CDW Documentation

Azure chatbot with open AI and slack integration

Azure chatbot with open AI and slack integration

Complete all the steps in the documentation Azure chatbot with Open AI (These steps are run in local)

npm install botbuilder-adapter-slack npm install @slack/web-api

index.js

```
const path = require('path');
const restify = require('restify');
const { BotFrameworkAdapter } = require('botbuilder');
const { AzureOpenAIBot } = require('./bot');

// Load environment variables
const ENV_FILE = path.join(__dirname, '.env');
require('dotenv').config({ path: ENV_FILE });

console.log('=== ENVIRONMENT CHECK ===');
console.log('NODE_ENV:', process.env.NODE_ENV);
console.log('Dotenv file path:', ENV_FILE);

console.log('=== BOT FRAMEWORK CREDENTIALS ===');
console.log('MicrosoftAppId:', process.env.MicrosoftAppId || 'NOT SET');
console.log('MicrosoftAppPassword:', process.env.MicrosoftAppPassword ? 'SET
(length: ' + process.env.MicrosoftAppPassword.length + ')' : 'NOT SET');
console.log('Port:', process.env.port || process.env.PORT || 3978);

// Create HTTP server
const server = restify.createServer();
server.use(restify.plugins.bodyParser());

console.log('=== CREATING BOT FRAMEWORK ADAPTER ===');

// Create adapter
const adapter = new BotFrameworkAdapter({
    appId: process.env.MicrosoftAppId,
    appPassword: process.env.MicrosoftAppPassword
});

console.log('Adapter created successfully');

// Error handler
adapter.onTurnError = async (context, error) => {
    console.error('=== ADAPTER TURN ERROR ===');
```

```
    console.error('Error message:', error.message);
    console.error('Error stack:', error.stack);
    try {
        await context.sendTraceActivity(
            'OnTurnError Trace',
            `${error}`,
            'https://www.botframework.com/schemas/error',
            'TurnError'
        );
        await context.sendActivity('The bot encountered an error or bug.');
    } catch (sendError) {
        console.error('Failed to send error message:', sendError);
    }
};

// Add this test function after creating the adapter
async function testBotAuthentication() {
    console.log('=== TESTING BOT AUTHENTICATION ===');
    const { MicrosoftAppCredentials } = require('botframework-connector');
    try {
        if (!process.env.MicrosoftAppId ||
!process.env.MicrosoftAppPassword) {
            console.log('□ Missing credentials');
            return false;
        }
        console.log('Testing with App ID:',
process.env.MicrosoftAppId.substring(0, 8) + '...');
        console.log('Password length:',
process.env.MicrosoftAppPassword.length);
        const credentials = new MicrosoftAppCredentials(
            process.env.MicrosoftAppId,
            process.env.MicrosoftAppPassword
        );
        const token = await credentials.getToken();
        console.log('□ Authentication successful - token obtained');
        console.log('Token starts with:', token.substring(0, 20) + '...');
        return true;
    } catch (error) {
        console.log('□ Authentication failed:');
        console.log('Error message:', error.message);
        console.log('Error code:', error.code);
        console.log('Full error:', error);
        return false;
    }
}

// Call this when server starts (add this line after creating the bot)
testBotAuthentication();

// Create bot
console.log('=== CREATING BOT INSTANCE ===');
```

```
const myBot = new AzureOpenAIBot();
console.log('Bot instance created successfully');

// Listen for incoming requests - MODIFIED TO HANDLE SLACK URL VERIFICATION
server.post('/api/messages', async (req, res) => {
    console.log('=== INCOMING REQUEST ===');
    console.log('Method:', req.method);
    console.log('URL:', req.url);
    console.log('Headers:', JSON.stringify(req.headers, null, 2));
    console.log('Request timestamp:', new Date().toISOString());
    // SLACK URL VERIFICATION HANDLER
    if (req.body && req.body.type === 'url_verification') {
        console.log('=== SLACK URL VERIFICATION ===');
        console.log('Challenge received:', req.body.challenge);
        // Respond with the challenge to verify the URL
        res.send(req.body.challenge);
        return;
    }
    // SLACK EVENT CALLBACK HANDLER
    if (req.body && req.body.type === 'event_callback') {
        console.log('=== SLACK EVENT RECEIVED ===');
        console.log('Event type:', req.body.event?.type);
        console.log('Event data:', JSON.stringify(req.body.event, null, 2));
        const event = req.body.event;
        // Handle message events
        if (event.type === 'message' && !event.bot_id) {
            try {
