Eu preciso que você monte um prd para uma aplicação em react native que será para o seguinte uso:

Este é um app que unifica dois projetos de sistemas embarcados com IOT, leve em conta que ambos os projetos estão sendo montados em ordem de serem apresentados em um simpósito. Os projetos estão melhor explicados nos pdf's anexados.

Nosso app em react native deve seguir todos os comentarios de funções feitos nos pdf's e adicionar mais algumas coisas que você diga que são importantes para isso. O app também deve ter uma funcionalidade de login simples mesmo sem encriptamento de senhas nem nada. Cada um dos projetos deve estar bem separado do outro. Uma vez que são projetos distintos mas unificados em um app só para facilidade de processo.

O backend que queremos fazer é usando o supabase. O envio de dados dos nossos microcontroladores vai ser usando um metodo POST. Ainda nao decidimos como fazer os calculos e codar essa parte, preciso de ajuda com isso também.

Se sentir que esqueci de algo para complementar, complemente, e me avise o que foi adicionado.

Leve em conta que vamos usar mesmo que escrito nos pdfs mqtt, um protocolo POST para envio de dados vindo dos micro controladores, precisamos saber também como fazer os calculos (e onde) para salvar as informacoes, como estruturar a DB etc.

Agora vou montar um PRD completo e profissional para a aplicação React Native que unifica os dois projetos IoT.[1] [2]

PRD - Product Requirements Document

App de Monitoramento IoT Unificado

Visão Geral do Produto

Aplicação móvel em React Native que unifica dois sistemas IoT distintos desenvolvidos para apresentação em simpósio acadêmico: um sistema de monitoramento de consumo de energia elétrica e um sistema de monitoramento de nível de água em reservatórios. A aplicação oferece dashboards interativos, análises históricas, alertas inteligentes e autenticação básica de usuários.[2] [1]

Objetivos do Produto

Primários:

- Consolidar dois projetos IoT independentes em uma única interface mobile amigável e profissional[1] [2]
- Fornecer monitoramento em tempo real de consumo energético e níveis de água através de dashboards interativos[2] [1]
- Facilitar a demonstração dos projetos durante o simpósio SBrT 2025 com interface intuitiva[1] [2]

Secundários:

- Promover consciência sobre consumo de energia e gestão hídrica em ambientes residenciais[2] [1]
- Demonstrar viabilidade de soluções IoT de baixo custo com tecnologias modernas[3] [4]

Usuários-Alvo

- Estudantes e professores do IBMEC-RJ que participarão do simpósio[1] [2]
- Avaliadores e visitantes do evento SBrT 2025[2] [1]
- Residentes interessados em monitoramento residencial inteligente[1] [2]

Arquitetura do Sistema

Stack Tecnológico

Frontend Mobile:

- React Native com Expo para desenvolvimento cross-platform[4] [3]
- TypeScript para type safety[4]
- React Navigation para navegação entre telas[4]
- Supabase JS Client para comunicação com backend[3] [4]
- React Native Chart Kit ou Victory Native para visualizações gráficas[4]
- AsyncStorage para cache local de sessões[4]

Backend:

- Supabase (PostgreSQL + REST API + Auth + Realtime)[3] [4]
- Row Level Security (RLS) para segurança de dados[5] [3]
- Edge Functions (opcional) para processamento de cálculos complexos[3]

Hardware/IoT:

- Projeto 1: ESP8266 + ACS712 (30A) sensor de corrente[1]

- Projeto 2: ESP8266 + HC-SR04 sensor ultrassônico[2]
- Comunicação: HTTP POST via Wi-Fi para API REST do Supabase[6] [7]

Estrutura do Banco de Dados (Supabase/PostgreSQL)

Tabela: users

```
CREATE TABLE users (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  email TEXT UNIQUE NOT NULL,
  password_hash TEXT NOT NULL,
  full_name TEXT,
  created_at TIMESTAMP DEFAULT NOW(),
  updated_at TIMESTAMP DEFAULT NOW()
);
```

Tabela: energy_readings

```
CREATE TABLE energy_readings (
  id BIGSERIAL PRIMARY KEY,
  user_id UUID REFERENCES users(id),
  device_id TEXT NOT NULL,
  timestamp TIMESTAMP DEFAULT NOW(),
  current_rms FLOAT NOT NULL, -- Corrente RMS em Amperes
  voltage FLOAT NOT NULL, -- Tensão (127V ou 220V)
  power_watts FLOAT NOT NULL, -- Potência instantânea calculada
  energy_kwh FLOAT, -- Energia acumulada (calculada)
  appliance_name TEXT -- Nome do aparelho monitorado
);
```

