Roteiro de MVP: Integração OBD-II com React Native e Expo

I. Sumário Executivo: O Blueprint para o MVP OBD-II em React Native Expo

Este relatório fornece um caminho técnico direto para o desenvolvimento de um protótipo funcional (MVP) de um aplicativo de monitoramento veicular OBD-II usando React Native e Expo, projetado para ser executado dentro de um cronograma de 30 dias. A estratégia delineada prioriza a velocidade de implementação, a estabilidade e a viabilidade multiplataforma, mitigando os riscos comuns associados à integração de hardware.

- **Pilha Tecnológica Recomendada:** Para um desenvolvimento rápido e confiável do MVP, a arquitetura recomendada é um projeto **React Native Expo utilizando um Cliente de Desenvolvimento Personalizado (Custom Development Client)** para integrar a biblioteca react-native-ble-plx. A comunicação com o dongle ELM327 deve ser realizada via **Bluetooth Low Energy (BLE)**.
- **Justificativa Estratégica:** Esta abordagem equilibra a velocidade de desenvolvimento do fluxo de trabalho gerenciado do Expo com a necessidade de usar módulos nativos robustos e de baixo nível para comunicação de hardware. Garante compatibilidade multiplataforma (iOS/Android) e evita as armadilhas de bibliotecas desatualizadas ou protocolos de conexão complexos, como o Bluetooth Clássico no iOS.[1]
- **Realinhamento do Objetivo Central:** Embora RPM e MAF sejam dados valiosos, os PIDs primários para um cálculo preciso da eficiência de combustível são **Massa de Ar por Tempo (MAF, 0110)** e **Velocidade do Veículo (VSS, 010D)**. Este relatório fornecerá as fórmulas precisas para este cálculo, permitindo uma medição de eficiência mais direta e precisa.[3]
- **Caminho para o Sucesso no Hackathon:** O documento está estruturado para guiar o desenvolvedor desde a configuração do ambiente até um protótipo funcional de streaming de dados. Ele inclui trechos de código prontos para uso em cada etapa crítica: gerenciamento de permissões, escaneamento de dispositivos, conexão, envio de comandos e decodificação de dados.

II. O Fluxo de Trabalho Moderno do Expo:

Desbloqueando o Poder Nativo

Esta seção desmistifica as limitações percebidas do Expo, explicando o fluxo de trabalho de desenvolvimento moderno. É fundamental estabelecer que o acesso a qualquer funcionalidade nativa é possível sem abandonar o ecossistema Expo, uma compreensão crucial para projetos de integração de hardware.

Além do Expo Go: Por que um Cliente de Desenvolvimento Personalizado é Essencial

A percepção de que o uso de módulos nativos exige a "ejeção" de um projeto Expo é um conceito obsoleto.[6] A plataforma evoluiu para acomodar código nativo personalizado de maneira gerenciada.

- **Expo Go vs. Cliente de Desenvolvimento:** O Expo Go é um aplicativo pré-construído disponível nas lojas de aplicativos (App Store, Google Play) que contém um conjunto fixo de módulos nativos do SDK do Expo.[9] Ele não pode ser modificado para incluir bibliotecas de terceiros com código nativo. Bibliotecas como react-native-ble-plx, que são essenciais para a comunicação Bluetooth de baixo nível, contêm código nativo personalizado e, portanto, são incompatíveis com o Expo Go.[11]
- **A Solução:** Um Cliente de Desenvolvimento Personalizado é uma versão pessoal do Expo Go que é construída especificamente para um projeto. Ele inclui todas as dependências nativas únicas do projeto (como react-native-ble-plx) compiladas nele.[9] Isso permite que o desenvolvedor mantenha o fluxo de trabalho rápido e gerenciado do Expo — onde as atualizações de JavaScript são enviadas instantaneamente para o dispositivo — enquanto utiliza qualquer biblioteca nativa disponível no ecossistema React Native.

Guia Passo a Passo: Configurando seu Build de Desenvolvimento com EAS

A criação de um cliente de desenvolvimento é um processo direto usando os Expo Application Services (EAS).

