Signal PoC - Complete Documentation

Version: 1.0.0
Date: October 2025
Status: Proof of Concept - Working ✅

📋 Table of Contents

Project Overview

What Is This?

A **complete web application** for sending Signal messages to groups, built with:

- **Frontend:** React 18 + Vite
- **Backend:** Node.js + Express
- **Signal Integration:** signal-cli-rest-api (Docker)

This is a **Proof of Concept (PoC)** designed to demonstrate Signal messaging automation through a web interface.

Use Cases

This PoC can be used for:

- 📊 **System monitoring alerts** - Send automated notifications to teams
- 📈 **CI/CD notifications** - Build status updates
- 🚨 **Security alerts** - Incident notifications
- 📅 **Scheduled messages** - Daily reports

- 🤖 **Bot integrations** - Automated responses
- 💼 **Team communication** - Group messaging automation

Tech Stack

Layer	Technology	Purpose
Frontend	React 18, Vite, Axios	User interface
Backend	Node.js, Express, Axios, Morgan	REST API server
Signal Integration	signal-cli-rest-api (Docker)	Signal protocol handler
Styling	Vanilla CSS	Modern gradient design
Data	Stateless (no database)	Real-time from Signal

What This Project Does

Core Features ✅

- ✅ **View all Signal groups** - List all groups you're a member of
- ✅ **Select any group** - Click to select from the list
- ✅ **Write messages** - Large textarea for composing
- ✅ **Send messages** - Messages appear in Signal for all members
- ✅ **Real-time error messages** - Errors shown in UI, console, and logs
- ✅ **Status indicators** - Green/red badges for connection status
- ✅ **Sync on refresh** - Update groups list on demand
- ✅ **Beautiful modern UI** - Purple/blue gradient design
- ✅ **Demo & real modes** - Test without Signal or use real integration
- ✅ **Character counter** - Track message length
- ✅ **Loading states** - Visual feedback during operations

Technical Features ✅

- ✅ React 18 frontend with modern hooks
- ✅ Express backend with comprehensive error handling
- ✅ Signal API integration via Docker
- ✅ CORS configured for cross-origin requests
- ✅ Morgan logging for HTTP requests
- ✅ Environment-based configuration
- ✅ Axios for HTTP client
- ✅ One-command startup scripts
- ✅ Automatic browser opening

What Works & What Doesn't

🎯 The Reality of signal-cli

✅ What signal-cli (and wrappers) CAN Do:

- ✅ **Send messages to groups** - Works perfectly!
- ✅ **Create groups** - Full support
- ✅ **Add members to groups** - Members get invited
- ✅ **When they accept on THEIR phone, it works** - Full functionality
- ✅ **List all groups** - Complete group information
- ✅ **Sync groups** - Keep groups up to date
- ✅ **Send to multiple recipients** - Broadcast messages
- ✅ **Group administration** - Manage group settings

❌ What signal-cli Struggles With:

- ❌ **Cannot accept invite links** (we can only create groups to be in them)
- ❌ **Cannot invite people to already existing groups** (only admin members can invite)
- ❌ **Accepting invitations FROM the project** - Invites must be accepted on phone
- ❌ **Some GroupsV2 advanced features** - Limited compared to mobile app
- ❌ **Certain group operations** - Some features unavailable via API

🛠️ Working API Operations

✅ 1. Create Group with Members

Command:

```
curl -X POST http://localhost:8080/v1/groups/+[YOUR_NUMBER] \
  -H 'Content-Type: application/json' \
  -d '{
    "name": "Group Name",
    "members": ["+40[PHONE_1]", "+40[PHONE_2]"]
  }'
```

Result:

- ✅ All members get invited
- ✅ Works perfectly
- ✅ Group is created immediately

✅ 2. Send Messages

Command:

```
curl -X POST http://localhost:8080/v2/send \
  -H 'Content-Type: application/json' \
  -d '{
    "message": "Hello!",
    "number": "+[YOUR_NUMBER]",
    "recipients": ["GROUP_ID"]
  }'
```

Result:

- ✅ Everyone sees messages
- ✅ Works great
- ✅ Messages sync to all devices

❌ What DOESN'T WORK:

Invite Links via API:

- ❌ signal-cli limitation
- ❌ Links stay empty
- ❌ `resetLink: true` doesn't generate them
- **Workaround:** Create groups directly with members

Quick Start Guide

🚀 Fastest Way (30 seconds)

```
cd /Users/thinslicesacademy8/projects/Signal_PoC
./START_PROJECT.sh
```

This single command:

1. ✅ Starts Signal API (Docker)
2. ✅ Starts backend server
3. ✅ Starts frontend app
4. ✅ Opens browser automatically to http://localhost:3000

To Stop:

```
./STOP_PROJECT.sh
```

Prerequisites

Required Software

Before you begin, ensure you have installed:

1. **Docker Desktop**
   - Download: https://www.docker.com/products/docker-desktop

   - Required for running signal-cli-rest-api
   - Verify: `docker --version`
   - Expected output: `Docker version 20.x.x` or higher

2. **Node.js** (v16 or higher)

   - Download: https://nodejs.org/
   - Verify: `node --version`
   - Expected output: `v16.x.x` or higher

3. **npm** (comes with Node.js)

   - Verify: `npm --version`
   - Expected output: `8.x.x` or higher

Required Resources

- ✅ **Phone number** for Signal (can receive SMS)
- ✅ **Signal mobile app** installed (iOS or Android)
- ✅ **Internet connection** on both development machine and phone

System Requirements

- **OS:** macOS, Linux, or Windows (with WSL2 for Docker)
- **RAM:** 4GB minimum, 8GB recommended
- **Disk Space:** 2GB free space
- **Ports Available:** 3000, 5001, 8080

Initial Setup (First Time Only)

⚙️ Prerequisites & Initial Setup

IMPORTANT: You MUST register your phone number as a **primary account** (not linked device) for the app to work properly.

Step 1: Install Dependencies

```
cd /Users/thinslicesacademy8/projects/Signal_PoC

# Install root dependencies
npm install

# Install backend dependencies
cd signal-poc/backend
npm install

# Install frontend dependencies
cd ../frontend
npm install
```

Step 2: Start Docker

```
cd /Users/thinslicesacademy8/projects/Signal_PoC/signal-api
docker-compose up -d
```

Verify it's running:

```
docker ps | grep signal
# Should show: signal-api container running

curl http://localhost:8080/v1/health
# Should return: {"status":"ok"}
```

Step 3: Register Your Phone Number

Important: Use a **dedicated phone number**, NOT your personal Signal number!

```
cd /Users/thinslicesacademy8/projects/Signal_PoC/signal-api
./register.sh
```

The script will:

1. Prompt you to enter your phone number (format: `+[YOUR_NUMBER]`)
2. Send a registration request to Signal
3. You'll receive an SMS with a 6-digit verification code

Example:

```
$ ./register.sh
Enter your phone number (with country code, e.g., +[YOUR_NUMBER]): +
[YOUR_NUMBER]
✅ Registration request sent!
📱 Check your phone for SMS with verification code
```

Step 4: Verify Your Number

After receiving the SMS code:

```
./verify.sh
```

The script will:

1. Prompt you to enter the 6-digit code from SMS
2. Verify your number with Signal
3. Complete registration

Example:

```
$ ./verify.sh
Enter the 6-digit verification code: 123456
✅ Verification successful!
Your Signal number is now registered!
```

Step 5: Sync Your Groups

```
./sync-groups.sh
```

This fetches all your Signal groups from the Signal network.

Example output:

```
$ ./sync-groups.sh
Syncing groups for +[YOUR_NUMBER]...
✅ Groups synced successfully!
Found 3 groups:
  - Project Team (5 members)
  - Family Chat (4 members)
  - Test Group (2 members)
```

Step 6: Configure Backend

```
cd /Users/thinslicesacademy8/projects/Signal_PoC/signal-poc/backend

# Create .env file (if not exists)
cp env.example .env

# Edit .env with your number
nano .env
```

Set your configuration:

```
PORT=5001
SIGNAL_API_URL=http://localhost:8080
SIGNAL_NUMBER=+[YOUR_NUMBER]
```

✅ Setup Complete!

Now you're ready! Your number is registered as primary and Docker is running.

