Documentação de Integração: Sistema de Controle de Motor IoT

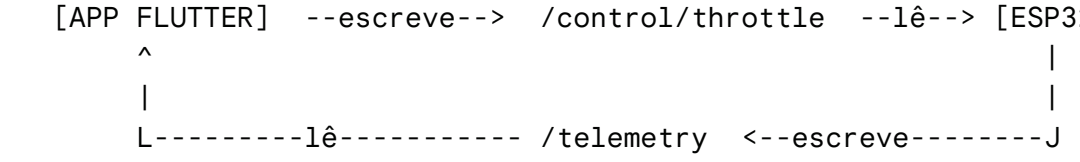
Stack: Flutter (Web/Mobile) + Firebase Realtime Database + ESP32 (C++)

1. Visão Geral da Arquitetura

O sistema utiliza uma arquitetura desacoplada baseada em nuvem. O Aplicativo (Frontend) e o Motor (Hardware) **não se comunicam diretamente**. Ambos utilizam o **Firebase Realtime Database** como intermediário.

Fluxo de Dados

1. **Comando (App -> Motor):** O usuário "Piloto" move a manete no App. O App escreve o valor (0.0 a 1.0) no Firebase. O ESP32 lê este valor e ajusta o PWM.
2. **Telemetria (Motor -> App):** O ESP32 lê os sensores (Hall/Corrente), calcula RPM/Potência e escreve no Firebase. O App lê estes dados e atualiza os gráficos para todos os espectadores.
3. **Fila (Lógica de Servidor/Cliente):** O gerenciamento de quem é o piloto é feito puramente no software (App/Firebase), o hardware não precisa saber sobre a fila.



2. Estrutura do Banco de Dados (Firebase)

Para a integração funcionar, o ESP32 deve ler e escrever nos caminhos exatos abaixo.

A. Controle (Downlink)

Caminho: `/control`

- `throttle` (float): Valor entre `0.0` (parado) e `1.0` (velocidade máxima).
  - *Exemplo:* `0.75`

B. Telemetria (Uplink)

Caminho: `/telemetry` O hardware deve escrever um objeto JSON neste caminho a cada ~200ms.

- `rpm` (float): Rotações por minuto medidas.
- `power` (float): Potência estimada em Watts.
- `efficiency` (float): Eficiência estimada em %.
- `timestamp` (long/int): Timestamp do sistema (millis ou epoch) para debugging de latência.

Exemplo de JSON esperado:

```
{
  "rpm": 120.5,
  "power": 2.1,
  "efficiency": 88.5,
  "timestamp": 1763570418
}
```

3. Implementação no Hardware (ESP32)

Dependências Necessárias

No Arduino IDE ou PlatformIO, instalar:

1. **Firebase Arduino Client Library for ESP8266 and ESP32** (por Mobizt).

Configuração de Segurança (`secrets.h`)

Para segurança, as credenciais **não** devem ficar no código principal. Crie um arquivo `secrets.h` ao lado do `.ino` :

```
#ifndef SECRETS_H
#define SECRETS_H

#define SECRET_WIFI_SSID "SEU_WIFI"
#define SECRET_WIFI_PASS "SUA_SENHA"
#define SECRET_API_KEY "AIzaSyAYSUlD7BDVGbYgyCwpjQxfrdScphcOEJ4" // Chave Web do Firebase
#define SECRET_DATABASE_URL "[https://motor-iot-project-default-rtdb.firebaseio.com/](https://motor-iot-p

#endif
```

Snippet Principal de Comunicação

O código abaixo ilustra como o ESP32 deve interagir com as rotas definidas.

```
// LER COMANDO (Dentro do loop)
if (Firebase.RTDB.getFloat(&fbdo, "/control/throttle")) {
  float target = fbdo.floatData();
```

```
  // Aplica PWM no motor...
}

// ENVIAR TELEMETRIA (A cada 200ms)
FirebaseJson json;
json.set("rpm", sensor_rpm);
json.set("power", calculo_potencia);
Firebase.RTDB.setJSON(&fbdo, "/telemetry", &json);
```

4. Lógica da Fila de Espera (Queue System)

O sistema de fila foi implementado para garantir acesso exclusivo ao motor.

1. **Entrada:** Ao entrar, o usuário recebe um `ID` com timestamp e é gravado em `/queue/{user_id}`.
2. **Prioridade:** A ordem é determinada pelo campo `joinedAt` (quem chegou primeiro).
3. **Sessão:**
   - A sessão dura **45 segundos**.
   - O tempo começa a contar apenas quando o usuário se torna o **#1 da fila** (campo `startedAt` é criado nesse momento).
4. **Timeout:**
   - O próprio aplicativo do usuário monitora o tempo. Ao chegar em 0, o app envia um comando de saída e zera o acelerador.
   - Se o usuário fechar a aba, o comando `onDisconnect().remove()` do Firebase garante que ele saia da fila para não travar os outros.

5. Modo de Simulação ("Host Mode")

Importante para a equipe de Hardware:

O aplicativo possui um sistema de **Failover/Simulação**.

- **Comportamento:** Se o usuário for o Piloto (Motorista), o aplicativo começa a calcular uma física simulada e **escreve na pasta** `/telemetry` para que os espectadores vejam algo acontecendo, caso o ESP32 esteja desligado.
- **Conflito Potencial:** Quando o Hardware real estiver pronto e conectado, o ESP32 escreverá em `/telemetry` ao mesmo tempo que o App.
- **Solução:** O hardware real sempre sobrescreverá os dados do App devido à frequência de envio, mas para a versão final de produção, recomenda-se desativar a flag `_isHost` no `data_service.dart` do Flutter.