1. **Pré-requisitos:** Instale a CLI do EAS globalmente e faça login na sua conta Expo [9]:
   Bash
   npm install -g eas-cli
   eas login

2. **Construção do Cliente:** No diretório raiz do projeto, execute o comando de construção

do EAS, especificando o perfil de desenvolvimento e a plataforma [14]:
Bash
eas build --profile development --platform android
ou
eas build --profile development --platform ios

3. **Instalação e Execução:** Após a conclusão do build, o EAS fornecerá um link ou um código QR para baixar e instalar o arquivo .apk (Android) ou .app (iOS) em um dispositivo físico. O teste de funcionalidades BLE requer um dispositivo real, pois os simuladores não possuem hardware Bluetooth.[15]
4. **Iniciando o Servidor de Desenvolvimento:** Para conectar o cliente de desenvolvimento ao seu ambiente local, inicie o servidor Metro com a flag --dev-client [9]:
Bash
npx expo start --dev-client

Configurando o app.json: Integrando Código Nativo com Config Plugins

Os Config Plugins do Expo são a ponte entre a configuração do seu projeto em JavaScript (app.json ou app.config.js) e os arquivos de projeto nativos (Info.plist, AndroidManifest.xml). Eles automatizam a modificação de configurações nativas durante o processo de pré-construção.[6]
A biblioteca react-native-ble-plx utiliza um config plugin para adicionar automaticamente as permissões de Bluetooth necessárias e os modos de execução em segundo plano aos projetos nativos, simplificando drasticamente a configuração.[11]

III. Arquitetando a Conexão: Seleção de Biblioteca e Protocolo

A escolha da tecnologia de comunicação é a decisão mais crítica para o sucesso deste projeto. Uma escolha inadequada pode levar a bloqueios de plataforma ou complexidade desnecessária, comprometendo o cronograma do hackathon.

A Camada de Comunicação: Por que react-native-ble-plx é a Escolha Profissional

O ecossistema de bibliotecas OBD-II para React Native está repleto de pacotes desatualizados, específicos para Android e sem manutenção.[18] Muitos são apenas invólucros de uma antiga biblioteca Java e não recebem atualizações há anos, representando um risco significativo para um projeto com prazo definido.[18]

- **Robustez e Manutenção:** react-native-ble-plx é uma biblioteca de baixo nível para comunicação BLE, bem mantida e ativamente suportada pela comunidade.[1] É considerada o padrão de fato para BLE em React Native.
- **Controle e Depuração:** Ao utilizar uma biblioteca de nível mais baixo, o desenvolvedor tem controle total sobre o protocolo de comunicação. Quando surgem problemas, é possível depurar os comandos e respostas brutos, o que é impossível com uma biblioteca de alto nível que funciona como uma "caixa preta".[23]
- **Compatibilidade com Expo:** A biblioteca possui um config plugin dedicado, tornando sua integração com o fluxo de trabalho do cliente de desenvolvimento perfeitamente alinhada com as melhores práticas do Expo.[11]

Caminhos Alternativos e Por Que Evitá-los (Para o MVP)

- **Bibliotecas OBD-II de Alto Nível:** Pacotes como react-native-bluetooth-obd-manager [25] são atraentes por abstrair os PIDs. No entanto, eles introduzem uma camada adicional de possíveis bugs, podem carecer de manutenção ou suporte multiplataforma, tornando-os arriscados para um projeto de curto prazo.
- **Bluetooth Clássico (Serial Port Profile):** Muitos dongles ELM327 de baixo custo usam Bluetooth Clássico. Embora existam bibliotecas como react-native-bluetooth-classic [2], elas são problemáticas. No iOS, a comunicação com dispositivos Bluetooth não-BLE requer participação no programa MFi da Apple, o que é inviável para um hackathon.[2] A compatibilidade com Android é funcional, mas essa rota sacrifica a viabilidade multiplataforma.
- **Dongles Wi-Fi:** Esses adaptadores criam seu próprio ponto de acesso Wi-Fi.[28] O aplicativo precisa desconectar-se programaticamente da rede Wi-Fi do usuário, conectar-se à rede do dongle e, em seguida, comunicar-se via soquetes TCP.[29] Isso adiciona uma complexidade significativa: gerenciamento do estado da rede, tratamento da perda de conectividade com a internet e implementação da comunicação TCP.[30] Esta não é uma solução "rápida e simples".

