Tech4Biz
Excellence Guaranteed!

AI Voice Agent Reminder & Check-In System

End-to-End Technical Implementation

Prepared by:Tech4Biz Solutions Pvt Ltd

https://tech4bizsolutions.com
contact@tech4biz.io

NGR Arcade, 4th FLoor, 27th Main Road, Sector 2
HSR Layout
Bengaluru 560102

Table of Contents

Executive Summary

This document provides a comprehensive technical implementation plan for the AI Voice Agent Reminder & Check-In System. The system will automatically place scheduled calls to seniors, children, and individuals with special needs, engage in pre-defined conversations, analyze responses, and alert caregivers based on results. The implementation leverages GoHighLevel (GHL) for contact management, n8n for workflow automation, and Retell AI for voice agent capabilities.

System Architecture Overview

The system operates through three interconnected platforms:

1. **GoHighLevel (GHL)** - Central database and user interface
2. **n8n** - Workflow automation engine
3. **Retell AI** - Voice interaction system

1. Data Structure & Organization

1.1 GoHighLevel Contact Record Structure

Core Contact Data

Field Group	Field Name	Type	Description
Basic Info	first_name	Text	Recipient's first name
Basic Info	last_name	Text	Recipient's last name
Basic Info	phone	Phone	Primary phone number for calls
Basic Info	email	Email	Email address (optional)
Basic Info	contact_type	Dropdown	Senior, Child, Special Needs
Basic Info	timezone	Dropdown	Recipient's timezone
Basic Info	special_instructions	Long Text	Special handling notes

Caregiver Network

Field Group	Field Name	Type	Description
Caregiver	primary_caregiver_name	Text	Primary contact name
Caregiver	primary_caregiver_phone	Phone	Primary contact phone
Caregiver	primary_caregiver_email	Email	Primary contact email
Caregiver	secondary_caregiver_name	Text	Backup contact name
Caregiver	secondary_caregiver_phone	Phone	Backup contact phone
Caregiver	secondary_caregiver_email	Email	Backup contact email
Caregiver	emergency_contact_name	Text	Emergency contact
Caregiver	emergency_contact_phone	Phone	Emergency phone
Caregiver	escalation_order	JSON	Ordered list of notification contacts

Call Configuration

Field Group	Field Name	Type	Description
Schedule	call_frequency	Multiple Select	Days of week to call
Schedule	call_time	Time	Time to place call (in recipient's timezone)
Schedule	call_active	Boolean	Whether calls are currently active
Schedule	retry_max	Number	Maximum retry attempts (default: 2)
Schedule	retry_interval	Number	Minutes between retries (default: 5)
Schedule	call_priority	Number	Call priority (1-5, 5 highest)

Script Configuration

Field Group	Field Name	Type	Description
Script	greeting_script	Long Text	Opening greeting message
Script	question_1	Long Text	First question text
Script	question_1_type	Dropdown	Yes/No, Multiple Choice, Open-ended
Script	question_1_options	Text	Options for multiple choice
Script	question_1_alert_on	Text	Responses that trigger alerts
Script	question_2	Long Text	Second question text
Script	question_2_type	Dropdown	Question response type
Script	question_2_options	Text	Options for multiple choice
Script	question_2_alert_on	Text	Responses that trigger alerts

Script	question_3	Long Text	Third question text
Script	question_3_type	Dropdown	Question response type
Script	question_3_options	Text	Options for multiple choice
Script	question_3_alert_on	Text	Responses that trigger alerts
Script	question_4	Long Text	Fourth question text (optional)
Script	question_4_type	Dropdown	Question response type
Script	question_4_options	Text	Options for multiple choice
Script	question_4_alert_on	Text	Responses that trigger alerts
Script	question_5	Long Text	Fifth question text (optional)
Script	question_5_type	Dropdown	Question response type
Script	question_5_options	Text	Options for multiple choice
Script	question_5_alert_on	Text	Responses that trigger alerts
Script	closing_script	Long Text	Closing message
Script	voice_gender	Dropdown	Preferred voice gender
Script	voice_style	Dropdown	Casual, Professional, Warm

Response Tracking

Field Group	Field Name	Type	Description
Responses	last_call_datetime	DateTime	When last call was made
Responses	last_call_status	Dropdown	Successful, No Answer, Failed
Responses	last_call_responses	JSON	Structured response data
Responses	response_history	JSON	Last 10 call responses
Responses	alert_count_7days	Number	Alerts in last 7 days
Responses	missed_call_count_7days	Number	Missed calls in last 7 days
Responses	concerning_response_log	Long Text	Log of concerning responses

1.2 Tags & Categories

The following tags will be used in GHL:

- Contact Type Tags: Senior, Child, Special_Needs
- Status Tags: Active, Paused, Terminated
- Alert Tags: Missed_Call, Concerning_Response, Emergency_Alert
- System Tags: Test_Contact, VIP, Special_Handling

1.3 Pipelines & Stages

Onboarding Pipeline:

1. Initial Setup (Contact created)
2. Script Configuration (Questions defined)
3. Test Call Completed (System validated)
4. Active Monitoring (Regular calls in progress)
5. Paused (Temporarily suspended)
6. Terminated (Service ended)

Alert Pipeline:

1. Alert Generated (System detected issue)
2. Caregiver Notified (Notification sent)
3. Awaiting Response (Waiting for caregiver)
4. Resolved (Issue addressed)
5. Escalated (Higher level intervention)

2. User Interfaces & Forms

2.1 New Contact Registration Form

A comprehensive form will be created in GHL capturing:

- Recipient personal information
- Caregiver network details
- Schedule preferences
- Special instructions and notes
- Legal consent for recording and monitoring

Form logic will include:

- Required fields validation
- Phone number formatting
- Timezone detection based on area code
- Conditional fields based on contact type

2.2 Call Script Configuration Form

A dedicated form for configuring the conversation script:

- Customizable greeting and closing
- Up to 5 configurable questions
- Response type selection for each question
- Alert trigger configuration
- Voice preference settings

Form logic will include:

- Script previewing capability
- Character count limitations
- Alert threshold configuration
- Sample response testing

2.3 Caregiver Dashboard

A custom GHL dashboard showing:

- Recent call statuses with color coding
- Upcoming scheduled calls
- Alert history and status
- Quick access to update contact information
- Response trends visualization

2.4 Admin Control Panel

A comprehensive administration interface:

- System-wide status monitoring
- Batch operations for multiple contacts
- Voice agent configuration
- Script template management
- Performance analytics

3. n8n Workflow Implementation

3.1 Core Workflow Architecture

The system will use these primary n8n workflows:

1. **Schedule Manager Workflow**
2. **Call Execution Workflow**
3. **Response Analysis Workflow**
4. **Retry & Escalation Workflow**
5. **Notification Workflow**
6. **System Monitoring Workflow**

3.2 Schedule Manager Workflow

Purpose: Identify contacts due for calls and initiate the call process

Trigger: Time-based cron job (runs every 5 minutes)

Steps:

```
1. Query GHL API for active contacts
2. // n8n HTTP Request node
3. {
4.   "method": "GET",
5.   "url": "https://rest.gohighlevel.com/v1/contacts/",
6.   "headers": {
7.     "Authorization": "Bearer {{$node['Credentials'].json.access_token}}"
8.   },
9.   "params": {
10.    "tags": "Active",
11.    "limit": 100
12.  }
13. }
14. Filter contacts by scheduled call time
15. // n8n Function node
16. const contacts = items[0].json.contacts;
17. const now = new Date();
18. const timeWindow = 5; // minutes
19.
20. return contacts.filter(contact => {
21.   // Get call schedule from contact custom fields
22.   const callDays = contact.customField.call_frequency || [];
23.   const callTimeStr = contact.customField.call_time || "09:00";
24.   const timezone = contact.customField.timezone || "America/New_York";
25.
26.   // Check if today is a scheduled day
27.   const today = now.toLocaleDateString('en-US', { weekday: 'long', timeZone: timezone });
28.   if (!callDays.includes(today)) return false;
29.
30.   // Check if current time is within window
31.   const [callHour, callMinute] = callTimeStr.split(':').map(Number);
```


```
  const contactTime = new Date(now.toLocaleDateString('en-US', { timeZone: timezone }));
  contactTime.setHours(callHour, callMinute, 0, 0);

  const diffMinutes = Math.abs((now - contactTime) / 60000);
  return diffMinutes <= timeWindow;
});
Sort contacts by priority
// n8n Function node
return items.sort((a, b) => {
  const priorityA = parseInt(a.json.customField.call_priority) || 3;
  const priorityB = parseInt(b.json.customField.call_priority) || 3;
  return priorityB - priorityA; // Descending (higher priority first)
});
Split into batches for processing
// n8n SplitInBatches node
{
  "batchSize": 10
}
For each contact, prepare script data
// n8n Function node
const contact = item.json;

// Build script with variable replacement
const greeting = contact.customField.greeting_script.replace(
  "{FirstName}",
  contact.firstName
);

// Format questions
const questions = [];
for (let i = 1; i <= 5; i++) {
  const qField = `question_${i}`;
  const qTypeField = `question_${i}_type`;
  if (contact.customField[qField]) {
    questions.push({
      text: contact.customField[qField],
      type: contact.customField[qTypeField] || "open",
      alertOn: contact.customField[`question_${i}_alert_on`] || ""
    });
  }
}

return {
  json: {
    contactId: contact.id,
    phone: contact.phone,
    name: `${contact.firstName} ${contact.lastName}`,
    script: {
      greeting: greeting,
      questions: questions,
```


```
    closing: contact.customField.closing_script
  },
  voicePreferences: {
    gender: contact.customField.voice_gender || "female",
    style: contact.customField.voice_style || "warm"
  },
  retrySettings: {
    maxRetries: parseInt(contact.customField.retry_max) || 2,
    interval: parseInt(contact.customField.retry_interval) || 5
  },
  caregivers: {
    primary: {
      name: contact.customField.primary_caregiver_name,
      phone: contact.customField.primary_caregiver_phone,
      email: contact.customField.primary_caregiver_email
    },
    secondary: {
      name: contact.customField.secondary_caregiver_name,
          phone: contact.customField.secondary_caregiver_phone,
          email: contact.customField.secondary_caregiver_email
        }
      }
    }
  };
  Call the Call Execution Workflow
  // n8n ExecuteWorkflow node
  {
    "workflowId": "12345", // ID of Call Execution Workflow
    "runWithContext": true
  }
```