Note:

- You only need to do this registration **once**
- Docker must be running whenever you use the app
- Use primary registration, NOT device linking for full functionality

---

Architecture Overview

System Architecture

```
┌─────────────────────────────────────────────────────────────┐
│  User Browser (http://localhost:3000)                       │
│                                                             │
│  ┌───────────────────────────────────────────────────────┐  │
│  │  React Frontend                                       │  │
│  │  - Group selection UI                                 │  │
│  │  - Message composition                                │  │
│  │  - Status indicators                                  │  │
│  │  - Error display                                      │  │
│  └───────────────────────────────────────────────────────┘  │
└─────────────────────────────────────────────────────────────┘
                          │
                          │ HTTP /api/* (proxied)
                          ↓
┌─────────────────────────────────────────────────────────────┐
│  Express Backend (http://localhost:5001)                    │
│                                                             │
│  ┌───────────────────────────────────────────────────────┐  │
│  │  Node.js + Express Server                             │  │
│  │  - GET  /api/health    → Health check                 │  │
│  │  - GET  /api/groups    → Fetch groups                 │  │
│  │  - POST /api/send      → Send message                 │  │
│  │  - POST /api/sync      → Sync with Signal             │  │
│  │  - GET  /api/config    → Get configuration            │  │
│  └───────────────────────────────────────────────────────┘  │
└─────────────────────────────────────────────────────────────┘
                          │
                          │ HTTP Requests
                          ↓
┌─────────────────────────────────────────────────────────────┐
│  Signal API (http://localhost:8080)                         │
│                                                             │
│  ┌───────────────────────────────────────────────────────┐  │
│  │  signal-cli-rest-api (Docker Container)               │  │
│  │  - RESTful wrapper for signal-cli                     │  │
│  │  - Handles Signal protocol                            │  │
│  │  - Manages encryption/decryption                      │  │
│  │  - Stores Signal data                                 │  │
│  └───────────────────────────────────────────────────────┘  │
└─────────────────────────────────────────────────────────────┘
                          │
                          │ Signal Protocol (encrypted)
                          ↓
```

```
┌───────────────────────────────────────────────────────────┐
│  Signal Servers (Official Infrastructure)                 │
│  - Message routing                                        │
│  - Group management                                       │
│  - End-to-end encryption                                  │
└───────────────────────────────────────────────────────────┘
```

Component Breakdown

1. Frontend (React + Vite)

Location: `signal-poc/frontend/`

Technologies:

- React 18 (UI framework)
- Vite (build tool, dev server)
- Axios (HTTP client)
- Vanilla CSS (styling)

Key Files:

- `src/App.jsx` - Main React component with all UI logic
- `src/App.css` - Component styles
- `vite.config.js` - Proxy configuration to backend

Features:

- Group list display
- Message composition
- Send button with loading states
- Error/success alerts
- Status indicators
- Refresh functionality

2. Backend (Node.js + Express)

Location: `signal-poc/backend/`

Technologies:

- Express (web framework)
- Axios (HTTP client for Signal API)
- Morgan (HTTP request logger)
- dotenv (environment configuration)
- CORS (cross-origin requests)

Key Files:

- `server.js` - Main server with all endpoints
- `.env` - Configuration (not in git)

- `package.json` - Dependencies

Features:

- RESTful API endpoints
- Error handling with detailed logging
- Health checks
- Request validation
- Group synchronization

3. Signal API (Docker)

Location: `signal-api/`

Technologies:

- Docker Compose
- signal-cli-rest-api (bbernhard/signal-cli-rest-api)
- signal-cli (AsamK/signal-cli)

Key Files:

- `docker-compose.yml` - Container configuration
- `signal-cli-config/` - Signal data storage (gitignored)
- Helper scripts for registration and management

Features:

- Signal protocol implementation
- RESTful API for Signal operations
- Message encryption/decryption
- Group management
- Account registration

Data Flow

Sending a Message:

```
1. User types message in React UI
2. User clicks "Send Message" button
3. Frontend sends POST to /api/send with {groupId, message}
4. Backend validates request
5. Backend calls Signal API: POST /v2/send
6. Signal API encrypts message using Signal protocol
7. Signal API sends to Signal servers
8. Signal servers route to all group members
9. Members receive message on their devices
10. Backend returns success to frontend
11. Frontend shows success alert
```

Fetching Groups:

```
1. User opens app or clicks "Refresh"
2. Frontend sends GET to /api/groups
3. Backend calls Signal API: GET /v1/groups/{number}
4. Signal API retrieves groups from local storage
5. Signal API returns group list
6. Backend formats and returns groups
7. Frontend displays groups in UI
```

Port Usage

Service	Port	URL	Purpose
Frontend	3000	http://localhost:3000	User interface
Backend	5001	http://localhost:5001	API server
Signal API	8080	http://localhost:8080	Signal protocol handler

File Structure

```
Signal_PoC/
├── signal-api/                     # Signal API (Docker)
│   ├── docker-compose.yml          # Docker configuration
│   ├── signal-cli-config/          # Signal data (gitignored)
│   ├── register.sh                 # Registration script
│   ├── verify.sh                   # Verification script
│   ├── sync-groups.sh              # Sync groups script
│   └── README.md                   # Signal API docs
│
├── signal-poc/                     # Web Application
│   ├── backend/
│   │   ├── server.js               # Express server
│   │   ├── demo-mode.js            # Demo server
│   │   ├── package.json            # Backend dependencies
│   │   ├── .env                    # Configuration (not in git)
│   │   └── env.example             # Configuration template
│   │
│   ├── frontend/
│   │   ├── src/
│   │   │   ├── App.jsx             # Main React component
│   │   │   ├── App.css             # Styles
│   │   │   ├── main.jsx            # React entry
│   │   │   └── index.css           # Global styles
│   │   ├── index.html              # HTML template
│   │   ├── package.json            # Frontend dependencies
│   │   └── vite.config.js          # Vite configuration
│   │
│   ├── start-all.sh                # Start backend + frontend
│   ├── switch-mode.sh           # Toggle demo/real mode
│   └── README.md                # Web app documentation
│
├── START_PROJECT.sh             # Start everything
├── STOP_PROJECT.sh              # Stop everything
├── package.json                 # Root dependencies
├── COMPLETE_DOCUMENTATION.md    # This file
└── PROJECT_COMPLETE.md          # Project summary

Documentation Files:
├── COMMANDS.md                  # Command reference
├── START_HERE.md                # Quick start guide
├── GETTING_STARTED.md           # Setup guide
└── signal_documentation/        # Deep dive guides
```

How to Run the Project

Method 1: One Command (Recommended)

Start Everything:

```
cd /Users/thinslicesacademy8/projects/Signal_PoC
./START_PROJECT.sh
```

This will:

1. Start Signal API Docker container
2. Wait for it to be healthy
3. Start backend server (port 5001)
4. Start frontend dev server (port 3000)
5. Open browser to http://localhost:3000

Stop Everything:

```
./STOP_PROJECT.sh
```

Method 2: Manual Step-by-Step

Terminal 1 - Signal API:

```
cd /Users/thinslicesacademy8/projects/Signal_PoC/signal-api
docker-compose up -d

# Verify
curl http://localhost:8080/v1/health
```

Terminal 2 - Backend:

```
cd /Users/thinslicesacademy8/projects/Signal_PoC/signal-poc/backend
npm start

# Expected output:
# Server running on port 5001
# ✅ Signal API connection successful
```

Terminal 3 - Frontend:

```
cd /Users/thinslicesacademy8/projects/Signal_PoC/signal-poc/frontend
npm run dev

# Expected output:
# Local: http://localhost:3000
```

Browser:

```
Open: http://localhost:3000
```

Method 3: Using Individual Scripts

Start Web App Only (assumes Signal API is running):

```
cd /Users/thinslicesacademy8/projects/Signal_PoC/signal-poc
./start-all.sh
```

Method 4: Demo Mode (No Signal Required)

Switch to demo mode:

```
cd /Users/thinslicesacademy8/projects/Signal_PoC/signal-poc
./switch-mode.sh demo
```

Start:

```
cd /Users/thinslicesacademy8/projects/Signal_PoC
./START_PROJECT.sh
```

Demo mode features:

- Mock groups (3 sample groups)
- Simulated sending (no actual Signal)
- Perfect for demonstrations
- No Docker or Signal registration needed

Daily Workflow

Morning - Start Work:

```
cd /Users/thinslicesacademy8/projects/Signal_PoC
./START_PROJECT.sh
```

During Work:

- Access at http://localhost:3000
- Send messages as needed
- Monitor logs in terminal

