CDW Documentation

Emotional support Bot with Text Analytics and OpenAI

Emotional support Bot with Text Analytics and OpenAI

This app is deployed in static webapp and uses node JS for managed Azure functions. Azure functions version 4 is used.

Below is the folder structure that needs to be deployed for Azure functions V4

screenshot_2025-07-20_at_11.08.02 pm.png

api/host.json

```
{
  "version": "2.0",
  "logging": {
    "applicationInsights": {
      "samplingSettings": {
        "isEnabled": true,
        "excludedTypes": "Request"
      }
    }
  },
  "extensionBundle": {
    "id": "Microsoft.Azure.Functions.ExtensionBundle",
    "version": "[3.*, 4.0.0)"
  },
  "functionApp": {
    "id": "emotional-support-bot"  // This is your entire app name
  }
}
```

api/package.json

```
{
  "name": "emotional-support-bot-api",
  "version": "1.0.0",
  "description": "Azure Functions for emotional support bot",
  "main": "src/functions/*.js",
  "scripts": {
    "start": "func start",
    "test": "echo \"No tests yet...\""
  },
  "dependencies": {
    "@azure/functions": "^4.0.0",
   "@azure/ai-text-analytics": "^5.1.0",
    "@azure/openai": "^2.0.0",
     "@azure/core-auth": "^1.5.0"
  },
```

```
  "devDependencies": {
    "azure-functions-core-tools": "^4.0.0"
  }
}
```

Please take a look at the section main. Main should contain the source to the node js function files which we need to run as part of static webapps

api/src/functions/analyze-sentiment.js

```
// src/functions/analyze-sentiment.js
const { app } = require('@azure/functions');

app.http('analyze-sentiment', {
    methods: ['POST'],
    authLevel: 'anonymous',
    route: 'analyze-sentiment',
    handler: async (request, context) => {
        context.log('Processing sentiment analysis request');
        try {
            const { text } = await request.json();
            if (!text) {
                return {
                    status: 400,
                    headers: {
                        'Content-Type': 'application/json'
                    },
                    body: JSON.stringify({ error: 'Text is required' })
                };
            }

            const endpoint = process.env.TEXT_ANALYTICS_ENDPOINT;
            const key = process.env.TEXT_ANALYTICS_KEY;
            // Logging for debugging
            context.log('Endpoint:', process.env.TEXT_ANALYTICS_ENDPOINT);
            context.log('Key exists:', !!process.env.TEXT_ANALYTICS_KEY);
            if (!endpoint || !key) {
                context.log.error('Text Analytics configuration missing');
                return {
                    status: 500,
                    headers: {
                        'Content-Type': 'application/json'
                    },
                    body: JSON.stringify({ error: 'Text Analytics
configuration missing' })
                };
            }

            const response = await
```

```
fetch(`${endpoint}/text/analytics/v3.1/sentiment`, {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json',
                    'Ocp-Apim-Subscription-Key': key
                },
                body: JSON.stringify({
                    documents: [
                        {
                            id: '1',
                            text: text,
                            language: 'en'
                        }
                    ]
                })
            });

            if (!response.ok) {
                const errorText = await response.text();
                context.log.error(`Text Analytics API failed:
${response.status} - ${errorText}`);
                throw new Error(`Text Analytics API failed:
${response.status}`);
            }

            const data = await response.json();
            const sentimentResult = data.documents[0];

            return {
                status: 200,
                headers: {
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify(sentimentResult)
            };

        } catch (error) {
            context.log.error('Error in analyze-sentiment:', error);
            return {
                status: 500,
                headers: {
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify({ error: 'Internal server error: ' +
error.message })
            };
        }
    }
});
```