3.3 Call Execution Workflow

Purpose: Execute the actual call via Retell AI

Trigger: Called from Schedule Manager Workflow

Steps:

```
Format Retell API request
// n8n Function node
const callData = items[0].json;

// Build questions in Retell format
const retellQuestions = callData.script.questions.map(q => ({
  question: q.text,
```


```
    responseType: mapResponseType(q.type),
    options: q.options ? q.options.split(',') : []
}));

function mapResponseType(type) {
  switch(type) {
    case 'yes_no': return 'boolean';
    case 'multiple_choice': return 'select';
    default: return 'open';
  }
}

return {
  json: {
    recipient: {
      phone: callData.phone,
      name: callData.name
    },
    script: {
      introduction: callData.script.greeting,
      questions: retellQuestions,
      conclusion: callData.script.closing
    },
    voice: {
      gender: callData.voicePreferences.gender,
      style: callData.voicePreferences.style,
      speed: 0.95 // slightly slower for clarity
    },
    webhooks: {
      completion: "{{$workflow.staticData.webhookBaseUrl}}/call-complete",
      response: "{{$workflow.staticData.webhookBaseUrl}}/response-received"
    },
    metadata: {
      contactId: callData.contactId,
      callId: "{{$workflow.id}}-{{$node.id}}",
      timestamp: new Date().toISOString()
    }
  }
};
Make Retell API call
// n8n HTTP Request node
{
  "method": "POST",
  "url": "https://api.retell.io/v1/calls",
  "headers": {
    "Authorization": "Bearer {{$credentials.retellApiKey}}",
    "Content-Type": "application/json"
  },
  "body": "={{$node['Format Retell Request'].json}}"
}
```


```
58. Create call log in GHL
59. // n8n HTTP Request node
60. {
61.   "method": "POST",
62.   "url": "https://rest.gohighlevel.com/v1/contacts/{{$node['Format Retell
Request'].json.metadata.contactId}}/notes",
63.   "headers": {
64.     "Authorization": "Bearer {{$credentials.ghlApiKey}}"
65.   },
66.   "body": {
67.     "title": "Outbound Call Initiated",
68.     "body": "Call initiated at {{$now.format('YYYY-MM-DD HH:mm:ss')}}. Call ID:
{{$node['Format Retell Request'].json.metadata.callId}}"
69.   }
70. }
71. Wait for Retell webhook callback
72. // n8n Webhook node
73. {
74.   "httpMethod": "POST",
75.   "path": "call-complete",
76.   "respondWith": "{\"status\":\"success\"}",
77.   "options": {
78.     "waitForWebhook": true,
79.     "timeout": 300 // 5 minutes timeout
80.   }
81. }
82. Process call results
83. // n8n Function node
84. const webhookData = items[0].json;
85. const callId = webhookData.metadata.callId;
86. const contactId = webhookData.metadata.contactId;
87. const callStatus = webhookData.status; // 'completed', 'no_answer', 'failed'
88. const transcript = webhookData.transcript || [];
89. const responses = webhookData.responses || [];
90.
91. // Format responses for storage
92. const formattedResponses = responses.map((r, i) => ({
93.   question: items[0].json.script.questions[i]?.text || 'Unknown',
94.   answer: r.text || 'No response',
95.   sentiment: r.sentiment || 'neutral'
96. }));
97.
98. // Determine if any responses need alerts
99. const alertTriggers = responses.filter(r =>
100.       r.sentiment === 'negative' ||
101.       r.alert === true ||
102.       containsAlertKeywords(r.text)
103.     );
104.
105.     const needsAlert = alertTriggers.length > 0;
```


```
106.
107.    function containsAlertKeywords(text) {
108.      if (!text) return false;
109.      const alertWords = ['help', 'pain', 'hurt', 'fell', 'emergency', 'hospital'];
110.      return alertWords.some(word => text.toLowerCase().includes(word));
111.    }
112.
113.    return {
114.      json: {
115.        contactId,
116.        callId,
117.        status: callStatus,
118.        responses: formattedResponses,
119.        transcript,
120.        needsAlert,
121.        alertTriggers
122.      }
123.    };
124.    Branch based on call outcome
125.    // n8n Switch node
126.    {
127.      "rules": [
128.        {
129.          "operation": "equals",
130.          "value1": "={{$node['Process Results'].json.status}}",
131.          "value2": "completed"
132.        },
133.        {
134.          "operation": "equals",
135.          "value1": "={{$node['Process Results'].json.status}}",
136.          "value2": "no_answer"
137.        },
138.        {
139.          "operation": "equals",
140.          "value1": "={{$node['Process Results'].json.status}}",
141.          "value2": "failed"
142.        }
143.      ]
144.    }
145.    For completed calls: Update GHL with responses
146.    // n8n HTTP Request node
147.    {
148.      "method": "PUT",
149.      "url": "https://rest.gohighlevel.com/v1/contacts/{{$node['Process
Results'].json.contactId}}",
150.      "headers": {
151.        "Authorization": "Bearer {{$credentials.ghlApiKey}}"
152.      },
153.      "body": {
154.        "customField": {
```


```
          "last_call_datetime": "{{$now.format('YYYY-MM-DD HH:mm:ss')}}",
          "last_call_status": "Successful",
          "last_call_responses": "={{JSON.stringify($node['Process
Results'].json.responses)}}",
          // Add response to history (maintaining last 10)
          "response_history": "={{
            const current = JSON.parse($node['Get
Contact'].json.customField.response_history || '[]');
            const newEntry = {
              date: $now.format('YYYY-MM-DD HH:mm:ss'),
              responses: $node['Process Results'].json.responses
            };
            const updated = [newEntry, ...current].slice(0, 10);
            JSON.stringify(updated)
          }}"
        }
      }
    }
    For completed calls: Check if alert needed
    // n8n If node
    {
      "condition": "={{$node['Process Results'].json.needsAlert}}"
    }
    If alert needed: Trigger Alert Workflow
    // n8n ExecuteWorkflow node
    {
      "workflowId": "67890", // ID of Alert Workflow
      "runWithContext": true
    }
    For no_answer: Update retry count and trigger Retry Workflow
    // n8n Function node
    const contactId = items[0].json.contactId;
    const currentRetries = parseInt(items[0].json.retryCount || 0);
    const maxRetries = parseInt(items[0].json.retrySettings.maxRetries);

    if (currentRetries < maxRetries) {
      // Schedule a retry
      return {
        json: {
          contactId,
          retryCount: currentRetries + 1,
          retryAfter: items[0].json.retrySettings.interval // minutes
        }
      };
    } else {
      // Max retries reached, trigger alert
      return {
        json: {
          contactId,
          status: "max_retries_reached",
```


```
          alertReason: "No answer after maximum retry attempts"
        }
      };
    }
```

3.4 Response Analysis Workflow

Purpose: Analyze call responses for patterns and potential concerns

Trigger: After successful call completion

Steps:

```
Retrieve contact history
// n8n HTTP Request node
{
  "method": "GET",
  "url": "https://rest.gohighlevel.com/v1/contacts/{{$input.item.json.contactId}}",
  "headers": {
    "Authorization": "Bearer {{$credentials.ghlApiKey}}"
  }
}
Analyze response patterns
// n8n Function node
const contact = items[0].json;
const currentResponses = $input.item.json.responses;
let responseHistory = [];

try {
  responseHistory = JSON.parse(contact.customField.response_history || '[]');
} catch (e) {
  responseHistory = [];
}

// Compare current responses with historical patterns
const analysis = currentResponses.map((response, i) => {
  const questionText = response.question;
  const currentAnswer = response.answer;

  // Find previous answers to the same question
  const previousAnswers = responseHistory
    .map(h => h.responses.find(r => r.question === questionText))
    .filter(Boolean)
    .map(r => r.answer);

  // Detect significant changes
```


```
  let changeDetected = false;
  let trend = "stable";

  if (previousAnswers.length > 0) {
    const lastAnswer = previousAnswers[0];

    // Simple change detection - can be enhanced with NLP
    if (response.sentiment === 'negative' &&
        previousAnswers.slice(0, 3).every(a => a.sentiment !== 'negative')) {
      changeDetected = true;
      trend = "declining";
    }

    // For yes/no questions, detect changes in pattern
    if ((response.question.toLowerCase().includes('medication') ||
         response.question.toLowerCase().includes('medicine')) &&
        isYesNoQuestion(response.question)) {

      const wasYes = isAffirmative(lastAnswer);
      const isYes = isAffirmative(currentAnswer);

      if (wasYes && !isYes) {
        changeDetected = true;
        trend = "concerning";
      }
    }
  }

  return {
    question: questionText,
    currentAnswer,
    changeDetected,
    trend,
    previousAnswers: previousAnswers.slice(0, 3) // Last 3 answers
  };
});

// Determine if follow-up is needed based on analysis
const needsFollowUp = analysis.some(a => a.changeDetected && a.trend === "concerning");

function isYesNoQuestion(question) {
  // Simple heuristic - can be improved
  return question.trim().endsWith('?') &&
         (question.toLowerCase().includes('did you') ||
          question.toLowerCase().includes('have you'));
}

function isAffirmative(text) {
  if (!text) return false;
  const affirmative = ['yes', 'yeah', 'yep', 'sure', 'of course', 'i did'];
```


```
  return affirmative.some(word => text.toLowerCase().includes(word));
}

return {
  json: {
    contactId: contact.id,
    analysis,
    needsFollowUp,
    riskLevel: needsFollowUp ? "medium" : "low"
  }
};
Update contact with analysis results
// n8n HTTP Request node
{
  "method": "PUT",
  "url": "https://rest.gohighlevel.com/v1/contacts/{{$node['Analyze
Patterns'].json.contactId}}",
      "headers": {
        "Authorization": "Bearer {{$credentials.ghlApiKey}}"
      },
      "body": {
        "customField": {
          "response_analysis": "={{JSON.stringify($node['Analyze
Patterns'].json.analysis)}}",
          "risk_level": "={{$node['Analyze Patterns'].json.riskLevel}}"
        }
      }
    }
    If follow-up needed, add to Alert Pipeline
    // n8n If node
    {
      "condition": "={{$node['Analyze Patterns'].json.needsFollowUp}}"
    }
    // n8n HTTP Request node - Create opportunity
    {
      "method": "POST",
      "url":
"https://rest.gohighlevel.com/v1/pipelines/{{$workflow.staticData.alertPipelineId}}/opportuni
ties",
      "headers": {
        "Authorization": "Bearer {{$credentials.ghlApiKey}}"
      },
      "body": {
        "title": "Follow-up Required: {{$node['Get Contact'].json.name}}",
        "contactId": "{{$node['Analyze Patterns'].json.contactId}}",
        "status": "Alert Generated",
        "monetaryValue": 0,
        "assignedTo": "{{$workflow.staticData.defaultAssigneeId}}",
        "note": "Pattern analysis detected concerning changes in responses. Please review
and follow up."
```


```
129.       }
130.     }
```