End of Day:

```
./STOP_PROJECT.sh
```

Verify Everything Is Running

```
# Check Docker
docker ps | grep signal
# Expected: signal-api container running

# Check Backend
curl http://localhost:5001/api/health
# Expected: {"status":"ok","backend":"running",...}

# Check Frontend
curl http://localhost:3000
# Expected: HTML content

# Check all ports
lsof -i :8080  # Signal API
lsof -i :5001  # Backend
lsof -i :3000  # Frontend
```

Commands Reference

🚀 Start/Stop Commands

Start Everything

```
cd /Users/thinslicesacademy8/projects/Signal_PoC
./START_PROJECT.sh
```

Stop Everything

```
./STOP_PROJECT.sh
```

Restart Everything

```
./STOP_PROJECT.sh && ./START_PROJECT.sh
```

🐳 Docker Commands

Start Signal API

```
cd signal-api
docker-compose up -d
```

Stop Signal API

```
cd signal-api
docker-compose down
```

Restart Signal API

```
cd signal-api
docker-compose restart
```

View Signal API Logs

```
docker logs signal-api --tail 50 --follow
```

Check Docker Status

```
docker ps | grep signal
```

🔧 Backend Commands

Start Backend

```
cd signal-poc/backend
npm start
```

Start Backend with Specific Config

```
cd signal-poc/backend
PORT=5001 SIGNAL_NUMBER="+[YOUR_NUMBER]"
SIGNAL_API_URL="http://localhost:8080" node server.js
```

Kill Backend

```
pkill -f "node server.js"
```

🎨 Frontend Commands

Start Frontend

```
cd signal-poc/frontend
npm run dev
```

Build Frontend for Production

```
cd signal-poc/frontend
npm run build
# Output: dist/
```

Preview Production Build

```
cd signal-poc/frontend
npm run preview
```

Kill Frontend

```
pkill -f "vite"
```

📱 Signal Operations

Register Phone Number

```
cd signal-api
./register.sh
# Follow prompts
```

Verify SMS Code

```
cd signal-api
./verify.sh
# Follow prompts
```

Sync Groups

```
cd signal-api
./sync-groups.sh
```

Check if Number is Registered

```
curl http://localhost:8080/v1/groups/+[YOUR_NUMBER]
```

🎮 Mode Switching

Switch to Demo Mode

```
cd signal-poc
./switch-mode.sh demo
```

Switch to Real Mode

```
cd signal-poc
./switch-mode.sh real
```

📊 Monitoring Commands

Check All Services Status

```
# Docker
docker ps | grep signal

# Backend
ps aux | grep "node server.js" | grep -v grep

# Frontend
ps aux | grep "vite" | grep -v grep

# All ports
lsof -i :8080,5001,3000
```

Test All Endpoints

```
# Signal API Health
curl http://localhost:8080/v1/health

# Backend Health
curl http://localhost:5001/api/health

# Get Groups
curl http://localhost:5001/api/groups

# Frontend
curl http://localhost:3000
```

🐛 Troubleshooting Commands

Kill Everything and Restart

```
# Kill all Node processes
pkill -f "node"

# Stop Docker
cd signal-api && docker-compose down
```

```
# Wait a moment
sleep 2

# Start fresh
cd ..
./START_PROJECT.sh
```

Kill Process on Specific Port

```
# Find process
lsof -i :5001

# Kill by port
lsof -ti :5001 | xargs kill -9
```

Clear Signal Data (Nuclear Option)

```
cd signal-api
docker-compose down
rm -rf signal-cli-config/data/*
docker-compose up -d
# Then register number again
```

View All Logs

```
# Signal API
docker logs signal-api --tail 100

# Backend (if logging to file)
tail -f signal-poc/backend.log

# Frontend (check browser console F12)
```

📡 API Testing Commands

Send Message via curl

```
curl -X POST http://localhost:5001/api/send \
  -H 'Content-Type: application/json' \
  -d '{
    "groupId": "YOUR_GROUP_ID",
    "message": "Hello from terminal!"
  }'
```

Get Groups via curl

```
curl http://localhost:5001/api/groups | jq '.'
```

Health Check All Services

```
echo "Signal API:" && curl -s http://localhost:8080/v1/health
echo "\nBackend:" && curl -s http://localhost:5001/api/health
echo "\nFrontend:" && curl -s -o /dev/null -w "%{http_code}"
http://localhost:3000
```

🔍 Debugging Commands

Check Environment Variables

```
cd signal-poc/backend
cat .env
```

Test Signal API Directly

```
# List groups
curl http://localhost:8080/v1/groups/+[YOUR_NUMBER]

# Send message directly
curl -X POST http://localhost:8080/v2/send \
  -H 'Content-Type: application/json' \
  -d '{
    "message": "Test",
    "number": "+[YOUR_NUMBER]",
    "recipients": ["group.YOUR_GROUP_ID"]
  }'
```

Command Output Examples

Successful Start

```
$ ./START_PROJECT.sh
Starting Signal PoC...
✅ Signal API is running
✅ Backend started on port 5001
✅ Frontend started on port 3000
🌐 Opening browser...
✅ Project started successfully!
```

Successful Health Check

```
$ curl http://localhost:5001/api/health
{
  "status": "ok",
  "backend": "running",
  "signalApi": {
    "status": "ok"
  }
}
```

Successful Group Fetch

```
$ curl http://localhost:5001/api/groups
{
  "success": true,
  "groups": [
    {
      "id": "group.abc123...",
      "name": "Project Team",
      "members": ["+[YOUR_NUMBER]", "+40[MEMBER_1]"],
      "memberCount": 2,
      "isAdmin": true,
      "isMember": true
    }
  ],
  "count": 1
}
```

API Documentation

Backend API Endpoints

1. Health Check

Endpoint: `GET /api/health`

Purpose: Check if backend and Signal API are running

Request:

```
curl http://localhost:5001/api/health
```

Response (Success):

```
{
  "status": "ok",
  "backend": "running",
  "signalApi": {
    "status": "ok"
  }
}
```

Response (Signal API Down):

```
{
  "status": "error",
  "backend": "running",
  "signalApi": "unreachable",
  "error": {
    "timestamp": "2025-10-14T10:30:00.000Z",
    "context": "Health Check",
    "message": "connect ECONNREFUSED 127.0.0.1:8080"
  }
}
```

---

2. Get Configuration

Endpoint: `GET /api/config`

Purpose: Get backend configuration (masked sensitive data)

Request:

```
curl http://localhost:5001/api/config
```

Response:

```
{
  "signalApiUrl": "http://localhost:8080",
  "signalNumberConfigured": true,
  "signalNumberMasked": "+[YOUR...BER]"
}
```

---

3. Fetch Groups

Endpoint: `GET /api/groups`

Purpose: Retrieve all Signal groups

Request:

```
curl http://localhost:5001/api/groups
```

Response (Success):

```
{
  "success": true,
  "groups": [
    {
      "id": "group.abc123def456...",
      "internalId": "internal-id-here",
      "name": "Project Team",
      "members": [
        "+[YOUR_NUMBER]",
        "+40[MEMBER_1]"
      ],
      "memberCount": 2,
      "isAdmin": true,
      "isMember": true,
      "isBlocked": false
    },
    {
      "id": "group.xyz789...",
      "internalId": "another-internal-id",
      "name": "Family Chat",
      "members": [
        "+[YOUR_NUMBER]",
        "+40[MEMBER_2]",
        "+40[MEMBER_3]"
      ],
      "memberCount": 3,
      "isAdmin": false,
      "isMember": true,
      "isBlocked": false
    }
  ],
  "count": 2
}
```

Response (Error):

```
{
  "success": false,
  "error": "Failed to fetch groups",
  "details": "SIGNAL_NUMBER not configured",
  "hint": "Configure SIGNAL_NUMBER in .env file"
}
```

Group Object Fields:

- `id` - Full group ID (use this for sending messages)
- `internalId` - Internal identifier (for reference)
- `name` - Group name
- `members` - Array of phone numbers
- `memberCount` - Number of members
- `isAdmin` - Whether you're an admin
- `isMember` - Whether you're a member
- `isBlocked` - Whether group is blocked

---

4. Send Message

Endpoint: `POST /api/send`

Purpose: Send a message to a Signal group

Request:

```
curl -X POST http://localhost:5001/api/send \
  -H 'Content-Type: application/json' \
  -d '{
    "groupId": "group.abc123def456...",
    "message": "Hello from the API!"
  }'
```

