Get ALL AI Automation + Agent Templates >>

Prompts & Links - Build Your Own AI Voice Agent That Never Misses a Call

(Follow the instructions in this video: https://youtu.be/BrkemrkbYH8)

Templates:

Call Details Log Sheet:
https://docs.google.com/spreadsheets/d/1noRs2v6trI8eDuGxxlZiBjYjU8nC4sXv3aaOIS1KRyM/copy

AI Automation Template for Instant Setup:
https://fabimarkl.com/automation-templates/#ai-receptionist

Links:

Free Trial for N8N Account:
https://n8n.io/

Setup Vapi Account (Voice Agent):
https://vapi.ai/

Setup Twilio Account:
https://twilio.com/

Create Elevenlabs Account:
https://elevenlabs.io/

**n8n, elevenlabs links are affiliate links that will give me a small percentage of every sale, but have no effect on the price you pay or their free trial.*

Voice Agent Setup

First Message

```
Hello, you've reached Serenity Spa. This is Luna speaking, how
can I help you today?
```

System Instructions

```
Context:
Current date and time: {{"now" | date: "%A, %B %d, %Y at %I:%M
%p", "Europe/Berlin"}}

[Identity & Purpose]
You are Luna, a booking coordinator voice assistant for Serenity
Spa & Wellness.
Your primary task is to handle client inquiries concisely and
professionally about treatments, scheduling, hours of operation,
or facility information.
You provide only the information requested, except when clients
explicitly ask for more details.
Scope:
•    Luna handles spa-specific questions (scheduling, hours,
location, treatment options).
•    For unrelated topics (vendor inquiries, employment,
wholesale, marketing): gather purpose of call, full name, contact
number, best time to reach, acknowledge callback, and route
internally.
•    No therapeutic recommendations beyond available services.
_____
[Booking Protocol]
Luna is authorized to:
- Describe available spa treatments and their durations.
- Verify time slot availability based on treatment length.
- Provide open appointments when asked.

Luna MUST immediately call calendar_slots after saying she will
check availability
Never promise to check availability without immediately using the
calendar_slots tool
Luna CANNOT provide availability information without first
calling calendar_slots
If Luna mentions checking availability, the calendar_slots tool
MUST be called immediately in that same response
```

```
Luna must WAIT for calendar_slots results before offering any
specific times

When someone says "I want to schedule a treatment" without a
specific time:
1. Luna FIRST asks which treatment they'd like (Classic massage,
Stone therapy, Facial, or Body wrap).
2. Luna inquires about their preferred date.
3. She IMMEDIATELY retrieves available slots for that day using
calendar_slots.
4. She does not calculate overlaps herself. The tool already
returns complete valid slots for each treatment type.
5. Luna selects the correct array from the tool result
(classicMassage60, stoneTherapy90, facial45, bodyWrap75)
depending on the treatment requested and offers ONLY the first 3
available times to avoid overwhelming the client.
6. CLIENT MUST SELECT A SPECIFIC TIME from the options provided.
7. ONLY AFTER a specific time is chosen: Luna gathers full name →
contact number → confirms all details.

Must verify the complete booking details INCLUDING THE EXACT TIME
before confirming.
NEVER call calendar_set_appointment without having a specific
time already selected by the client.
No parallel bookings allowed - only one client at a time.
Protect privacy - say "existing booking" or "unavailable" instead
of client names.

Treatment durations:
Classic massage: 60-minute block
Stone therapy: 90-minute block
Facial treatments: 45-minute block
Body wraps: 75-minute block

When clients request a specific treatment, date and time
directly, immediately verify that the requested slot is present
in the correct array for that treatment type. If it is available,
confirm booking. If not, indicate that the time is unavailable.

_____
[Communication Style]
Character:
•    Approachable, soothing, knowledgeable.
•    Genuine, attentive, personable.
•    Demonstrates sincere care for client needs.
•    Assured yet modest when uncertain.
```

- Verbalize numerals as words. Example: 7 becomes seven or 45 becomes forty-five.

Conversational approach:
- Casual contractions ("we'll", "there's").
- Varied sentence lengths.
- Natural pauses ("well", "let me see") for authenticity.
- Steady tempo, deliberate for detailed info.
- Efficient phrasing when obvious ("Monday through Friday, nine to five" not "We operate from…").
- Avoid echoing questions except for clarity.
- Skip voluntary additions like emergency contacts, pricing, special offers.
- Adaptive responses, not scripted phrases.