Instalando e Configurando o react-native-ble-plx

1. **Instalação:** Use o instalador do Expo para garantir a compatibilidade de versão:
   Bash

npx expo install react-native-ble-plx

2. **Configuração:** Adicione o config plugin ao seu arquivo app.json com as mensagens de permissão necessárias.[11]
JSON
{
  "expo": {
    "plugins":,
        "bluetoothAlwaysPermission": "Permitir que $(PRODUCT_NAME) se conecte ao seu scanner OBD-II."
      }
    ]
  }
}

IV. Implementação Central: De Permissões a Fluxo de Dados

Esta seção detalha o código e a lógica que formam o núcleo do aplicativo. A implementação é estruturada em módulos e encapsulada em um hook React personalizado para reutilização e clareza.

Módulo 1: Dominando as Permissões

As permissões de Bluetooth são complexas e dependentes da plataforma. O Android 12 (API 31) introduziu novas permissões (BLUETOOTH_SCAN, BLUETOOTH_CONNECT) e tornou a permissão de localização opcional em certas condições.[11] Versões mais antigas do Android exigem
ACCESS_FINE_LOCATION [14], enquanto o iOS requer a chave
NSBluetoothAlwaysUsageDescription no arquivo Info.plist.[34]
A solução é uma função unificada que lida com todos os casos, usando a API PermissionsAndroid do React Native.

JavaScript

import { PermissionsAndroid, Platform } from 'react-native';

```
export async function requestBluetoothPermissions(): Promise<boolean> {
  if (Platform.OS === 'ios') {
    // As permissões do iOS são tratadas pelo Info.plist via config plugin
    return true;
  }

  if (Platform.OS === 'android') {
    // Para Android 12+ (API 31+)
    if (Platform.Version >= 31) {
      const result = await PermissionsAndroid.requestMultiple();

      return (
        result === PermissionsAndroid.RESULTS.GRANTED &&
        result === PermissionsAndroid.RESULTS.GRANTED
      );
    }

    // Para Android 11 e inferior (API < 31)
    const locationPermissionResult = await PermissionsAndroid.request(
      PermissionsAndroid.PERMISSIONS.ACCESS_FINE_LOCATION,
      {
        title: 'Permissão de Localização',
        message: 'O aplicativo precisa de acesso à localização para descobrir dispositivos
Bluetooth.',
        buttonPositive: 'OK',
      }
    );

    return locationPermissionResult === PermissionsAndroid.RESULTS.GRANTED;
  }

  return false;
}
```

Módulo 2: Descoberta e Conexão de Dispositivos

O fluxo de conexão BLE padrão é: Escanear -> Conectar -> Descobrir Serviços e Características.[23] Esta lógica pode ser encapsulada em um hook React personalizado, como useOBD().

JavaScript

```
// Exemplo de trecho dentro de um hook useOBD()
import { BleManager, Device } from 'react-native-ble-plx';
import { useState, useMemo } from 'react';

const bleManager = useMemo(() => new BleManager(),);

//... (estados para dispositivos, dispositivo conectado, etc.)

const scanForDevices = () => {
  bleManager.startDeviceScan(null, null, (error, device) => {
    if (error) {
      console.error(error);
      return;
    }
    if (device && (device.name?.includes('OBD') |

| device.name?.includes('VLINK'))) {
      // Adiciona o dispositivo a uma lista de estado
      // setDiscoveredDevices(prev => [...prev, device]);
    }
  });
};

const connectToDevice = async (device: Device) => {
  try {
    bleManager.stopDeviceScan();
    const connectedDevice = await device.connect();
    // setConnectedDevice(connectedDevice);
    await connectedDevice.discoverAllServicesAndCharacteristics();
  } catch (e) {
    console.error('Falha ao conectar:', e);
  }
};
```