Request Body:

```
{
  "groupId": "group.abc123def456...",  // Required: Full group ID
  "message": "Your message here"        // Required: Message text
}
```

Response (Success):

```
{
  "success": true,
  "message": "Message sent successfully",
  "timestamp": 1697284800000
}
```

Response (Error - Missing Fields):

```
{
  "success": false,
  "error": "Missing required fields",
  "details": "groupId and message are required"
}
```

Response (Error - Signal API Failed):

```
{
  "success": false,
  "error": "Failed to send message",
  "details": "Network error: connect ECONNREFUSED",
  "hint": "Ensure Signal API Docker container is running"
}
```

Notes:

- Message can be multi-line
- Maximum length depends on Signal limits (~1000 characters)
- Group ID must include "group." prefix
- Uses Signal API v2 endpoint for sending

---

5. Sync Groups

Endpoint: `POST /api/sync`

Purpose: Force sync with Signal API to fetch latest groups

Request:

```
curl -X POST http://localhost:5001/api/sync
```

Response (Success):

```
{
  "success": true,
  "message": "Groups synced successfully"
}
```

Response (Error):

```
{
  "success": false,
  "error": "Failed to sync",
  "details": "Request timeout"
}
```

Signal API Endpoints (Direct)

These are the underlying Signal API endpoints. The backend proxies to these.

Get Groups

```
GET http://localhost:8080/v1/groups/{phoneNumber}
```

Send Message

```
POST http://localhost:8080/v2/send
Body: {
  "message": "text",
  "number": "+[YOUR_NUMBER]",
  "recipients": ["group.id"]
}
```

Register Number

```
POST http://localhost:8080/v1/register/{phoneNumber}
Body: {"use_voice": false}
```

Verify Registration

```
POST http://localhost:8080/v1/register/{phoneNumber}/verify/{code}
```

Health Check

```
GET http://localhost:8080/v1/health
```

Receive Messages

```
GET http://localhost:8080/v1/receive/{phoneNumber}?timeout=30
```

Using the API from Code

JavaScript/Node.js Example

```javascript
const axios = require('axios');

const BASE_URL = 'http://localhost:5001';

// Fetch groups
async function getGroups() {
  try {
    const response = await axios.get(`${BASE_URL}/api/groups`);
    console.log('Groups:', response.data.groups);
    return response.data.groups;
  } catch (error) {
    console.error('Error:', error.response?.data);
  }
}

// Send message
async function sendMessage(groupId, message) {
  try {
    const response = await axios.post(`${BASE_URL}/api/send`, {
      groupId,
      message
    });
    console.log('Success:', response.data);
    return response.data;
  } catch (error) {
    console.error('Error:', error.response?.data);
  }
}

// Usage
(async () => {
  const groups = await getGroups();
  if (groups && groups.length > 0) {
    await sendMessage(groups[0].id, 'Hello from Node.js!');
  }
})();
```

Python Example

```python
import requests

BASE_URL = 'http://localhost:5001'

def get_groups():
    """Fetch all groups"""
    try:
        response = requests.get(f'{BASE_URL}/api/groups')
        response.raise_for_status()
        data = response.json()
        print(f"Found {data['count']} groups")
        return data['groups']
    except requests.exceptions.RequestException as e:
        print(f"Error: {e}")
        return []

def send_message(group_id, message):
    """Send message to group"""
    try:
        response = requests.post(
            f'{BASE_URL}/api/send',
            json={
                'groupId': group_id,
                'message': message
            }
        )
        response.raise_for_status()
        print("Message sent successfully!")
        return response.json()
    except requests.exceptions.RequestException as e:
        print(f"Error: {e}")
        return None

# Usage
if __name__ == '__main__':
    groups = get_groups()
    if groups:
        send_message(groups[0]['id'], 'Hello from Python!')
```

curl Examples

Fetch and parse groups with jq:

```
curl -s http://localhost:5001/api/groups | jq '.groups[] | {name, id,
memberCount}'
```

Send message to first group:

```
GROUP_ID=$(curl -s http://localhost:5001/api/groups | jq -r
'.groups[0].id')
curl -X POST http://localhost:5001/api/send \
  -H 'Content-Type: application/json' \
  -d "{\"groupId\": \"$GROUP_ID\", \"message\": \"Test from bash\"}"
```

Loop through all groups:

```
curl -s http://localhost:5001/api/groups | jq -c '.groups[]' | while read
group; do
  name=$(echo $group | jq -r '.name')
  id=$(echo $group | jq -r '.id')
  echo "Group: $name (ID: $id)"
done
```

How to Create Groups

Method 1: Using Signal Mobile App (Recommended)

This is the **easiest and most reliable** method.

Step 1: Open Signal App

1. Open Signal on your mobile device
2. Tap the pencil/compose icon
3. Select "New group"

Step 2: Add Members

1. Select contacts to add to the group
2. You can search by name or phone number
3. Select at least one contact
4. Tap "Next"

Step 3: Name the Group

1. Enter a group name
2. Optionally add a group photo
3. Tap "Create"

Step 4: Sync to Web App

The group will automatically appear in the web app!

To see it immediately:

```
# Option 1: Use sync script
cd signal-api
./sync-groups.sh

# Option 2: Click "Refresh" button in web UI

# Option 3: Restart the app
cd ..
./STOP_PROJECT.sh
./START_PROJECT.sh
```

Method 2: Using API (Advanced)

You can also create groups programmatically using the Signal API.

Create Group with Members

```
curl -X POST http://localhost:8080/v1/groups/+[YOUR_NUMBER] \
  -H 'Content-Type: application/json' \
  -d '{
    "name": "API Created Group",
    "members": ["+40[MEMBER_1]", "+40[MEMBER_2]"]
  }'
```

Response:

```
{
  "id": "group.abc123...",
  "name": "API Created Group",
  "members": ["+[YOUR_NUMBER]", "+40[MEMBER_1]", "+40[MEMBER_2]"],
  "admins": ["+[YOUR_NUMBER]"]
}
```

Notes:

- ✅ Your number is automatically added as admin
- ✅ All members get invited
- ⚠ Members must accept invite on their phones
- ⚠ Group appears immediately in your app

Method 3: Using Helper Script

Create a helper script for easy group creation:

File: `signal-api/create-group.sh`

```bash
#!/bin/bash

# Create a new Signal group

read -p "Enter group name: " GROUP_NAME
read -p "Enter member phone (e.g., +40[MEMBER_2]): " MEMBER_PHONE

curl -X POST http://localhost:8080/v1/groups/+[YOUR_NUMBER] \
  -H 'Content-Type: application/json' \
  -d "{
    \"name\": \"$GROUP_NAME\",
    \"members\": [\"$MEMBER_PHONE\"]
  }"

echo ""
echo "✅ Group created! Syncing..."
./sync-groups.sh
```

Make it executable:

```bash
chmod +x signal-api/create-group.sh
```

Use it:

```bash
cd signal-api
./create-group.sh
```

Important Notes About Groups

✅ What Works:

- Creating groups with members via API
- Members get invited automatically
- You're automatically the admin
- Group appears in your web app immediately

⚠ Important Limitations:

- **Cannot accept invite links** - signal-cli limitation
- **Cannot invite to existing groups you didn't create** - only admins can invite
- **Members must accept on their phones** - cannot accept programmatically
- **Group links don't work via API** - `resetLink: true` doesn't generate them

💡 **Best Practices:**

1. **Use mobile app for initial group creation** - Most reliable
2. **Add members when creating** - Don't try to add later
3. **Sync after creating** - Run `./sync-groups.sh` or click refresh
4. **Test with small groups first** - Ensure everything works
5. **Keep groups simple** - Basic features work best

Verifying Group Creation

Check if group was created:

```
# Method 1: Via backend API
curl http://localhost:5001/api/groups

# Method 2: Via Signal API directly
curl http://localhost:8080/v1/groups/+[YOUR_NUMBER]

# Method 3: In web UI
# Open http://localhost:3000 and click "Refresh"
```

Send test message to new group:

```
# Get the group ID from the response above
GROUP_ID="group.abc123..."

# Send test message
curl -X POST http://localhost:5001/api/send \
  -H 'Content-Type: application/json' \
  -d "{
    \"groupId\": \"$GROUP_ID\",
    \"message\": \"Welcome to the group!\"
  }"
```

Environment Configuration

Backend Configuration (.env)