api/src/functions/generate-response.js

```
// src/functions/generate-response.js
const { app } = require('@azure/functions');

function getSystemPrompt(sentiment) {
    const sentimentLabel = sentiment.sentiment;
    const confidence = sentiment.confidenceScores;

    let prompt = `You are an empathetic AI support assistant. The user's
message has been analyzed with sentiment: ${sentimentLabel} (positive:
${confidence.positive.toFixed(2)}, neutral:
${confidence.neutral.toFixed(2)}, negative:
${confidence.negative.toFixed(2)}).

Based on this sentiment analysis, provide appropriate emotional support:

`;

    if (sentimentLabel === 'positive') {
        prompt += `- The user seems to be feeling good. Acknowledge their
positive emotions and encourage them to continue.
- Be supportive and celebratory while staying genuine.
- Ask follow-up questions to understand what's going well.`;
    } else if (sentimentLabel === 'negative') {
        prompt += `- The user seems to be struggling or feeling down.
Provide compassionate support.
- Validate their feelings and offer comfort.
- Suggest coping strategies or gentle encouragement.
- Be careful not to dismiss their concerns.`;
    } else {
        prompt += `- The user's sentiment is neutral. Provide balanced
support.
- Try to understand more about their situation.
- Offer helpful guidance without making assumptions.`;
    }

    prompt += `\n\nKeep responses concise (2-3 sentences), warm, and
genuinely supportive. Focus on being helpful rather than clinical.`;

    return prompt;
}

app.http('generate-response', {
    methods: ['POST'],
    authLevel: 'anonymous',
    route: 'generate-response',
    handler: async (request, context) => {
        context.log('Processing response generation request');
        try {
            const { messages, sentiment } = await request.json();
            if (!messages || !sentiment) {
                return {
```

```
                    status: 400,
                    headers: {
                        'Content-Type': 'application/json'
                    },
                    body: JSON.stringify({ error: 'Messages and sentiment
are required' })
                };
            }

            const endpoint = process.env.AZURE_OPENAI_ENDPOINT;
            const key = process.env.AZURE_OPENAI_KEY;
            const deploymentName = process.env.AZURE_OPENAI_DEPLOYMENT_NAME
|| 'gpt-35-turbo';
            const apiVersion = process.env.AZURE_OPENAI_API_VERSION ||
'2024-02-01';

            // Debug logging
            context.log('Azure OpenAI Configuration:');
            context.log('Endpoint:', endpoint);
            context.log('Deployment Name:', deploymentName);
            context.log('API Version:', apiVersion);
            context.log('Key exists:', !!key);

            if (!endpoint || !key) {
                context.error('Azure OpenAI configuration missing');
                return {
                    status: 500,
                    headers: {
                        'Content-Type': 'application/json'
                    },
                    body: JSON.stringify({ error: 'Azure OpenAI
configuration missing' })
                };
            }

            const systemPrompt = getSystemPrompt(sentiment);
            const fullMessages = [
                { role: 'system', content: systemPrompt },
                ...messages
            ];

            context.log('Sending request to Azure OpenAI...');

            // Use REST API directly to avoid SDK issues
            const url =
`${endpoint}/openai/deployments/${deploymentName}/chat/completions?api-
version=${apiVersion}`;
            const response = await fetch(url, {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json',
```

```
                    'api-key': key
                },
                body: JSON.stringify({
                    messages: fullMessages,
                    max_tokens: 300,
                    temperature: 0.7
                })
            });

            if (!response.ok) {
                const errorText = await response.text();
                throw new Error(`Azure OpenAI API error: ${response.status}
${response.statusText} - ${errorText}`);
            }

            const data = await response.json();

            if (!data.choices || data.choices.length === 0) {
                throw new Error('No response from Azure OpenAI');
            }

            const generatedResponse = data.choices[0].message.content;

            context.log('Successfully generated response');

            return {
                status: 200,
                headers: {
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify({ content: generatedResponse })
            };

        } catch (error) {
            context.error('Error in generate-response:', error);
            // More detailed error logging
            if (error.code) {
                context.error('Error code:', error.code);
            }
            if (error.message) {
                context.error('Error message:', error.message);
            }
            if (error.stack) {
                context.error('Error stack:', error.stack);
            }
            return {
                status: 500,
                headers: {
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify({
```

```
                    error: 'Internal server error: ' + error.message,
                    code: error.code || 'unknown'
                })
            };
        }
    }
});
```

src/index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Sentiment Support Bot</title>
    <style>
        * {
            margin: 0;
            padding: 0;
            box-sizing: border-box;
        }

        body {
            font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI',
Roboto, sans-serif;
            background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
            min-height: 100vh;
            display: flex;
            align-items: center;
            justify-content: center;
            padding: 20px;
        }

        .container {
            background: rgba(255, 255, 255, 0.95);
            backdrop-filter: blur(10px);
            border-radius: 20px;
            padding: 30px;
            max-width: 600px;
            width: 100%;
            box-shadow: 0 20px 40px rgba(0, 0, 0, 0.1);
        }

        .header {
            text-align: center;
            margin-bottom: 30px;
```

```
        }

        .header h1 {
            color: #333;
            font-size: 2.5em;
            font-weight: 700;
            margin-bottom: 10px;
        }

        .header p {
            color: #666;
            font-size: 1.1em;
        }

        .chat-container {
            background: #f8f9fa;
            border-radius: 15px;
            padding: 20px;
            margin-bottom: 20px;
            min-height: 300px;
            max-height: 400px;
            overflow-y: auto;
        }

