

Plan de Implementación del MVP - SmartRehabBar

1. Visión General del MVP










1.1 Objetivo del MVP

Crear un prototipo funcional desplegado en Heroku que permita al equipo validar:





- La arquitectura propuesta
- La experiencia de usuario del dashboard
- La sincronización de mediciones asíncronas
- La viabilidad técnica del concepto

1.2 Alcance del MVP

Incluye:

-  Backend con endpoints para recibir mediciones
-  Base de datos PostgreSQL con esquema completo
-  Frontend React con dashboard en tiempo real
-  Gestión básica de pacientes (nombre)
-  Control de sesiones (inicio/fin)
-  Visualización de balance con colores
-  Gráfico simple de últimas pisadas
-  Simulador de plataformas para testing
-  Despliegue en Heroku

NO Incluye (para versiones futuras):

-  Autenticación de usuarios
-  Exportación de datos (se agregará en Fase 2)
-  Estadísticas avanzadas
-  Configuración de parámetros desde UI

- ✗ Historial detallado de sesiones
- ✗ Modo kiosko (solo para Raspberry Pi)

1.3 Criterios de Éxito del MVP

1. **Funcional:** El sistema recibe, procesa y visualiza mediciones en tiempo real
2. **Sincronización:** Las mediciones de ambos pies se emparejan correctamente
3. **Usabilidad:** El equipo puede crear pacientes, iniciar sesiones y ver resultados
4. **Estabilidad:** El sistema funciona sin errores durante 30 minutos de uso continuo
5. **Despliegue:** Accesible vía URL de Heroku sin configuración adicional

2. Fases de Desarrollo

FASE 1: Configuración Inicial del Proyecto (Estimado: 2-3 horas)

Objetivo: Establecer la estructura base del proyecto y configurar herramientas

Tareas:

1. Crear estructura de carpetas
2. Inicializar repositorio Git
3. Configurar backend (Node.js + Express)
4. Configurar frontend (React + Vite)
5. Configurar Prisma con PostgreSQL
6. Configurar variables de entorno
7. Crear scripts de desarrollo

Dependencias: Ninguna

Entregable: Proyecto inicializado con estructura completa, dependencias instaladas

Validación:

- `npm run dev` ejecuta backend y frontend simultáneamente
- No hay errores de configuración
- Git está configurado con `.gitignore` apropiado

FASE 2: Base de Datos y Modelos (Estimado: 1-2 horas)

Objetivo: Definir y crear el esquema de base de datos

Tareas:

1. Crear `schema.prisma` con modelos (Patient, Session, Measurement)
2. Configurar conexión a PostgreSQL local para desarrollo
3. Crear y ejecutar migraciones
4. Generar Prisma Client
5. Crear seeds para datos de prueba

Dependencias: FASE 1 completada

Entregable: Base de datos funcional con esquema completo

Validación:

- `npx prisma studio` muestra las tablas correctamente
- Seeds crean datos de prueba exitosamente
- Relaciones entre tablas funcionan

FASE 3: Backend - API REST (Estimado: 4-5 horas)

Objetivo: Implementar endpoints para gestión de pacientes, sesiones y mediciones

Tareas:

1. Crear estructura de rutas (routes/)
2. Implementar endpoints de pacientes:
 - `GET /api/patients` (listar)

- POST /api/patients (crear)
- GET /api/patients/:id (obtener uno)
- 3. Implementar endpoints de sesiones:
 - POST /api/sessions (crear/iniciar)
 - PATCH /api/sessions/:id (finalizar)
 - GET /api/sessions/:id (obtener detalles)
- 4. Implementar endpoints de mediciones:
 - POST /api/measurements/left
 - POST /api/measurements/right
 - GET /api/sessions/:id/measurements
- 5. Crear middleware de validación
- 6. Crear middleware de manejo de errores
- 7. Implementar logging básico

Dependencias: FASE 2 completada

Entregable: API REST funcional con todos los endpoints

Validación:

- Probar cada endpoint con Postman/Thunder Client
- Validación de datos funciona correctamente
- Errores se manejan apropiadamente
- Logs se generan correctamente

FASE 4: Backend - Lógica de Negocio (Estimado: 4-5 horas)

Objetivo: Implementar servicios para sincronización y procesamiento de mediciones

Tareas:

1. Crear MeasurementService:
 - Buffer temporal para mediciones
 - Lógica de sincronización (ventana de 3 segundos)
 - Cálculo de distribución de peso
 - Validación de datos (umbrales, rangos)

2. Crear SessionService:
 - Gestión de sesión activa
 - Cálculo de estadísticas de sesión
3. Implementar lógica de emparejamiento:
 - Detectar mediciones dentro de ventana temporal
 - Crear relación paired_measurement_id
 - Manejar mediciones sin pareja (timeout)
4. Crear constantes configurables (config/constants.js)

Dependencias: FASE 3 completada

Entregable: Lógica de negocio completa y testeada

Validación:

- Mediciones se sincronizan correctamente
- Timeout funciona para mediciones sin pareja
- Cálculos de balance son correctos
- Datos se persisten correctamente en DB

FASE 5: Backend - Socket.IO (Estimado: 2-3 horas)

Objetivo: Implementar comunicación en tiempo real con frontend

Tareas:

1. Configurar Socket.IO en servidor Express
2. Crear SocketService:
 - Gestión de conexiones
 - Broadcast de eventos
3. Definir eventos:
 - measurement:new (nueva medición sincronizada)
 - session:started (sesión iniciada)
 - session:ended (sesión finalizada)
4. Integrar SocketService con MeasurementService
5. Implementar manejo de desconexiones

Dependencias: FASE 4 completada

Entregable: Comunicación en tiempo real funcional

Validación:

- Cliente puede conectarse a Socket.IO
- Eventos se emiten correctamente
- Múltiples clientes reciben eventos
- Reconexión automática funciona

FASE 6: Frontend - Configuración y Estructura (Estimado: 2-3 horas)

Objetivo: Configurar React y crear estructura de componentes

Tareas:

1. Configurar Vite con React
2. Instalar y configurar TailwindCSS
3. Instalar dependencias (React Router, Socket.IO Client, Recharts)
4. Crear estructura de carpetas (components, hooks, services)
5. Configurar React Router con rutas básicas
6. Crear componentes base (vacíos):
 - App.jsx
 - PatientManager.jsx
 - Dashboard.jsx
 - SessionControl.jsx
 - FeetVisualization.jsx
 - WeightIndicators.jsx
 - BalanceChart.jsx

Dependencias: FASE 1 completada

Entregable: Frontend configurado con estructura de componentes

Validación:

- Aplicación React se ejecuta sin errores
- TailwindCSS funciona correctamente
- Navegación entre rutas funciona
- Hot reload funciona

FASE 7: Frontend - Servicios y Hooks (Estimado: 3-4 horas)

Objetivo: Implementar capa de servicios y hooks personalizados

Tareas:

1. Crear api.js (cliente REST):
 - Funciones para todos los endpoints
 - Manejo de errores
 - Configuración de base URL
2. Crear socket.js (cliente Socket.IO):
 - Configuración de conexión
 - Funciones helper para eventos
3. Crear hooks personalizados:
 - useSocket() - Gestión de conexión Socket.IO
 - useSession() - Estado de sesión activa
 - useMeasurements() - Buffer de mediciones recientes
4. Crear Context para estado global (paciente activo, sesión)

Dependencias: FASE 6 completada

Entregable: Capa de servicios y hooks funcionales

Validación:

- API client puede hacer peticiones al backend
- Socket.IO se conecta correctamente
- Hooks mantienen estado correctamente
- Context provee datos a componentes

FASE 8: Frontend - Gestión de Pacientes (Estimado: 2-3 horas)

Objetivo: Implementar UI para crear y seleccionar pacientes

Tareas:

1. Implementar PatientManager.jsx:
 - Formulario para crear paciente (solo nombre)
 - Lista de pacientes existentes
 - Selección de paciente activo
 - Indicador visual de paciente seleccionado
2. Integrar con API (GET y POST /api/patients)
3. Implementar validación de formulario
4. Agregar feedback visual (loading, success, error)

Dependencias: FASE 7 completada

Entregable: Gestión de pacientes funcional

Validación:

- Se pueden crear pacientes nuevos
- Lista se actualiza automáticamente
- Selección de paciente funciona
- Validación previene datos inválidos

FASE 9: Frontend - Control de Sesiones (Estimado: 2-3 horas)

Objetivo: Implementar UI para iniciar/finalizar sesiones

Tareas:

1. Implementar SessionControl.jsx:
 - Botón "Iniciar Sesión" (solo si hay paciente seleccionado)
 - Botón "Finalizar Sesión" (solo si hay sesión activa)
 - Indicador de sesión activa

- Timer de duración de sesión
- Contador de pisadas
- 2. Integrar con API (POST y PATCH /api/sessions)
- 3. Actualizar estado global al iniciar/finalizar
- 4. Mostrar modal con resumen al finalizar

Dependencias: FASE 8 completada

Entregable: Control de sesiones funcional

Validación:

- Sesión se inicia correctamente
- Mediciones se asocian a sesión activa
- Sesión se finaliza correctamente
- Resumen muestra datos correctos

FASE 10: Frontend - Dashboard en Tiempo Real (Estimado: 5-6 horas)

Objetivo: Implementar visualización de mediciones en tiempo real

Tareas:

1. Implementar WeightIndicators.jsx:
 - Indicadores numéricos grandes (peso por pie)
 - Unidades (kg)
 - Animación al actualizar
2. Implementar FeetVisualization.jsx:
 - Representación visual de pies (SVG o divs)
 - Codificación por colores:
 - Verde: balance < 10%
 - Amarillo: balance 10-20%
 - Rojo: balance > 20%
 - Porcentaje de distribución
3. Implementar BalanceChart.jsx:
 - Gráfico de líneas (Recharts)

- Mostrar últimas 20 pisadas
 - Dos líneas (pie izquierdo y derecho)
 - Eje X: número de pisada
 - Eje Y: peso (kg)
4. Integrar Dashboard.jsx:
- Componer todos los componentes
 - Layout responsivo
 - Escuchar eventos Socket.IO
 - Actualizar estado en tiempo real

Dependencias: FASE 9 completada

Entregable: Dashboard completo y funcional

Validación:

- Mediciones se muestran en tiempo real
- Colores cambian según balance
- Gráfico se actualiza correctamente
- UI es responsiva y clara

FASE 11: Simulador de Plataformas (Estimado: 2-3 horas)

Objetivo: Crear herramienta para simular envío de mediciones

Tareas:

1. Crear platformSimulator.js:
 - Configuración (URL del servidor, frecuencia)
 - Generación de pesos aleatorios realistas (40-100 kg)
 - Simulación de duración de pisada (500-1200 ms)
 - Envío HTTP POST a endpoints
2. Crear escenarios.js:
 - Escenario 1: Balance perfecto (50/50)
 - Escenario 2: Desequilibrio leve (55/45)
 - Escenario 3: Desequilibrio severo (70/30)
 - Escenario 4: Mediciones asíncronas (delays variables)

3. Crear CLI interactivo:
 - Seleccionar escenario
 - Configurar parámetros
 - Iniciar/detener simulación
4. Agregar logging de envíos

Dependencias: FASE 5 completada (backend con endpoints)

Entregable: Simulador funcional con múltiples escenarios

Validación:

- Simulador envía datos correctamente
- Backend recibe y procesa datos
- Frontend muestra datos simulados
- Diferentes escenarios funcionan

FASE 12: Testing y Validación (Estimado: 3-4 horas)

Objetivo: Asegurar que el MVP funciona correctamente

Tareas:

1. Testing manual:
 - Flujo completo: crear paciente → iniciar sesión → recibir mediciones → finalizar
 - Probar con simulador en diferentes escenarios
 - Verificar sincronización de mediciones
 - Probar con múltiples clientes simultáneos
2. Testing de edge cases:
 - Mediciones sin pareja (timeout)
 - Sesión sin mediciones
 - Reconexión de Socket.IO
 - Datos inválidos
3. Verificar persistencia:
 - Datos se guardan correctamente en DB
 - Relaciones entre tablas son correctas

4. Testing de UI:
 - Responsividad en diferentes tamaños
 - Feedback visual apropiado
 - Manejo de errores en UI
5. Crear checklist de validación

Dependencias: FASES 1-11 completadas

Entregable: Sistema validado y lista de issues encontrados

Validación:

- Todos los flujos principales funcionan
- No hay errores críticos
- Performance es aceptable
- UI es usable

FASE 13: Configuración para Heroku (Estimado: 2-3 horas)

Objetivo: Preparar el proyecto para despliegue en Heroku

Tareas:

1. Crear Procfile
2. Configurar package.json raíz con scripts de build
3. Configurar backend para servir frontend estático
4. Crear .env.example con variables necesarias
5. Configurar Prisma para migraciones en Heroku
6. Actualizar CORS para permitir dominio de Heroku
7. Configurar PORT dinámico (process.env.PORT)
8. Crear documentación de despliegue

Dependencias: FASE 12 completada

Entregable: Proyecto listo para desplegar

Validación:

- Build de producción funciona localmente

- Backend sirve frontend correctamente
- Variables de entorno están documentadas
- Migraciones están listas

FASE 14: Despliegue en Heroku (Estimado: 1-2 horas)

Objetivo: Desplegar el MVP en Heroku y validar funcionamiento

Tareas:

1. Crear aplicación en Heroku
2. Agregar addon PostgreSQL
3. Configurar variables de entorno
4. Conectar repositorio Git
5. Hacer push a Heroku
6. Ejecutar migraciones en Heroku
7. Verificar logs
8. Probar aplicación desplegada
9. Configurar dominio personalizado (opcional)

Dependencias: FASE 13 completada

Entregable: MVP desplegado y accesible públicamente

Validación:

- Aplicación carga correctamente
- Base de datos funciona
- Socket.IO se conecta
- Simulador puede enviar datos a URL de Heroku
- No hay errores en logs

FASE 15: Documentación y Entrega (Estimado: 2-3 horas)

Objetivo: Documentar el MVP para el equipo

Tareas:

1. Crear README.md completo:
 - Descripción del proyecto
 - Instrucciones de instalación local
 - Instrucciones de uso
 - Arquitectura resumida
2. Crear guía de usuario:
 - Cómo crear pacientes
 - Cómo iniciar sesiones
 - Cómo interpretar el dashboard
3. Crear guía para desarrolladores:
 - Estructura del código
 - Cómo agregar nuevas funcionalidades
 - Cómo ejecutar el simulador
4. Documentar API (endpoints, payloads, respuestas)
5. Crear video demo (opcional)
6. Preparar presentación para el equipo

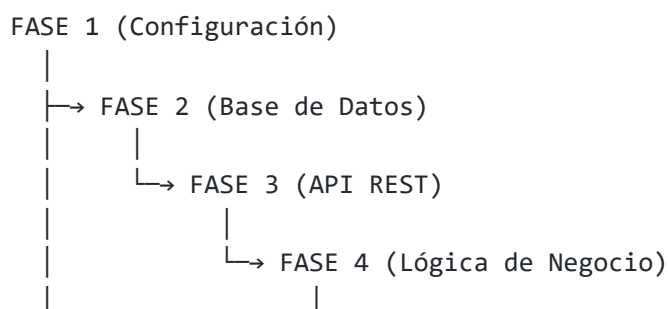
Dependencias: FASE 14 completada

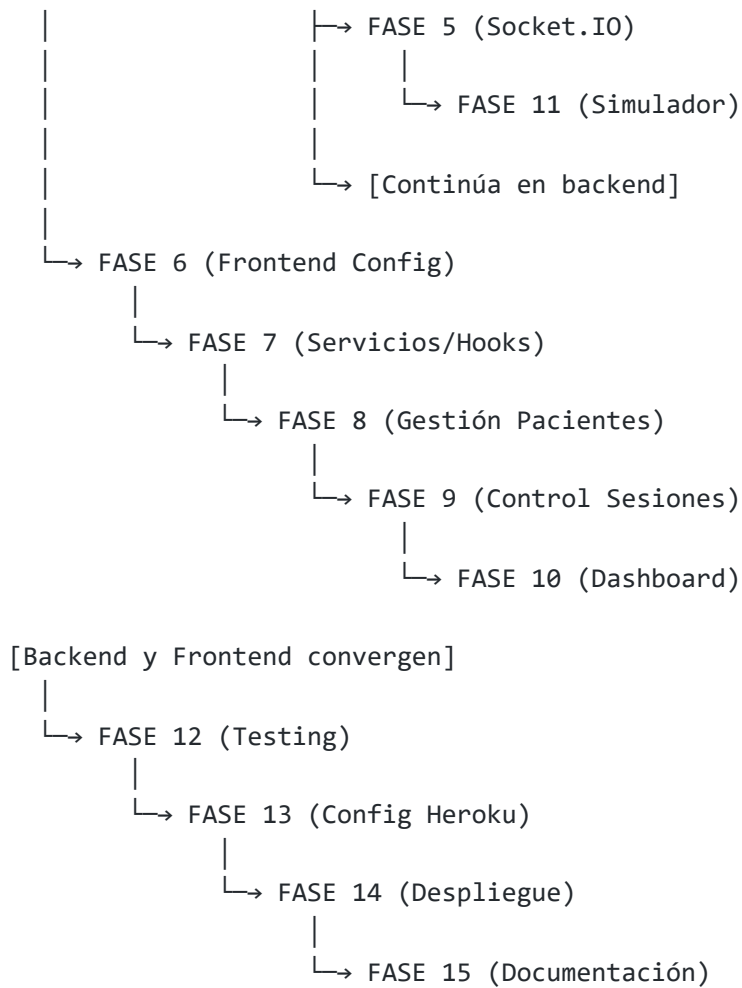
Entregable: Documentación completa y presentación