                console.log('Processing Slack message...');
                // Get user message
                const userMessage = event.text;
                const userId = event.user;
                const channel = event.channel;
                console.log(`Message from user ${userId}: ${userMessage}`);
                // Call your bot's logic using the new method
                const botResponse = await
myBot.processSlackMessage(userMessage);
                // Send response back to Slack
                await sendSlackMessage(channel, botResponse);
                console.log('Slack message processed successfully');
            } catch (error) {
                console.error('Error processing Slack message:', error);
                // Send error response to Slack
                try {
                    await sendSlackMessage(event.channel, 'Sorry, I
encountered an error processing your message.');
                } catch (sendError) {
                    console.error('Failed to send error message to Slack:',
sendError);
                }
            }
        }
```

```
        // Acknowledge the event
        res.send('OK');
        return;
    }

    try {
        // Regular Bot Framework processing
        await adapter.processActivity(req, res, async (context) => {
            console.log('Processing activity with bot...');
            await myBot.run(context);
            console.log('Bot processing completed');
        });
    } catch (error) {
        console.error('=== REQUEST PROCESSING ERROR ===');
        console.error('Error:', error.message);
        console.error('Stack:', error.stack);
        // Send error response if not already sent
        if (!res.headersSent) {
            res.status(500);
            res.json({ error: 'Internal server error' });
        }
    }
});

// Test endpoint to verify server is running
server.get('/api/health', (req, res, next) => {
    console.log('Health check requested');
    res.json({
        status: 'OK',
        timestamp: new Date().toISOString(),
        environment: {
            hasAppId: !!process.env.MicrosoftAppId,
            hasAppPassword: !!process.env.MicrosoftAppPassword,
            hasOpenAIKey: !!process.env.AZURE_OPENAI_KEY,
            hasOpenAIEndpoint: !!process.env.AZURE_OPENAI_ENDPOINT,
            hasSlackToken: !!process.env.SLACK_BOT_TOKEN
        }
    });
    return next();
});

const port = process.env.PORT || 3978;

server.listen(port, () => {
    console.log('=== SERVER STARTED ===');
    console.log(`Bot started, listening on port ${port}`);
    console.log(`Health check: http://localhost:${port}/api/health`);
    console.log(`Messages endpoint: http://localhost:${port}/api/messages`);
    console.log(`Using Azure OpenAI deployment:
${process.env.AZURE_OPENAI_DEPLOYMENT_NAME}`);
    // Test if we can access environment variables
```

```
    if (!process.env.MicrosoftAppId || !process.env.MicrosoftAppPassword) {
        console.log('⚠️  WARNING: Bot Framework credentials not set - 
authentication will fail');
    } else {
        console.log('✅ Bot Framework credentials are set');
    }
    if (!process.env.AZURE_OPENAI_KEY || !process.env.AZURE_OPENAI_ENDPOINT)
{
        console.log('⚠️  WARNING: Azure OpenAI credentials not set');
    } else {
        console.log('✅ Azure OpenAI credentials are set');
    }
    if (!process.env.SLACK_BOT_TOKEN) {
        console.log('⚠️  WARNING: Slack bot token not set');
    } else {
        console.log('✅ Slack bot token is set');
    }
});

// Handle server errors
server.on('error', (error) => {
    console.error('=== SERVER ERROR ===');
    console.error('Error:', error);
});

// Graceful shutdown
process.on('SIGINT', () => {
    console.log('Shutting down gracefully...');
    server.close(() => {
        console.log('Server closed');
        process.exit(0);
    });
});

// SLACK MESSAGE SENDER - UPDATED
async function sendSlackMessage(channel, message) {
    const { WebClient } = require('@slack/web-api');
    if (!process.env.SLACK_BOT_TOKEN) {
        console.error('SLACK_BOT_TOKEN not set in environment variables');
        return;
    }
    const slack = new WebClient(process.env.SLACK_BOT_TOKEN);
    try {
        console.log(`Sending message to Slack channel ${channel}: 
${message}`);
        const result = await slack.chat.postMessage({
            channel: channel,
            text: message
        });
        console.log('Message sent successfully to Slack:', result.ts);
    } catch (error) {
```

```
        console.error('Error sending Slack message:', error);
    }
}
```