Location: `signal-poc/backend/.env`

Template: `signal-poc/backend/env.example`

Required Variables:

```
# Server port
PORT=5001

# Signal API URL
SIGNAL_API_URL=http://localhost:8080

# Your Signal phone number (E.164 format)
SIGNAL_NUMBER=+[YOUR_NUMBER]
```

Creating the .env file:

```
cd signal-poc/backend
cp env.example .env
nano .env  # or use your preferred editor
```

Environment Variable Descriptions:

Variable	Required	Default	Description
`PORT`	No	5001	Port for backend server
`SIGNAL_API_URL`	No	http://localhost:8080	URL of Signal API
`SIGNAL_NUMBER`	Yes	None	Your registered Signal number

Important Notes:

- ✅ Use E.164 format for phone number: `+CountryCode + Number`
- ✅ No spaces in phone number
- ✅ Must match registered number in Signal API
- ❌ Never commit .env to git (it's in .gitignore)

Frontend Configuration

Location: `signal-poc/frontend/vite.config.js`

The frontend proxies API requests to the backend. Configuration:

```
export default defineConfig({
  plugins: [react()],
  server: {
    port: 3000,
    proxy: {
      '/api': {
        target: 'http://localhost:5001',
        changeOrigin: true,
      }
    }
  }
})
```

What this does:

- Frontend runs on port 3000
- Any request to `/api/*` is proxied to `http://localhost:5001/api/*`
- Solves CORS issues
- No frontend environment variables needed

Docker Configuration

Location: `signal-api/docker-compose.yml`

```yaml
version: "3"
services:
  signal-cli-rest-api:
    image: bbernhard/signal-cli-rest-api:latest
    container_name: signal-api
    ports:
      - "8080:8080"
    volumes:
      - "./signal-cli-config:/home/.local/share/signal-cli"
    environment:
      - MODE=native
    restart: unless-stopped
```

Configuration Options:

Option	Value	Purpose
`image`	bbernhard/signal-cli-rest-api:latest	Docker image
`container_name`	signal-api	Container name
`ports`	8080:8080	Port mapping (host:container)
`volumes`	./signal-cli-config:/home/.local/...	Data persistence
`MODE`	native	Signal client mode
`restart`	unless-stopped	Auto-restart policy

Customization:

Change port:

```
ports:
  - "9090:8080"  # Access via localhost:9090
```

Add environment variables:

```
environment:
  - MODE=native
  - AUTO_RECEIVE_SCHEDULE=0 */2 * * *  # Receive every 2 minutes
```

Production Configuration

For production deployment, consider:

1. **Use environment-specific .env files:**

```
# Development
.env.development

# Production
.env.production
```

2. **Use Docker Compose for all services:**

```
version: "3"
services:
  signal-api:
    # ... existing config

  backend:
    build: ./signal-poc/backend
    ports:
      - "5001:5001"
    env_file:
      - ./signal-poc/backend/.env
    depends_on:
      - signal-api

  frontend:
    build: ./signal-poc/frontend
    ports:
      - "80:80"
    depends_on:
      - backend
```

3. **Use secrets management:**

```
# Use Docker secrets or environment variable services
# Never hardcode sensitive values
```

4. **Configure logging:**

```
// backend/server.js
const winston = require('winston');

const logger = winston.createLogger({
  level: process.env.LOG_LEVEL || 'info',
  format: winston.format.json(),
  transports: [
    new winston.transports.File({ filename: 'error.log', level: 'error'
}),
    new winston.transports.File({ filename: 'combined.log' })
  ]
});
```

Configuration Validation

Check if configuration is correct:

```
# 1. Check .env file exists
ls -la signal-poc/backend/.env

# 2. Check .env contents (without exposing values)
cd signal-poc/backend
grep -E "^(PORT|SIGNAL_API_URL|SIGNAL_NUMBER)=" .env

# 3. Test backend configuration endpoint
curl http://localhost:5001/api/config

# Expected output:
# {
#   "signalApiUrl": "http://localhost:8080",
#   "signalNumberConfigured": true,
#   "signalNumberMasked": "+[YOUR...BER]"
# }
```

Troubleshooting

Common Issues & Solutions

1. "Cannot connect to Signal API"

Symptoms:

- Backend shows "Signal API: unreachable"
- Red status indicator in UI
- Error: `connect ECONNREFUSED 127.0.0.1:8080`

Solutions:

```
# Check if Docker container is running
docker ps | grep signal

# If not running, start it
cd signal-api
docker-compose up -d

# Check logs
docker logs signal-api

# Test directly
curl http://localhost:8080/v1/health
```

---

2. "SIGNAL_NUMBER not configured"

Symptoms:

- Backend returns error when fetching groups
- Error message: "SIGNAL_NUMBER not configured in .env file"

Solution:

```
# Check if .env exists
ls signal-poc/backend/.env

# If not, create it
cd signal-poc/backend
cp env.example .env

# Edit and add your number
nano .env
# Add: SIGNAL_NUMBER=+[YOUR_NUMBER]

# Restart backend
pkill -f "node server.js"
npm start
```

---

3. "No groups found"

Symptoms:

- Empty groups list in UI
- Message: "No groups available"

Solutions:

```
# 1. Create a group in Signal mobile app first

# 2. Sync groups
cd signal-api
./sync-groups.sh

# 3. Refresh UI
# Click "Refresh" button in browser

# 4. Check if number is registered
curl http://localhost:8080/v1/groups/+[YOUR_NUMBER]
```

---

4. "Port already in use"

Symptoms:

- Error: `EADDRINUSE: address already in use :::5001`
- Service won't start

Solutions:

```
# Find what's using the port
lsof -i :5001

# Kill the process
lsof -ti :5001 | xargs kill -9

# Or change port in .env
echo "PORT=5002" >> signal-poc/backend/.env
```

---

5. "Message not sending"

Symptoms:

- Click "Send" but message doesn't appear
- Error in console or UI

Solutions:

A. Check group ID format:

```
# Group ID should start with "group."
# Correct: group.abc123def456...
# Wrong: abc123def456...

# Verify in API response
curl http://localhost:5001/api/groups | jq '.groups[].id'
```

B. Check Signal API:

```
# Send directly to Signal API
curl -X POST http://localhost:8080/v2/send \
  -H 'Content-Type: application/json' \
  -d '{
    "message": "Test",
    "number": "+[YOUR_NUMBER]",
    "recipients": ["group.YOUR_GROUP_ID"]
  }'
```

C. Check logs:

```
# Backend logs
tail -f signal-poc/backend.log

# Signal API logs
docker logs signal-api --tail 50
```

---

6. "Docker container keeps restarting"

Symptoms:

- `docker ps` shows container restarting
- Signal API not accessible

Solutions:

```
# Check logs
docker logs signal-api

# Common cause: corrupted data
cd signal-api
docker-compose down
rm -rf signal-cli-config/data/*
docker-compose up -d
```

```
# Re-register number
./register.sh
./verify.sh
```

---

7. "Frontend shows blank page"

Symptoms:

- Browser shows empty page
- No errors in console

Solutions:

```
# Check if frontend is running
ps aux | grep vite

# Check port
lsof -i :3000

# Restart frontend
cd signal-poc/frontend
pkill -f "vite"
npm run dev

# Check for build errors
npm run build
```

---

8. "Messages appear as 'Note to Self'"

Cause: Group only has you as a member.

Solution:

1. Open Signal mobile app
2. Open the group
3. Add at least one other contact
4. Try sending again

Note: Messages to groups with only you as a member will always appear as notes.

---

9. "Cannot accept group invites"

This is a signal-cli limitation!

