

Documentação Completa: Bot de Integração WhatsApp-GLPI

1. Visão Geral

Este documento detalha a arquitetura, instalação, configuração e funcionamento de um bot para WhatsApp projetado para interagir com o sistema de help desk GLPI. A solução permite que os usuários abram, consultem, atualizem e cancelem chamados diretamente de suas conversas no WhatsApp, otimizando o processo de suporte e melhorando a experiência do usuário.

O bot é construído em Node.js e utiliza a biblioteca @whiskeysockets/baileys para comunicação com o WhatsApp e axios para interagir com a API REST do GLPI.

1.1. Principais Funcionalidades

- **Abertura de Chamados:** Um fluxo conversacional guiado para que o usuário forneça categoria, título, descrição, IP e anexe arquivos (imagens e documentos).
- **Consulta de Chamados:** Permite que o usuário visualize seus chamados em aberto, selecione um e veja todo o histórico de interações.
- **Adicionar Acompanhamento:** Após consultar um chamado, o usuário pode adicionar novas informações ou anexos.
- **Cancelamento de Chamados:** O usuário pode listar seus chamados abertos e optar por fechar (cancelar) um deles.
- **Notificações Ativas (Webhooks):** O bot pode receber notificações do GLPI (ex: um técnico adicionou um acompanhamento) e avisar proativamente o usuário no WhatsApp.
- **Gerenciamento de Sessão:** Controle de estado por usuário e encerramento automático da sessão por inatividade.
- **Persistência de Dados:** Armazena o e-mail do usuário associado ao seu número de WhatsApp para agilizar interações futuras.

2. Preparação do Ambiente (Servidor Linux)

Este guia pressupõe o uso de um sistema operacional baseado em Debian/Ubuntu.

2.1. Pré-requisitos

- Um servidor Linux (Ubuntu 20.04 LTS ou superior é recomendado).
- Node.js (versão 16.x ou superior).
- NPM (geralmente instalado com o Node.js).
- Acesso à API REST do seu GLPI, com um App Token e um User Token válidos.

2.2. Guia de Instalação Passo a Passo

Passo 1: Acessar o Servidor

Conecte-se ao seu servidor via SSH.

```
ssh seu_usuario@ip_do_servidor
```

Passo 2: Atualizar o Sistema

Garanta que todos os pacotes do sistema estejam atualizados.

```
sudo apt update && sudo apt upgrade -y
```

Passo 3: Instalar o Node.js (via NVM)

Recomendamos usar o NVM (Node Version Manager) para instalar e gerenciar versões do Node.js, pois oferece mais flexibilidade.

```
# Baixar e executar o script de instalação do NVM
```

```
curl -o-
```

```
[https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.3/install.sh](https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.3/install.sh) | bash
```

```
# Carregar o NVM no shell atual
```

```
export NVM_DIR="$HOME/.nvm"
```

```
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh"
```

```
# Instalar a versão recomendada do Node.js (LTS - Long Term Support)
```

```
nvm install --lts
```

```
# Verificar a instalação
```

```
node -v
```

```
npm -v
```

Passo 4: Criar o Diretório do Projeto

Crie uma pasta para o seu bot e navegue até ela.

```
mkdir glpi-whatsapp-bot
```

```
cd glpi-whatsapp-bot
```

Passo 5: Iniciar o Projeto Node.js

Crie um arquivo package.json para gerenciar as dependências.

```
npm init -y
```

Passo 6: Instalar as Dependências

Instale todas as bibliotecas necessárias para o projeto.

```
npm install @whiskeysockets/baileys pino axios qrcode-terminal fs form-data express
```

Passo 7: Criar o Arquivo do Bot

Crie o arquivo principal do bot. Você pode usar o editor de texto nano.

```
nano bot.js
```

Copie e cole **todo o código-fonte fornecido** dentro deste arquivo. Salve e saia (Ctrl+X, depois Y, depois Enter).

3. Adaptações Manuais Obrigatórias

Antes de executar o bot, você **PRECISA** customizar as seções a seguir no arquivo bot.js para que correspondam à sua infraestrutura GLPI.