        .message {
            margin-bottom: 15px;
            padding: 15px;
            border-radius: 12px;
            max-width: 80%;
            word-wrap: break-word;
            animation: fadeIn 0.3s ease-in;
        }

        .user-message {
            background: #007bff;
            color: white;
            margin-left: auto;
            text-align: right;
        }

        .bot-message {
            background: white;
            color: #333;
            border: 1px solid #e0e0e0;
            margin-right: auto;
        }

        .sentiment-indicator {
            display: inline-block;
            padding: 4px 8px;
            border-radius: 20px;
```

```
            font-size: 0.8em;
            font-weight: 600;
            margin-bottom: 8px;
        }

        .sentiment-positive { background: #d4edda; color: #155724; }
        .sentiment-negative { background: #f8d7da; color: #721c24; }
        .sentiment-neutral { background: #e2e3e5; color: #383d41; }

        .input-container {
            display: flex;
            gap: 10px;
            margin-bottom: 20px;
        }

        .input-field {
            flex: 1;
            padding: 15px;
            border: 2px solid #e0e0e0;
            border-radius: 12px;
            font-size: 1em;
            outline: none;
            transition: border-color 0.3s ease;
        }

        .input-field:focus {
            border-color: #007bff;
        }

        .send-button {
            background: linear-gradient(45deg, #007bff, #0056b3);
            color: white;
            border: none;
            padding: 15px 25px;
            border-radius: 12px;
            font-size: 1em;
            font-weight: 600;
            cursor: pointer;
            transition: transform 0.2s ease, box-shadow 0.2s ease;
        }

        .send-button:hover {
            transform: translateY(-2px);
            box-shadow: 0 10px 20px rgba(0, 123, 255, 0.3);
        }

        .send-button:disabled {
            background: #ccc;
            cursor: not-allowed;
            transform: none;
            box-shadow: none;
```

```
        }



        .loading {
            display: none;
            text-align: center;
            color: #666;
            font-style: italic;
        }

        .error {
            color: #dc3545;
            background: #f8d7da;
            padding: 10px;
            border-radius: 8px;
            margin-bottom: 15px;
        }

        @keyframes fadeIn {
            from { opacity: 0; transform: translateY(10px); }
            to { opacity: 1; transform: translateY(0); }
        }

        @media (max-width: 600px) {
            .input-container {
                flex-direction: column;
            }
        }
    </style>
</head>
<body>
    <div class="container">
        <div class="header">
            <h1> Sentiment Support Bot</h1>
            <p>AI-powered emotional support with sentiment analysis</p>
        </div>



        <div class="chat-container" id="chatContainer">
            <div class="message bot-message">
                <div class="sentiment-indicator sentiment-
positive">Positive</div>
                Hello! I'm here to provide emotional support. Please share
what's on your mind, and I'll analyze your sentiment and respond
accordingly.
            </div>
        </div>

        <div class="loading" id="loading">Analyzing sentiment and generating
```

```
response...</div>
        <div class="error" id="error" style="display: none;"></div>

        <div class="input-container">
            <input type="text" id="userInput" class="input-field"
placeholder="Share your thoughts or feelings..."
onkeypress="handleKeyPress(event)">
            <button onclick="sendMessage()" class="send-button"
id="sendButton">Send</button>
        </div>
    </div>

    <script>
        class SentimentSupportBot {
            constructor() {
                this.chatContainer =
document.getElementById('chatContainer');
                this.userInput = document.getElementById('userInput');
                this.sendButton = document.getElementById('sendButton');
                this.loading = document.getElementById('loading');
                this.error = document.getElementById('error');
                this.conversationHistory = [];
            }

            async analyzeSentiment(text) {
                const response = await fetch('/api/analyze-sentiment', {
                    method: 'POST',
                    headers: {
                        'Content-Type': 'application/json'
                    },
                    body: JSON.stringify({ text })
                });

                if (!response.ok) {
                    throw new Error(`Sentiment analysis failed:
${response.status}`);
                }

                const data = await response.json();
                return data;
            }

            async generateSupportResponse(userMessage, sentiment) {
                const messages = [
                    ...this.conversationHistory,
                    { role: 'user', content: userMessage }
                ];

                const response = await fetch('/api/generate-response', {
                    method: 'POST',
                    headers: {
```

```
                        'Content-Type': 'application/json'
                    },
                    body: JSON.stringify({
                        messages,
                        sentiment
                    })
                });

                if (!response.ok) {
                    throw new Error(`Response generation failed:
${response.status}`);
                }

                const data = await response.json();
                return data.content;
            }

            getSystemPrompt(sentiment) {
                const sentimentLabel = sentiment.sentiment;
                const confidence = sentiment.confidenceScores;

                let prompt = `You are an empathetic AI support assistant.
The user's message has been analyzed with sentiment: ${sentimentLabel}
(positive: ${confidence.positive.toFixed(2)}, neutral:
${confidence.neutral.toFixed(2)}, negative:
${confidence.negative.toFixed(2)}).