Workaround:

- Accept invites on your phone, not via API
- Or create new groups instead (you'll be admin automatically)

---

10. "Registration fails with captcha error"

Symptoms:

- Registration returns captcha required error

Solution:

```
# 1. Get captcha token
# Visit: https://signalcaptchas.org/registration/generate.html
# Solve captcha and copy the link

# 2. Register with captcha
curl -X POST http://localhost:8080/v1/register/+[YOUR_NUMBER] \
  -H 'Content-Type: application/json' \
  -d '{
    "captcha": "signalcaptcha://signal-recaptcha-v2.YOUR_TOKEN"
  }'

# 3. Verify as usual
```

Debug Checklist

When something doesn't work, check in this order:

1. Docker/Signal API:

```
docker ps | grep signal            # Should be running
curl http://localhost:8080/v1/health  # Should return {"status":"ok"}
```

2. Backend:

```
ps aux | grep "node server.js"     # Should be running
curl http://localhost:5001/api/health # Should return success
cat signal-poc/backend/.env        # Should have SIGNAL_NUMBER
```

3. Frontend:

```
ps aux | grep "vite"               # Should be running
curl http://localhost:3000         # Should return HTML
# Open browser console (F12) for errors
```

4. Network:

```
lsof -i :8080                    # Signal API
lsof -i :5001                    # Backend
lsof -i :3000                    # Frontend
```

5. Configuration:

```
# Check Signal number is registered
curl http://localhost:8080/v1/groups/+[YOUR_NUMBER]

# Check backend config
curl http://localhost:5001/api/config
```

Getting Help

1. Check logs:

```
# Signal API
docker logs signal-api --tail 100

# Backend
# Check terminal where backend is running
# Or: tail -f signal-poc/backend.log

# Frontend
# Press F12 in browser → Console tab
```

2. Verify setup:

```
# Run health checks
curl http://localhost:8080/v1/health
curl http://localhost:5001/api/health

# Check all documentation
ls -la *.md signal-poc/*.md signal-api/*.md
```

3. Clean restart:

```
# Nuclear option - restart everything
./STOP_PROJECT.sh
```

```
sleep 2
./START_PROJECT.sh
```

Extension Guide

Adding New Features

This section is for developers who want to extend the PoC.

---

1. Add New Backend Endpoint

Example: Add endpoint to get single group details

File: `signal-poc/backend/server.js`

```
// Add this new endpoint
app.get('/api/groups/:groupId', async (req, res) => {
  try {
    const { groupId } = req.params;

    if (!SIGNAL_NUMBER) {
      throw new Error('SIGNAL_NUMBER not configured');
    }

    console.log(`Fetching group: ${groupId}`);

    // Fetch all groups
    const response = await axios.get(
      `${SIGNAL_API_URL}/v1/groups/${SIGNAL_NUMBER}`,
      { timeout: 10000 }
    );

    const groups = Array.isArray(response.data) ? response.data : [];
    const group = groups.find(g => g.id === groupId);

    if (!group) {
      return res.status(404).json({
        success: false,
        error: 'Group not found'
      });
    }

    res.json({
      success: true,
      group: {
        id: group.id,
        name: group.name || 'Unnamed Group',
        members: group.members || [],
        memberCount: (group.members || []).length,
        isAdmin: group.isAdmin || false
      }
    });

  } catch (error) {
    const errorInfo = logError('Fetch Single Group', error);
    res.status(500).json({
      success: false,
      error: 'Failed to fetch group',
      details: errorInfo.message
    });
  }
});
```

Usage:

```
curl http://localhost:5001/api/groups/group.abc123...
```

---

2. Add New Frontend Feature

Example: Add message history display

File: `signal-poc/frontend/src/App.jsx`

```
// Add state for message history
const [messageHistory, setMessageHistory] = useState([]);

// Add function to fetch history
const fetchHistory = async () => {
  try {
    const response = await axios.get('/api/history');
    setMessageHistory(response.data.messages);
  } catch (error) {
    console.error('Failed to fetch history:', error);
  }
};

// Add UI component
<div className="message-history">
  <h3>Recent Messages</h3>
  {messageHistory.map((msg, index) => (
    <div key={index} className="history-item">
      <span className="timestamp">{msg.timestamp}</span>
      <span className="message">{msg.text}</span>
    </div>
  ))}
</div>
```

3. Add Message Scheduling

File: `signal-poc/backend/server.js`

```
const schedule = require('node-schedule');

// Store scheduled messages
const scheduledMessages = new Map();

// Endpoint to schedule message
app.post('/api/schedule', async (req, res) => {
  try {
    const { groupId, message, sendAt } = req.body;

    if (!groupId || !message || !sendAt) {
      return res.status(400).json({
        success: false,
        error: 'Missing required fields'
      });
    }

    const scheduleDate = new Date(sendAt);
    const jobId = `${groupId}-${Date.now()}`;

    // Schedule the job
    const job = schedule.scheduleJob(scheduleDate, async () => {
      try {
        await axios.post(`${SIGNAL_API_URL}/v2/send`, {
          message,
          number: SIGNAL_NUMBER,
          recipients: [groupId]
        });
        console.log(`Scheduled message sent: ${jobId}`);
        scheduledMessages.delete(jobId);
      } catch (error) {
        console.error(`Failed to send scheduled message: ${jobId}`,
error);
      }
    });

    scheduledMessages.set(jobId, { job, groupId, message, sendAt });

    res.json({
      success: true,
      jobId,
      scheduledFor: scheduleDate
    });

  } catch (error) {
    logError('Schedule Message', error);
    res.status(500).json({
      success: false,
      error: 'Failed to schedule message'
    });
  }
});

// Endpoint to list scheduled messages
app.get('/api/scheduled', (req, res) => {
  const scheduled = Array.from(scheduledMessages.entries()).map(([id,
data]) => ({
    jobId: id,
    groupId: data.groupId,
    message: data.message,
    sendAt: data.sendAt
  }));

  res.json({
    success: true,
    scheduled
  });
});

// Endpoint to cancel scheduled message
app.delete('/api/schedule/:jobId', (req, res) => {
  const { jobId } = req.params;
  const scheduled = scheduledMessages.get(jobId);

  if (!scheduled) {
    return res.status(404).json({
      success: false,
      error: 'Scheduled message not found'
    });
  }

  scheduled.job.cancel();
  scheduledMessages.delete(jobId);

  res.json({
    success: true,
    message: 'Scheduled message cancelled'
  });
});
```

Install dependency:

```
cd signal-poc/backend
npm install node-schedule
```

---

4. Add File/Image Support

Backend changes:

```
const multer = require('multer');
const upload = multer({ dest: 'uploads/' });

app.post('/api/send-attachment', upload.single('file'), async (req, res)
=> {
  try {
    const { groupId, message } = req.body;
    const file = req.file;

    if (!file) {
      return res.status(400).json({
        success: false,
        error: 'No file uploaded'
      });
    }

    const formData = new FormData();
    formData.append('message', message);
    formData.append('number', SIGNAL_NUMBER);
    formData.append('recipients', groupId);
    formData.append('attachment', fs.createReadStream(file.path));

    const response = await axios.post(
      `${SIGNAL_API_URL}/v2/send`,
      formData,
      {
        headers: formData.getHeaders()
      }
    );

    // Clean up uploaded file
    fs.unlinkSync(file.path);

    res.json({
      success: true,
      message: 'File sent successfully'
    });

  } catch (error) {
    logError('Send Attachment', error);
    res.status(500).json({
      success: false,
      error: 'Failed to send attachment'
    });
  }
});
```

Install dependencies:

```
npm install multer form-data
```

---

5. Add User Authentication

Basic JWT authentication example:

```
const jwt = require('jsonwebtoken');
const JWT_SECRET = process.env.JWT_SECRET || 'your-secret-key';

// Middleware to verify token
const authenticateToken = (req, res, next) => {
  const authHeader = req.headers['authorization'];
  const token = authHeader && authHeader.split(' ')[1];

  if (!token) {
    return res.status(401).json({ error: 'Access denied' });
  }

  jwt.verify(token, JWT_SECRET, (err, user) => {
    if (err) {
      return res.status(403).json({ error: 'Invalid token' });
    }
    req.user = user;
    next();
  });
};

// Login endpoint
app.post('/api/login', (req, res) => {
  const { username, password } = req.body;

  // Verify credentials (use proper password hashing in production!)
  if (username === 'admin' && password === 'password') {
    const token = jwt.sign(
      { username },
      JWT_SECRET,
      { expiresIn: '24h' }
    );

    res.json({
      success: true,
      token
    });
  } else {
    res.status(401).json({
      success: false,
      error: 'Invalid credentials'
    });
  }
});
```

```
// Protect endpoints
app.get('/api/groups', authenticateToken, async (req, res) => {
  // ... existing code
});

app.post('/api/send', authenticateToken, async (req, res) => {
  // ... existing code
});
```

