

Plano de Migração: MQTT para API REST







Sistema de Configuração de Telas - AutoTech HMI Display v2

Resumo Executivo





Escopo da Mudança

Este plano detalha a migração do sistema de carregamento de configurações de telas do protocolo **MQTT** para **API REST**, mantendo todos os outros aspectos do sistema (comandos de relés, telemetria, etc.) utilizando MQTT.

Benefícios da Migração

-  **Performance:** Carregamento mais rápido de configurações (HTTP vs MQTT)
-  **Flexibilidade:** Possibilidade de paginação para grandes configurações
-  **Consistência:** APIs REST são mais previsíveis que MQTT para dados estruturados
-  **Cache:** HTTP permite implementar cache HTTP nativo
-  **Debug:** Fácil teste com ferramentas HTTP (curl, Postman, etc.)
-  **Escalabilidade:** Reduz carga no broker MQTT

Riscos Identificados

-  **Memória RAM:** HTTPClient adiciona ~10-15KB de uso de RAM
-  **Dependência:** Necessita conectividade HTTP além de MQTT
-  **Latência:** Possível latência maior para configurações pequenas
-  **Compatibilidade:** Período de transição pode gerar instabilidade

Timeline Estimado

- **Desenvolvimento:** 12-16 horas
- **Testes:** 4-8 horas
- **Implementação:** 2-4 horas
- **Total:** 18-28 horas

Análise Técnica Atual

Arquivos Envolvidos na Configuração MQTT

Arquivos Principais - Configuração

Arquivo	Função	Linhas Relevantes	Modificação
src/communication/ConfigReceiver.cpp	Recebe config via MQTT	L90-230	SUBSTITUIR

Arquivo	Função	Linhas Relevantes	Modificação
include/communication/ConfigReceiver.h	Interface MQTT config	L14-55	REFATORAR
src/core/ConfigManager.cpp	Gerencia config recebida	L17-61	MANTER
include/core/ConfigManager.h	Interface ConfigManager	L18-56	MANTER
src/main.cpp	Inicialização e setup	L141-195	MODIFICAR

Arquivos Secundários - Uso da Config

Arquivo	Função	Modificação
src/ui/ScreenManager.cpp	Constrói UI a partir da config	MANTER
src/ui/ScreenFactory.cpp	Cria telas dinâmicas	MANTER
src/core/MQTTClient.cpp	Cliente MQTT (outros usos)	MANTER

Fluxo Atual MQTT (Diagrama)



Estrutura de Dados Atual

Tópicos MQTT Utilizados

```
// ConfigReceiver.cpp - L21-25
String configTopic = "autotech/" + deviceId + "/config";           //
Device specific
String requestTopic = "autotech/gateway/config/request";           //
Request config
String responseTopic = "autotech/gateway/config/response";         //
Receive config
String updateTopic = "autotech/config/update";                     // Hot-
reload
```

Formato JSON Atual (MQTT)

```
{
  "config": {
    "version": "2.0.0",
    "screens": {...},
    "devices": {...},
    "presets": {...}
  }
}
```

Dependências Atuais

```
# platformio.ini - L19-25
lib_deps =
  bodmer/TFT_eSPI@2.5.0
  lvgl/lvgl@8.3.11
  bblanchon/ArduinoJson@7.0.2
  knolleary/PubSubClient@2.8      # MQTT only
  https://github.com/PaulStoffregen/XPT2046_Touchscreen.git
  WiFi
```

Arquitetura Proposta

Novo Fluxo com API REST

1. ESP32 Liga
 - ↓
2. Conecta WiFi
 - ↓
3. Conecta MQTT (comandos e telemetria)
 - ↓
4. ScreenApiClient::loadConfiguration()
 - ↓

5. HTTP GET /api/screens
↓
6. HTTP GET /api/screens/{screen_id}/items (para cada tela)
↓
7. ConfigManager::loadConfig() (mesmo formato)
↓
8. ScreenManager::buildFromConfig()
↓
9. Interface construída ✅