Based on this sentiment analysis, provide appropriate emotional support:

`;

                if (sentimentLabel === 'positive') {
                    prompt += `- The user seems to be feeling good.
Acknowledge their positive emotions and encourage them to continue.
- Be supportive and celebratory while staying genuine.
- Ask follow-up questions to understand what's going well.`;
                } else if (sentimentLabel === 'negative') {
                    prompt += `- The user seems to be struggling or feeling
down. Provide compassionate support.
- Validate their feelings and offer comfort.
- Suggest coping strategies or gentle encouragement.
- Be careful not to dismiss their concerns.`;
                } else {
                    prompt += `- The user's sentiment is neutral. Provide
balanced support.
- Try to understand more about their situation.
- Offer helpful guidance without making assumptions.`;
                }

                prompt += `\n\nKeep responses concise (2-3 sentences), warm,
and genuinely supportive. Focus on being helpful rather than clinical.`;
```

```
                return prompt;
            }

            addMessage(content, isUser = false, sentiment = null) {
                const messageDiv = document.createElement('div');
                messageDiv.className = `message ${isUser ? 'user-message' :
'bot-message'}`;
                if (!isUser && sentiment) {
                    const sentimentIndicator =
document.createElement('div');
                    sentimentIndicator.className = `sentiment-indicator
sentiment-${sentiment.sentiment}`;
                    sentimentIndicator.textContent =
sentiment.sentiment.charAt(0).toUpperCase() + sentiment.sentiment.slice(1);
                    messageDiv.appendChild(sentimentIndicator);
                }
                const contentDiv = document.createElement('div');
                contentDiv.textContent = content;
                messageDiv.appendChild(contentDiv);
                this.chatContainer.appendChild(messageDiv);
                this.chatContainer.scrollTop =
this.chatContainer.scrollHeight;
            }

            showError(message) {
                this.error.textContent = message;
                this.error.style.display = 'block';
                setTimeout(() => {
                    this.error.style.display = 'none';
                }, 5000);
            }

            setLoading(isLoading) {
                this.loading.style.display = isLoading ? 'block' : 'none';
                this.sendButton.disabled = isLoading;
                this.userInput.disabled = isLoading;
            }

            async sendMessage() {
                const message = this.userInput.value.trim();
                if (!message) return;

                this.addMessage(message, true);
                this.userInput.value = '';
                this.setLoading(true);

                try {
                    // Analyze sentiment
                    const sentimentResult = await
this.analyzeSentiment(message);
                    // Generate response
```

```
                    const supportResponse = await
this.generateSupportResponse(message, sentimentResult);
                    // Add to conversation history
                    this.conversationHistory.push(
                        { role: 'user', content: message },
                        { role: 'assistant', content: supportResponse }
                    );
                    // Keep only last 10 messages for context
                    if (this.conversationHistory.length > 10) {
                        this.conversationHistory =
this.conversationHistory.slice(-10);
                    }
                    this.addMessage(supportResponse, false,
sentimentResult);
                } catch (error) {
                    this.showError(error.message);
                } finally {
                    this.setLoading(false);
                }
            }
        }

        // Initialize the bot
        const bot = new SentimentSupportBot();

        // Global functions for HTML events
        function sendMessage() {
            bot.sendMessage();
        }

        function handleKeyPress(event) {
            if (event.key === 'Enter') {
                sendMessage();
            }
        }

        // Focus on input when page loads
        document.addEventListener('DOMContentLoaded', function() {
            document.getElementById('userInput').focus();
        });
    </script>
</body>
</html>
```

staticwebapp.config.json

```
{
  "routes": [
    {
      "route": "/api/*",
```

```
      "methods": ["GET", "POST", "PUT", "DELETE"],
      "allowedRoles": ["anonymous"]
    },
    {
      "route": "/*",
      "serve": "/index.html",
      "statusCode": 200
    }
  ],
  "navigationFallback": {
    "rewrite": "/index.html"
  },
  "mimeTypes": {
    ".json": "application/json"
  },
  "defaultHeaders": {
    "content-security-policy": "default-src 'self' 'unsafe-inline' https:",
    "x-frame-options": "DENY",
    "x-content-type-options": "nosniff"
  }
}
```

Key points to note: 1. do not use function.json for Azure functions v4 2. check the build and deploy step to verify function is created.