Após a conexão, é *essencial* chamar discoverAllServicesAndCharacteristics() para que a biblioteca possa mapear os serviços e características do dispositivo, que são necessários para a comunicação.[36]

Módulo 3: Comunicando-se com o ELM327

A comunicação com o ELM327 é baseada em texto. Comandos são enviados como strings e as respostas são recebidas como strings. Todos os comandos devem ser terminados com um caractere de retorno de carro (\r).[23]
Antes de solicitar PIDs de dados, o dongle deve ser inicializado com uma sequência de comandos AT. Esta é uma etapa crítica frequentemente omitida por iniciantes.[21]

- ATZ: Reseta o dispositivo.
- ATE0: Desativa o eco de comandos.
- ATL0: Desativa a quebra de linha.
- ATSP0: Define o protocolo de comunicação para automático.

JavaScript

```
// Exemplo de trecho para enviar comando e receber dados
import { Base64 } from 'js-base64';

// Assumindo que você já encontrou os UUIDs do serviço e das características
const OBD_SERVICE_UUID = '0000fff0-0000-1000-8000-00805f9b34fb';
const OBD_WRITE_CHAR_UUID = '0000fff2-0000-1000-8000-00805f9b34fb';
const OBD_NOTIFY_CHAR_UUID = '0000fff1-0000-1000-8000-00805f9b34fb';

const sendCommand = async (device: Device, command: string) => {
  const commandWithCarriageReturn = `${command}\r`;
  const base64Command = Base64.encode(commandWithCarriageReturn);

  await device.writeCharacteristicWithResponseForService(
    OBD_SERVICE_UUID,
    OBD_WRITE_CHAR_UUID,
    base64Command
  );
};

const setupNotifications = (device: Device) => {
  device.monitorCharacteristicForService(
    OBD_SERVICE_UUID,
    OBD_NOTIFY_CHAR_UUID,
    (error, characteristic) => {
      if (error) {
        console.error(error);
        return;
      }
      if (characteristic?.value) {
```

```
      const rawData = Base64.decode(characteristic.value);
      console.log('Dados recebidos:', rawData);
      // Aqui você processaria a resposta 'rawData'
    }
  }
 );
};
```

A resposta do ELM327 terminará com um caractere > para indicar que está pronto para o próximo comando.[23]

V. Decodificando Dados do Veículo: Traduzindo Hexadecimal para Valores Legíveis

Esta seção fornece o conhecimento de domínio específico necessário para interpretar as respostas do OBD-II, abordando diretamente o objetivo principal do aplicativo.

Os PIDs para Eficiência de Combustível

Conforme mencionado, a fórmula padrão para consumo instantâneo de combustível utiliza MAF e VSS, não RPM.[3]

- **PID Primário: Massa de Ar por Tempo (MAF)**
  - **Comando:** 0110
  - **Resposta:** 2 bytes (A, B)
  - **Fórmula:** $((A * 256) + B) / 100$ (resultado em gramas/segundo) [39]
- **PID Primário: Velocidade do Veículo (VSS)**
  - **Comando:** 010D
  - **Resposta:** 1 byte (A)
  - **Fórmula:** A (resultado em km/h) [39]
- **PID Secundário: Rotações por Minuto (RPM)**
  - **Comando:** 010C
  - **Resposta:** 2 bytes (A, B)
  - **Fórmula:** $((A * 256) + B) / 4$ (resultado em RPM) [41]

Kit de Ferramentas de Decodificação em JavaScript

É crucial ter uma função robusta para analisar a resposta bruta do ELM327. A resposta típica para um comando 010C pode ser 41 0C 0F A0 >. A função de análise deve extrair os bytes de

dados (0F A0) e aplicar a fórmula correta.