HTTP Client Implementation

```
class ScreenApiClient {  
private:  
    HTTPClient httpClient;  
    String baseUrl;  
    String deviceId;  
    unsigned long cacheExpiry;  
  
public:  
    bool loadConfiguration();  
    bool loadScreens();  
    bool loadScreenItems(int screenId);  
    bool validateConnection();  
    void setCacheTimeout(unsigned long timeout);  
};
```

Estratégia de Cache

```
struct CachedConfig {  
    String jsonData;  
    unsigned long timestamp;  
    String etag;  
    bool isValid() { return (millis() - timestamp) < CACHE_DURATION; }  
};
```

Fallback/Recovery Strategy

1. **Primário:** API REST
2. **Fallback 1:** Cache local em SPIFFS/LittleFS
3. **Fallback 2:** Configuração padrão hardcoded mínima
4. **Recovery:** Retry automático com backoff exponencial

Coexistência MQTT/REST

```
// Manter MQTT para:
- Comandos de relés: "autotech/relay_board_1/command"
- Status: "autotech/hmi_display_1/status"
- Telemetria: "autotech/hmi_display_1/telemetry"
- Hot-reload: "autotech/config/update" (trigger para recarregar via API)

// Migrar para REST:
- Carregamento inicial de configuração
- Lista de telas
- Itens das telas
```



Mudanças Necessárias

1. Dependências (platformio.ini)

```
lib_deps =
  bodmer/TFT_eSPI@^2.5.0
  lvgl/lvgl@^8.3.11
  bblanchon/ArduinoJson@^7.0.2
  knolleary/PubSubClient@^2.8
+  HTTPClient                # ESP32 built-in HTTP client
  https://github.com/PaulStoffregen/XPT2046_Touchscreen.git
  WiFi

+build_flags =
+  -D ENABLE_API_CONFIG=1      # Feature flag
+  -D API_BASE_URL="\http://10.0.10.100:8081\"
+  -D API_TIMEOUT_MS=5000
```

2. Novo Arquivo: `include/communication/ScreenApiClient.h`

```
#ifndef SCREEN_API_CLIENT_H
#define SCREEN_API_CLIENT_H

#include <Arduino.h>
#include <HTTPClient.h>
#include <ArduinoJson.h>
#include <WiFi.h>

class ScreenApiClient {
private:
  HTTPClient httpClient;
  String baseUrl;
  String deviceId;

  // Cache
  String cachedScreensJson;
```

```

    unsigned long cacheTimestamp;
    static const unsigned long CACHE_DURATION_MS = 300000; // 5 min

    // Internal methods
    bool makeRequest(const String& endpoint, JsonDocument& response);
    String buildScreensJson(JsonArray screens);
    bool isCacheValid();
    void saveToCache(const String& data);
    String loadFromCache();

public:
    ScreenApiClient(const String& baseUrl, const String& deviceId);
    ~ScreenApiClient();

    // Main API
    bool loadConfiguration(JsonDocument& config);
    bool testConnection();
    void clearCache();

    // Low-level API access
    bool getScreens(JsonArray& screens);
    bool getScreenItems(int screenId, JsonArray& items);

    // Error handling
    String getLastError() const { return lastError; }
    int getLastHttpCode() const { return lastHttpCode; }

private:
    String lastError;
    int lastHttpCode;
};

#endif

```

3. Novo Arquivo: `src/communication/ScreenApiClient.cpp`

```

#include "communication/ScreenApiClient.h"
#include "core/Logger.h"

extern Logger* logger;

ScreenApiClient::ScreenApiClient(const String& baseUrl, const String&
deviceId)
    : baseUrl(baseUrl), deviceId(deviceId), cacheTimestamp(0),
lastHttpCode(0) {

    httpClient.setTimeout(API_TIMEOUT_MS);
    httpClient.setUserAgent("AutoTech-HMI-v2.0.0");
}

bool ScreenApiClient::loadConfiguration(JsonDocument& config) {