3.1. Configurações da API e Parâmetros Gerais

Este é o bloco de configuração principal. Substitua os valores de exemplo pelos dados da sua instância GLPI.

```
// === CONFIGURAÇÕES ===
const GLPI_API_URL = "http://SEU_IP_OU_DOMINIO/glpi/apirest.php";
const GLPI_APP_TOKEN = "SEU_APP_TOKEN_GERADO_NO_GLPI";
const GLPI_USER_TOKEN = "SEU_USER_TOKEN_DE_UM_TECNICO_NO_GLPI";
const INACTIVITY_MINUTES = 5;
const USER_EMAILS_PATH = './user_emails.json';
const WEBHOOK_PORT = 3000;
const DEBUG_MODE = true; // Mude para false em produção
```

- GLPI_API_URL: A URL completa para o endpoint apirest.php da sua instância GLPI.
- GLPI_APP_TOKEN: Token de aplicação gerado no GLPI em Configurar > Geral > API.
- GLPI_USER_TOKEN: Token pessoal de um usuário técnico do GLPI que tenha as permissões necessárias para abrir e gerenciar chamados.
- WEBHOOK_PORT: Porta de rede que será usada para receber as notificações do GLPI.

Certifique-se de que esta porta esteja liberada no firewall do seu servidor.

3.2. Mapeamento de Categorias de Chamados

Esta seção é crucial. Você deve mapear as categorias que deseja exibir no WhatsApp para os IDs correspondentes no seu GLPI.

```
const CATEGORIES_DISPLAY = {
  '1': '💻 GW sistemas',
  '2': '✉️ E-mail',
  '3': '🌐 Rede interna e internet',
  // Adicione ou modifique as categorias conforme sua necessidade
};
const CATEGORIES_API_MAP = {
  '💻 GW sistemas': 35,
  '✉️ E-mail': 9,
  '🌐 Rede interna e internet': 42,
  // Verifique o ID de cada categoria no seu GLPI e atualize aqui
};
```


Como encontrar o ID da Categoria no GLPI?

1. Acesse sua instância do GLPI.
2. Navegue até Configurar > Intitulados > Categorias de Chamados.
3. Clique na categoria que você deseja usar.
4. O ID da categoria estará visível na barra de endereço do seu navegador (ex: `.../front/itilcategory.form.php?id=35`). Neste caso, o ID é 35.

4. Execução e Gerenciamento

4.1. Primeira Execução

`node bot.js`

1. Um QR Code será exibido no terminal.
2. Abra o WhatsApp no seu celular, vá em Aparelhos conectados e escaneie o código.
3. Após a conexão, a mensagem  Bot WhatsApp conectado! aparecerá.
4. O bot criará uma pasta chamada `auth_info` para armazenar a sessão.

4.2. Execução em Produção com PM2

Para manter o bot rodando 24/7, use o gerenciador de processos PM2.

Instalar o PM2 globalmente

```
sudo npm install pm2 -g
```

Iniciar o bot com o PM2

```
pm2 start bot.js --name "glpi-whatsapp-bot"
```

Para monitorar os logs em tempo real

```
pm2 logs glpi-whatsapp-bot
```

Para fazer o PM2 iniciar automaticamente com o sistema

```
pm2 startup
```

(siga as instruções que o comando acima fornecer)

Salvar a lista de processos atual

```
pm2 save
```

4.3. Depuração (Debugging)

- **Modo de Depuração:** Mantenha `DEBUG_MODE` como `true` para logs detalhados.
- **Logs do PM2:** O comando `pm2 logs glpi-whatsapp-bot` é essencial para ver saídas e erros em tempo real.

5. Análise Detalhada do Código

5.1. Importações e Variáveis Globais

Código:

```
const { default: makeWASocket, useMultiFileAuthState, DisconnectReason, getContentType,
downloadContentFromMessage } = require('@whiskeysockets/baileys');
const P = require('pino');
const axios = require('axios');
const qrcode = require('qrcode-terminal');
const fs = require('fs');
const FormData = require('form-data');
const express = require('express');

// ... (Bloco de configurações omitido, já detalhado na seção 3)

let userStates = {};
let userEmails = {};

// Estruturas de dados para gerenciar webhooks e estados
const pendingWebhookTimers = new Map();
const userPendingWebhookContent = new Map();
const botActionSuppressions = new Map();

// ... (Blocos de mapeamento de categorias e status omitidos)
```

Explicação:

- **@whiskeysockets/baileys**: Biblioteca principal para a conexão com a API do WhatsApp.
- **axios**: Cliente HTTP para fazer as requisições à API REST do GLPI.
- **qrcode-terminal**: Gera o QR Code de autenticação no terminal.
- **fs, form-data, express**: Módulos para manipulação de arquivos, envio de formulários com anexos e criação do servidor de webhooks, respectivamente.
- **userStates**: Objeto-chave que funciona como uma "memória" de curto prazo. Armazena o estado atual da conversa para cada usuário (ex: 'awaiting_title'), além dos dados coletados (título, descrição, etc.).
- **userEmails**: Objeto que mapeia o ID do WhatsApp do usuário ao seu e-mail, carregado de user_emails.json para persistência.
- **Estruturas de Webhook**: Maps para gerenciar timers e conteúdos de webhooks, garantindo que notificações não se percam ou sejam processadas incorretamente.