==== Bot.js ====

```
const { ActivityHandler, MessageFactory } = require('botbuilder');
const OpenAI = require('openai');

class AzureOpenAIBot extends ActivityHandler {
    constructor() {
        super();

        console.log('=== BOT INITIALIZATION ===');
        console.log('Azure OpenAI Endpoint:',
process.env.AZURE_OPENAI_ENDPOINT ? 'Set' : 'Not set');
        console.log('Azure OpenAI Key:', process.env.AZURE_OPENAI_KEY ? 'Set
(length: ' + process.env.AZURE_OPENAI_KEY.length + ')' : 'Not set');
        console.log('Deployment Name:',
process.env.AZURE_OPENAI_DEPLOYMENT_NAME || 'Not set');

        // Initialize OpenAI client configured for Azure OpenAI
        try {
            this.openai = new OpenAI({
                apiKey: process.env.AZURE_OPENAI_KEY,
                baseURL:
`${process.env.AZURE_OPENAI_ENDPOINT}/openai/deployments/${process.env.AZURE
_OPENAI_DEPLOYMENT_NAME}`,
                defaultQuery: { 'api-version': '2024-02-15-preview' },
                defaultHeaders: {
                    'api-key': process.env.AZURE_OPENAI_KEY,
                }
            });
            console.log('OpenAI client initialized successfully');
            console.log('Base URL:',
`${process.env.AZURE_OPENAI_ENDPOINT}/openai/deployments/${process.env.AZURE
_OPENAI_DEPLOYMENT_NAME}`);
        } catch (error) {
            console.error('Failed to initialize OpenAI client:', error);
        }

        // Set system message for the bot's personality
        this.systemMessage = `You are a helpful assistant for a company
called TechCorp.
        You help answer questions about our products, services, and general
support.
        Keep responses concise and professional. If you don't know something
specific
        about TechCorp, say so and offer to connect them with a human
```

```
agent.`;

        this.onMessage(async (context, next) => {
            console.log('=== MESSAGE RECEIVED ===');
            console.log('Timestamp:', new Date().toISOString());
            console.log('User message:', context.activity.text);
            console.log('Activity type:', context.activity.type);
            console.log('Channel ID:', context.activity.channelId);

            const userMessage = context.activity.text;

            // First, try a simple echo to test basic functionality
            if (userMessage && userMessage.toLowerCase().includes('test
echo')) {
                console.log('Echo test requested');
                try {
                    await context.sendActivity(MessageFactory.text(`Echo:
${userMessage}`));
                    console.log('Echo response sent successfully');
                } catch (echoError) {
                    console.error('Even simple echo failed:', echoError);
                }
                await next();
                return;
            }

            try {
                console.log('About to call Azure OpenAI...');
                // Add timeout to the OpenAI call
                const openaiCall = this.openai.chat.completions.create({
                    messages: [
                        { role: "system", content: this.systemMessage },
                        { role: "user", content: userMessage }
                    ],
                    max_tokens: 150,
                    temperature: 0.7
                });

                const timeoutPromise = new Promise((_, reject) =>
                    setTimeout(() => reject(new Error('OpenAI API timeout
after 30 seconds')), 30000)
                );

                const response = await Promise.race([openaiCall,
timeoutPromise]);
                console.log('OpenAI response received');
                console.log('Response choices length:',
response.choices?.length || 0);
                if (!response.choices || response.choices.length === 0) {
                    throw new Error('No choices returned from OpenAI API');
                }
```

```
                const aiResponse = response.choices[0].message.content;
                console.log('AI response content:', aiResponse);
                if (!aiResponse) {
                    throw new Error('Empty response content from OpenAI');
                }

                await context.sendActivity(MessageFactory.text(aiResponse));
                console.log('Response sent to user successfully');

            } catch (error) {
                console.error('=== ERROR DETAILS ===');
                console.error('Error type:', error.constructor.name);
                console.error('Error message:', error.message);
                console.error('Error stack:', error.stack);
                // Check for specific Azure OpenAI errors
                if (error.response) {
                    console.error('HTTP Status:', error.response.status);
                    console.error('Response data:',
JSON.stringify(error.response.data, null, 2));
                }

                // Check for network/connection errors
                if (error.code) {
                    console.error('Error code:', error.code);
                }

                // Always try to send an error response to the user
                try {
                    let errorMessage = 'Sorry, I encountered an error
processing your request. Please try again.';
                    // Provide more specific error messages for common
issues
                    if (error.message.includes('timeout')) {
                        errorMessage = 'The request timed out. Please try
again with a shorter message.';
                    } else if (error.message.includes('rate')) {
                        errorMessage = 'Service is currently busy. Please
wait a moment and try again.';
                    } else if (error.response?.status === 401) {
                        errorMessage = 'Authentication error. Please contact
support.';
                    } else if (error.response?.status === 404) {