```

```
        logger->info("Loading configuration from API...");

        // Try cache first
        if (isCacheValid()) {
            logger->info("Using cached configuration");
            DeserializationError error = deserializeJson(config,
cachedScreensJson);
            return !error;
        }

        // Load from API
        JsonArray screens;
        if (!getScreens(screens)) {
            logger->error("Failed to load screens from API");
            return false;
        }

        // Load items for each screen
        for (JsonObject screen : screens) {
            int screenId = screen["id"];
            JsonArray items;

            if (getScreenItems(screenId, items)) {
                screen["screen_items"] = items;
            }
        }

        // Build final config format
        config["screens"] = screens;
        config["version"] = "2.0.0";

        // Cache the result
        String jsonStr;
        serializeJson(config, jsonStr);
        saveToCache(jsonStr);

        logger->info("Configuration loaded successfully from API");
        return true;
    }

    bool ScreenApiClient::getScreens(JsonArray& screens) {
        JsonDocument response;
        if (!makeRequest("/api/screens", response)) {
            return false;
        }

        if (response.is<JsonArray>()) {
            screens = response.as<JsonArray>();
            return true;
        }

        lastError = "Invalid response format for screens";
        return false;
    }
```

```
bool ScreenApiClient::getScreenItems(int screenId, JsonArray& items) {
    String endpoint = "/api/screens/" + String(screenId) + "/items";
    JsonDocument response;

    if (!makeRequest(endpoint, response)) {
        return false;
    }

    if (response.is<JsonArray>()) {
        items = response.as<JsonArray>();
        return true;
    }

    return false;
}

bool ScreenApiClient::makeRequest(const String& endpoint, JsonDocument&
response) {
    String url = baseUrl + endpoint;
    logger->debug("API Request: " + url);

    httpClient.begin(url);
    httpClient.addHeader("Accept", "application/json");

    int httpCode = httpClient.GET();
    lastHttpCode = httpCode;

    if (httpCode == HTTP_CODE_OK) {
        String payload = httpClient.getString();
        DeserializationError error = deserializeJson(response, payload);

        if (!error) {
            logger->debug("API Response OK, size: " +
String(payload.length()));
            httpClient.end();
            return true;
        } else {
            lastError = "JSON parsing failed: " + String(error.c_str());
        }
    } else {
        lastError = "HTTP Error: " + String(httpCode);
        logger->error("API Request failed: " + lastError);
    }

    httpClient.end();
    return false;
}

bool ScreenApiClient::testConnection() {
    JsonDocument response;
    return makeRequest("/api/screens", response);
}
```



```
// Cache implementation
bool ScreenApiClient::isCacheValid() {
    return (cachedScreensJson.length() > 0) &&
        ((millis() - cacheTimestamp) < CACHE_DURATION_MS);
}

void ScreenApiClient::saveToCache(const String& data) {
    cachedScreensJson = data;
    cacheTimestamp = millis();
}

String ScreenApiClient::loadFromCache() {
    return isCacheValid() ? cachedScreensJson : "";
}
```

4. Refatoração: `include/communication/ConfigReceiver.h`

```
class ConfigReceiver {
private:
    MQTTClient* mqttClient;
    ConfigManager* configManager;
+   ScreenApiClient* apiClient;           // NEW

-   String configTopic;                  // Remove config-specific topics
-   String requestTopic;
-   String responseTopic;
    String updateTopic;                  // Keep for hot-reload triggers

-   bool waitingForConfig;               // Remove MQTT-specific state
-   unsigned long requestTime;

    std::function<void()> onConfigUpdateCallback;

public:
-   ConfigReceiver(MQTTClient* mqtt, ConfigManager* config);
+   ConfigReceiver(MQTTClient* mqtt, ConfigManager* config,
+   ScreenApiClient* api);

-   void requestConfig();                // Replace with loadFromApi()
+   bool loadFromApi();
+   bool testApiConnection();
}
```

