

# Arquitectura del Sistema SmartRehabBar

---

## 1. Visión General

---

SmartRehabBar es un sistema de rehabilitación que monitoriza la distribución del peso corporal entre ambos pies durante la marcha, diseñado para ayudar a pacientes en recuperación post-accidente a corregir desequilibrios.

### 1.1 Objetivos del Sistema

- Capturar mediciones de peso de dos plataformas instrumentadas (una por pie)
- Procesar y sincronizar mediciones que pueden llegar de forma asíncrona
- Visualizar en tiempo real la distribución del peso y detectar desequilibrios
- Almacenar datos para análisis histórico y seguimiento de progreso
- Soportar múltiples pacientes y sesiones de rehabilitación

### 1.2 Restricciones y Consideraciones

- Las mediciones de ambos pies NO son simultáneas
- Múltiples mediciones pueden ocurrir durante un solo paso
- Sistema debe funcionar offline (sin acceso a internet)
- Comunicación vía HTTP desde plataformas al servidor
- Despliegue dual: Heroku (validación) → Raspberry Pi (producción)

## 2. Stack Tecnológico

---

### 2.1 Backend

Tecnología	Versión	Justificación
Node.js	18.x LTS	Runtime JavaScript eficiente, excelente para I/O asíncrono
Express	4.x	Framework web minimalista y maduro para APIs REST
Socket.IO	4.x	Comunicación bidireccional en tiempo real con fallback automático
Prisma	5.x	ORM moderno con soporte PostgreSQL y SQLite, type-safe
PostgreSQL	15.x	Base de datos para Heroku (addon gratuito disponible)
SQLite	3.x	Base de datos para Raspberry Pi (sin servidor, archivo local)

#### Justificación del Stack Backend:

- **Node.js + Express:** Ideal para manejar múltiples conexiones HTTP concurrentes de las plataformas
- **Socket.IO:** Permite actualizar el dashboard en tiempo real sin polling
- **Prisma:** Facilita migración entre PostgreSQL (Heroku) y SQLite (Raspberry Pi)
- **Event-driven architecture:** Perfecto para sincronización de mediciones asíncronas

## 2.2 Frontend

Tecnología	Versión	Justificación
React	18.x	Librería UI reactiva, ideal para dashboards en tiempo real
Vite	5.x	Build tool rápido con HMR, mejor experiencia de desarrollo
Socket.IO Client	4.x	Cliente para comunicación en tiempo real con backend

Tecnología	Versión	Justificación
Recharts	2.x	Librería de gráficos React-friendly, declarativa y responsiva
TailwindCSS	3.x	Framework CSS utility-first, desarrollo rápido de UI
React Router	6.x	Navegación entre vistas (pacientes, dashboard, historial)

**Justificación del Stack Frontend:**

- **React:** Componentes reutilizables, estado reactivo ideal para datos en tiempo real
- **Vite:** Desarrollo más rápido que Create React App, mejor para prototipado
- **Recharts:** Gráficos responsivos sin configuración compleja
- **TailwindCSS:** Prototipado rápido sin escribir CSS custom