3.5 Retry & Escalation Workflow

Purpose: Handle missed calls with retries and alerts

Trigger: After call with no answer

Steps:

```
1.  Check retry count against maximum
2.  // n8n Function node
3.  const retryCount = $input.item.json.retryCount;
4.  const maxRetries = $input.item.json.retrySettings.maxRetries;
5.
6.  return {
7.    json: {
8.      ...items[0].json,
9.      shouldRetry: retryCount < maxRetries,
10.     isLastRetry: retryCount === maxRetries - 1,
11.     maxRetriesReached: retryCount >= maxRetries
12.   }
13. };
14. If retries available, schedule retry
15. // n8n If node
16. {
17.   "condition": "={{$node['Check Retry Status'].json.shouldRetry}}"
18. }
19.
20. // n8n Wait node
21. {
22.   "unit": "minutes",
23.   "amount": "={{$input.item.json.retrySettings.interval}}"
24. }
25.
26. // n8n HTTP Request node - Update contact
27. {
28.   "method": "PUT",
29.   "url": "https://rest.gohighlevel.com/v1/contacts/{{$node['Check Retry
Status'].json.contactId}}",
30.   "headers": {
31.     "Authorization": "Bearer {{$credentials.ghlApiKey}}"
32.   },
33.   "body": {
34.     "customField": {
```


```
      "last_call_status": "Retry Scheduled",
      "retry_count": "={{$node['Check Retry Status'].json.retryCount}}"
    }
  }
}

// n8n ExecuteWorkflow node - Call again
{
  "workflowId": "12345", // ID of Call Execution Workflow
  "runWithContext": true
}
If max retries reached, trigger caregiver notification
// n8n If node
{
  "condition": "={{$node['Check Retry Status'].json.maxRetriesReached}}"
}

// n8n HTTP Request node - Update contact
{
  "method": "PUT",
  "url": "https://rest.gohighlevel.com/v1/contacts/{{$node['Check Retry
Status'].json.contactId}}",
  "headers": {
    "Authorization": "Bearer {{$credentials.ghlApiKey}}"
  },
  "body": {
    "customField": {
      "last_call_status": "Max Retries Reached",
      "missed_call_count_7days": "={{
        parseInt($node['Get Contact'].json.customField.missed_call_count_7days || 0) + 1
      }}"
    },
    "tags": ["Missed_Call"]
  }
}
Send SMS to primary caregiver
// n8n HTTP Request node
{
  "method": "POST",
  "url": "https://rest.gohighlevel.com/v1/sms/messages",
  "headers": {
    "Authorization": "Bearer {{$credentials.ghlApiKey}}"
  },
  "body": {
    "from": "{{$workflow.staticData.smsNumber}}",
    "to": "{{$node['Get Contact'].json.customField.primary_caregiver_phone}}",
    "message": "ALERT: We've been unable to reach {{$node['Get Contact'].json.firstName}}
after {{$node['Check Retry Status'].json.maxRetries}} attempts. Please check on them and
reply CONFIRM when they're okay."
  }
```


```
82. }
83. Call primary caregiver via Retell AI
84. // n8n HTTP Request node
85. {
86.   "method": "POST",
87.   "url": "https://api.retell.io/v1/calls",
88.   "headers": {
89.     "Authorization": "Bearer {{$credentials.retellApiKey}}",
90.     "Content-Type": "application/json"
91.   },
92.   "body": {
93.     "recipient": {
94.       "phone": "{{$node['Get Contact'].json.customField.primary_caregiver_phone}}",
95.       "name": "{{$node['Get Contact'].json.customField.primary_caregiver_name}}"
96.     },
97.     "script": {
98.       "introduction": "Hello, this is an urgent alert from the care monitoring system.",
99.       "body": "We've been unable to reach {{$node['Get Contact'].json.firstName}}
{{$node['Get Contact'].json.lastName}} after multiple attempts. Our system tried calling at
{{$now.format('h:mm a')}} but received no answer. Please check on them as soon as possible
and follow your care protocol.",
100.          "conclusion": "To confirm you've received this message, please say 'confirmed'
or press any key."
101.        },
102.        "voice": {
103.          "gender": "female",
104.          "style": "professional",
105.          "speed": 1.0
106.        },
107.        "webhooks": {
108.          "completion": "{{$workflow.staticData.webhookBaseUrl}}/caregiver-call-complete"
109.        },
110.        "metadata": {
111.          "contactId": "{{$node['Check Retry Status'].json.contactId}}",
112.          "alertId": "{{$workflow.id}}-{{$node.id}}",
113.          "timestamp": "{{$now.format()}}"
114.        }
115.      }
116.    }
117.    Create alert record in GHL opportunity
118.    // // n8n HTTP Request node - Create opportunity
119.    {
120.      "method": "POST",
121.      "url":
"https://rest.gohighlevel.com/v1/pipelines/{{$workflow.staticData.alertPipelineId}}/opportuni
ties",
122.      "headers": {
123.        "Authorization": "Bearer {{$credentials.ghlApiKey}}"
124.      },
125.      "body": {
```


```
126.         "title": "Missed Call Alert: {{$node['Get Contact'].json.firstName}} {{$node['Get
Contact'].json.lastName}}",
127.         "contactId": "{{$node['Check Retry Status'].json.contactId}}",
128.         "stageId": "{{$workflow.staticData.alertStageId}}",
129.         "assignedTo": "{{$workflow.staticData.defaultAssigneeId}}",
130.         "monetaryValue": 0,
131.         "note": "Failed to reach contact after {{$node['Check Retry
Status'].json.maxRetries}} attempts. Caregiver has been notified."
132.       }
133.     }
```