5. Modificação: `src/communication/ConfigReceiver.cpp`

```
-ConfigReceiver::ConfigReceiver(MQTTClient* mqtt, ConfigManager* config)
+ConfigReceiver::ConfigReceiver(MQTTClient* mqtt, ConfigManager* config,
+ScreenApiClient* api)
-    : mqttClient(mqtt), configManager(config), waitingForConfig(false),
  requestTime(0),
```

```

+   : mqttClient(mqtt), configManager(config), apiClient(api),
      onConfigUpdateCallback(nullptr) {

    instance = this;

-
-   // Setup topics - REMOVE CONFIG TOPICS
-   String deviceId = mqttClient->getDeviceId();
-   configTopic = "autotech/" + deviceId + "/config";
-   requestTopic = "autotech/gateway/config/request";
-   responseTopic = "autotech/gateway/config/response";
+
+   // Keep only hot-reload topic
    updateTopic = "autotech/config/update";
}

void ConfigReceiver::begin() {
    logger->info("ConfigReceiver: Starting...");

-   // Remove config-specific subscriptions
-   mqttClient->subscribe(configTopic, onConfigReceived);
-   mqttClient->subscribe(responseTopic, onConfigReceived);

    // Keep only hot-reload subscription
    mqttClient->subscribe(updateTopic, onConfigUpdate);
    logger->info("ConfigReceiver: Hot reload enabled on " + updateTopic);
}

-void ConfigReceiver::requestConfig() {
+bool ConfigReceiver::loadFromApi() {
-   // Remove MQTT request logic
+   if (!apiClient) {
+       logger->error("API client not initialized");
+       return false;
+   }
+
+   JsonDocument config;
+   if (apiClient->loadConfiguration(config)) {
+       return configManager->loadConfig(serializeJson(config));
+   } else {
+       logger->error("Failed to load config from API: " + apiClient-
>getLastError());
+       return false;
+   }
}

// Keep hot-reload for MQTT triggers
void ConfigReceiver::handleConfigUpdate(const String& payload) {
    JsonDocument doc;
    DeserializationError error = deserializeJson(doc, payload);

    if (!error && doc["command"].as<String>() == "reload") {
        logger->info("Hot reload triggered via MQTT, reloading from
API...");
        if (loadFromApi() && onConfigUpdateCallback) {

```

```

        onConfigUpdateCallback();
    }
}
}

```

6. Modificação: `src/main.cpp`

```

// Add includes
#include "communication/ScreenApiClient.h"

// Global components
+ScreenApiClient* screenApiClient = nullptr;

void setupMQTT() {
+ // Initialize API client first
+ screenApiClient = new ScreenApiClient(API_BASE_URL, DEVICE_ID);
+
+ // Test API connectivity
+ if (!screenApiClient->testConnection()) {
+     logger->warning("API not reachable, will use fallback");
+ }

    mqttClient = new MQTTClient(DEVICE_ID, MQTT_BROKER, MQTT_PORT);
- configReceiver = new ConfigReceiver(mqttClient, configManager);
+ configReceiver = new ConfigReceiver(mqttClient, configManager,
screenApiClient);

    // ... rest of MQTT setup

    // Replace config request
- configReceiver->requestConfig();
+ if (!configReceiver->loadFromApi()) {
+     logger->error("Failed to load configuration from API");
+     // TODO: Implement fallback strategy
+ }
}

```

Implementação Step-by-Step

Fase 1: Preparação (2-3 horas)

1. Adicionar dependências HTTP

- Modificar `platformio.ini`
- Adicionar feature flags

2. Criar estrutura base

- `ScreenApiClient.h`

- `ScreenApiClient.cpp` (apenas esqueleto)

3. Testes básicos

- Compilação
- Conectividade HTTP básica

Fase 2: Implementação Core (4-6 horas)

1. Implementar `ScreenApiClient`

- Métodos de requisição HTTP
- Parsing de respostas JSON
- Error handling

2. Integração com `ConfigReceiver`

- Refatorar construtor
- Implementar `loadFromApi()`
- Manter hot-reload

3. Modificar `main.cpp`

- Inicialização do API client
- Fluxo de carregamento

Fase 3: Cache e Otimização (3-4 horas)

1. Implementar cache

- Cache em memória
- Validação de expiração
- Cache persistente (opcional)

2. Estratégia de fallback

- Configuração padrão
- Recovery automático
- Retry logic

3. Otimizações

- Compressão HTTP (gzip)
- Timeout configurável
- Connection pooling

Fase 4: Testes e Validação (3-4 horas)