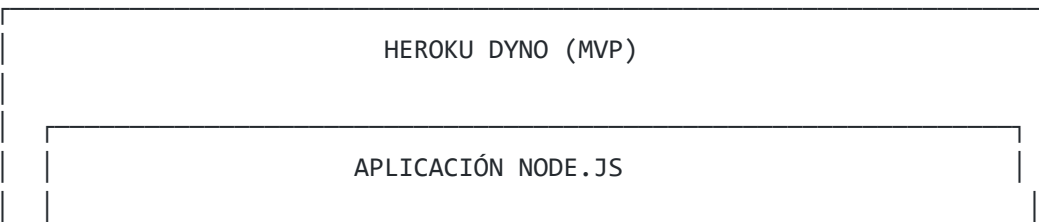
**2.3 Herramientas de Desarrollo**

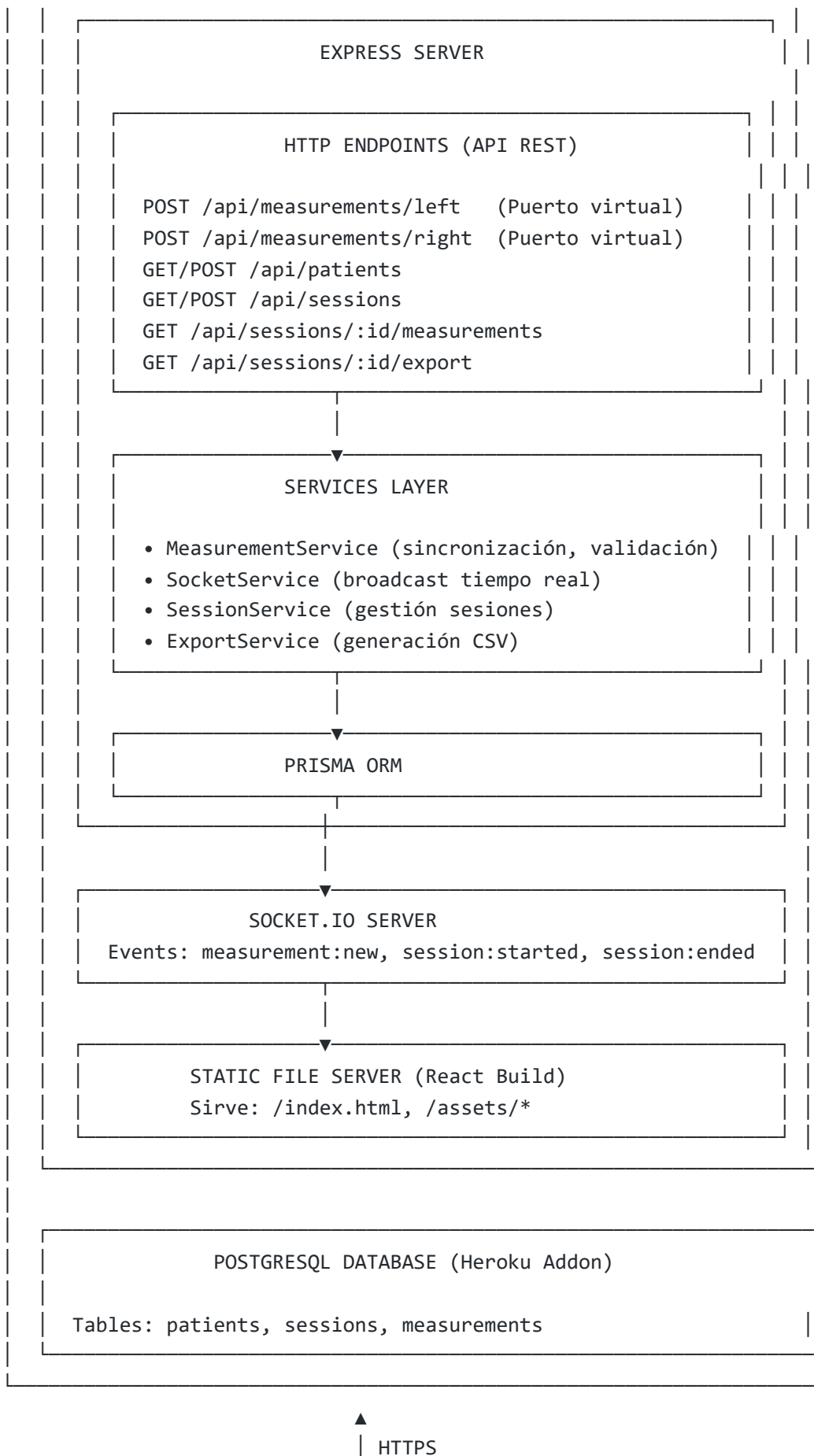
- **Git:** Control de versiones
- **ESLint:** Linting de código JavaScript
- **Prettier:** Formateo consistente de código
- **Nodemon:** Auto-reload del servidor en desarrollo
- **Concurrently:** Ejecutar backend y frontend simultáneamente