AI Voice Agent Reminder & Check-In System: Technical Implementation (Continued)

3.5 Retry & Escalation Workflow (continued)

```
javascript
// n8n HTTP Request node - Create opportunity
{
  "method": "POST",
  "url":
"https://rest.gohighlevel.com/v1/pipelines/{{$workflow.staticData.alertPipelineId}}/opportuni
ties",
  "headers": {
    "Authorization": "Bearer {{$credentials.ghlApiKey}}"
  },
  "body": {
    "title": "Missed Call Alert: {{$node['Get Contact'].json.firstName}} {{$node['Get
Contact'].json.lastName}}",
    "contactId": "{{$node['Check Retry Status'].json.contactId}}",
    "stageId": "{{$workflow.staticData.alertStageId}}",
    "assignedTo": "{{$workflow.staticData.defaultAssigneeId}}",
    "monetaryValue": 0,
    "note": "Failed to reach contact after {{$node['Check Retry Status'].json.maxRetries}}
attempts. Caregiver has been notified."
  }
}
```

3.6 Notification Workflow

Purpose: Handle caregiver notifications for various alert scenarios Trigger: From other workflows when notifications are needed

Steps:

Get notification parameters

```
javascript
// n8n Function node
const params = {
  contactId: $input.item.json.contactId,
  alertType: $input.item.json.alertType || "general",
  message: $input.item.json.message || "",
  severity: $input.item.json.severity || "medium",
  requiresConfirmation: $input.item.json.requiresConfirmation === true
};

return { json: params };
```

Get contact and caregiver details

```
javascript
// n8n HTTP Request node
{
  "method": "GET",
  "url": "https://rest.gohighlevel.com/v1/contacts/{{$node['Get Parameters'].json.contactId}}",
  "headers": {
    "Authorization": "Bearer {{$credentials.ghlApiKey}}"
  }
}

// n8n Function node - Extract caregiver info
const contact = items[0].json;
const params = $node['Get Parameters'].json;

// Extract caregivers based on escalation order
let escalationOrder = ['primary'];
try {
  if (contact.customField.escalation_order) {
    escalationOrder = JSON.parse(contact.customField.escalation_order);
  }
} catch (e) {
  // Use default if parsing fails
}

// Get caregiver contacts in order
const caregivers = escalationOrder.map(level => {
  const prefix = level === 'emergency' ? 'emergency' : `${level}_caregiver`;
  return {
    level,
    name: contact.customField[`${prefix}_name`] || '',
    phone: contact.customField[`${prefix}_phone`] || '',
    email: contact.customField[`${prefix}_email`] || ''
  };
```


```
}).filter(cg => cg.phone || cg.email);

// Prepare message based on alert type
let messageText = params.message;
if (!messageText) {
  switch(params.alertType) {
    case "missed_call":
      messageText = `ALERT: We've been unable to reach ${contact.firstName} after multiple attempts. Please check on them and reply CONFIRM when they're okay.`;
      break;
    case "concerning_response":
      messageText = `ALERT: ${contact.firstName} provided concerning responses during their check-in call. Please review the details in your dashboard and follow up as needed.`;
      break;
    case "no_medication":
      messageText = `ALERT: ${contact.firstName} indicated they did not take their medication during today's check-in call. Please follow up as soon as possible.`;
      break;
    default:
      messageText = `ALERT: There's an issue with ${contact.firstName}'s check-in. Please review their status in your dashboard.`;
  }
}

return {
  json: {
    contactId: contact.id,
    contactName: `${contact.firstName} ${contact.lastName}`,
    caregivers,
    messageText,
    severity: params.severity,
    requiresConfirmation: params.requiresConfirmation
  }
};

```

Send SMS notifications to caregivers

```
javascript
// n8n HTTP Request node
{
  "method": "POST",
  "url": "https://rest.gohighlevel.com/v1/sms/messages",
  "headers": {
    "Authorization": "Bearer {{$credentials.ghlApiKey}}"
  },
  "body": {
    "from": "{{$workflow.staticData.smsNumber}}",
    "to": "{{$node['Prepare Notification'].json.caregivers[0].phone}}",
```


```
    "message": "{{$node['Prepare Notification'].json.messageText}}",
    "contactId": "{{$node['Prepare Notification'].json.contactId}}"
  }
}
```

If confirmation required, set up webhook listener

```
javascript
// n8n If Node
{
  "condition": "={{$node['Prepare Notification'].json.requiresConfirmation}}"
}

// n8n Webhook node
{
  "httpMethod": "POST",
  "path": "confirmation-{{$node['Prepare Notification'].json.contactId}}",
  "respondWith": "{\"status\":\"success\"}",
  "options": {
    "waitForWebhook": true,
    "timeout": 43200 // 12 hours timeout
  }
}
```

Handle confirmation timeout with escalation

```
javascript
// n8n NoOp node - Capture webhook timeout
{
  "errorMessage": "Webhook timed out without confirmation"
}

// n8n Function node - Prepare escalation
const contactId = $node['Prepare Notification'].json.contactId;
const caregivers = $node['Prepare Notification'].json.caregivers;

// Move to next caregiver in escalation chain
if (caregivers.length > 1) {
  return {
    json: {
      contactId,
      nextCaregiver: caregivers[1],
      messageText: $node['Prepare Notification'].json.messageText + " (Escalated alert - previous notification was not confirmed)",
      requiresConfirmation: true,
      severity: "high"
    }
  };
} else {
  // No more caregivers to escalate to
```


```
  return {
    json: {
      contactId,
      escalationFailed: true,
      status: "critical",
      message: "All caregiver notifications failed to receive confirmation"
    }
  };
}