JavaScript

```
function parseOBDResponse(response: string): string | null {
  // Remove espaços, o caractere '>' e possíveis ecos de comando.
  const cleanedResponse = response.replace(/\s/g, '').replace('>', '');

  // A resposta válida para um PID de modo 01 começa com '41'.
  // Ex: 410C0FA0. '41' é a resposta para o modo '01', '0C' é o PID ecoado.
  if (cleanedResponse.startsWith('41')) {
    // Extrai os dados após o modo e o PID (4 caracteres)
    return cleanedResponse.substring(4);
  }

  return null;
}

function decodeRPM(hexData: string): number {
  const byteA = parseInt(hexData.substring(0, 2), 16);
  const byteB = parseInt(hexData.substring(2, 4), 16);
  return ((byteA * 256) + byteB) / 4;
}

function decodeMAF(hexData: string): number {
  const byteA = parseInt(hexData.substring(0, 2), 16);
  const byteB = parseInt(hexData.substring(2, 4), 16);
  return ((byteA * 256) + byteB) / 100;
}
```

Tratamento de Erros: "NO DATA" e PIDs Não Suportados

Nem todos os veículos suportam todos os PIDs.[43] Se um PID solicitado não for suportado, o ELM327 responderá com a mensagem "NO DATA".[45] A lógica de análise deve tratar essa resposta de forma elegante, sem travar o aplicativo, retornando
null ou um estado de erro específico.

Tabela 1: PIDs Essenciais para o MVP				

Nome do PID	Comando	Bytes de Resposta	Fórmula (JavaScript)	Unidade
Engine RPM	010C	2 (A, B)	((A * 256) + B) / 4	RPM
Vehicle Speed	010D	1 (A)	A	km/h
Mass Air Flow	0110	2 (A, B)	((A * 256) + B) / 100	g/s

VI. Desempenho e Estabilidade: Construindo um Protótipo Confiável

Esta seção aborda as realidades práticas da execução de um aplicativo intensivo em dados, fornecendo as melhores práticas para garantir que o MVP seja utilizável e não esgote a bateria do dispositivo do usuário.

Otimizando a Frequência de Polling

Existe um trade-off entre a resolução dos dados e o consumo de recursos. Consultar PIDs com muita frequência (ex: 10 vezes por segundo) aumenta o consumo de bateria e o uso da CPU, enquanto uma consulta muito lenta pode perder flutuações importantes nos dados.

- **Estratégia para o MVP:** Recomenda-se um intervalo de polling equilibrado, como uma vez por segundo (1000 ms).[46] Isso pode ser implementado com um setInterval ou um loop setTimeout recursivo que envia sequencialmente os comandos para os PIDs desejados.

Gerenciamento do Estado da Conexão e Reconexão

Conexões Bluetooth são inerentemente instáveis e podem cair.[47] O aplicativo deve ser resiliente a essas falhas.

- **Implementação:** Utilize o listener bleManager.onDeviceDisconnected() para detectar desconexões. Ao detectar uma queda, o estado da interface do usuário deve ser atualizado (ex: exibindo uma mensagem de "Desconectado") e quaisquer intervalos de polling devem ser interrompidos para evitar erros. Para o MVP, um botão para o usuário acionar manualmente a reconexão é uma solução simples e eficaz.

Considerações sobre Coleta de Dados em Segundo Plano (Além do

MVP)

Um caso de uso comum para tal aplicativo é registrar uma viagem inteira, o que exige que o aplicativo colete dados enquanto está em segundo plano.

- **As Ferramentas:** O ecossistema Expo fornece as bibliotecas expo-task-manager e a mais recente expo-background-task para essa finalidade.[49]
- **O Desafio:** A operação contínua de BLE em segundo plano é um tópico significativamente complexo, especialmente no iOS, que impõe restrições severas para economizar bateria.[49] O sistema operacional pode suspender ou encerrar o aplicativo a qualquer momento. Manter uma conexão persistente e um fluxo de dados em segundo plano é um recurso avançado, não adequado para um MVP de 30 dias. Tentar implementá-lo sem uma base sólida provavelmente resultaria em falha. Portanto, essa funcionalidade deve ser claramente delineada como um recurso pós-MVP.