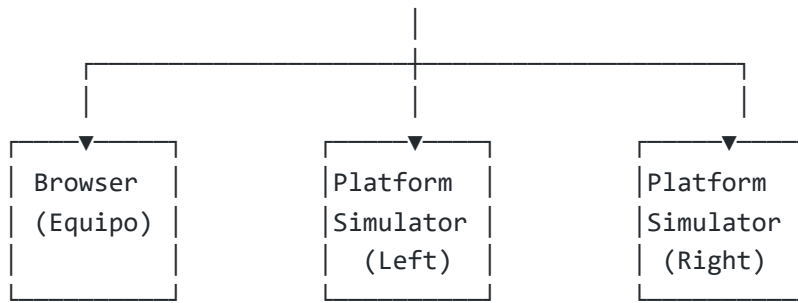
**3. Arquitectura del Sistema**

---

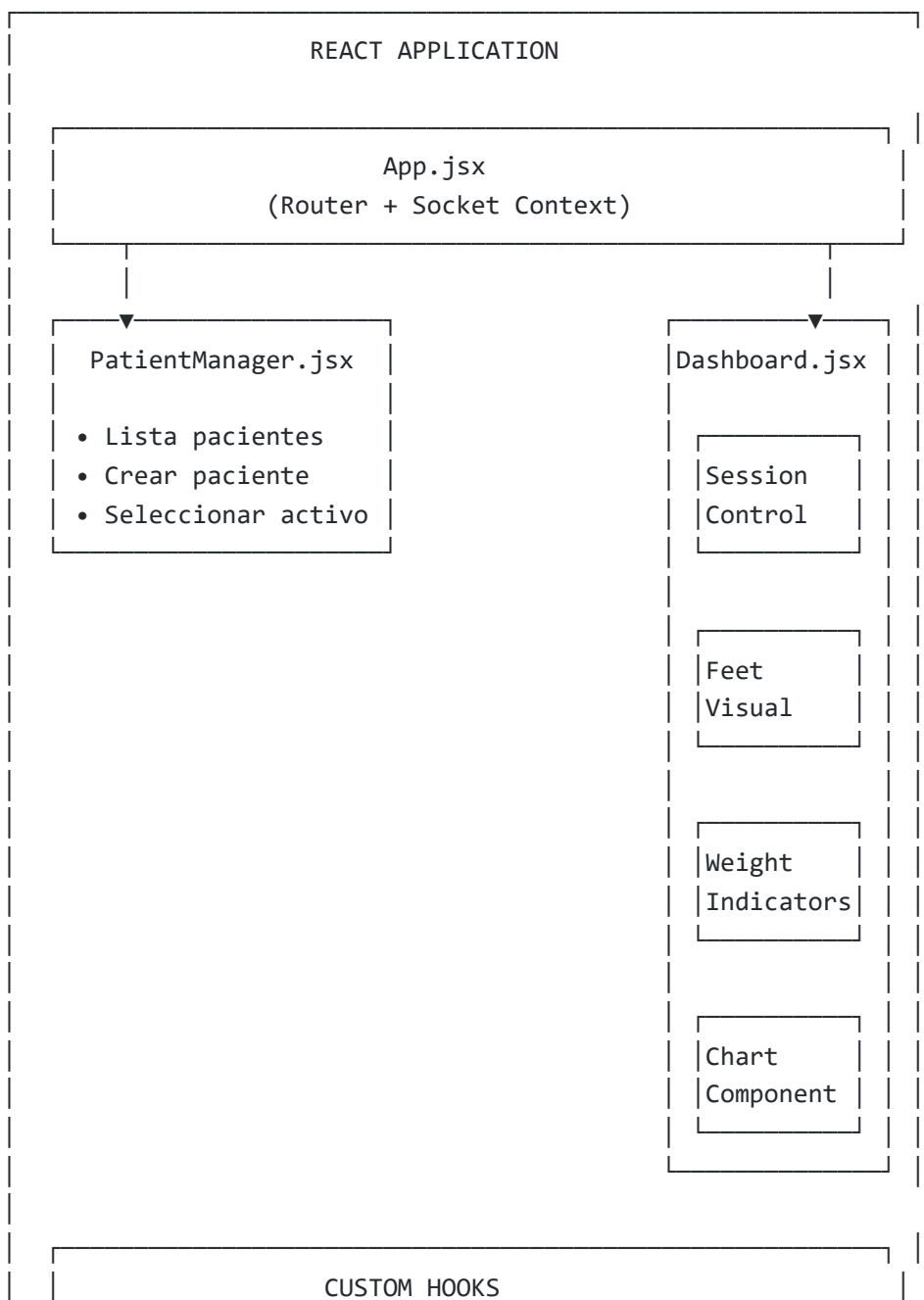
**3.1 Diagrama de Arquitectura Completo**