                        errorMessage = 'Service configuration error. Please
contact support.';
                    }

                    await
context.sendActivity(MessageFactory.text(errorMessage));
                    console.log('Error message sent to user');
                } catch (sendError) {
                    console.error('Failed to send error message to user:',
```

```
sendError);
                }
            }

            console.log('=== MESSAGE PROCESSING COMPLETE ===');
            await next();
        });

        this.onMembersAdded(async (context, next) => {
            console.log('=== MEMBER ADDED EVENT ===');
            console.log('Members added count:',
context.activity.membersAdded?.length || 0);
            const welcomeText = 'Hello! I\'m your TechCorp assistant powered
by Azure OpenAI. How can I help you today?';
            try {
                for (let cnt = 0; cnt <
context.activity.membersAdded.length; ++cnt) {
                    if (context.activity.membersAdded[cnt].id !==
context.activity.recipient.id) {
                        console.log('Sending welcome message to:',
context.activity.membersAdded[cnt].id);
                        await
context.sendActivity(MessageFactory.text(welcomeText));
                    }
                }
                console.log('Welcome messages sent successfully');
            } catch (error) {
                console.error('Error sending welcome message:', error);
            }
            await next();
        });

        // Add error handler for the activity handler
        this.onTurnError = async (context, error) => {
            console.error('=== TURN ERROR ===');
            console.error('Turn error:', error);
            console.error('Context activity:',
JSON.stringify(context.activity, null, 2));
            try {
                await context.sendActivity(MessageFactory.text('Sorry, an
unexpected error occurred.'));
            } catch (sendError) {
                console.error('Failed to send turn error message:',
sendError);
            }
        };

        console.log('=== BOT INITIALIZATION COMPLETE ===');
    }

    // NEW METHOD: Process Slack messages directly
```

```
    async processSlackMessage(userMessage) {
        console.log('=== PROCESSING SLACK MESSAGE ===');
        console.log('Message:', userMessage);

        try {
            // Test echo functionality
            if (userMessage && userMessage.toLowerCase().includes('test
echo')) {
                return `Echo: ${userMessage}`;
            }

            console.log('Calling Azure OpenAI for Slack message...');
            const openaiCall = this.openai.chat.completions.create({
                messages: [
                    { role: "system", content: this.systemMessage },
                    { role: "user", content: userMessage }
                ],
                max_tokens: 150,
                temperature: 0.7
            });

            const timeoutPromise = new Promise((_, reject) =>
                setTimeout(() => reject(new Error('OpenAI API timeout after
30 seconds')), 30000)
            );

            const response = await Promise.race([openaiCall,
timeoutPromise]);
            console.log('OpenAI response received for Slack');
            if (!response.choices || response.choices.length === 0) {
                throw new Error('No choices returned from OpenAI API');
            }

            const aiResponse = response.choices[0].message.content;
            console.log('AI response for Slack:', aiResponse);
            if (!aiResponse) {
                throw new Error('Empty response content from OpenAI');
            }

            return aiResponse;

        } catch (error) {
            console.error('=== SLACK MESSAGE ERROR ===');
            console.error('Error type:', error.constructor.name);
            console.error('Error message:', error.message);
            if (error.response) {
                console.error('HTTP Status:', error.response.status);
                console.error('Response data:',
JSON.stringify(error.response.data, null, 2));
            }
```

```
            // Return appropriate error message
            if (error.message.includes('timeout')) {
                return 'The request timed out. Please try again with a
shorter message.';
            } else if (error.message.includes('rate')) {
                return 'Service is currently busy. Please wait a moment and
try again.';
            } else if (error.response?.status === 401) {
                return 'Authentication error. Please contact support.';
            } else if (error.response?.status === 404) {
                return 'Service configuration error. Please contact
support.';
            } else {
                return 'Sorry, I encountered an error processing your
request. Please try again.';
            }
        }
    }

    // Add a method to test OpenAI connection separately
    async testOpenAIConnection() {
        console.log('=== TESTING OPENAI CONNECTION ===');
        try {
            const response = await this.openai.chat.completions.create({
                messages: [
                    { role: "user", content: "Hello, this is a test
message." }
                ],
                max_tokens: 10,
                temperature: 0.1
            });
            console.log('OpenAI test successful:',
response.choices[0].message.content);
            return true;
        } catch (error) {
            console.error('OpenAI test failed:');
            console.error('Error:', error.message);
            if (error.response) {
                console.error('Status:', error.response.status);
                console.error('Data:', error.response.data);
            }
            return false;
        }
    }
}

module.exports.AzureOpenAIBot = AzureOpenAIBot;
```