```

For critical escalations, trigger emergency protocol

```
javascript
// n8n If Node
{
  "condition": "={{$node['Prepare Escalation'].json.escalationFailed === true}}"
}

// n8n HTTP Request node - Update contact status
{
  "method": "PUT",
  "url": "https://rest.gohighlevel.com/v1/contacts/{{$node['Prepare
Escalation'].json.contactId}}",
  "headers": {
    "Authorization": "Bearer {{$credentials.ghlApiKey}}"
  },
  "body": {
    "customField": {
      "alert_status": "CRITICAL",
      "last_critical_alert": "{{$now.format('YYYY-MM-DD HH:mm:ss')}}"
    },
    "tags": ["Emergency_Alert"]
  }
}

// n8n HTTP Request node - Create high priority opportunity
{
  "method": "POST",
  "url":
"https://rest.gohighlevel.com/v1/pipelines/{{$workflow.staticData.alertPipelineId}}/op
portunities",
  "headers": {
    "Authorization": "Bearer {{$credentials.ghlApiKey}}"
  },
  "body": {
    "title": "EMERGENCY: Escalation Chain Exhausted",
    "contactId": "{{$node['Prepare Escalation'].json.contactId}}",
    "stageId": "{{$workflow.staticData.emergencyStageId}}",
    "assignedTo": "{{$workflow.staticData.escalationManagerId}}",
```


```
    "monetaryValue": 0,
    "note": "Critical situation: All caregivers in the escalation chain failed to confirm alert notification. Emergency protocol should be initiated according to care plan."
  }
}
```

3.7 System Monitoring Workflow

Purpose: Ensure system health and detect operational issues Trigger: Scheduled (every 4 hours)

Steps:

Check system components

```
javascript
// n8n Function node
const components = [
  {name: "GHL API", endpoint: "https://rest.gohighlevel.com/v1/custom/health"},
  {name: "Retell API", endpoint: "https://api.retell.io/health"},
  {name: "n8n API", endpoint: "http://localhost:5678/api/v1/health"}
];

return components.map(component => ({json: component}));

// n8n HTTP Request node
{
  "method": "GET",
  "url": "={{$json.endpoint}}",
  "headers": {
    "Authorization": "Bearer {{$credentials[$json.name + 'ApiKey']}}"
  }
}
```

Analyze component status

```
javascript
// n8n Function node
const results = items.map(item => {
  const component = item.json;
  let status = "operational";
  let errorMessage = "";

  try {
    if (!component.statusCode || component.statusCode >= 400) {
```


```
      status = "down";
      errorMessage = component.error || "Error connecting to service";
    }
  } catch (e) {
    status = "error";
    errorMessage = e.message;
  }

  return {
    component: component.name,
    status,
    errorMessage,
    timestamp: new Date().toISOString()
  };
});

const systemStatus = results.every(r => r.status === "operational")
  ? "operational"
  : "degraded";

return {
  json: {
    status: systemStatus,
    components: results,
    timestamp: new Date().toISOString()
  }
};

```

Store status history

```
javascript
// n8n Function node
const currentStatus = $node['Analyze Status'].json;
let statusHistory = [];

try {
  statusHistory = JSON.parse($workflow.staticData.statusHistory || '[]');
} catch (e) {
  statusHistory = [];
}

// Keep last 24 entries (4-hour intervals = 24 hours)
statusHistory.unshift({
  timestamp: currentStatus.timestamp,
  status: currentStatus.status
});

if (statusHistory.length > 24) {
  statusHistory = statusHistory.slice(0, 24);
```


```
}

// Store updated history
$workflow.staticData.statusHistory = JSON.stringify(statusHistory);

return { json: currentStatus };

```

If issues detected, send admin notification

```
javascript
// n8n If Node
{
  "condition": "={{$node['Analyze Status'].json.status !== 'operational'}}"
}

// n8n Send Email node
{
  "to": "{{$workflow.staticData.adminEmail}}",
  "subject": "AI Voice Agent System Alert: Service Degradation",
  "text": "The following system components are experiencing issues:\n\n{{
    $node['Analyze Status'].json.components
      .filter(c => c.status !== 'operational')
      .map(c => `- ${c.component}: ${c.errorMessage}`)
      .join('\n')
  }}\n\nPlease check the system dashboard for more details.",
  "attachments": []
}

```

4. Integration with Retell AI

4.1 Retell AI Configuration

The system uses Retell AI for voice agent capabilities. The following configurations are required:

Voice agent settings

```
json
{
  "voice": {
    "gender": "female",
    "style": "warm",
    "speed": 0.95,
    "pitch": 1.0
  },
  "interaction": {
```


```
    "interruption_ms": 500,
    "endpointing_ms": 1000,
    "silence_trigger_ms": 2000
  },
  "behavior": {
    "max_turns": 20,
    "max_duration_seconds": 300,
    "speech_timeout_seconds": 5
  }
}
```

Custom error handling

```
javascript
const errorMessages = {
  "no_answer": "I'm sorry, but I didn't receive a response. Let me ask again.",
  "unclear_response": "I'm sorry, I didn't quite catch that. Could you please repeat your answer?",
  "call_dropped": "It seems our call was disconnected. I'll try calling back shortly.",
  "service_error": "I apologize, but we're experiencing a technical issue. A caregiver will follow up with you soon."
};

function handleRetellError(errorType) {
  return errorMessages[errorType] || "I apologize for the inconvenience. Let's continue our conversation.";
}
```

Response parsing and classification

```
javascript
function classifyResponse(question, responseText) {
  // Basic yes/no detection
  const yesPatterns = ['yes', 'yeah', 'yep', 'sure', 'of course', 'i did', 'i have'];
  const noPatterns = ['no', 'nope', 'haven\'t', 'didn\'t', 'not'];

  const questionLower = question.toLowerCase();
  const responseLower = responseText.toLowerCase();

  // Detect sentiment
  let sentiment = "neutral";
  const negativeWords = ['bad', 'pain', 'hurt', 'sick', 'terrible', 'awful', 'help'];
  const positiveWords = ['good', 'great', 'fine', 'well', 'okay', 'happy', 'better'];

  if (negativeWords.some(word => responseLower.includes(word))) {
    sentiment = "negative";
  } else if (positiveWords.some(word => responseLower.includes(word))) {
```


```
    sentiment = "positive";
  }

  // Detect if response needs alert
  const alertKeywords = ['emergency', 'help', 'call doctor', 'hospital', 'fell', 'fall', 'pain'];
  const needsAlert = alertKeywords.some(word => responseLower.includes(word));

  // Determine response type
  let responseType = "open";
  if (yesPatterns.some(p => responseLower.includes(p))) {
    responseType = "affirmative";
  } else if (noPatterns.some(p => responseLower.includes(p))) {
    responseType = "negative";
  }

  return {
    text: responseText,
    type: responseType,
    sentiment,
    needsAlert,
    keywords: extractKeywords(responseText)
  };
}

function extractKeywords(text) {
  // Simple keyword extraction
  const stopWords = ['i', 'me', 'my', 'the', 'a', 'an', 'and', 'but', 'or', 'in', 'on', 'at'];
  return text.toLowerCase()
    .split(/\s+/)
    .filter(word => word.length > 2)
    .filter(word => !stopWords.includes(word))
    .slice(0, 5);
}