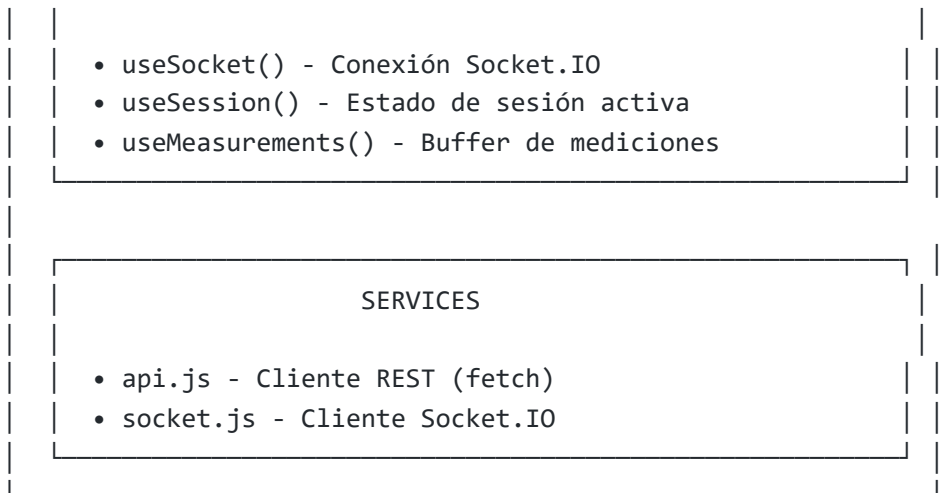






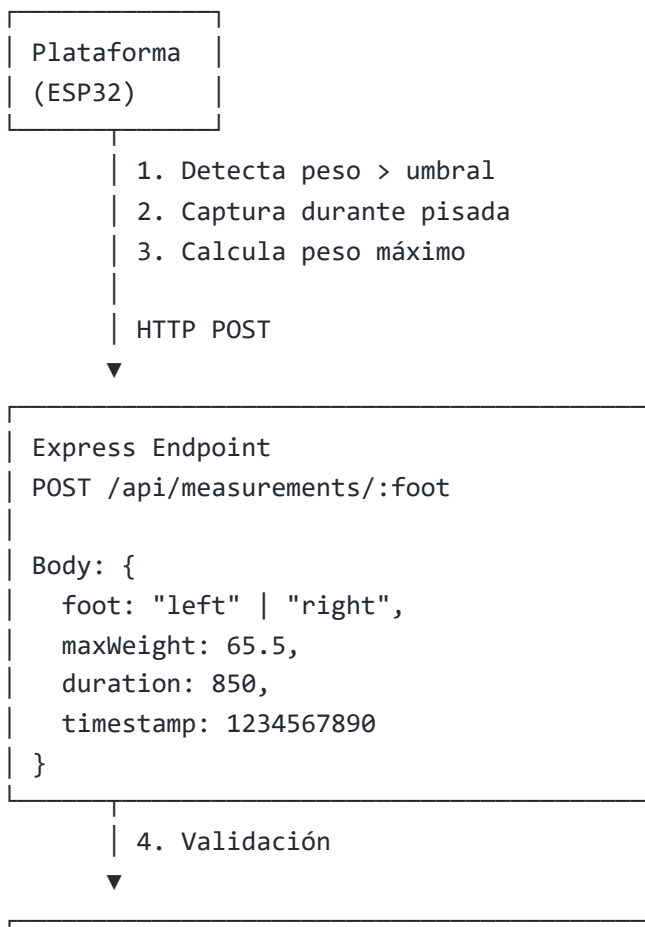
## 3.2 Arquitectura de Componentes Frontend