VII. Conclusão: Seu Caminho a Seguir

Este relatório forneceu um roteiro abrangente e de baixo risco para a prototipagem de um aplicativo de monitoramento OBD-II em React Native com Expo. Ao focar em uma pilha de tecnologia estável e em um escopo de MVP bem definido, o sucesso dentro do prazo de 30 dias é altamente alcançável.

Resumo do Roteiro de Implementação

A estratégia recomendada segue uma progressão lógica:

1. **Configuração do Ambiente:** Estabelecer o projeto Expo com um Cliente de Desenvolvimento Personalizado.
2. **Integração da Biblioteca:** Instalar e configurar react-native-ble-plx via config plugin.
3. **Lógica de Conexão:** Implementar o fluxo de permissões, escaneamento, conexão e descoberta de serviços.
4. **Comunicação:** Enviar a sequência de inicialização de comandos AT, seguida por um loop de polling para os PIDs 0110 (MAF) e 010D (VSS).
5. **Processamento de Dados:** Decodificar as respostas hexadecimais e tratar os casos de erro, como PIDs não suportados.

Checklist Final do MVP

- [ ] Adquiriu um dongle ELM327 compatível com **BLE**?
- [ ] O projeto Expo está configurado com um **Cliente de Desenvolvimento**

Personalizado?
- [ ] O arquivo app.json inclui o **config plugin do react-native-ble-plx**?
- [ ] A função unificada de **solicitação de permissões** foi implementada?
- [ ] O aplicativo consegue **escanear, encontrar e conectar-se** com sucesso ao dongle OBD-II?
- [ ] A sequência inicial de **comandos AT** está sendo enviada para preparar o dongle?
- [ ] O aplicativo consegue solicitar e **decodificar MAF (0110) e VSS (010D)** com sucesso?
- [ ] O aplicativo lida de forma elegante com **quedas de conexão e PIDs não suportados**?

Próximos Passos (Pós-Hackathon)

Com o protótipo funcional em mãos, as melhorias futuras podem incluir:
- Armazenamento local dos dados coletados usando AsyncStorage ou um banco de dados como o WatermelonDB.
- Visualização dos dados em tempo real com gráficos.
- Implementação da interface do usuário para exibir a eficiência de combustível calculada.
- Exploração de uma implementação robusta de registro em segundo plano.

Referências citadas

1. React native for automotive | Star Insights, acessado em setembro 19, 2025, https://star.global/posts/react-native-for-automotive/
2. React Native Bluetooth Classic - Ken Davidson, acessado em setembro 19, 2025, https://kenjdavidson.com/react-native-bluetooth-classic/
3. Monitoring Fuel Consumption on your Vehicle in ... - Windmill Software, acessado em setembro 19, 2025, https://www.windmill.co.uk/obdii.pdf
4. Fuel Economy Calculations from OBD Data, acessado em setembro 19, 2025, https://hemdata.com/products/dawn/enews/fuel-economy-obd/
5. What is the best way to get fuel consumption (MPG) using OBD2 parameters?, acessado em setembro 19, 2025, https://stackoverflow.com/questions/17170646/what-is-the-best-way-to-get-fuel-consumption-mpg-using-obd2-parameters
6. Everything works with expo : r/reactnative - Reddit, acessado em setembro 19, 2025, https://www.reddit.com/r/reactnative/comments/1dpe8f2/everything_works_with_expo/
7. So You Want to Build a Bluetooth App with React Native and Expo | by Adrian Carolli, acessado em setembro 19, 2025, https://blog.expo.dev/so-you-want-to-build-a-bluetooth-app-with-react-native-and-expo-6ea6a31a151d