---

6. Add Database Integration

Example with SQLite:

```
npm install sqlite3
```

```
const sqlite3 = require('sqlite3').verbose();
const db = new sqlite3.Database('./signal-poc.db');

// Initialize database
db.serialize(() => {
  db.run(`
    CREATE TABLE IF NOT EXISTS message_history (
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      group_id TEXT,
      group_name TEXT,
      message TEXT,
      sent_at DATETIME DEFAULT CURRENT_TIMESTAMP
    )
  `);
});

// Save message to database
const saveMessage = (groupId, groupName, message) => {
  return new Promise((resolve, reject) => {
    db.run(
      'INSERT INTO message_history (group_id, group_name, message) VALUES
(?, ?, ?)',
      [groupId, groupName, message],
      function(err) {
        if (err) reject(err);
        else resolve(this.lastID);
      }
    );
  });
};

// Get message history
```

```
app.get('/api/history', authenticateToken, async (req, res) => {
  db.all(
    'SELECT * FROM message_history ORDER BY sent_at DESC LIMIT 100',
    [],
    (err, rows) => {
      if (err) {
        return res.status(500).json({
          success: false,
          error: 'Failed to fetch history'
        });
      }

      res.json({
        success: true,
        messages: rows
      });
    }
  );
});

// Update send endpoint to save history
app.post('/api/send', authenticateToken, async (req, res) => {
  // ... existing send code ...

  // After successful send
  try {
    await saveMessage(groupId, 'Group Name', message);
  } catch (error) {
    console.error('Failed to save to history:', error);
  }

  // ... return response
});
```

---

7. Add Webhook Support

Allow external services to send notifications:

```
// Webhook endpoint
app.post('/api/webhook', async (req, res) => {
  try {
    const { secret, groupId, message } = req.body;

    // Verify webhook secret
    if (secret !== process.env.WEBHOOK_SECRET) {
      return res.status(401).json({
        success: false,
        error: 'Invalid webhook secret'
      });
    }
    // Send message
    await axios.post(`${SIGNAL_API_URL}/v2/send`, {
      message,
      number: SIGNAL_NUMBER,
      recipients: [groupId]
    });

    res.json({
      success: true,
      message: 'Webhook processed'
    });

  } catch (error) {
    logError('Webhook', error);
    res.status(500).json({
      success: false,
      error: 'Failed to process webhook'
    });
  }
});
```

Usage:

```
curl -X POST http://localhost:5001/api/webhook \
  -H 'Content-Type: application/json' \
  -d '{
    "secret": "your-webhook-secret",
    "groupId": "group.abc123...",
    "message": "Alert: Server is down!"
  }'
```

Development Best Practices

1. **Always validate input:**

```
if (!groupId || !message) {
  return res.status(400).json({
    success: false,
    error: 'Missing required fields'
  });
}
```

2. **Use try-catch blocks:**

```
try {
  // Your code
} catch (error) {
  logError('Context', error);
  res.status(500).json({ error: 'Failed' });
}
```

3. **Log important actions:**

```
console.log(`User ${req.user} sent message to ${groupId}`);
```

4. **Return consistent JSON:**

```
// Success
{ success: true, data: {...} }

// Error
{ success: false, error: 'message', details: '...' }
```

5. **Use environment variables:**

```
const API_KEY = process.env.API_KEY;
if (!API_KEY) throw new Error('API_KEY not configured');
```

Code Structure

Backend Architecture

File: `signal-poc/backend/server.js`

```
Lines 1-18:    Imports and setup
Lines 19-30:   Configuration and logging
Lines 32-52:   Health check endpoint
Lines 54-100:  Get groups endpoint
Lines 102-150: Send message endpoint
Lines 152-180: Sync endpoint
Lines 182-195: Config endpoint
Lines 197-210: Server startup
```

Key Functions:

```
// Error logging utility
const logError = (context, error) => {
  // Returns structured error info
}

// Main endpoints
app.get('/api/health', ...)    // Health check
app.get('/api/groups', ...)    // Fetch groups
app.post('/api/send', ...)     // Send message
app.post('/api/sync', ...)     // Sync with Signal
app.get('/api/config', ...)    // Get config
```

Frontend Architecture

File: `signal-poc/frontend/src/App.jsx`

```
Lines 1-10:    Imports
Lines 12-25:   State management
Lines 27-50:   fetchGroups function
Lines 52-95:   sendMessage function
Lines 97-120:  checkHealth function
Lines 122-140: useEffect hooks
Lines 142-350: JSX/UI rendering
```

React State:

```
const [groups, setGroups] = useState([])                 // All groups
const [selectedGroup, setSelectedGroup] = useState(null) // Current group
const [message, setMessage] = useState('')               // Message text
const [error, setError] = useState('')                   // Error message
const [success, setSuccess] = useState('')               // Success message
const [loading, setLoading] = useState(false)            // Loading state
const [backendStatus, setBackendStatus] = useState('') // Backend status
const [signalStatus, setSignalStatus] = useState('')  // Signal status
```

Key Functions:

```
fetchGroups()      // Fetch groups from API
sendMessage()      // Send message to group
checkHealth()      // Check backend/Signal status
handleGroupClick() // Handle group selection
handleSend()       // Handle send button click
```

Component Structure

```
App (Main Component)
├── Header
│   ├── Title
│   └── Status Indicators
│       ├── Backend Status Badge
│       └── Signal API Status Badge
│
├── Groups Card
│   ├── Title & Refresh Button
│   └── Groups List
│       └── Group Items (clickable)
│
├── Send Message Card
│   ├── Group Selector Dropdown
│   ├── Message Textarea
│   ├── Character Counter
│   └── Send Button
│
└── Alert Messages
    ├── Error Alert (red)
    └── Success Alert (green)
```

Data Flow Diagram

```
User Interaction
       ↓
React State Update
       ↓
API Call (Axios)
       ↓
Backend Endpoint
       ↓
Validation
       ↓
Signal API Call
       ↓
Signal Servers
       ↓
Response
       ↓
Backend Processing
       ↓
JSON Response
       ↓
Frontend Update
       ↓
UI Update
```

File Dependencies

Backend:

```
server.js
  ├── express (web framework)
  ├── cors (CORS handling)
  ├── axios (HTTP client)
  ├── morgan (logging)
  ├── dotenv (environment)
  └── process.env (configuration)
```

Frontend:

```
App.jsx
  ├── react (UI framework)
  ├── axios (HTTP client)
  ├── App.css (styles)
  └── /api/* (backend proxy)

main.jsx
  ├── react
  ├── react-dom
  ├── App.jsx
  └── index.css

vite.config.js
  └── @vitejs/plugin-react
```

Adding New Files

To add a new backend module:

1. Create file: `signal-poc/backend/modules/newFeature.js`

```
module.exports = {
  doSomething: (param) => {
    // Your code
    return result;
  }
};
```

2. Import in server.js:

```
const newFeature = require('./modules/newFeature');

app.get('/api/new-feature', (req, res) => {
  const result = newFeature.doSomething(req.query.param);
  res.json(result);
});
```

To add a new React component:

1. Create file: `signal-poc/frontend/src/components/NewComponent.jsx`

```
import React from 'react';
import './NewComponent.css';

function NewComponent({ prop1, prop2 }) {
  return (
    <div className="new-component">
      {/* Your JSX */}
    </div>
  );
}

export default NewComponent;
```

2. Import in App.jsx:

```
import NewComponent from './components/NewComponent';

// Use in JSX
<NewComponent prop1={value1} prop2={value2} />
```

FAQs

General Questions

Q: What is this project?
A: A Proof of Concept web application for sending Signal messages to groups through a web interface.

Q: Is this production-ready?
A: No, this is a PoC. For production, you need authentication, HTTPS, rate limiting, and proper security measures.

Q: Do I need a database?
A: No! The app is stateless. Groups are fetched from Signal in real-time.

Q: Can I use my personal Signal number?
A: Not recommended! Use a dedicated test number. Registering here will unregister your phone.

Q: Is this official Signal software?
A: No, it uses signal-cli (community project) and signal-cli-rest-api (community wrapper).

Setup Questions

Q: I don't have Docker, can I still use this?
A: Docker is required for signal-cli-rest-api. It's the easiest way to run Signal integration.

Q: Can I run this on Windows?
A: Yes! Use WSL2 for Docker, or run backend/frontend natively with slight script modifications.

Q: What ports do I need available?
A: Ports 3000, 5001, and 8080 must be free.

Q: Can I change the ports?
A: Yes! Modify:

- Backend: `.env` file (`PORT=5001`)
- Frontend: `vite.config.js` (server.port)
- Signal API: `docker-compose.yml` (ports section)

Registration Questions

Q: What's the difference between primary registration and device linking?
A:

- **Primary registration:** Full functionality, but unregisters phone
- **Device linking:** Like Signal Desktop, limited features, messages may appear as "notes"
- **Recommendation:** Use primary registration with a test number

Q: Can I register without a real phone number?
A: No, Signal requires a real phone number that can receive SMS.