Tabela: water_readings

```
CREATE TABLE water_readings (
  id BIGSERIAL PRIMARY KEY,
  user_id UUID REFERENCES users(id),
  device_id TEXT NOT NULL,
  timestamp TIMESTAMP DEFAULT NOW(),
  distance_cm FLOAT NOT NULL, -- Distância medida pelo sensor
  water_level_percent FLOAT NOT NULL, -- Percentual do nível
  volume_liters FLOAT NOT NULL, -- Volume calculado
  tank_height_cm FLOAT NOT NULL, -- Altura total do tanque
  tank_capacity_liters FLOAT NOT NULL -- Capacidade total
);
```

Tabela: devices

```
CREATE TABLE devices (
  id TEXT PRIMARY KEY, -- MAC address do ESP8266
  user_id UUID REFERENCES users(id),
  device_type TEXT CHECK (device_type IN ('energy', 'water')),
```

```
  device_name TEXT NOT NULL,
  location TEXT,
  is_active BOOLEAN DEFAULT TRUE,
  created_at TIMESTAMP DEFAULT NOW()
);
```

Tabela: alerts

```
CREATE TABLE alerts (
  id BIGSERIAL PRIMARY KEY,
  user_id UUID REFERENCES users(id),
  device_id TEXT REFERENCES devices(id),
  alert_type TEXT NOT NULL, -- 'low_water', 'high_energy', etc.
  message TEXT NOT NULL,
  severity TEXT CHECK (severity IN ('info', 'warning', 'critical')),
  is_read BOOLEAN DEFAULT FALSE,
  created_at TIMESTAMP DEFAULT NOW()
);
```

Integração ESP8266 → Supabase via HTTP POST

Firmware ESP8266 (Arduino C++):

```
// Bibliotecas necessárias
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#include <WiFiClientSecure.h>
#include <ArduinoJson.h>

// Configurações
const char* ssid = "YOUR_WIFI_SSID";
const char* password = "YOUR_WIFI_PASSWORD";
const char* supabaseUrl = "https://YOUR_PROJECT.supabase.co";
const char* supabaseKey = "YOUR_ANON_KEY";
const char* deviceId = "DEVICE_MAC_ADDRESS";

// Envio de dados para Supabase
void sendDataToSupabase(float measurement, String table) {
  if (WiFi.status() == WL_CONNECTED) {
    WiFiClientSecure client;
    client.setInsecure(); // Para testes; use certificado em produção
    HTTPClient http;

    String endpoint = String(supabaseUrl) + "/rest/v1/" + table;
    http.begin(client, endpoint);
    http.addHeader("Content-Type", "application/json");
    http.addHeader("apikey", supabaseKey);
    http.addHeader("Authorization", "Bearer " + String(supabaseKey));

    // Montar JSON payload
    StaticJsonDocument<256> doc;
    doc["device_id"] = deviceId;
    doc["timestamp"] = getISOTimestamp();
```

```
    if (table == "energy_readings") {
      doc["current_rms"] = measurement;
      doc["voltage"] = 127.0;
      doc["power_watts"] = measurement * 127.0;
    } else if (table == "water_readings") {
      doc["distance_cm"] = measurement;
      doc["water_level_percent"] = calculateWaterLevel(measurement);
      doc["volume_liters"] = calculateVolume(measurement);
      doc["tank_height_cm"] = 200.0;
      doc["tank_capacity_liters"] = 1000.0;
    }

    String payload;
    serializeJson(doc, payload);

    int httpCode = http.POST(payload);
    http.end();
  }
}
```

Observações importantes:

- O ESP8266 deve enviar dados a cada 5-10 segundos para monitoramento em tempo real[2] [1]
- Cálculos básicos (RMS, potência, volume) devem ser realizados no microcontrolador para reduzir carga no backend[1] [2]
- Use HTTPS com certificado SSL válido em produção[7]
- Implemente retry logic para falhas de conexão[7]

Processamento de Cálculos

Cálculos no ESP8266 (preferencial):

- Cálculo de corrente RMS a partir de múltiplas amostras do ADC[1]
- Cálculo de potência instantânea (P = V × I)[1]
- Conversão de distância ultrassônica para percentual e volume de água[2]
- Validação básica de dados (range checking)[2] [1]

Cálculos no Backend (Supabase Edge Functions ou Database Functions):

- Acumulação de energia consumida (kWh) através de integração temporal[1]
- Análise de padrões de consumo histórico[2] [1]
- Detecção de anomalias e geração de alertas[2] [1]
- Agregações para dashboards (médias horárias, diárias, mensais)[3]