1. Testes funcionais

- Carregamento de configuração
- Hot-reload via MQTT
- Fallback scenarios

2. Testes de performance

- Tempo de carregamento
- Uso de memória
- Stability testing

3. Testes de integração

- API disponível/indisponível
- Configurações grandes/pequenas
- Múltiplos dispositivos

Compatibilidade

Manter MQTT Para

-  **Comandos de relés:** `autotech/relay_board_1/command`
-  **Status de sistema:** `autotech/hmi_display_1/status`
-  **Telemetria:** `autotech/hmi_display_1/telemetry`
-  **Hot-reload trigger:** `autotech/config/update`
-  **Emergency commands:** `autotech/system/emergency_stop`

Migrar para API REST

-  **Lista de telas:** `GET /api/screens`
-  **Itens de tela:** `GET /api/screens/{id}/items`
-  **Configuração inicial:** Combinação das APIs acima

Período de Transição

```
// Feature flag para rollback
#ifdef ENABLE_MQTT_CONFIG_FALLBACK
    if (!configReceiver->loadFromApi()) {
        logger->warning("API failed, falling back to MQTT");
        configReceiver->requestConfigMqtt(); // Keep old method
    }
#endif
```

Backward Compatibility

- Manter código MQTT como fallback por 30 dias
- Feature flag para desabilitar API se necessário
- Logs detalhados para debug durante transição

Otimizações

Compressão de Dados

```
// HTTP headers para compressão
httpClient.addHeader("Accept-Encoding", "gzip, deflate");

// Parsing de response comprimida
if (httpClient.header("Content-Encoding") == "gzip") {
    // Decompress response
}
```

Paginação (Para Futuro)

```
// Se configuração ficar muito grande (>20KB)
bool ScreenApiClient::getScreensPaginated(int page, int pageSize,
JsonArray& screens) {
    String endpoint = "/api/screens?page=" + String(page) + "&size=" +
String(pageSize);
    // Implementation...
}
```

Cache Inteligente

```
struct ConfigCache {
    String etag;           // HTTP ETag para validação
    String lastModified;   // HTTP Last-Modified
    String data;           // JSON data
    unsigned long expiry;  // Timestamp de expiração

    bool needsRefresh() {
        return millis() > expiry;
    }
};
```

Redução de Memória

```
// Streaming JSON parsing para configurações grandes
class StreamingJsonParser {
public:
    bool parseScreens(Stream& stream, std::function<void(JsonObject)>
onScreen);
    bool parseItems(Stream& stream, std::function<void(JsonObject)>
onItem);
};
```



Unit Tests

```
// tests/test_screen_api_client.cpp
TEST(ScreenApiClient, LoadConfiguration) {
    MockHTTPServer server;
    server.expectGET("/api/screens").andReturn(200, "[...]");

    ScreenApiClient client("http://localhost:8080", "test_device");
    JsonDocument config;

    EXPECT_TRUE(client.loadConfiguration(config));
    EXPECT_EQ(config["version"], "2.0.0");
}

TEST(ScreenApiClient, HandleHttpError) {
    MockHTTPServer server;
    server.expectGET("/api/screens").andReturn(404, "Not Found");

    ScreenApiClient client("http://localhost:8080", "test_device");
    JsonDocument config;

    EXPECT_FALSE(client.loadConfiguration(config));
    EXPECT_EQ(client.getLastHttpCode(), 404);
}
```

Integration Tests

```
// tests/test_config_integration.cpp
TEST(ConfigIntegration, FullFlow) {
    // Start mock API server
    MockAPIServer server;
    server.addScreens({screen1, screen2});
    server.start();

    // Initialize system
    ConfigManager configManager;
    ScreenApiClient apiClient("http://localhost:8081", "test");
    ConfigReceiver receiver(nullptr, &configManager, &apiClient);

    // Test full flow
    EXPECT_TRUE(receiver.loadFromApi());
    EXPECT_TRUE(configManager.hasConfig());
    EXPECT_EQ(configManager.getScreens().size(), 2);
}
```

