WhatsApp MCP Server

A Model Context Protocol (MCP) server for WhatsApp integration. This server provides standardized API endpoints for various WhatsApp operations including sending messages, searching contacts, and managing chats.

Project Structure

- **whatsapp-mcp-server**: Python-based MCP server
- **whatsapp-bridge**: Go-based bridge to WhatsApp API (running separately)

Configuration

The server can be configured using the following environment variables:

- `MCP_TRANSPORT` - Transport protocol (`stdio` or `http`, default: `stdio`)
- `PORT` - HTTP server port when using HTTP transport (default: `8000`)
- `WHATSAPP_API_URL` - URL of the WhatsApp API bridge (default: `http://localhost:8080/api`)

Deployment Instructions for Render

1. Create a new Web Service on Render
    - Connect your repository or use Render's direct deployment
    - Select the "Web Service" option
2. Configure the service:
    - **Name**: whatsapp-mcp
    - **Build Command**: `pip install -r whatsapp-mcp-server/requirements.txt`
    - **Start Command**: `cd whatsapp-mcp-server && MCP_TRANSPORT=http python main.py`
    - **Environment**: Python 3
3. Add environment variables:
    - `WHATSAPP_API_URL`: URL of your WhatsApp bridge service
4. Deploy

Important Notes

- The WhatsApp Bridge service must be accessible from the Render hosted service
- The database files (`messages.db` and `whatsapp.db`) need to be populated with relevant data
- For production use, consider moving to a proper database backend instead of SQLite

WhatsApp Bridge

The WhatsApp Bridge component needs to be deployed separately and made accessible to this MCP server. The bridge service URL should be provided to the MCP server using the `WHATSAPP_API_URL` environment variable.

Installation

Prerequisites

- Go
- Python 3.6+
- Anthropic Claude Desktop app (or Cursor)
- UV (Python package manager), install with `curl -LsSf https://astral.sh/uv/install.sh | sh`
- FFmpeg (*optional*) - Only needed for audio messages. If you want to send audio files as playable WhatsApp voice messages, they must be in `.ogg` Opus format. With FFmpeg installed, the MCP server will automatically convert non-Opus audio files. Without FFmpeg, you can still send raw audio files using the `send_file` tool.

Steps

1. **Clone this repository**

```
git clone https://github.com/lharries/whatsapp-mcp.git
cd whatsapp-mcp
```

2. **Run the WhatsApp bridge**

   Navigate to the whatsapp-bridge directory and run the Go application:

```
cd whatsapp-bridge
go run main.go
```

   The first time you run it, you will be prompted to scan a QR code. Scan the QR code with your WhatsApp mobile app to authenticate.

   After approximately 20 days, you will might need to re-authenticate.

3. **Connect to the MCP server**

   Copy the below json with the appropriate {{PATH}} values:

```
{
  "mcpServers": {
    "whatsapp": {
      "command": "{{PATH_TO_UV}}", // Run `which uv` and place the output
here
```

```
      "args": [
        "--directory",
        "{{PATH_TO_SRC}}/whatsapp-mcp/whatsapp-mcp-server", // cd into
the repo, run `pwd` and enter the output here + "/whatsapp-mcp-server"
        "run",
        "main.py"
      ]
    }
  }
}
```

For **Claude**, save this as `claude_desktop_config.json` in your Claude Desktop configuration directory at:

```
~/Library/Application Support/Claude/claude_desktop_config.json
```

For **Cursor**, save this as `mcp.json` in your Cursor configuration directory at:

```
~/.cursor/mcp.json
```

4. **Restart Claude Desktop / Cursor**

   Open Claude Desktop and you should now see WhatsApp as an available integration.

   Or restart Cursor.

Windows Compatibility

If you're running this project on Windows, be aware that `go-sqlite3` requires **CGO to be enabled** in order to compile and work properly. By default, **CGO is disabled on Windows**, so you need to explicitly enable it and have a C compiler installed.

Steps to get it working:

1. **Install a C compiler**
   We recommend using MSYS2 to install a C compiler for Windows. After installing MSYS2, make sure to add the `ucrt64\bin` folder to your `PATH`.
   → A step-by-step guide is available here.