Exemplo de Database Function para energia acumulada:

```
CREATE OR REPLACE FUNCTION calculate_energy_consumption(
  device_id_param TEXT,
  start_time TIMESTAMP,
  end_time TIMESTAMP
) RETURNS FLOAT AS $$
  SELECT SUM(power_watts * EXTRACT(EPOCH FROM (timestamp - LAG(timestamp) OVER (ORDER BY
  FROM energy_readings
  WHERE device_id = device_id_param
    AND timestamp BETWEEN start_time AND end_time;
$$ LANGUAGE SQL;
```

Funcionalidades do Aplicativo

1. Autenticação e Gerenciamento de Usuários

Tela de Login:

- Campos: email e senha (sem encriptação complexa conforme solicitado)[4] [3]
- Validação básica de formato de email[4]
- Mensagens de erro amigáveis[4]
- Botão "Esqueci minha senha" (opcional)[4]

Tela de Registro:

- Campos: nome completo, email, senha, confirmação de senha[4]
- Armazenamento de hash de senha usando bcrypt no backend[3]
- Criação automática de usuário na tabela users[3]

Gerenciamento de Sessão:

- Uso de AsyncStorage para persistência de token JWT[4]
- Auto-login se token válido existir[4]
- Logout com limpeza de cache local[4]

2. Dashboard Principal (Home)

Layout:

- Cartões separados e visualmente distintos para cada projeto IoT[1] [2]
- Navegação por tabs ou cartões clicáveis[4]
- Indicadores visuais de status (online/offline) dos dispositivos[2] [1]

Conteúdo dos Cartões:

Projeto 1 - Monitoramento de Energia:

- Potência atual em tempo real (Watts)[1]

- Consumo acumulado do dia (kWh)[1]
- Custo estimado (baseado em tarifa configurável)[1]
- Gráfico de consumo nas últimas 24 horas[1]
- Badge de alerta se consumo anormal detectado[1]

Projeto 2 - Monitoramento de Água:

- Nível atual do reservatório (percentual e visual)[2]
- Volume em litros[2]
- Indicador de consumo diário[2]
- Estimativa de tempo até esvaziar (baseado em média de consumo)[2]
- Alertas de nível baixo/crítico[2]

3. Tela Detalhada - Monitoramento de Energia

Visualizações em Tempo Real:

- Gauge circular mostrando potência instantânea[1]
- Medidor de corrente RMS (Amperes)[1]
- Tensão configurada (127V ou 220V)[1]

Análise Histórica:

- Gráfico de linha: consumo por hora nas últimas 24h[1]
- Gráfico de barras: consumo diário da última semana[1]
- Gráfico de pizza: distribuição de consumo por aparelho (se múltiplos sensores)[1]
- Tabela de estatísticas: consumo total, média, pico[1]

Funcionalidades Adicionais:

- Seleção de período customizado para análise[8]
- Exportação de dados em CSV[8]
- Configuração de alertas por threshold de potência[1]
- Cadastro de aparelhos monitorados[1]

4. Tela Detalhada - Monitoramento de Água

Visualizações em Tempo Real:

- Animação de tanque preenchido proporcionalmente ao nível[2]
- Percentual numérico grande e legível[2]
- Volume em litros[2]
- Distância medida pelo sensor (para debug)[2]

Análise Histórica:

- Gráfico de área: variação do nível ao longo do tempo[2]
- Identificação de padrões de consumo (horários de pico)[2]
- Gráfico de tendência: consumo diário da última semana[2]
- Estatísticas: consumo médio diário, tempo de reabastecimento[2]

Funcionalidades Adicionais:

- Configuração de alertas para níveis baixo (20%), crítico (10%) e alto (95%)[2]
- Configuração das dimensões do tanque (altura, capacidade)[2]
- Histórico de alertas recebidos[2]
- Notificações push quando nível crítico[2]

5. Sistema de Alertas e Notificações

Tipos de Alertas:

Energia:

- Consumo acima de threshold configurado[1]
- Pico anormal de potência[1]
- Dispositivo offline[1]

Água:

- Nível baixo (20%)[2]
- Nível crítico (10%)[2]
- Nível muito alto (possível vazamento - 95%+)[2]
- Dispositivo offline[2]

Implementação:

- Notificações push usando Expo Notifications[4]
- Badge de contador na tab de alertas[4]
- Centro de notificações no app com histórico[8]
- Marcação de alertas como lidos[8]