Cenários de Falha

```

TEST(ScreenApiClient, NetworkFailure) {
    // No server running
    ScreenApiClient client("http://localhost:9999", "test");
    JsonDocument config;

    EXPECT_FALSE(client.loadConfiguration(config));
    EXPECT_NE(client.getLastError(), "");
}

TEST(ScreenApiClient, InvalidJson) {
    MockHTTPServer server;
    server.expectGET("/api/screens").andReturn(200, "invalid json{");

    ScreenApiClient client("http://localhost:8080", "test");
    JsonDocument config;

    EXPECT_FALSE(client.loadConfiguration(config));
}

```

Performance Tests

```

TEST(PerformanceTest, LoadTime) {
    auto start = millis();

    ScreenApiClient client(API_URL, "test");
    JsonDocument config;
    client.loadConfiguration(config);

    auto duration = millis() - start;
    EXPECT_LT(duration, 2000); // Must load in < 2 seconds
}

TEST(PerformanceTest, MemoryUsage) {
    size_t heapBefore = ESP.getFreeHeap();

    {
        ScreenApiClient client(API_URL, "test");
        JsonDocument config;
        client.loadConfiguration(config);

        size_t heapDuring = ESP.getFreeHeap();
        EXPECT_GT(heapBefore - heapDuring, 0); // Memory used
        EXPECT_LT(heapBefore - heapDuring, 20480); // Less than 20KB
    }

    // Check for memory leaks
    delay(100);
    size_t heapAfter = ESP.getFreeHeap();
    EXPECT_NEAR(heapBefore, heapAfter, 1024); // Within 1KB (allow for

```



```
fragmentation)
}
```

Estimativas

Esforço de Desenvolvimento

Tarefa	Complexidade	Horas
Setup inicial	Baixa	2-3h
ScreenApiClient core	Média	4-6h
Integração ConfigReceiver	Média	2-3h
Sistema de cache	Média	2-3h
Estratégia fallback	Alta	2-4h
Testes unitários	Baixa	2-3h
Testes integração	Média	2-3h
Debug e polish	Baixa	2-4h
TOTAL	-	18-29h

Impacto em Memória

Componente	RAM (KB)	Flash (KB)
HTTPClient library	+8-12	+15-20
ScreenApiClient class	+2-3	+5-8
JSON response buffer	+5-10	-
Cache storage	+3-8	-
TOTAL ADICIONAL	+18-33	+20-28
% do ESP32	+5-10%	+1-2%

Latência Adicional

Operação	MQTT (ms)	API REST (ms)	Diferença
Config pequena (<5KB)	200-500	300-800	+100-300ms
Config média (5-15KB)	500-1200	600-1500	+100-300ms
Config grande (15-30KB)	1200-3000	1000-2500	-200-500ms

Nota: API REST pode ser mais rápida para configurações grandes devido ao HTTP/1.1 e possível compressão

Tamanho do Firmware

Cenário	Tamanho Atual	Novo Tamanho	Aumento
Build Debug	~1.8MB	~1.85MB	+~50KB
Build Release	~1.2MB	~1.25MB	+~30KB
OTA Update	1.2MB	1.25MB	+2%

Complexidade (1-10)

Aspecto	Complexidade	Justificativa
Implementação	6/10	HTTPClient + JSON parsing + integration
Testes	5/10	Mock servers + network conditions
Deploy	4/10	Feature flags + gradual rollout
Manutenção	3/10	Código mais simples que MQTT
TOTAL	4.5/10	Complexidade Média

✔ Checklist de Validação Pós-Migração

✔ Funcionalidade

- ☐ Configuração carrega corretamente via API
- ☐ Todas as telas são criadas como antes
- ☐ Hot-reload via MQTT funciona
- ☐ Fallback funciona se API estiver offline
- ☐ Cache funciona corretamente
- ☐ Comandos MQTT continuam funcionando

✔ Performance

- ☐ Tempo de boot ≤ 10 segundos
- ☐ Carregamento config ≤ 3 segundos
- ☐ Uso de RAM não aumentou >20KB
- ☐ Interface responde em <100ms após config