```

5. GoHighLevel (GHL) Configuration

5.1 Custom Forms

New Contact Registration Form

Fields:

- Contact Information
  - First Name
  - Last Name

    - Phone Number
    - Email (optional)
    - Contact Type (dropdown: Senior, Child, Special Needs)
    - Timezone
    - Special Instructions (text area)
- Caregiver Network
    - Primary Caregiver Name
    - Primary Caregiver Phone
    - Primary Caregiver Email
    - Secondary Caregiver Name
    - Secondary Caregiver Phone
    - Secondary Caregiver Email
    - Emergency Contact Name
    - Emergency Contact Phone
- Call Configuration
    - Call Frequency (checkbox: Mon, Tue, Wed, Thu, Fri, Sat, Sun)
    - Call Time (time picker)
    - Call Active (toggle)
    - Retry Maximum (dropdown: 1, 2, 3)
    - Retry Interval (dropdown: 5, 10, 15 minutes)

Call Script Configuration Form

Fields:

- Greeting Script (text area)
- Question 1 (text area)
- Question 1 Type (dropdown: Yes/No, Multiple Choice, Open-ended)
- Question 1 Options (if Multiple Choice)
- Question 1 Alert Triggers (text area)
- [Repeat for Questions 2-5]
- Closing Script (text area)
- Voice Gender (dropdown: Female, Male)
- Voice Style (dropdown: Casual, Professional, Warm)

5.2 Custom Pipelines

Onboarding Pipeline Stages

- Initial Setup
- Script Configuration
- Test Call Completed
- Active Monitoring
- Paused
- Terminated

Alert Pipeline Stages

- Alert Generated
- Caregiver Notified
- Awaiting Response
- Resolved
- Escalated

6. Security & Compliance Implementation

6.1 Data Security

API key storage

```
javascript
// Use environment variables or n8n credentials vault
const credentials = {
  ghlApiKey: process.env.GHL_API_KEY,
  retellApiKey: process.env.RETELL_API_KEY
};
```

Data encryption

- All data transmitted between services uses HTTPS
- Sensitive information stored in GHL custom fields marked as "Protected"
- Call recordings and transcripts handled according to retention policy

6.2 Compliance Features

Consent management

```
javascript
// n8n Function node - Check consent before placing call
const contact = items[0].json;
const hasConsent = contact.customField.recording_consent === 'true';

if (!hasConsent) {
  // Add consent gathering to script
  let greeting = contact.customField.greeting_script || '';
  greeting += " This call may be recorded for quality and safety purposes. Do you consent to
this recording? Please say yes or no.";

  return {
    json: {
      ...contact,
      requiresConsent: true,
```


```
      customField: {
        ...contact.customField,
        greeting_script: greeting
      }
    }
  };
}

return { json: contact };
```

Audit trail implementation

```
javascript
// n8n Function node - Log audit event
function logAuditEvent(eventType, details) {
  const timestamp = new Date().toISOString();
  const event = {
    type: eventType,
    timestamp,
    details,
    user: $workflow.staticData.currentUser || 'system'
  };

  // Store in database or send to audit service
  $workflow.staticData.auditLog = $workflow.staticData.auditLog || [];
  $workflow.staticData.auditLog.push(event);

  // Trim if needed
  if ($workflow.staticData.auditLog.length > 1000) {
    $workflow.staticData.auditLog = $workflow.staticData.auditLog.slice(-1000);
  }

  return event;
}
```

7. Testing Framework

7.1 Test Contacts and Scenarios

```
javascript
// n8n Function node - Create test contacts
const testScenarios = [
  {
    name: "Standard Senior Check-in",
    contactType: "Senior",
    script: {
      greeting: "Hello {FirstName}, this is your daily wellness check-in. How are you feeling
today?",
```


```
      questions: [
        "Did you take your morning medications?",
        "Have you eaten breakfast today?",
        "Do you need anything from the store?"
      ],
      closing: "Thank you for checking in. Have a wonderful day!"
    }
  },
  {
    name: "Child After-School Check",
    contactType: "Child",
    script: {
      greeting: "Hi {FirstName}, it's time for your after-school check-in!",
      questions: [
        "Did you get home safely?",
        "Have you had your snack?",
        "Do you have homework to complete today?"
      ],
      closing: "Great job checking in! Your parents will be home soon."
    }
  },
  {
    name: "Emergency Scenario Test",
    contactType: "Senior",
    script: {
      greeting: "Hello {FirstName}, this is your wellness check. How are you doing?",
      questions: [
        "Are you feeling well today?",
        "Do you need medical assistance?",
        "Can you confirm your emergency contact information?"
      ],
      closing: "Thank you for your responses. Someone will follow up with you shortly."
    },
    simulatedResponses: [
      "Not well",
      "Yes, I'm in pain",
      "My daughter Sarah"
    ]
  }
];

return testScenarios.map(scenario => ({json: scenario}));
```

7.2 Automated Test Workflow

```
javascript
// n8n Function node - Execute test sequence
const testCase = items[0].json;
```


```
// Log test execution
console.log(`Executing test: ${testCase.name}`);

// Create temporary test contact
const testContactId = `test-${Date.now()}`;
const testPhone = $workflow.staticData.testPhoneNumbers[testCase.contactType.toLowerCase()];

// Set up the test case
return {
  json: {
    id: testContactId,
    firstName: "Test",
    lastName: testCase.name.replace(/\s+/g, ''),
    phone: testPhone,
    contactType: testCase.contactType,
    script: testCase.script,
    simulatedResponses: testCase.simulatedResponses || [],
    testStartTime: new Date().toISOString()
  }
};
```

8. Performance Optimization

8.1 Batching Strategy

```
javascript
// n8n Function node - Implement call batching
function createCallBatches(contacts, maxConcurrent = 10) {
  // Sort by priority
  const sortedContacts = [...contacts].sort((a, b) => {
    return (parseInt(b.customField.call_priority) || 1) -
(parseInt(a.customField.call_priority) || 1);
  });

  // Group into batches
  const batches = [];
  for (let i = 0; i < sortedContacts.length; i += maxConcurrent) {
    batches.push(sortedContacts.slice(i, i + maxConcurrent));
  }

  return batches;
}
```