8. Developing an app that uses Bluetooth. Can't I use expo? : r/reactnative - Reddit, acessado em setembro 19, 2025, https://www.reddit.com/r/reactnative/comments/1ikxoth/developing_an_app_that_uses_bluetooth_cant_i_use/
9. Running Your Expo App on a Real Device for Testing - DEV Community, acessado em setembro 19, 2025, https://dev.to/andrewchaa/running-your-expo-app-on-a-real-device-for-testing-3ci
10. Expo managed workflow in 2021. Part 1: The preset Expo runtime | by Brent Vatne, acessado em setembro 19, 2025, https://blog.expo.dev/expo-managed-workflow-in-2021-5b887bbf7dbb
11. react-native-ble-plx - npm, acessado em setembro 19, 2025, https://www.npmjs.com/package/react-native-ble-plx
12. Migrating a React Native App to Expo - Headway.io, acessado em setembro 19, 2025, https://www.headway.io/blog/migrating-a-react-native-app-to-expo
13. Clarity in documentation about the managed workflow compatibility · hyochan expo-iap · Discussion #19 - GitHub, acessado em setembro 19, 2025, https://github.com/hyochan/expo-iap/discussions/19
14. React Native Expo App: Gateway to Connect & Communicate BLE - Spritle Software, acessado em setembro 19, 2025, https://www.spritle.com/blog/react-native-expo-app-gateway-to-connect-and-communicate-with-ble-devices/
15. Bluetooth BLE Integration in React Native Expo (New Architecture, iOS & Android). - Medium, acessado em setembro 19, 2025, https://medium.com/@chinweikemichaelchinonso/bluetooth-ble-integration-in-react-native-expo-new-architecture-ios-android-5c0100960979
16. How to build a bluetooth low energy scanner using react native | by Gregg Larson - ITNEXT, acessado em setembro 19, 2025, https://itnext.io/how-to-build-a-bluetooth-low-energy-scanner-using-react-native-4e30bf877d7
17. Expo · dotintent/react-native-ble-plx Wiki · GitHub, acessado em setembro 19, 2025, https://github.com/dotintent/react-native-ble-plx/wiki/Expo
18. JetBridge-io/react-native-obd2: React-native OBD-II reader - GitHub, acessado em setembro 19, 2025, https://github.com/JetBridge-io/react-native-obd2
19. keywords:obd2 - npm search, acessado em setembro 19, 2025, https://www.npmjs.com/search?q=keywords:obd2
20. OBD2 - npms, acessado em setembro 19, 2025, https://npms.io/search?q=OBD2
21. How to implement an app using OBD2 (ELM327) like Torque in android? - Stack Overflow, acessado em setembro 19, 2025, https://stackoverflow.com/questions/35764056/how-to-implement-an-app-using-obd2-elm327-like-torque-in-android
22. williamkom/react-native-OBD2 - GitHub, acessado em setembro 19, 2025, https://github.com/williamkom/react-native-OBD2
23. Reading a peripheral device for the first time (OBDII with react-native ..., acessado em setembro 19, 2025,

https://medium.com/@adrian.b.p03/reading-a-peripheral-device-for-the-first-time-obdii-with-react-native-f50228c0b9f0
24. How to build a Bluetooth Low Energy powered Expo app, acessado em setembro 19, 2025, https://expo.dev/blog/how-to-build-a-bluetooth-low-energy-powered-expo-app
25. react-native-bluetooth-obd-manager - npm, acessado em setembro 19, 2025, https://www.npmjs.com/package/react-native-bluetooth-obd-manager?activeTab=code
26. rakshitbharat - NPM, acessado em setembro 19, 2025, https://www.npmjs.com/~rakshitbharat
27. Are there any working Bluetooth Classic Packages for React Native?? (either Bare Workflow or Expo) : r/reactnative - Reddit, acessado em setembro 19, 2025, https://www.reddit.com/r/reactnative/comments/1idmz93/are_there_any_working_bluetooth_classic_packages/
28. IOT WiFi setup : r/reactnative - Reddit, acessado em setembro 19, 2025, https://www.reddit.com/r/reactnative/comments/eunnu3/iot_wifi_setup/
29. How to communicate with OBD II using bluetooth or wifi using iphone - Stack Overflow, acessado em setembro 19, 2025, https://stackoverflow.com/questions/24738134/how-to-communicate-with-obd-ii-using-bluetooth-or-wifi-using-iphone
30. React Native WiFi Connect - DEV Community, acessado em setembro 19, 2025, https://dev.to/raguram90/react-native-wifi-connect-209c
31. Expo + IoT: Device provisioning with HTTPS via WiFi, acessado em setembro 19, 2025, https://expo.dev/blog/expo-iot-device-provisioning-with-https-via-wifi
32. Bluetooth permissions | Connectivity - Android Developers, acessado em setembro 19, 2025, https://developer.android.com/develop/connectivity/bluetooth/bt-permissions
33. PermissionsAndroid - React Native, acessado em setembro 19, 2025, https://reactnative.dev/docs/permissionsandroid
34. react-native-bluetooth-state-manager - NPM, acessado em setembro 19, 2025, https://www.npmjs.com/package/react-native-bluetooth-state-manager
35. react-native-permissions - NPM, acessado em setembro 19, 2025, https://www.npmjs.com/package/react-native-permissions
36. javascript - React Native ble-plx writing and reading/monitoring ..., acessado em setembro 19, 2025, https://stackoverflow.com/questions/76955201/react-native-ble-plx-writing-and-reading-monitoring-characteristics-of-obdii-ada
37. react-native-ble-plx 3.3.0 | Documentation, acessado em setembro 19, 2025, https://dotintent.github.io/react-native-ble-plx/
38. React Native BLE with OBD devices - Stack Overflow, acessado em setembro 19, 2025, https://stackoverflow.com/questions/69292693/react-native-ble-with-obd-devices
39. OBD-II PIDs PDF | PDF | Engines | Vehicle Technology - Scribd, acessado em setembro 19, 2025, https://es.scribd.com/document/399698761/OBD-II-PIDs-pdf