✔ Robustez

- ☐ Sistema funciona com API offline
- ☐ Recovery automático quando API volta
- ☐ Não trava com JSON inválido
- ☐ Não vaza memória após múltiplas cargas

✅ Compatibilidade

- ☐ MQTT para relés funciona 100%
- ☐ Telemetria continua sendo enviada
- ☐ Status reports funcionam
- ☐ Emergency stop via MQTT funciona

✅ User Experience

- ☐ Telas carregam visualmente igual
- ☐ Navegação funciona identicamente
- ☐ Hot-reload é imperceptível ao usuário
- ☐ Mensagens de erro são claras

🔧 Código Exemplo Completo

Exemplo de Uso da Nova API

```
void setup() {
    // Initialize components
    configManager = new ConfigManager();
    screenApiClient = new ScreenApiClient(API_BASE_URL, DEVICE_ID);
    configReceiver = new ConfigReceiver(mqttClient, configManager,
    screenApiClient);

    // Test API connectivity
    if (screenApiClient->testConnection()) {
        logger->info("API connection successful");

        // Load configuration
        if (configReceiver->loadFromApi()) {
            logger->info("Configuration loaded successfully");

            // Build UI
            screenManager->buildFromConfig(configManager->getConfig());

        } else {
            logger->error("Failed to load configuration: " +
screenApiClient->getLastError());
            // Implement fallback here
        }
    } else {
        logger->warning("API not reachable, implementing fallback
strategy");
        // Fallback implementation
    }
}
```

Error Handling Completo

```

bool ConfigReceiver::loadFromApi() {
    static int retryCount = 0;
    const int MAX_RETRIES = 3;

    for (int i = 0; i <= MAX_RETRIES; i++) {
        JsonDocument config;

        if (apiClient->loadConfiguration(config)) {
            String configStr;
            serializeJson(config, configStr);

            if (configManager->loadConfig(configStr)) {
                logger->info("Configuration loaded successfully on attempt
" + String(i + 1));
                retryCount = 0; // Reset on success
                return true;
            }

            // Log error and retry
            logger->warning("Config load attempt " + String(i + 1) + " failed:
" + apiClient->getLastError());

            if (i < MAX_RETRIES) {
                delay(1000 * (i + 1)); // Exponential backoff
            }

            logger->error("Failed to load configuration after " +
String(MAX_RETRIES + 1) + " attempts");
            return false;
        }
    }
}

```

Integration Points

```

// main.cpp - Modified initialization
void setupUI() {
    logger->info("Setting up UI components");

    configManager = new ConfigManager();
    screenManager = new ScreenManager();
    navigator = new Navigator(screenManager);

    // Initialize API client
    screenApiClient = new ScreenApiClient(API_BASE_URL, DEVICE_ID);

    // Modified config receiver with API support
    configReceiver = new ConfigReceiver(mqttClient, configManager,
screenApiClient);
}

```

```
logger->info("UI components ready");  
}
```

Plano de Rollout

Fase 1: Development (Semana 1)

- ☒ Implementar `ScreenApiClient`
- ☒ Modificar `ConfigReceiver`
- ☒ Testes básicos
- ☒ Feature flag preparado

Fase 2: Testing (Semana 2)

- ☒ Testes de integração
- ☒ Performance testing
- ☒ Memory leak detection
- ☒ Fallback scenarios

Fase 3: Deployment (Semana 3)

- ☒ Deploy em ambiente de teste
- ☒ Validação com configurações reais
- ☒ Monitoring de performance
- ☒ Ajustes finais

Fase 4: Production (Semana 4)

- ☒ Feature flag habilitada gradualmente
- ☒ Monitoring de logs e métricas
- ☒ Rollback plan pronto
- ☒ Remoção do código MQTT após estabilização

Autor: Sistema AutoTech

Versão: 1.0

Data: 12/08/2025

Status: Pronto para Implementação

Referências

- [ESP32 HTTPClient Library](#)
- [ArduinoJson Documentation](#)
- [AutoTech Architecture Documentation](#)
- [MQTT Protocol Reference](#)