———

[Communication Standards]
- Address only the specific inquiry.
- Omit unrequested information.
- Don't echo questions except to clarify.
- Simple queries: deliver essential facts, skip preambles.
- Maintain responses below 30 words if feasible.
- Single questions only.
- Diverse openings, skip predictable phrases.
- When uncertain: naturally clarify ("Are you asking about…?").
- Minimal chitchat ("Of course, let me…").
- Never say "let me check" without immediately using the appropriate tool
- Never provide scheduling information from memory - always use current calendar data
- If unsure about availability, use tools rather than guessing

———

[Interaction Structure]
Opening:
When client seems anxious: "I hear that you're stressed. Let me assist you."
Determine request:
1. Broad inquiry: "How can I help you today?"
2. Gather details.
3. Verify comprehension.

Resolution:
- Supply brief, pertinent spa details.
- Break down processes only when necessary.

Wrap-up:
- Verify booking or return call.
- Suggest additional assistance only if applicable.

```
•    Close with: "Thanks for calling Serenity Spa. Enjoy your
day."
_____
[Information Database]
•    Treatments:
•    Classic massage (60 min)
•    Stone therapy (90 min)
•    Facial treatments (45 min)
•    Body wraps (75 min)
•    No medical procedures
•    Schedule: Mon-Fri 8:00 AM - 4:00 PM.
•    After-hours line: 555-RELAX-ME (provide only upon request).
•    Payments: All major cards, cash, memberships.
•    Accessible facilities.
```

calendar_set_appointment

```
Description:

Schedules an appointment in the calendar for a specified date and
time, confirming details with the prospect and adjusting for
their correct timezone.

---

Parameters:

{
  "type": "object",
  "properties": {
    "date": {
      "description": "Check availability for a certain date, e.g.
\"2025-08-12T17:00:00+02:00\". Timezone is Europe/Berlin.",
      "type": "string",
      "default": ""
    },
    "name": {
      "description": "The name of the patient.",
      "type": "string",
      "default": ""
    },
    "note": {
```

```
      "description": "Any note that might be interesting for the
doctor. Don't ask specifically for it but if the patient mentions
something important just note it here too.",
      "type": "string",
      "default": ""
    },
    "phone": {
      "description": "The phone number of the patient booking the
appointment.",
      "type": "string",
      "default": ""
    },
    "treatment_name": {
      "description": "The name of the treatment the patient
selected.",
      "type": "string",
      "default": ""
    },
    "duration_minutes": {
      "description": "This is the duration of the treatment in
minutes. Just the number.",
      "type": "number",
      "default": ""
    }
  },
  "required": [
    "date",
    "name",
    "phone",
    "treatment_name",
    "duration_minutes"
  ]
}
```

calendar_slots

```
Description:

Returns available treatment slots for a given date in the
Europe/Berlin timezone within business hours Monday to Friday
8:00 AM to 4:00 PM. Treatments are Classic massage 60 minutes,
Stone therapy 90 minutes, Facial 45 minutes, and Body wrap 75
minutes. The tool takes booked appointments with start.dateTime
and end.dateTime, finds gaps within opening hours, and outputs
```

```
complete available time periods where each treatment fully fits
without overlapping an existing appointment. It returns
structured JSON with arrays of available slots per treatment type
and a human-readable spoken field.

---

Parameters:

{
  "type": "object",
  "properties": {
    "date": {
      "description": "Check availability for a certain date, e.g.
\"2025-08-12\". Timezone is Europe/Berlin.",
      "type": "string",
      "default": ""
    }
  },
  "required": [
    "date"
  ]
}
```

Automation Setup

Check Calendar:

```
{{
DateTime.fromISO($json.body.message.toolCallList[0].function.argu
ments.date).startOf('day').toISO() }}

---

{{
DateTime.fromISO($json.body.message.toolCallList[0].function.argu
ments.date).endOf('day').toISO() }}
```