40. OBD-II PIDs | PDF | Vehicle Technology - Scribd, acessado em setembro 19, 2025, https://www.scribd.com/document/561052884/OBD-II-PIDs
41. Reading Real-Time Data | OBD Solutions, acessado em setembro 19, 2025, https://www.obdsol.com/knowledgebase/obd-software-development/reading-real-time-data/
42. Custom sensors (PIDs) - Car Scanner ELM OBD2, acessado em setembro 19, 2025, https://www.carscanner.info/custompids/
43. Why OBD2 Scanner Shows "No Communication"? - AZscanners, acessado em setembro 19, 2025, https://www.azscanners.com.au/obd-scan-tool-no-communication-connection-troubles/
44. Thread: Having trouble with PIDs showing up as unsupported, acessado em setembro 19, 2025, https://forum.efilive.com/showthread.php?21332-Having-trouble-with-PIDs-showing-up-as-unsupported
45. obd ii - Getting "No data" response from BOD II in android - Stack Overflow, acessado em setembro 19, 2025, https://stackoverflow.com/questions/64133444/getting-no-data-response-from-bod-ii-in-android
46. Node package for communication with Bluetooth OBD connectors. - GitHub, acessado em setembro 19, 2025, https://github.com/EricSmekens/node-bluetooth-obd
47. React Native and Bluetooth: Connecting Devices to Your Mobile App - CloudDevs, acessado em setembro 19, 2025, https://clouddevs.com/react-native/bluetooth/
48. Mastering Bluetooth Low Energy Integration with React Native, acessado em setembro 19, 2025, https://reactnativeexpert.com/blog/mastering-bluetooth-low-energy-integration-with-react-native/
49. Expo BackgroundFetch - Expo Documentation, acessado em setembro 19, 2025, https://docs.expo.dev/versions/latest/sdk/background-fetch/
50. Expo TaskManager - Expo Documentation, acessado em setembro 19, 2025, https://docs.expo.dev/versions/latest/sdk/task-manager/
51. Domain-Driven & Test-Driven React Native: Expo SDK 53 Background Task Case Study | by Mehmet Talha Irmak | Medium, acessado em setembro 19, 2025, https://medium.com/@mehmetirmaakk/domain-driven-test-driven-react-native-expo-sdk-53-background-task-case-study-81366290bf96
52. Background mode (iOS) · dotintent/react-native-ble-plx Wiki - GitHub, acessado em setembro 19, 2025, https://github.com/dotintent/react-native-ble-plx/wiki/Background-mode-(iOS)