Q: Can I use VoIP numbers?
A: Some work, some don't. Google Voice (US) works. Twilio can work. Try and see!

Q: What happens to my phone when I register here?
A: Your phone loses Signal access. You'll need to re-register it in the app.

Usage Questions

Q: Why don't I see any groups?
A: You need to create groups in Signal mobile app first, then sync with `./sync-groups.sh`.

Q: Why do messages show as "Note to Self"?
A: Group only has you as a member. Add more people to the group.

Q: Can I send images/files?
A: Not in the current PoC, but you can extend it (see Extension Guide).

Q: Can I send to multiple groups at once?
A: Not in UI, but you can write a script to loop through groups.

Q: How many messages can I send?
A: Signal has rate limits (~30-60 per minute). Respect them to avoid being flagged.

Q: Can I receive messages?
A: The API can receive, but the PoC doesn't display them. You can extend it.

Troubleshooting Questions

Q: Port 8080 is already in use, what do I do?
A:

```
# Find what's using it
lsof -i :8080

# Kill it or change Signal API port in docker-compose.yml
```

Q: Backend won't start, says SIGNAL_NUMBER not found?
A: Create `.env` file in `signal-poc/backend/` with your number.

Q: Groups won't load?
A:

1. Check Signal API is running: `docker ps | grep signal`
2. Check number is registered: `curl http://localhost:8080/v1/groups/+[YOUR_NUMBER]`
3. Sync groups: `cd signal-api && ./sync-groups.sh`

Q: I get "Network Error" in the UI?
A: Backend is down or wrong URL. Check `npm start` is running in backend directory.

Q: Docker container keeps restarting?
A: Check logs: `docker logs signal-api`. May need to clear data and re-register.

Feature Questions

Q: Can I create groups via the web app?
A: Yes, via API (see "How to Create Groups" section), but easier in mobile app.

Q: Can I invite people to existing groups?
A: Only if you're the admin of the group. signal-cli limitation.

Q: Can I use group invite links?
A: No, signal-cli doesn't support generating or accepting invite links.

Q: Can I schedule messages?
A: Not by default, but you can add it (see Extension Guide → Add Message Scheduling).

Q: Can I add authentication?
A: Yes! See Extension Guide → Add User Authentication.

Q: Can I deploy this to a server?
A: Yes, but add security measures first (auth, HTTPS, rate limiting, etc.).

Technical Questions

Q: What version of Node.js do I need?
A: v16 or higher. Check with `node --version`.

Q: Can I use Yarn instead of npm?
A: Yes! Replace `npm install` with `yarn` and `npm start` with `yarn start`.

Q: How do I add new dependencies?
A:

```
cd signal-poc/backend
npm install package-name

cd signal-poc/frontend
npm install package-name
```

Q: Can I use TypeScript?
A: Yes! Convert files to `.ts`/`.tsx` and add TypeScript config. Vite supports TS out of the box.

Q: How do I debug?
A:

- Backend: Add `console.log()` or use Node debugger
- Frontend: Use React DevTools and browser console
- Signal API: Check `docker logs signal-api`

Q: Can I run tests?
A: No tests included in PoC. You can add Jest for backend, React Testing Library for frontend.

Integration Questions

Q: Can I integrate this with other systems?
A: Yes! The backend API can be called from any system that can make HTTP requests.

Q: Can I use this for automated notifications?
A: Yes! Perfect use case. Call `/api/send` from your monitoring/CI/CD system.

Q: Can I integrate with webhooks?
A: Yes! See Extension Guide → Add Webhook Support.

Q: Can I use this with Python/Java/etc?
A: Yes! Any language that can make HTTP requests can use the API.

Demo Mode Questions

Q: What is demo mode?
A: A mode that shows mock data without requiring Signal setup. Perfect for demos.

Q: How do I enable demo mode?
A: `cd signal-poc && ./switch-mode.sh demo`

Q: Does demo mode actually send messages?
A: No, it just simulates sending. No Signal connection needed.

Q: When should I use demo mode?
A: When showing the UI to stakeholders, testing UI changes, or don't have Signal set up yet.

Performance Questions

Q: How fast can it send messages?
A: ~500-1000ms per message (depends on Signal servers and network).

Q: Can I send bulk messages?
A: Yes, but respect Signal's rate limits. Add delays between messages.

Q: How many groups can it handle?
A: No limit from the app. Signal limits you to being in ~1000 groups max.

Q: Does it use a lot of resources?
A: No. Minimal CPU/RAM usage. Docker container uses ~200-300MB RAM.

Security Questions

Q: Is this secure?
A: For a PoC, yes. For production, you need to add authentication, HTTPS, input validation, etc.

Q: Where is my Signal data stored?
A: In `signal-api/signal-cli-config/`. This is gitignored and should never be committed.

Q: Can other people access my messages?
A: Only if they have access to your server. Add authentication to prevent unauthorized access.

Q: Should I commit my .env file?
A: NO! Never commit .env files with credentials. Use .gitignore.

Q: Is the Signal protocol secure?
A: Yes! Signal uses end-to-end encryption. This app just sends messages via the official protocol.

Deployment Questions

Q: Can I deploy this to production?
A: After adding security measures (auth, HTTPS, input validation, rate limiting).

Q: Can I deploy to Heroku/AWS/GCP?
A: Yes! You'll need Docker support or deploy Signal API separately.

Q: Do I need HTTPS?
A: For production, absolutely yes. Use Let's Encrypt or cloud provider SSL.

Q: How do I keep it running 24/7?
A: Use process manager (PM2) for Node, and Docker restart policies for Signal API.

Q: Can multiple users use the same instance?
A: Yes, but add authentication first. Currently, it's single-user.

Project Metrics

Code Statistics:

- Total Files: 50+
- Lines of Code: ~3,500+
- Documentation Files: 30+
- Scripts: 15+

Component Breakdown:

- Frontend: 1 main React component
- Backend: 1 Express server
- Signal Integration: Docker container + REST API

Time to Start: 1 command (~30 seconds)
Setup Complexity: Simple (Docker + npm)
Learning Curve: Low (basic React + Node.js knowledge)

Credits & Resources

Built With

- signal-cli - Command-line Signal client
- signal-cli-rest-api - REST API wrapper
- React - UI framework
- Vite - Build tool
- Express - Backend framework

- Axios - HTTP client

Documentation

- Signal Protocol
- signal-cli GitHub
- signal-cli-rest-api Docs

Support

For issues with:

- **This PoC:** Check TROUBLESHOOTING section above
- **signal-cli:** https://github.com/AsamK/signal-cli/issues
- **signal-cli-rest-api:** https://github.com/bbernhard/signal-cli-rest-api/issues
- **Signal Protocol:** https://signal.org/

License

MIT License - Feel free to use and modify!

Final Notes

✅ What Works Great

- Sending messages to groups
- Creating groups with members
- Listing all groups
- Real-time status indicators
- Error handling
- Beautiful UI
- One-command startup

⚠ Known Limitations

- Cannot accept invite links (signal-cli limitation)
- Cannot invite to existing groups you don't admin
- Some GroupsV2 features limited
- No message history (can be added)
- No file attachments (can be added)

🚀 Ready for Production?

No, but you can make it production-ready by adding:

- User authentication (JWT/OAuth)
- HTTPS/SSL certificates
- Input validation & sanitization

- Rate limiting
- Database for history
- Logging & monitoring
- Error tracking (Sentry)
- Load balancing (if needed)

🎯 Perfect For

- ✅ Proof of concepts
- ✅ Internal tools
- ✅ Automated notifications
- ✅ System monitoring alerts
- ✅ CI/CD integrations
- ✅ Team communication automation
- ✅ Learning Signal integration

Quick Reference Card

Start:

```
./START_PROJECT.sh
```

Stop:

```
./STOP_PROJECT.sh
```

URLs:

- Frontend: http://localhost:3000
- Backend: http://localhost:5001
- Signal API: http://localhost:8080

Configuration:

```
signal-poc/backend/.env
```

Logs:

```
docker logs signal-api
tail -f signal-poc/backend.log
```

Sync Groups:

```
cd signal-api && ./sync-groups.sh
```

Test:

```
curl http://localhost:5001/api/health
```

Documentation Version: 1.0.0
Last Updated: October 2025
Status: Complete ✅

Happy Messaging! 📱 ✨