ChatGPT

Documentação do Sistema de Gerenciamento de WhatsApp

Visão geral

Este projeto implementa um **painel de controle e uma API** para gerenciar contas do WhatsApp Web de forma centralizada. Ele se apoia na biblioteca Baileys para criar sessões de WhatsApp, no **Express** para expor uma API HTTP e no **WebSocket** para enviar eventos em tempo real ao painel e a outros clientes. A interface web (construída com Tailwind CSS) permite que um usuário não-técnico crie conexões, leia mensagens recebidas e envie textos ou mídias sem precisar tocar no código.

O servidor aceita várias sessões simultâneas e grava as credenciais de autenticação em diretórios separados, de modo que as conexões sobrevivem a reinicializações. Cada sessão possui um identificador (`sessionId`), um token de sessão gerado após o login e informações básicas como nome e número do telefone conectado.

Estrutura principal

- `index.js` – Inicializa o Express, injeta a função de broadcast no módulo WhatsApp, configura o servidor WebSocket, carrega sessões salvas e inicia o painel na porta configurada.
- `whatsapp.js` – Centraliza toda a lógica de sessão: cria ou restaura conexões, gera o QR Code, monitora status de conexão, retransmite mensagens recebidas, envia atualizações de presença e gerencia reconexões automáticas. As credenciais são salvas em `./auth/<sessionId>`.
- `websocket.js` – Sobe um `WebSocketServer` e implementa um **broadcast** simples que envia mensagens para todos os clientes conectados. Quando um cliente envia `{"type": "get-all-sessions"}`, o servidor retorna a lista de sessões.
- `router.js` – Define todas as rotas da API REST. Existem duas categorias: rotas **administrativas** (requerem a chave de API global) e rotas **operacionais** (requerm o token da sessão). O roteador lida com envio de mensagens, operações de grupos, modificações de chat, consulta de status, atualização de perfil, gestão de privacidade, etc.
- `index.html` / `index.js` – Código do painel. Conecta-se ao WebSocket, exibe as sessões disponíveis, mostra o QR Code e permite o envio de mensagens de texto ou mídia.

Funcionalidades suportadas

A seguir estão as principais funções já implementadas. Elas podem ser acessadas pela API ou pelo painel.

Gerenciamento de sessões

- **Criar/iniciar sessão** (`POST /sessions/start`) – Cria uma nova sessão informando um `sessionId`. O servidor guarda as credenciais em `auth/<sessionId>` e emite um QR Code para login. Após ler o código com o telefone, a sessão muda de `PENDING` para `CONNECTED` e recebe um **token de sessão** (exibido no painel) que autoriza as chamadas operacionais.

- **Listar sessões** (`GET /sessions`) – Retorna todas as sessões conhecidas (ativas, pendentes ou desconectadas) com seus status. Somente rotas administrativas exigem a chave de API global.
- **Remover sessão** (`DELETE /sessions/:sessionId`) – Faz logout, remove as credenciais do disco e emite uma atualização de status `DELETED` .
- **Recuperar sessão específica** (`GET /sessions/:sessionId`) – Disponibiliza as informações básicas (somente via WebSocket, pois o roteador não expõe essa rota diretamente).
- **Persistência automática** – Ao iniciar, o servidor procura subdiretórios dentro de `./auth` e restaura todas as sessões previamente autenticadas.

Envio de mensagens

As rotas a seguir requerem autenticação do tipo **Bearer** usando o token de sessão (enviado no cabeçalho `Authorization: Bearer <token>`). Além disso, o número de destino deve estar no formato internacional sem `+` (por exemplo, `5531999998888`).