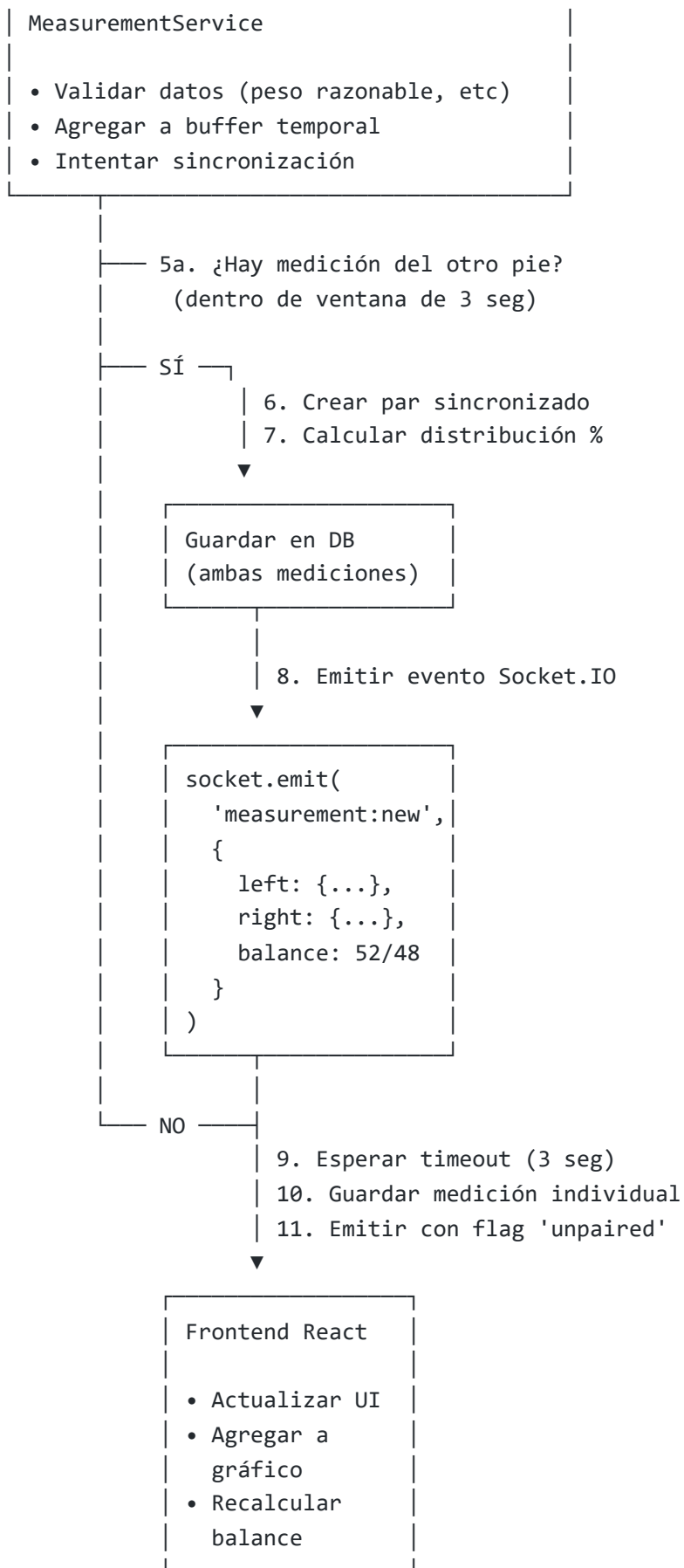




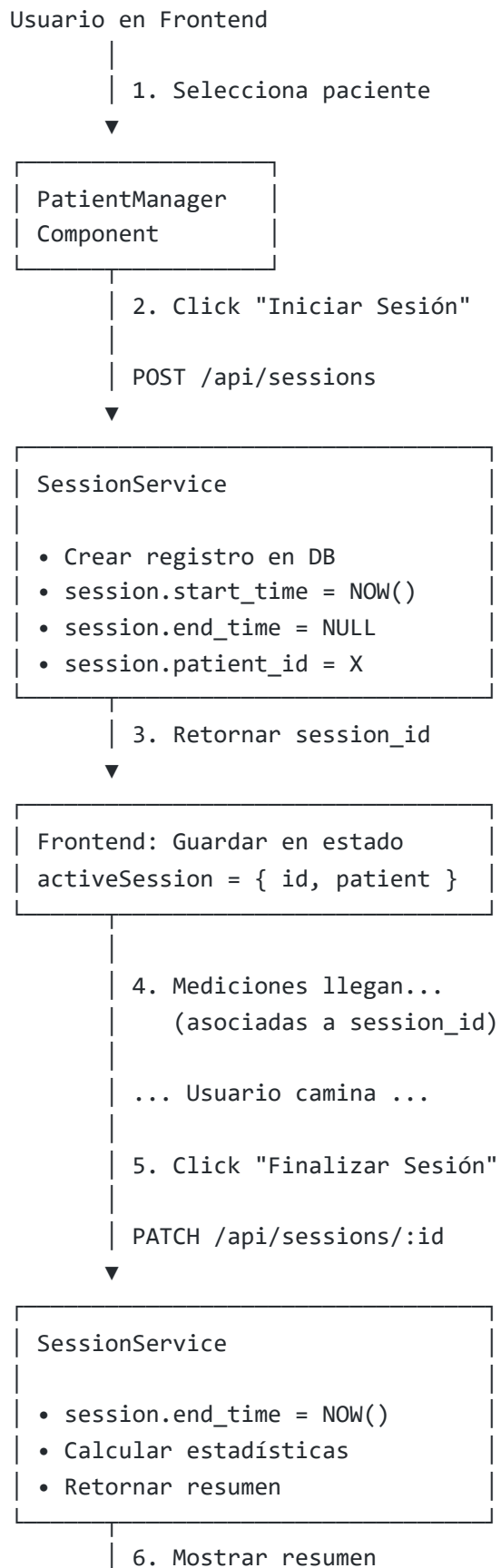
## 4. Flujo de Datos Detallado

### 4.1 Flujo de Captura de Mediciones





## 4.2 Flujo de Gestión de Sesiones







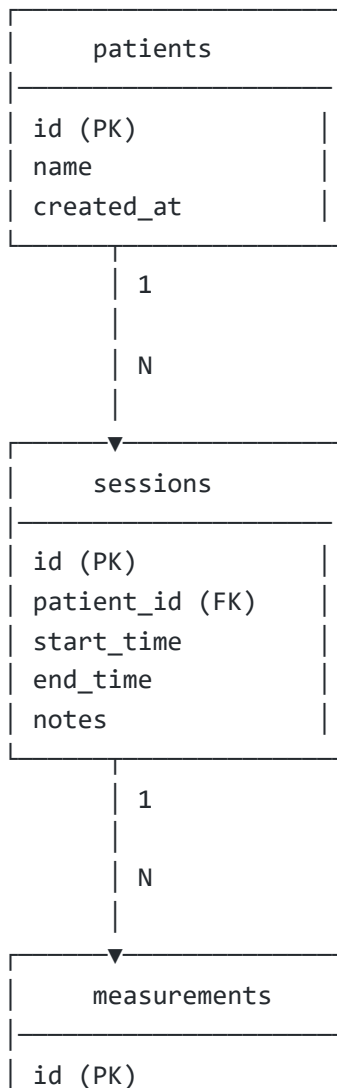
Frontend: Modal con resultados

- Total pisadas
- Balance promedio
- Duración sesión
- Opción exportar CSV

## 5. Esquema de Base de Datos

---

### 5.1 Diagrama Entidad-Relación



session_id (FK)	
foot (left/right)	
weight	
duration	
timestamp	
paired_measurement_id (FK)	

