asistencia
- Pagos separados de clases

### Panel de Métricas Avanzado:

- Dashboard con KPIs
- Comparación mes a mes
- Predicción de churn (abandono)
- Análisis de retención

## 12.3 Visión a Largo Plazo

### Multi-Sede:

- Gestionar múltiples sucursales
- Acceso a cualquier sede con un solo abono
- Reportes consolidados

### Sistema de Puntos/Recompensas:

- Acumular puntos por visitas
- Canjear por descuentos o upgrades
- Gamificación para fidelización

### Integración con Wearables:

- Pulseras NFC para acceso
- Sin necesidad de escanear QR
- Más rápido y práctico

### Inteligencia Artificial:

- Predicción de demanda

- Recomendaciones de precios
  - Detección automática de patrones sospechosos
- 

## SOPORTE Y CONTACTO

### Información del Desarrollador

**Desarrollador:** Juan (Ingeniero Químico y Programador)

**Ubicación:** General Belgrano, Buenos Aires, Argentina

**Empresa:** Boros (Materiales Reciclados)

### Soporte Técnico

Para consultas técnicas, modificaciones o soporte:

- Contactar al desarrollador directamente
- Incluir descripción detallada del problema
- Adjuntar logs de error si es posible
- Especificar versión del sistema

### Actualizaciones del Sistema

El sistema puede recibir actualizaciones para:

- Corrección de bugs
- Nuevas funcionalidades
- Mejoras de seguridad
- Optimización de rendimiento

**Recomendación:** Mantener contacto con el desarrollador para recibir actualizaciones.

---

## NOTAS FINALES

### Licencia y Propiedad

Este sistema fue desarrollado específicamente para gestión de natatorios/piscinas. Los derechos de uso se transfieren al comprador según lo acordado en el contrato de venta.

### Garantía

El sistema se entrega funcionando según las especificaciones descritas en este manual. Cualquier bug o error crítico será corregido sin costo adicional durante el período de garantía acordado.

### Responsabilidad

El comprador es responsable de:

- Mantener copias de seguridad de la base de datos
- Configurar correctamente el sistema según este manual
- Cumplir con las leyes de protección de datos vigentes
- Mantener actualizado el software base (Node.js, MongoDB)

### Privacidad de Datos

El sistema almacena datos personales de usuarios. El comprador debe:

- Cumplir con GDPR / Ley de Protección de Datos Personal (Argentina)
  - Informar a usuarios sobre qué datos se recopilan
  - Permitir a usuarios solicitar eliminación de datos
  - Mantener datos seguros y encriptados
- 

## CHECKLIST DE IMPLEMENTACIÓN

Usa esta lista para verificar que todo está correcto:

### Instalación:

- Node.js 18+ instalado
- MongoDB instalado o Atlas configurado
- Repositorio clonado
- Dependencias instaladas (backend y frontend)
- Archivos .env creados y configurados
- Base de datos inicializada
- Usuario admin creado

### Configuración:

- Tipos de abono configurados
- Al menos 1 usuario de prueba creado
- QR del usuario de prueba generado
- Escaneo de QR probado y funcionando

### Testing:

- Login funciona correctamente
- Crear usuario funciona
- Crear abono funciona
- Escaneo QR funciona
- Cargar apto médico funciona
- Reportes muestran datos correctos

### Producción:

- Variables de entorno de producción configuradas
  - Base de datos en la nube (MongoDB Atlas)
  - Backend deployado
  - Frontend deployado
  - HTTPS configurado (certificado SSL)
  - Backups automáticos configurados
  - Contraseña del admin cambiada
- 

## ¡FELICITACIONES!

Has adquirido un sistema profesional de gestión de natatorio. Este manual te proporciona toda la información necesaria para instalarlo, configurarlo y operarlo exitosamente.

### Recuerda:

- Hacer backups regularmente
- Mantener el sistema actualizado

- Contactar al desarrollador ante cualquier duda
- Aprovechar todas las funcionalidades

¡Éxito con tu natatorio! 🏊💪

---

**Versión del Manual:** 1.0

**Fecha:** Enero 2025

**Sistema:** Gestión de Natatorio v1.0

**Desarrollado por:** Juan - Boros

---

*Este manual es un documento vivo y puede ser actualizado según evolucione el sistema.*