- **Mensagem de texto** (`POST /:sessionId/messages/send`) – Envia uma mensagem simples para um contato ou grupo. Parâmetros: `to` , `message` e opcionalmente `options` (p. ex., scheduling, mentions).
- **Responder mensagem (citar)** (`POST /:sessionId/messages/send-quote`) – Envia uma resposta citando a mensagem original (`quoted`), útil para dar contexto em grupos.
- **Mensagem com menções** (`POST /:sessionId/messages/send-mention`) – Permite mencionar usuários em uma mensagem de texto, informando o array `mentions` .
- **Encaminhar mensagem** (`POST /:sessionId/messages/forward`) – Encaminha uma mensagem recebida para outro contato ou grupo.
- **Enviar localização** (`POST /:sessionId/messages/location`) – Compartilha coordenadas (latitude/longitude) e, opcionalmente, nome e endereço.
- **Enviar contato** (`POST /:sessionId/messages/contact`) – Envia um cartão de contato (vCard) com nome e dados do contato.
- **Enviar reação** (`POST /:sessionId/messages/reaction`) – Envia um emoji como reação a uma mensagem específica.
- **Enviar enquete (poll)** (`POST /:sessionId/messages/poll`) – Cria uma enquete com opções e número de seleções permitido.
- **Enviar GIF** (`POST /:sessionId/messages/send-gif`) – Envia um arquivo GIF. O upload é feito via `multipart/form-data` (campo `file`).
- **Enviar mídia (imagem, vídeo, áudio ou documento)** (`POST /:sessionId/messages/send-media`) – Envia qualquer tipo de mídia. É possível acrescentar uma legenda (`caption`). Para imagens/vídeos/áudios, o módulo Baileys define automaticamente o tipo correto; para documentos é preciso enviar também `fileName` e `mimetype` .
- **View-once (visualização única)** (`POST /:sessionId/messages/view-once`) – Envia uma foto ou vídeo que só pode ser visto uma vez pelo destinatário.
- **Broadcast (Status)** (`POST /:sessionId/messages/broadcast`) – Envia um texto para todos os contatos de uma lista de transmissão (`recipients`) via Status.
- **Marcar mensagens como lidas** (`POST /:sessionId/messages/read`) – Recebe um array de keys de mensagens e marca como lidas para o WhatsApp.
- **Apagar mensagem** (`POST /:sessionId/messages/delete`) – Remove uma mensagem para todos os participantes usando o identificador da mensagem (`key`).

Leitura de conversas e histórico

- **Histórico** (`POST /:sessionId/chat/:jid/history`) – Busca mensagens mais antigas de um chat (grupo ou privado). É necessário informar o número de mensagens (`count`), a key da última mensagem conhecida e seu timestamp.
- **Modificação de chat** (`POST /:sessionId/chat/:jid/modify`) – Permite arquivar, silenciar, fixar, marcar como lido ou remover mensagens de um chat. O corpo deve seguir o formato aceito pelo método `chatModify` do Baileys.

Grupos

- **Listar grupos** (`GET /:sessionId/groups`) – Retorna todos os grupos em que a conta participa.
- **Criar grupo** (`POST /:sessionId/groups/create`) – Cria um novo grupo com assunto e lista de participantes.
- **Metadados** (`GET /:sessionId/groups/:groupId/metadata`) – Traz detalhes de um grupo, incluindo nome, descrição, participantes e permissões.
- **Gerenciar participantes** (`POST /:sessionId/groups/:groupId/participants`) – Adiciona, remove, promove ou rebaixa participantes (campo `action`: `add`, `remove`, `promote`, `demote`).
- **Mudar nome** (`PUT /:sessionId/groups/:groupId/subject`) – Atualiza o assunto do grupo.
- **Sair de um grupo** (`POST /:sessionId/groups/:groupId/leave`) – Abandona o grupo.
- **Código de convite** (`GET /:sessionId/groups/:groupId/invite-code`) – Gera o código de convite do grupo. Também há rotas para revogar o código e para obter informações de um convite.
- **Aprovar/rejeitar solicitações** (`POST /:sessionId/groups/:groupId/requests`) – Quando o modo de aprovação está habilitado, aprova ou rejeita pedidos de entrada. Há outra rota (`GET /:sessionId/groups/:groupId/requests`) para listar os pedidos pendentes.
- **Configurações** (`PUT /:sessionId/groups/:groupId/settings`) – Altera configurações como `announcement` (somente admins enviam mensagens) ou `locked` / `unlocked`.
- **Mensagens efêmeras** (`POST /:sessionId/groups/:groupId/ephemeral`) – Define a duração das mensagens temporárias em segundos.
- **Modo de adição** (`POST /:sessionId/groups/:groupId/member-add-mode`) – Define se todos podem adicionar participantes ou somente administradores.