Validación:

- README es claro y completo
- Equipo puede entender cómo usar el sistema
- Documentación técnica es suficiente

3. Dependencias entre Fases





4. Estrategia de Testing

4.1 Testing Durante Desarrollo

Backend:

- Probar cada endpoint con Thunder Client/Postman
- Verificar respuestas y códigos de estado
- Probar validaciones con datos inválidos
- Verificar persistencia en base de datos

Frontend:

- Desarrollo con React DevTools
- Verificar estado de componentes

- Probar interacciones de usuario
- Verificar actualización en tiempo real

Integración:

- Ejecutar backend y frontend simultáneamente
- Usar simulador para generar datos
- Verificar flujo completo end-to-end

4.2 Checklist de Validación del MVP

Funcionalidad Básica:

- ☐ Se puede crear un paciente nuevo
- ☐ Se puede seleccionar un paciente existente
- ☐ Se puede iniciar una sesión
- ☐ Se pueden recibir mediciones del pie izquierdo
- ☐ Se pueden recibir mediciones del pie derecho
- ☐ Las mediciones se sincronizan correctamente
- ☐ El dashboard se actualiza en tiempo real
- ☐ Se puede finalizar una sesión
- ☐ Los datos persisten en la base de datos

Visualización:

- ☐ Los indicadores numéricos muestran el peso correcto
- ☐ Los colores cambian según el balance
- ☐ El gráfico muestra las últimas pisadas
- ☐ El porcentaje de distribución es correcto

Sincronización:

- ☐ Mediciones dentro de 3 segundos se emparejan
- ☐ Mediciones sin pareja se registran individualmente
- ☐ El balance se calcula correctamente

Robustez:

- ☐ El sistema maneja datos inválidos apropiadamente

- ☐ Los errores se muestran al usuario
- ☐ La reconexión de Socket.IO funciona
- ☐ El sistema funciona con múltiples clientes

Despliegue:

- ☐ La aplicación se despliega en Heroku sin errores
- ☐ La base de datos PostgreSQL funciona
- ☐ Socket.IO funciona en Heroku
- ☐ El simulador puede enviar datos a la URL de Heroku

5. Estimación Total

Fase	Descripción	Estimación
1	Configuración Inicial	2-3 horas
2	Base de Datos	1-2 horas
3	API REST	4-5 horas
4	Lógica de Negocio	4-5 horas
5	Socket.IO	2-3 horas
6	Frontend Config	2-3 horas
7	Servicios/Hooks	3-4 horas
8	Gestión Pacientes	2-3 horas
9	Control Sesiones	2-3 horas
10	Dashboard	5-6 horas
11	Simulador	2-3 horas
12	Testing	3-4 horas
13	Config Heroku	2-3 horas
14	Despliegue	1-2 horas

Fase	Descripción	Estimación
15	Documentación	2-3 horas
TOTAL		39-52 horas

Estimación realista: 5-7 días de trabajo a tiempo completo

6. Próximos Pasos Después del MVP

Una vez validado el MVP con el equipo, las siguientes funcionalidades serían:

Fase 2 (Post-MVP):

1. Exportación de datos a CSV/Excel
2. Historial de sesiones con filtros
3. Estadísticas por paciente
4. Configuración de parámetros desde UI
5. Gráficos avanzados (comparación entre sesiones)

Fase 3 (Producción):

1. Migración a Raspberry Pi
2. Modo kiosko
3. Integración con plataformas reales
4. Backup automático de base de datos
5. Optimizaciones de performance

Documento creado: 2025-10-03

Versión: 1.0

Autor: Equipo SmartRehabBar