6. Configurações

Perfil do Usuário:

- Edição de nome e email[4]
- Alteração de senha[4]
- Logout[4]

Gerenciamento de Dispositivos:

- Listagem de todos os dispositivos cadastrados[3]
- Adição de novos dispositivos (via QR code ou código manual)[8]
- Edição de nome e localização[8]
- Remoção de dispositivos[8]

Preferências:

- Tarifa de energia (R$/kWh) para cálculo de custos[1]
- Thresholds de alertas customizáveis[1] [2]
- Unidades de medida (Watts/kW, Litros/m³)[8]
- Tema claro/escuro[4]
- Idioma (PT-BR por padrão)[8]

Design e UX

Paleta de Cores

Projeto Energia:

- Primária: Amarelo/Laranja (#FFA500) para representar energia[8]
- Secundária: Azul escuro (#1E3A8A) para contraste[8]
- Alertas: Vermelho (#EF4444) para alto consumo[8]
- Sucesso: Verde (#10B981) para consumo normal[8]

Projeto Água:

- Primária: Azul claro (#3B82F6) para representar água[8]
- Secundária: Ciano (#06B6D4)[8]
- Alertas: Vermelho (#EF4444) para nível baixo, Laranja (#F59E0B) para atenção[8]
- Sucesso: Verde (#10B981) para nível adequado[8]

Componentes Principais

- Cards com sombras sutis para separação visual[4]
- Gráficos interativos com animações suaves[4]
- Botões com feedback tátil (haptic)[4]
- Loading states para todas as operações assíncronas[4]
- Empty states informativos quando sem dados[8]
- Skeleton loaders para melhor UX[4]

Navegação

Bottom Tab Navigator:

- Home (ícone casa)[4]
- Energia (ícone raio)[4]
- Água (ícone gota)[4]
- Alertas (ícone sino com badge)[4]
- Configurações (ícone engrenagem)[4]

Requisitos Não-Funcionais

Performance

- Tempo de carregamento inicial < 3 segundos[8]
- Atualização de dados em tempo real com latência < 2 segundos[1] [2]
- Suporte a mínimo 100 leituras/minuto por dispositivo[3]
- Cache local para visualização offline de dados históricos[9] [4]

Segurança

- Autenticação JWT com refresh tokens[3]
- Row Level Security (RLS) no Supabase para isolamento de dados por usuário[5] [3]
- Validação de entrada no frontend e backend[3]
- Rate limiting na API para prevenir abuse[3]
- HTTPS obrigatório para todas as comunicações[7]

Escalabilidade

- Suporte a múltiplos dispositivos por usuário (energia e água)[1] [2]
- Arquitetura preparada para adição de novos tipos de sensores IoT[8]
- Particionamento de tabelas por data para grandes volumes[3]
- Uso de indexes para queries de histórico otimizadas[5]

Compatibilidade

- iOS 13+ e Android 8+[4]
- Responsivo para tablets[4]
- Compatibilidade com ESP8266 (Wi-Fi 2.4GHz)[1] [2]

Roadmap de Desenvolvimento

Sprint 1 - Fundação (2 semanas)

- Setup do projeto React Native com Expo[4]
- Configuração do Supabase (database schema, RLS policies)[5] [3]
- Implementação de autenticação (login/registro)[3] [4]
- Estrutura de navegação básica[4]

Sprint 2 - Integração IoT (2 semanas)

- Desenvolvimento do firmware ESP8266 para ambos os projetos[2] [1]
- Implementação de HTTP POST para Supabase[6] [7]
- Testes de comunicação e troubleshooting[7]
- Implementação de retry logic e error handling[7]

Sprint 3 - Dashboards (2 semanas)

- Dashboard principal com cards dos projetos[1] [2]
- Tela detalhada de monitoramento de energia[1]
- Tela detalhada de monitoramento de água[2]
- Implementação de gráficos e visualizações[4]

Sprint 4 - Funcionalidades Avançadas (2 semanas)

- Sistema de alertas e notificações push[2] [1] [4]
- Análise histórica e estatísticas[1] [2]
- Configurações e gerenciamento de dispositivos[8]
- Cálculos de energia acumulada e estimativas[2] [1]

Sprint 5 - Polimento e Testes (1 semana)

- Testes end-to-end[8]
- Refinamentos de UX/UI[8]
- Otimização de performance[8]
- Preparação para demonstração no simpósio[1] [2]

Métricas de Sucesso

Métricas Técnicas

- Uptime do sistema > 99% durante o simpósio[8]
- Latência média de atualização < 2 segundos[8]
- Taxa de sucesso de POST dos ESP8266 > 95%[7]
- Zero crashes durante demonstrações[8]