## 5.2 Esquema Prisma

```
// prisma/schema.prisma

datasource db {
  provider = "postgresql" // Cambiar a "sqlite" para Raspberry Pi
  url      = env("DATABASE_URL")
}

generator client {
  provider = "prisma-client-js"
}

model Patient {
  id          Int          @id @default(autoincrement())
  name        String
  createdAt   DateTime     @default(now()) @map("created_at")
  sessions    Session[]

  @@map("patients")
}

model Session {
  id          Int          @id @default(autoincrement())
  patientId   Int          @map("patient_id")
  startTime   DateTime     @map("start_time")
  endTime     DateTime?    @map("end_time")
  notes       String?
  patient     Patient      @relation(fields: [patientId], references
  measurements Measurement[]

  @@map("sessions")
}

model Measurement {
  id          Int          @id @default(autoincrement())
  sessionId   Int          @map("session_id")
```

```

    foot          String          // "left" or "right"
    weight         Float
    duration        Int?           // milliseconds
    timestamp       DateTime
    pairedMeasurementId Int?       @map("paired_measurement_id")
    session         Session        @relation(fields: [sessionId], ref
    pairedMeasurement Measurement? @relation("MeasurementPair", field
    pairOf          Measurement? @relation("MeasurementPair")

    @@index([sessionId])
    @@index([timestamp])
    @@map("measurements")
  }

```

## 6. Estructura de Carpetas y Archivos

```

smartrehabbar/
├── docs/
│   ├── ARQUITECTURA.md          (este documento)
│   ├── PLAN_IMPLEMENTACION.md    (plan de fases)
│   └── API.md                    (documentación de endpoints)
├── backend/
│   ├── src/
│   │   ├── server.js             # Punto de entrada, configuración Exp
│   │   ├── config/
│   │   │   ├── database.js       # Configuración Prisma
│   │   │   └── constants.js      # Constantes del sistema
│   │   ├── routes/
│   │   │   ├── measurements.js   # POST /api/measurements/:foot
│   │   │   ├── patients.js       # CRUD pacientes
│   │   │   └── sessions.js       # CRUD sesiones
│   │   ├── services/
│   │   │   ├── measurementService.js # Lógica sincronización
│   │   │   ├── socketService.js    # Gestión Socket.IO
│   │   │   ├── sessionService.js   # Lógica sesiones
│   │   │   └── exportService.js     # Generación CSV
│   │   ├── middleware/
│   │   │   ├── errorHandler.js    # Manejo de errores
│   │   │   └── validation.js       # Validación de datos
│   │   └── utils/
│   │       └── logger.js           # Logging

```

- ├─ prisma/
  - ├─ schema.prisma # Esquema de base de datos
  - └─ migrations/ # Migraciones
- ├─ tests/
  - ├─ measurements.test.js
  - └─ sessions.test.js
- ├─ package.json
- ├─ .env.example
- └─ .env
- ├─ frontend/
  - ├─ src/
    - ├─ App.jsx # Componente raíz
    - ├─ main.jsx # Punto de entrada
    - ├─ components/
      - ├─ PatientManager.jsx
      - ├─ SessionControl.jsx
      - ├─ Dashboard.jsx
      - ├─ FeetVisualization.jsx
      - ├─ WeightIndicators.jsx
      - ├─ BalanceChart.jsx
      - └─ MeasurementList.jsx
    - ├─ hooks/
      - ├─ useSocket.js # Hook Socket.IO
      - ├─ useSession.js # Hook sesión activa
      - └─ useMeasurements.js
    - ├─ services/
      - ├─ api.js # Cliente REST
      - └─ socket.js # Cliente Socket.IO
    - ├─ utils/
      - └─ calculations.js # Cálculos de balance
    - └─ styles/
      - └─ index.css # Tailwind imports
  - ├─ public/
  - ├─ index.html
  - ├─ package.json
  - ├─ vite.config.js
  - └─ tailwind.config.js
- ├─ simulator/
  - ├─ platformSimulator.js # Simula envío de datos
  - ├─ escenarios.js # Escenarios de prueba
  - └─ package.json
- ├─ .gitignore
- └─ Procfile # Heroku: web: cd backend && npm sta

```
|— package.json
|— README.md
```

```
# Root: scripts de instalación
```