5.2. Funções Auxiliares

Código:

```
function getExtensionFromMime(mimeType) {
  switch (mimeType) {
    case 'image/jpeg': return '.jpeg';
    case 'image/png': return '.png';
    case 'image/gif': return '.gif';
    case 'application/pdf': return '.pdf';
    default: return '';
  }
}

function loadUserEmails() {
  try {
    if (fs.existsSync(USER_EMAILS_PATH)) {
      let fileContent = fs.readFileSync(USER_EMAILS_PATH, 'utf8');
      const cleanedContent = fileContent.replace(/,\s*([}\]])/g, '$1');
      userEmails = JSON.parse(cleanedContent);
      console.log('✅ Emails dos usuários carregados.');
```

else {

```
      console.log('❗ Arquivo de emails não encontrado, um novo será criado.');
```

}

```
    } catch (error) {
      console.error('❌ Erro ao carregar o arquivo de emails:', error);
    }
  }

  async function saveUserEmails() {
    try {
      await fs.promises.writeFile(USER_EMAILS_PATH, JSON.stringify(userEmails, null, 2));
      if (DEBUG_MODE) console.log('DEBUG: Emails dos usuários salvos.');
```

} catch (error) {

```
      console.error('❌ Erro ao salvar o arquivo de emails:', error);
    }
  }

  function stripHtmlTags(htmlContent) {
    if (!htmlContent) return '';
    return htmlContent.toString()
      .replace(/&/g, '&')
      .replace(/</g, '<')
      // ... (outras substituições)
      .replace(/<[^\>]*>?/gm, '')
      .replace(/(\r\n|\n\r){2,}/g, '\n')
      .trim();
  }
}
```

Explicação:

- **getExtensionFromMime:** Converte um MIME type (ex: image/jpeg) em uma extensão de arquivo (.jpeg), útil para nomear anexos.
- **loadUserEmails e saveUserEmails:** Gerenciam a leitura e escrita do arquivo user_emails.json, garantindo a persistência dos e-mails dos usuários entre reinicializações do bot.
- **stripHtmlTags:** Remove tags HTML e decodifica entidades (ex: <) do conteúdo vindo do GLPI, limpando o texto para ser exibido de forma legível no WhatsApp.

5.3. Bloco Principal (startBot e messages.upsert)

Código:

```
async function startBot() {
  loadUserEmails();
  const { state, saveCreds } = await useMultiFileAuthState('./auth_info');
  const sock = makeWASocket({
    logger: P({ level: 'silent' }),
    auth: state
  });

  sock.ev.on('creds.update', saveCreds);

  sock.ev.on('connection.update', (update) => {
    // ... (lógica de conexão, QR code e reconexão)
  });

  function resetInactivityTimer(from) {
    // ... (lógica para limpar a sessão do usuário após inatividade)
  }

  sock.ev.on('messages.upsert', async (m) => {
    const msg = m.messages[0];
    if (msg.key.fromMe || !msg.message) return;

    const from = msg.key.remoteJid;
    // ... (extração de dados da mensagem)

    if (normalizedText === '0') {
      // ... (lógica para encerrar o processo)
      return;
    }
  })
}
```



```

const currentState = userStates[from]?.state;
if (!currentState) {
  await showMainMenu(from, senderName);
  return;
}

switch (currentState) {
  case 'menu': await handleMenuOption(from, normalizedText, senderName); break;
  case 'awaiting_category': await handleCategorySelection(from, normalizedText); break;
  // ... (outros estados da conversa)
}
});

// ... (restante do código, como servidor de webhook e funções de fluxo)
}

startBot();

```

Explicação:

- **startBot:** Função principal que inicializa tudo. Carrega e-mails, configura a autenticação com o WhatsApp e estabelece os listeners de eventos.
- **sock.ev.on('connection.update', ...):** Gerencia o ciclo de vida da conexão, exibindo o QR Code quando necessário e tentando reconectar automaticamente.
- **sock.ev.on('messages.upsert', ...):** O coração do bot. Este evento é disparado para cada nova mensagem.
 - Ele ignora as próprias mensagens do bot (fromMe).
 - Implementa o comando global 'O' para cancelar a operação a qualquer momento.
 - **Máquina de Estados:** Verifica o currentState do usuário. Se não houver estado, mostra o menu principal. Caso contrário, o switch direciona a mensagem para a função de tratamento correta (ex: handleCategorySelection), guiando a conversa passo a passo.