2. **Enable CGO and run the app**

```
cd whatsapp-bridge
go env -w CGO_ENABLED=1
go run main.go
```

Without this setup, you'll likely run into errors like:

```
Binary was compiled with 'CGO_ENABLED=0', go-sqlite3 requires cgo to work.
```

Architecture Overview

This application consists of two main components:

1. **Go WhatsApp Bridge** (`whatsapp-bridge/`): A Go application that connects to WhatsApp's web API, handles authentication via QR code, and stores message history in SQLite. It serves as the bridge between WhatsApp and the MCP server.

2. **Python MCP Server** (`whatsapp-mcp-server/`): A Python server implementing the Model Context Protocol (MCP), which provides standardized tools for Claude to interact with WhatsApp data and send/receive messages.

Data Storage

- All message history is stored in a SQLite database within the `whatsapp-bridge/store/` directory
- The database maintains tables for chats and messages
- Messages are indexed for efficient searching and retrieval

Usage

Once connected, you can interact with your WhatsApp contacts through Claude, leveraging Claude's AI capabilities in your WhatsApp conversations.

MCP Tools

Claude can access the following tools to interact with WhatsApp:

- **search_contacts**: Search for contacts by name or phone number
- **list_messages**: Retrieve messages with optional filters and context
- **list_chats**: List available chats with metadata
- **get_chat**: Get information about a specific chat
- **get_direct_chat_by_contact**: Find a direct chat with a specific contact
- **get_contact_chats**: List all chats involving a specific contact
- **get_last_interaction**: Get the most recent message with a contact
- **get_message_context**: Retrieve context around a specific message
- **send_message**: Send a WhatsApp message to a specified phone number or group JID
- **send_file**: Send a file (image, video, raw audio, document) to a specified recipient
- **send_audio_message**: Send an audio file as a WhatsApp voice message (requires the file to be an .ogg opus file or ffmpeg must be installed)
- **download_media**: Download media from a WhatsApp message and get the local file path

Media Handling Features

The MCP server supports both sending and receiving various media types:

Media Sending

You can send various media types to your WhatsApp contacts:

- **Images, Videos, Documents**: Use the `send_file` tool to share any supported media type.
- **Voice Messages**: Use the `send_audio_message` tool to send audio files as playable WhatsApp voice messages.
    - For optimal compatibility, audio files should be in `.ogg` Opus format.
    - With FFmpeg installed, the system will automatically convert other audio formats (MP3, WAV, etc.) to the required format.
    - Without FFmpeg, you can still send raw audio files using the `send_file` tool, but they won't appear as playable voice messages.

Media Downloading

By default, just the metadata of the media is stored in the local database. The message will indicate that media was sent. To access this media you need to use the download_media tool which takes the `message_id` and `chat_jid` (which are shown when printing messages containing the meda), this downloads the media and then returns the file path which can be then opened or passed to another tool.

Technical Details

1. Claude sends requests to the Python MCP server
2. The MCP server queries the Go bridge for WhatsApp data or directly to the SQLite database
3. The Go accesses the WhatsApp API and keeps the SQLite database up to date
4. Data flows back through the chain to Claude
5. When sending messages, the request flows from Claude through the MCP server to the Go bridge and to WhatsApp

Troubleshooting

- If you encounter permission issues when running uv, you may need to add it to your PATH or use the full path to the executable.
- Make sure both the Go application and the Python server are running for the integration to work properly.

Authentication Issues

- **QR Code Not Displaying**: If the QR code doesn't appear, try restarting the authentication script. If issues persist, check if your terminal supports displaying QR codes.
- **WhatsApp Already Logged In**: If your session is already active, the Go bridge will automatically reconnect without showing a QR code.
- **Device Limit Reached**: WhatsApp limits the number of linked devices. If you reach this limit, you'll need to remove an existing device from WhatsApp on your phone (Settings > Linked Devices).
- **No Messages Loading**: After initial authentication, it can take several minutes for your message history to load, especially if you have many chats.
- **WhatsApp Out of Sync**: If your WhatsApp messages get out of sync with the bridge, delete both database files (`whatsapp-bridge/store/messages.db` and `whatsapp-`

`bridge/store/whatsapp.db`) and restart the bridge to re-authenticate.

For additional Claude Desktop integration troubleshooting, see the MCP documentation. The documentation includes helpful tips for checking logs and resolving common issues.