## 7. Configuración para Heroku

---

### 7.1 Variables de Entorno

```
# .env (backend)
DATABASE_URL="postgresql://user:pass@host:5432/dbname" # Heroku lo pr
NODE_ENV="production"
PORT=3000 # Heroku lo asigna dinámicamente
CORS_ORIGIN="*" # En producción, especificar dominio

# Configuración del sistema
WEIGHT_THRESHOLD_START=5
WEIGHT_THRESHOLD_END=3
SYNC_WINDOW_MS=3000
DISPLAY_STEPS=20
BALANCE_GOOD_PERCENT=10
BALANCE_WARNING_PERCENT=20
```

### 7.2 Procfile

```
web: cd backend && npm start
```

### 7.3 package.json (raíz)

```
{
  "name": "smartrehabbar",
  "version": "1.0.0",
  "engines": {
    "node": "18.x",
    "npm": "9.x"
  },
  "scripts": {
    "install-backend": "cd backend && npm install",
    "install-frontend": "cd frontend && npm install",
```

```
    "install-all": "npm run install-backend && npm run install-frontend",
    "build-frontend": "cd frontend && npm run build",
    "heroku-postbuild": "npm run install-all && npm run build-frontend"
  }
}
```

## 7.4 Comandos de Despliegue

```
# Crear app en Heroku
heroku create smartrehabbar

# Agregar addon PostgreSQL
heroku addons:create heroku-postgresql:mini

# Configurar variables de entorno
heroku config:set NODE_ENV=production

# Desplegar
git push heroku main

# Ver logs
heroku logs --tail

# Ejecutar migraciones
heroku run cd backend && npx prisma migrate deploy
```

## 8. Migración a Raspberry Pi

---

### 8.1 Cambios Necesarios

1. **Base de Datos:** Cambiar provider = "postgresql" a provider = "sqlite" en schema.prisma
2. **DATABASE\_URL:** Cambiar a file:./dev.db
3. **Puerto:** Configurar puerto fijo (ej: 3000)
4. **Modo Kiosko:** Configurar Chromium en fullscreen
5. **Autostart:** Configurar PM2 para iniciar al boot

### 8.2 Script de Instalación Raspberry Pi

```
#!/bin/bash
# install-rpi.sh

# Instalar Node.js
curl -fsSL https://deb.nodesource.com/setup_18.x | sudo -E bash -
sudo apt-get install -y nodejs

# Instalar PM2
sudo npm install -g pm2

# Clonar repositorio
git clone <repo-url> smartrehabbar
cd smartrehabbar

# Instalar dependencias
npm run install-all

# Configurar base de datos SQLite
cd backend
cp .env.example .env
# Editar .env: DATABASE_URL="file:./dev.db"
npx prisma migrate deploy
npx prisma generate

# Build frontend
cd ../frontend
npm run build





# Iniciar con PM2
cd ../backend
pm2 start src/server.js --name smartrehabbar
pm2 save
pm2 startup

# Configurar Chromium en modo kiosko
echo "@chromium-browser --kiosk --app=http://localhost:3000" >> ~/.con
```





## 9. Consideraciones de Seguridad

---

### 9.1 Para MVP en Heroku

-  HTTPS automático (Heroku lo provee)
-  Variables de entorno para secretos
-  CORS configurado para permitir cualquier origen (ajustar en producción)
-  Sin autenticación (no necesaria para MVP)

## 9.2 Para Producción en Raspberry Pi

-  Red WiFi local (sin exposición a internet)
-  Base de datos local (SQLite)
-  Considerar autenticación básica si múltiples usuarios
-  Backup periódico de base de datos

# 10. Escalabilidad y Extensiones Futuras

---

## 10.1 Sensores Adicionales

- Arquitectura permite agregar nuevos endpoints fácilmente
- Modificar `MeasurementService` para manejar más tipos de datos
- Extender esquema de base de datos con nuevas tablas

## 10.2 Análisis Avanzado

- Agregar tabla `analytics` para métricas calculadas
- Implementar algoritmos de detección de patrones
- Integración con servicios de ML (TensorFlow.js)

## 10.3 Multi-Dispositivo

- Socket.IO ya soporta múltiples clientes
- Agregar sincronización de estado entre dispositivos
- Implementar roles (terapeuta vs paciente)

Documento creado: 2025-10-03

Versión: 1.0



**Autor:** Equipo SmartRehabBar