Perfil e privacidade

- **Verificar existência de número** (`GET /:sessionId/users/:jid/exists`) – Checa se um número está registrado no WhatsApp.
- **Foto de perfil** (`GET /:sessionId/users/:jid/avatar`) – Retorna a URL da foto de perfil, caso haja permissão.
- **Enviar atualização de presença** (`POST /:sessionId/users/presence`) – Atualiza o status de presença (digitando, gravando, pausado, disponível, etc.) para um contato.
- **Alterar nome e status** (`PUT /:sessionId/profile/name` e `/status`) – Modifica o nome ou texto de status da conta conectada.
- **Atualizar foto de perfil** (`PUT /:sessionId/profile/picture`) – Envia uma nova foto via multipart. Há também rotas para remover a foto.
- **Configurações de privacidade** (`GET /:sessionId/privacy`) – Obtém as configurações atuais (last seen, foto, status, grupos, recibos de leitura). Rotas específicas permitem atualizar

cada item (por exemplo, `/privacy/last-seen`, `/privacy/profile-picture`, `/privacy/status`, etc.).
- **Bloquear/Desbloquear** (`POST /:sessionId/users/:jid/block`) – Bloqueia ou desbloqueia um contato. A rota `/users/blocklist` recupera todos os bloqueados.
- **Atualizar modo de mensagens desaparecendo** (`PUT /:sessionId/privacy/default-disappearing`) – Define a duração padrão das mensagens temporárias em novas conversas.

Eventos em tempo real (WebSocket)

O servidor WebSocket funciona na mesma porta do HTTP e envia eventos para todos os clientes conectados. Esses eventos são extremamente úteis para integrar outros sistemas ou para atualizar o painel em tempo real.

Conectando-se

1. Abra uma conexão WebSocket para `ws://<host>:<porta>` (por padrão, `ws://localhost:3001`).
2. Ao conectar, o cliente deve enviar a seguinte mensagem JSON para solicitar a lista de sessões:

```
{"type": "get-all-sessions"}
```

1. O servidor responde com `{"type": "all-sessions", "data": [...]}`, contendo um array de sessões (cada uma com `sessionId`, `status`, `qrCode`, `token`, `name` e `phoneNumber`).

Tipos de eventos emitidos

- `session-update` – Enviado sempre que uma sessão muda de status. O payload inclui os mesmos campos citados acima.
- `qr` – Contém a propriedade `sessionId` e uma URL base64 (`src`) com o QR Code para autenticação. O código permanece válido até expirar; se fechar o modal, você pode reabrir e reutilizar o mesmo `src` enquanto a sessão estiver pendente.
- `message` – Dispara para cada mensagem recebida pela conta. Inclui `sessionId`, `from` (JID do remetente), `text` e `timestamp`.
- `presence` – Notifica mudanças de presença (disponível, digitando, gravando, etc.) de usuários para os quais você se inscreveu via API (`users/presence-subscribe`).

Qualquer aplicação externa pode se conectar a esse WebSocket para ouvir e tratar esses eventos. Por exemplo, um bot pode receber as mensagens em JSON e responder de acordo.

Como usar o sistema

1. **Pré-requisitos** – Tenha o Node.js (versão 16 ou superior) instalado. Faça o download deste projeto e execute `npm install` para instalar as dependências.
2. **Configuração** – Crie um arquivo `.env` na raiz, definindo pelo menos:
3. `PORT=3001` – Porta onde a API e o WebSocket ficarão disponíveis.
4. `API_KEY=<suaChaveSecreta>` – Chave global utilizada para iniciar ou remover sessões via API.
5. `LOG_LEVEL=info` (opcional) – Nível de log.