5.4. Servidor de Webhooks

Código:

```
// --- SERVIDOR DE WEBHOOKS ---
const app = express();
app.use(express.text({ type: '*/' }));

async function processDelayedWebhook(userJid, rawBody) {
  // ... (extrai dados do corpo do webhook)
  const suppression = botActionSuppressions.get(userJidForWebhook);
  if (suppression && ...) {
    // Ignora a notificação se foi uma ação do próprio bot
    return;
  }
  // ... (formata e envia a mensagem de notificação para o usuário)
}

app.post('/glpi-webhook', async (req, res) => {
  // ... (recebe a notificação do GLPI)
  const timer = setTimeout(() => {
    // ... (chama processDelayedWebhook após um atraso)
  }, WEBHOOK_PROCESSING_DELAY_MS);
  // ...
});

app.listen(WEBHOOK_PORT, '0.0.0.0', () => {
  console.log(`🚀 Servidor de webhooks rodando...`);
});
```

Explicação:

- **app.post('/glpi-webhook', ...):** Define a rota que o GLPI usará para enviar notificações. Quando uma notificação chega, ela não é processada imediatamente.
- **Lógica de Atraso (setTimeout):** Inicia um temporizador antes de processar. Isso evita uma "condição de corrida", onde a notificação do GLPI (ex: "chamado fechado") poderia chegar ao usuário antes da mensagem de confirmação do próprio bot (ex: "Seu chamado foi fechado com sucesso!").
- **processDelayedWebhook(...):** É executada após o atraso. Ela analisa o webhook, formata uma mensagem amigável e a envia ao usuário.
- **Supressão de Notificação:** Uma lógica inteligente verifica se a atualização do GLPI foi causada por uma ação recente do próprio bot. Se sim, a notificação é ignorada para evitar mensagens duplicadas e confusas para o usuário.

5.5. Funções de Fluxo e Interação com GLPI

Exemplo de Código (Criação de Chamado):

```
async function handleTicketCreation(from, senderName, ticketData) {
  let sessionToken = null;
  try {
    // 1. Inicia a sessão com a API do GLPI
    sessionToken = (await axios.get(`${GLPI_API_URL}/initSession`, ...)).data.session_token;

    // 2. Busca o ID do usuário no GLPI usando o e-mail
    const { glpiUserId, glpiUserName } = await getGlpiUser(sessionToken, ticketData.email,
from);

    // 3. Cria o chamado no GLPI
    const ticketId = await createGlpiTicket(sessionToken, ticketData, glpiUserId, ...);

    // 4. Processa e anexa arquivos (PDFs, etc.) que não são imagens
    await processAttachments(sessionToken, ticketId, ticketData);
    await sock.sendMessage(from, { text: `✅ Chamado *#${ticketId}* aberto com sucesso!`
});
  } catch (error) {
    // ... (tratamento de erro)
  } finally {
    // 5. Encerra a sessão com a API
    if (sessionToken) await closeSession(sessionToken);
    delete userStates[from];
  }
}

async function createGlpiTicket(sessionToken, ticketData, glpiUserId, ...) {
  const { title, category, description, attachments } = ticketData;
  // ... (monta o conteúdo do chamado em HTML)
  // Anexa imagens diretamente no corpo do chamado como base64
  const imagesContent = attachments.filter(...).map(att => `<p></p>`).join("");
  if (imagesContent) ticketContent += ...;
  const ticketInput = { name: `${title} - ${senderName} via WhatsApp`, content:
ticketContent, ... };
  if (glpiUserId) ticketInput._users_id_requester = glpiUserId;

  const { data } = await axios.post(`${GLPI_API_URL}/Ticket`, { input: ticketInput }, ...);
  return data.id;
}
```

Explicação:

- **Funções handle...:** Orquestram o fluxo da conversa. Cada uma é responsável por uma etapa: receber a entrada, armazená-la em `userStates`, atualizar o estado para o próximo passo e enviar a próxima pergunta.
- **Funções de API (`getGlpidUser`, `createGlpiTiket`, etc.):** Contêm a lógica de comunicação direta com o GLPI.
 - Elas sempre seguem um padrão: iniciar sessão (`initSession`), realizar a operação (GET para buscar, POST/PUT para criar/atualizar) e, crucialmente, encerrar a sessão (`killSession`) dentro de um bloco `finally` para garantir que a conexão seja sempre fechada, mesmo em caso de erro.
 - **Tratamento de Anexos:** O bot diferencia imagens de outros tipos de arquivos. Imagens são embutidas diretamente no corpo do chamado em formato `base64`, enquanto outros documentos (como PDFs) são enviados para o endpoint `/Document` do GLPI e associados ao ticket.