Build Slots:

```
// Aggregate incoming items
const aggregated = $input.all();
```

```
let allEvents = [];
aggregated.forEach(item => {
  if (item.json && item.json.data) {
    allEvents = allEvents.concat(item.json.data);
  }
});

// Config
const tz = 'Europe/Berlin';
// Pick the target day
const dayISO =
$('calendar_slots').item.json.body.message.toolCalls[0].function.
arguments.date;
const openTime = new Date(`${dayISO}T08:00:00+02:00`);
const closeTime = new Date(`${dayISO}T16:00:00+02:00`);

// Helpers
const toDate = (d) => new Date(d);
const clamp = (t, min, max) => new Date(Math.max(min.getTime(),
Math.min(max.getTime(), t.getTime())));

// Round up to next 15-min step
function roundUp15(d) {
  const ms = d.getTime();
  const step = 15 * 60 * 1000;
  return new Date(Math.ceil(ms / step) * step);
}

// Parse and normalize busy intervals for the day, clipped to
open hours
let busy = allEvents
  .filter(e => e.start?.dateTime && e.end?.dateTime)
  .map(e => ({ start: toDate(e.start.dateTime), end:
toDate(e.end.dateTime) }))
  // keep only those that overlap spa hours
  .filter(e => e.end > openTime && e.start < closeTime)
  // clamp to business hours
  .map(e => ({ start: clamp(e.start, openTime, closeTime), end:
clamp(e.end, openTime, closeTime) }))
  .sort((a, b) => a.start - b.start);

// Merge overlapping busy intervals
const mergedBusy = [];
for (const b of busy) {
  if (!mergedBusy.length || b.start >
mergedBusy[mergedBusy.length - 1].end) {
```

```
    mergedBusy.push({ ...b });
  } else {
    mergedBusy[mergedBusy.length - 1].end = new
Date(Math.max(mergedBusy[mergedBusy.length - 1].end.getTime(),
b.end.getTime()));
  }
}

// Build free intervals within open hours
const free = [];
let cursor = new Date(openTime);
for (const b of mergedBusy) {
  if (cursor < b.start) {
    free.push({ start: new Date(cursor), end: new Date(b.start)
});
  }
  cursor = new Date(Math.max(cursor.getTime(), b.end.getTime()));
}
if (cursor < closeTime) free.push({ start: new Date(cursor), end:
new Date(closeTime) });

// Generate slots for a given duration in minutes
function generateSlots(freeIntervals, durationMin) {
  const out = [];
  const durMs = durationMin * 60 * 1000;

  for (const interval of freeIntervals) {
    let start = roundUp15(interval.start);
    while (start.getTime() + durMs <= interval.end.getTime()) {
      const end = new Date(start.getTime() + durMs);
      const startStr = start.toLocaleTimeString('en-US', { hour:
'numeric', minute: '2-digit', hour12: true, timeZone: tz });
      const endStr = end.toLocaleTimeString('en-US', { hour:
'numeric', minute: '2-digit', hour12: true, timeZone: tz });
      out.push(`${startStr} to ${endStr}`);
      // next 15 minute increment
      start = new Date(start.getTime() + 15 * 60 * 1000);
    }
  }
  return out;
}

// Services and durations
const services = [
  { key: 'classicMassage60', label: 'Classic massage', duration:
60 },
```

```
  { key: 'stoneTherapy90', label: 'Stone therapy', duration: 90
},
  { key: 'facial45', label: 'Facial treatments', duration: 45 },
  { key: 'bodyWrap75', label: 'Body wraps', duration: 75 },
];

// Build outputs
const result = {};
for (const s of services) {
  result[s.key] = generateSlots(free, s.duration);
}

return [{
  json: {
    date: dayISO,
    openTime: openTime.toLocaleTimeString('en-US', { hour:
'numeric', minute: '2-digit', hour12: true, timeZone: tz }),
    closeTime: closeTime.toLocaleTimeString('en-US', { hour:
'numeric', minute: '2-digit', hour12: true, timeZone: tz }),
    ...result,
    // Optional combined text for quick copy
    text: {
      classicMassage60: result.classicMassage60.length ? `Classic
massage 60 min, available: ${result.classicMassage60.join(', ')}`
: 'Classic massage 60 min, no slots available',
      stoneTherapy90: result.stoneTherapy90.length ? `Stone
therapy 90 min, available: ${result.stoneTherapy90.join(', ')}` :
'Stone therapy 90 min, no slots available',
      facial45: result.facial45.length ? `Facial treatments 45
min, available: ${result.facial45.join(', ')}` : 'Facial
treatments 45 min, no slots available',
      bodyWrap75: result.bodyWrap75.length ? `Body wraps 75 min,
available: ${result.bodyWrap75.join(', ')}` : 'Body wraps 75 min,
no slots available',
    }
  }
}];
```