6. **Inicialização** – Inicie o servidor com `node index.js`. Na primeira execução serão criados os diretórios `./tmp` e `./auth`.
7. **Acessar o painel** – Abra o navegador em `http://localhost:3001`. No painel, informe sua chave de API na área lateral e clique em **Nova Conexão**, definindo um nome único para a sessão (p. ex., `suporte`).
8. **Escanear QR Code** – Ao criar uma sessão, um modal exibirá um QR. Leia com o WhatsApp do celular para autenticar. Uma vez conectado, o status passa para `CONNECTED` e o painel mostrará o **token de sessão**. Copie esse token; será necessário para usar a API.
9. **Enviar mensagens pelo painel** – Selecione a sessão e utilize os campos de envio de texto ou mídia. As mensagens recebidas aparecem no painel em tempo real.
10. **Usar a API REST** – Utilize um cliente HTTP (cURL, Postman, etc.). As rotas administrativas (`/sessions`) exigem o cabeçalho `x-api-key` com sua chave. As rotas de operação exigem `Authorization: Bearer <tokenDaSessao>`.
11. **Sair ou remover sessão** – Clique no botão **Desconectar e Apagar** no cabeçalho da sessão. Isso encerra a conexão, apaga as credenciais e atualiza todos os clientes via WebSocket.

Integração com outros sistemas

Existem diversas formas de integrar este sistema a outros aplicativos (bots, CRMs, ferramentas de marketing, etc.). Duas opções recomendadas são:

1. Conectar ao WebSocket existente

Seu sistema pode abrir uma conexão WebSocket para o mesmo servidor que o painel utiliza e receber todos os eventos de sessão, QR Code e mensagens. Esse método é o mais simples, pois reaproveita o mecanismo de broadcast já existente. Uma vez conectado, basta filtrar os eventos desejados e tratá-los de acordo.

Exemplo em pseudocódigo:

```
const ws = new WebSocket('ws://localhost:3001');
ws.onopen = () => ws.send(JSON.stringify({ type: 'get-all-sessions' }));
ws.onmessage = (ev) => {
  const msg = JSON.parse(ev.data);
  if (msg.type === 'message') {
    // trate a mensagem recebida
  }
  if (msg.type === 'qr') {
    // exiba o QR ao operador humano
  }
};
```

2. Injetar uma função de broadcast personalizada

Se você preferir que o servidor envie eventos também para outro destino (como um webhook HTTP ou outro WebSocket), pode **injetar** uma função de broadcast própria no módulo WhatsApp. O arquivo `whatsapp.js` expõe a função `initialize(broadcastFn)`. No `index.js`, o servidor já injeta a função padrão de broadcast, mas você pode modificá-la assim:

```
// index.js
const axios = require('axios');
const whatsapp = require('./whatsapp');

// Define uma função personalizada que encaminha eventos para um webhook
function customBroadcast(type, data) {
  // envia para o painel via broadcast padrão
  broadcast(type, data);
  // envia para outro sistema via HTTP
  axios.post('https://meu-sistema.com/webhook/whatsapp', { type, data })
    .catch(err => console.error('Falha ao enviar webhook', err));
}

// Injeta a função customizada
whatsapp.initialize(customBroadcast);
```

Com isso, cada chamada a `emit()` no `whatsapp.js` enviará o evento tanto para o painel quanto para seu webhook. Você pode adaptar a função para publicar em filas de mensagens, gravar em banco de dados ou qualquer outro mecanismo.

Considerações finais

- **Persistência das credenciais** – As credenciais de cada sessão são armazenadas em `./auth/<sessionId>`. Faça backup desse diretório se não quiser perder logins após reinstalar o projeto.
- **Tokens de sessão** – O token Bearer é regenerado sempre que a sessão se conecta. Se você desconectar e reconectar, precisará atualizar o token nos seus clientes.
- **Segurança** – Guarde sua `API_KEY` e os tokens de sessão em local seguro. Eles concedem controle total sobre as contas. Considere expor a API por trás de um proxy e usar HTTPS em produção.
- **Limitações** – A API depende da estabilidade da biblioteca Baileys e pode ter limitações impostas pelo WhatsApp Web. Evite usos em larga escala sem testes.
- **Atualizações** – O sistema verifica a versão mais recente do Baileys ao iniciar e registra no log se está desatualizado. Recomenda-se atualizar periodicamente as dependências do projeto.

Com esta documentação você possui uma visão completa das funcionalidades oferecidas, sabe como utilizar o painel e a API e entende como integrar outros sistemas para receber eventos ou enviar mensagens programaticamente.