Métricas de Experiência

- Tempo médio para setup de novo dispositivo < 2 minutos[8]
- Satisfação dos avaliadores do simpósio (feedback qualitativo)[2] [1]
- Facilidade de navegação e compreensão das informações[8]

Riscos e Mitigações

Risco	Impacto	Probabilidade	Mitigação
Instabilidade da conexão Wi-Fi durante simpósio	Alto	Média	Buffer local no ESP8266, retry automático, fallback para hotspot [7]
Latência excessiva em queries históricas	Médio	Baixa	Indexes otimizados, cache de agregações, paginação [5]
Problemas de compatibilidade Expo/React Native	Médio	Baixa	Testes frequentes em dispositivos reais iOS/Android [4]
Limite de requisições no plano gratuito Supabase	Alto	Média	Monitoramento de quotas, otimização de polling intervals [3]
Falha de hardware (ESP8266/sensores) durante demo	Alto	Baixa	Redundância de dispositivos, dados mock de backup [1] [2]

Adições e Melhorias Recomendadas

Além das funcionalidades descritas nos PDFs, foram adicionadas as seguintes melhorias para tornar o produto mais robusto :[8]

1. **Sistema de gerenciamento de dispositivos**: Permite cadastrar múltiplos sensores de energia e água, facilitando escalabilidade[8]
2. **Notificações push**: Essencial para alertas em tempo real de níveis críticos e anomalias[4]
3. **Análise preditiva básica**: Estimativa de tempo até esvaziar o reservatório e previsão de custos mensais[8]
4. **Exportação de dados**: Permite exportar histórico para análise externa ou documentação acadêmica[8]
5. **Modo offline**: Cache de dados históricos para visualização sem internet[9] [4]

6. **Tema escuro**: Melhora experiência do usuário em ambientes com pouca luz [4]
7. **Onboarding interativo**: Tutorial inicial para novos usuários entenderem o sistema rapidamente [8]
8. **Database functions**: Processamento de cálculos complexos no backend reduz carga no app [5]
9. **Rate limiting e validação**: Segurança adicional contra abuse e dados inválidos [3]
10. **Estrutura modular**: Separação clara entre projetos permite fácil manutenção e adição de novos módulos IoT [8]

Considerações Finais

Este PRD fornece uma base sólida para o desenvolvimento do aplicativo unificado de monitoramento IoT. A arquitetura proposta com React Native, Supabase e comunicação via HTTP POST é moderna, escalável e adequada para demonstração em ambiente acadêmico. O uso de cálculos híbridos (parte no ESP8266, parte no backend) otimiza performance e confiabilidade do sistema. [7] [5] [3] [8] [1] [2] [4]

⁂

1. SBrT2026_Modelo_da_parte_escrita_para_AP1_AP2_AC_G6__IBM3118_2025_2.pdf
2. SBrT2026_IOT_Based_Smart_Water_Tank_Monitoring_System_G6__IBM3119_2025_2.pdf
3. https://docs.expo.dev/guides/using-supabase/
4. https://supabase.com/docs/guides/getting-started/tutorials/with-expo-react-native
5. https://supabase.com/docs/guides/getting-started/quickstarts/reactjs
6. https://www.youtube.com/watch?v=wI209WK3-VM
7. https://github.com/jhagas/ESPSupabase
8. https://airfocus.com/templates/product-requirements-document/
9. https://supabase.com/blog/react-native-offline-first-watermelon-db
10. https://www.notion.com/templates/category/product-requirements-doc
11. https://www.chatprd.ai/templates
12. https://pm3.com.br/glossario/prd-product-requirements-document/
13. https://www.atlassian.com/agile/product-management/requirements
14. https://www.reddit.com/r/Supabase/comments/15rgsxg/how_to_use_supabase_auth_whilst_using_mondodb_for/
15. https://miro.com/templates/prd/
16. https://productschool.com/blog/product-strategy/product-template-requirements-document-prd
17. https://www.figma.com/community/file/1108838632250762265/prd-product-requirement-doc
18. https://supabase.com/docs/guides/auth/quickstarts/react-native
19. https://github.com/topics/supabase?l=c%2B%2B&o=desc&s=forks
20. https://www.atlassian.com/software/confluence/templates/product-requirements

21. https://www.b4x.com/android/forum/threads/banano-supabase-firebase-alternative-crud-example-with-sse.144817/
22. https://www.notion.com/templates/product-requirement-document-prd