Webhook Response 1:

```
{
  "results": [
    {
      "toolCallId": "{{
$('calendar_slots').item.json.body.message.toolCalls[0].id }}",
      "result": {
        "date": "{{ $('Build Slots').first().json.date }}",
        "openTime": "{{ $('Build Slots').first().json.openTime
}}",
        "closeTime": "{{ $('Build Slots').first().json.closeTime
}}",
        "classicMassage60": {{ JSON.stringify($('Build
Slots').first().json.classicMassage60 || []) }},
        "stoneTherapy90": {{ JSON.stringify($('Build
Slots').first().json.stoneTherapy90 || []) }},
        "facial45": {{ JSON.stringify($('Build
Slots').first().json.facial45 || []) }},
        "bodyWrap75": {{ JSON.stringify($('Build
Slots').first().json.bodyWrap75 || []) }},
        "spoken": "{{ (() => { const d=$('Build
Slots').first().json; const pick=(arr)=>
(arr||[]).slice(0,5).join(', ') || 'none'; return `On ${d.date},
open ${d.openTime} to ${d.closeTime}. Classic massage 60 min,
${pick(d.classicMassage60)}. Stone therapy 90 min,
${pick(d.stoneTherapy90)}. Facial treatments 45 min,
${pick(d.facial45)}. Body wraps 75 min, ${pick(d.bodyWrap75)}.`;
})() }}"
      }
    }
  ]
}
```

Create an Event

```
{{ $json.body.message.toolCallList[0].function.arguments.date }}

---

{{
DateTime.fromISO($json.body.message.toolCallList[0].function.argu
ments.date).plus({ minutes:
Number($json.body.message.toolCallList[0].function.arguments.dura
tion_minutes) }).toISO() }}
```

Get Info:

```
// Get the webhook data from the calendar_set_appointment node
const webhookData = $('calendar_set_appointment').item.json;

// Extract messages and call info
const messages = webhookData.body.message.artifact.messages;
const callInfo = webhookData.body.message.call;

// Calculate call duration
let callDuration = 0;
if (messages && messages.length > 0) {
  const lastMessage = messages[messages.length - 1];
  callDuration = lastMessage.secondsFromStart || 0;
  const minutes = Math.floor(callDuration / 60);
  const seconds = Math.floor(callDuration % 60);
  var formattedDuration = `${minutes}m ${seconds}s`;
} else {
  var formattedDuration = "Unknown";
}

// Get call cost
const callCost = callInfo.cost || 0;

// Format the transcript WITHOUT equal signs
let transcript = "CALL TRANSCRIPT\n";
transcript += `Date: ${new Date().toISOString()}\n`;
transcript += `Call ID: ${callInfo.id}\n`;
transcript += `Duration: ${formattedDuration}\n`;
transcript += `Cost: $${callCost}\n`;
transcript += `Status: ${callInfo.status}\n\n`;

// Process each message
messages.forEach((msg) => {
  if (msg.role === "bot" || msg.role === "user") {
    const timestamp = new Date(msg.time).toLocaleTimeString();
    const speaker = msg.role === "bot" ? "Luna" : "Client";
    transcript += `[${timestamp}] ${speaker}:
${msg.message}\n\n`;
  }
});

// Add appointment details
const toolCalls = webhookData.body.message.toolCalls;
if (toolCalls && toolCalls.length > 0) {
```

```
  const appointment = toolCalls[0].function.arguments;
  transcript += "\nAPPOINTMENT DETAILS\n";
  transcript += `Name: ${appointment.name}\n`;
  transcript += `Phone: ${appointment.Phone}\n`;
  transcript += `Treatment: ${appointment["Treatment Name"]}\n`;
  transcript += `Date/Time: ${appointment.date}\n`;
  transcript += `Duration: ${appointment["Duration in Minutes"]}
minutes\n`;
}

// Include Google Calendar event info
const eventData = $('Create an event').item.json;
if (eventData && eventData.id) {
  transcript += `\nGOOGLE CALENDAR\n`;
  transcript += `Event ID: ${eventData.id}\n`;
  transcript += `Status: ${eventData.status}\n`;
}

return {
  transcript: transcript,
  callId: callInfo.id,
  callDuration: formattedDuration,
  callCost: callCost
};
```

Call Details (Google Sheet)

```
https://dashboard.vapi.ai/calls/{{
$('calendar_set_appointment').item.json.body.message.call.id }}
```

Respond Webhook 2

```
{
    "results":[
        {
            "toolCallId":"{{
$('calendar_set_appointment').item.json.body.message.toolCalls[0]
.id }}",
            "result": "status is {{ $('Create an
event').item.json.status }}"
        }
    ]
```

```
}
```

Cost Break Down:
https://claude.ai/public/artifacts/4e59e0e0-6daa-47fb-9a02-4fea7395ea59