Slack configuration

1. Create a Slack App

Go to api.slack.com/apps Click “Create New App” → “From scratch” Give your app a name and select your workspace Click “Create App”

2. Configure Bot Permissions

In your app settings, go to “OAuth & Permissions” Under “Scopes” → “Bot Token Scopes”, add these permissions:

chat:write (send messages) app_mentions:read (read mentions) channels:read (read channel info) im:read (read DMs) im:write (send DMs)

3. Install the App to Your Workspace

Still in “OAuth & Permissions”, click “Install to Workspace” Authorize the app Copy the “Bot User OAuth Token” (starts with xoxb-)

4. Enable Events

Go to “Event Subscriptions” and toggle it on You'll need a public URL for your app (use ngrok for local development) Set the Request URL to https://your-ngrok-url.com/slack/events Under “Subscribe to bot events”, add:

app_mention message.im

5. Update Your Azure Bot Configuration

In Azure Portal, go to your Bot Service Go to “Channels” and add “Slack” channel Enter your Slack app credentials:

Client ID (from Slack app “Basic Information”) Client Secret (from Slack app “Basic Information”) Verification Token (from Slack app “Basic Information”)

6. Update env variables

SLACK_BOT_TOKEN=xoxb-your-bot-token SLACK_CLIENT_SIGNING_SECRET=your-signing-secret SLACK_CLIENT_ID=your-client-id SLACK_CLIENT_SECRET=your-client-secret

7. Set the Request URL in Slack

Copy your ngrok URL (e.g., https://abc123.ngrok.io) In your Slack app “Event Subscriptions”, set Request URL to: https://abc123.ngrok.io/api/slack/messages Slack will verify the URL

8. OAuth Slack App Configuration Make sure your Slack app has all required settings: In your Slack App (api.slack.com):

Go to “OAuth & Permissions” Add redirect URL: https://slack.botframework.com Make sure you have a Bot User created (“App Home” → “Bot Users”) Install the app to your workspace first

9. Test the app by tagging the bot in the messages in the channel. You should be able to see the logs in the local as well