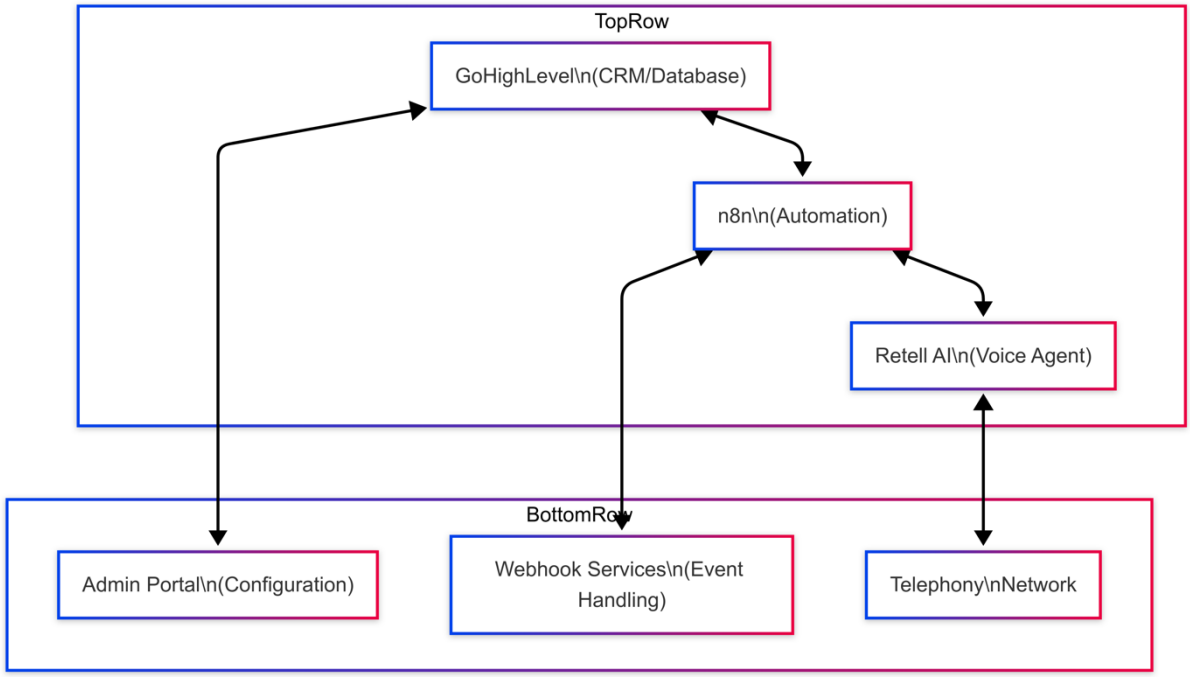
8.2 Resource Management

```
javascript
// n8n Function node - Monitor resource usage
```


```
function checkSystemLoad() {
  const metrics = {
    activeWorkflows: 0,
    pendingCalls: 0,
    currentRate: 0,
    timeOfDay: new Date().getHours()
  };

  // Adjust concurrent limits based on time of day
  // Peak hours: 8-10am and 5-7pm
  let isPeakHour = (metrics.timeOfDay >= 8 && metrics.timeOfDay <= 10) ||
                   (metrics.timeOfDay >= 17 && metrics.timeOfDay <= 19);

  const maxConcurrent = isPeakHour ? 5 : 10;

  return {
    metrics,
    maxConcurrent,
    systemStatus: metrics.activeWorkflows > 20 ? "high-load" : "normal"
  };
}
```

9. Documentation and Handover

9.1 System Architecture Diagram



9.2 Deployment Checklist

Initial Setup

- Create GHL account and configure API access
- Set up n8n instance (cloud or self-hosted)
- Register Retell AI account and obtain API keys
- Configure webhook endpoints

Data Structure Setup

- Create custom fields in GHL
- Set up tags and categories
- Define pipelines and stages

Workflow Implementation

- Deploy Schedule Manager Workflow
- Deploy Call Execution Workflow
- Deploy Response Analysis Workflow
- Deploy Retry & Escalation Workflow
- Deploy Notification Workflow
- Deploy System Monitoring Workflow

Testing

- Create test contacts
- Run test scenarios
- Validate notifications and escalations
- Load testing with simulated calls

Go-Live

- Review security settings
- Set up monitoring alerts
- Client training session
- Documentation handover

10. Maintenance and Support Plan

10.1 Monitoring Strategy

```javascript
// n8n Function node - Generate system health report
function generateHealthReport() {
  const metrics = {
```


```
    totalActiveContacts: 0,
    callsLast24Hours: 0,
    successRate: 0,
    alertsGenerated: 0,
    avgCallDuration: 0,
    systemErrors: 0
  };

  // Generate report with summary and detailed sections
  const report = {
    summary: `System Status: ${metrics.systemErrors > 0 ? 'Issues Detected' :
'Operational'}`,
    details: metrics,
    recommendations: []
  };

  if (metrics.successRate < 90) {
    report.recommendations.push("Call success rate below threshold. Check network
connectivity.");
  }

  return report;
}
```

10.2 Backup and Recovery

```
javascript
// n8n Function node - Backup configuration
function backupConfiguration() {
  const timestamp = new Date().toISOString().replace(/[:.]/g, '-');
  const backupId = `config-backup-${timestamp}`;

  const configData = {
    ghlCustomFields: [],
    workflows: [],
    retellScripts: [],
    systemSettings: {}
  };

  // Store backup metadata
  const backupMetadata = {
    id: backupId,
    timestamp: new Date().toISOString(),
    size: JSON.stringify(configData).length,
    type: 'full'
  };

  return {
    backupId,
```


```
    metadata: backupMetadata,
    configData
  };
}
```

11. SLA and Performance Metrics

Metric	Target	Critical Threshold
Call Success Rate	>98%	<90%
System Availability	0.999	<99%
Alert Delivery Time	<1 minute	>5 minutes
False Positive Alerts	<2%	>5%
Call Audio Quality	MOS >4.0	MOS <3.5

Performance Tracking Workflow

```javascript
// n8n Function node - Calculate SLA metrics
function calculateSLAMetrics(timeframe = '24h') {
  // Get data from monitoring database

  const metrics = {
    callSuccess: {
      target: 98,
      current: 99.1,
      status: 'meeting'
    },
    availability: {
      target: 99.9,
      current: 99.95,
      status: 'meeting'
    },
    alertDelivery: {
      target: 60, // seconds
      current: 32,
      status: 'meeting'
    },
    falsePositives: {
      target: 2,
      current: 1.2,
      status: 'meeting'
    },
    audioQuality: {
      target: 4.0,
```


```
      current: 4.2,
      status: 'meeting'
    }
  };

  // Calculate overall SLA status
  const slaStatus = Object.values(metrics).every(m => m.status === 'meeting')
    ? 'All SLAs Met'
    : 'SLA Violations Detected';

  return {
    timeframe,
    metrics,
    status: slaStatus,
    timestamp: new Date().toISOString()
  };
}
```