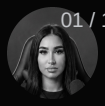


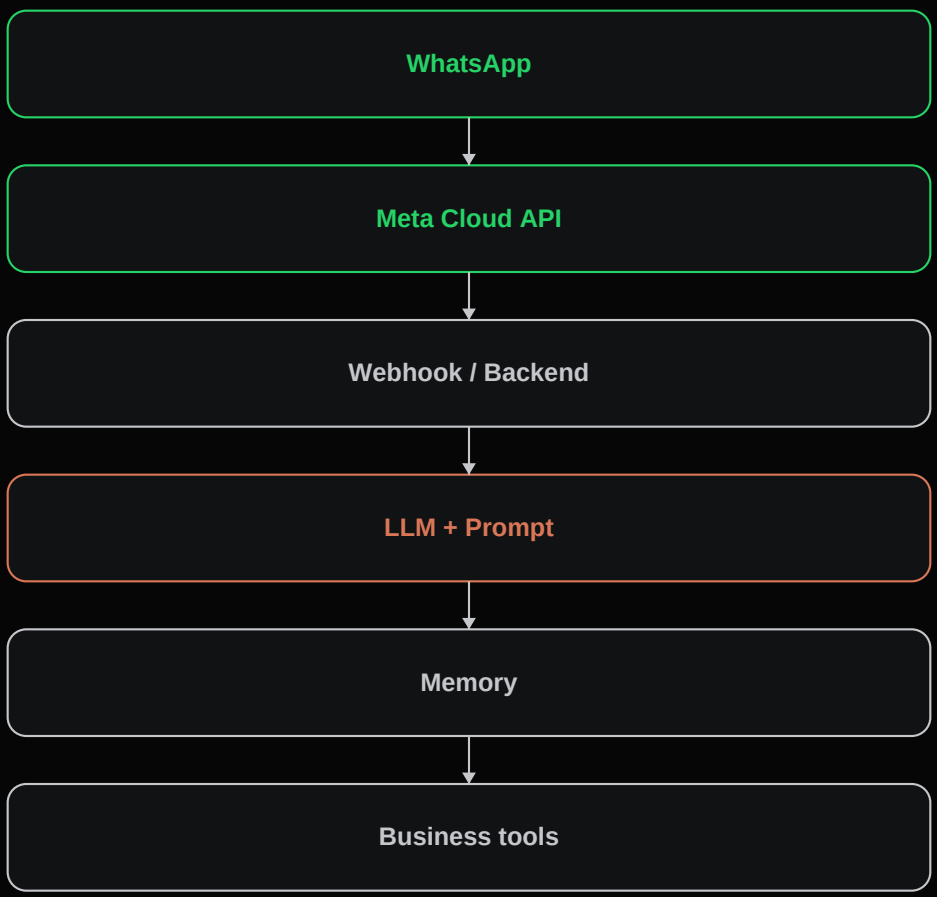
Build a WhatsApp AI Agent in 30 Minutes

Using Claude Code + Meta WhatsApp API

A practical implementation playbook for a WhatsApp assistant that can answer customers, use business knowledge, remember context, take booking actions and run 24/7.

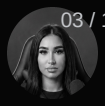
What you are actually building

Think of the system as a message pipeline. WhatsApp is only the channel; your backend decides what context to load, which model to call and which business action to execute.



RULE OF THUMB

Keep messaging, reasoning, memory and actions modular. That makes the system easier to debug and change.

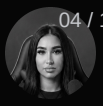


Before you start

Meta Developer account	WhatsApp Cloud API test number
Claude Code + VS Code	Project folder + Git
OpenRouter account	API key stored as an environment variable
Booking tool	Cal.com, CRM or your own scheduling API
Hosting	A VPS or managed host with HTTPS

SECURITY

Do not paste production secrets into screenshots or public repositories. Use environment variables and rotate any key that is exposed.



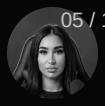
Define the agent's job

Start with the business outcome. The agent should know what it is responsible for and, just as importantly, what it must not do.

DO	Answer FAQs • recommend services • collect details • check availability • book
DON'T	Invent prices • promise unavailable slots • bypass policy • hide uncertainty

OUTCOME FIRST

A good initial instruction describes the customer experience and required actions. Let Claude Code propose the architecture before you ask it to implement.

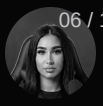


Use a build prompt that is specific

```
I am building a WhatsApp assistant for [BUSINESS].

It must:
1. Answer FAQs using approved business knowledge.
2. Understand the customer's goal before recommending.
3. Collect only the details needed for the next action.
4. Check availability and create bookings using our booking tool.
5. Remember recent conversation context.
6. Escalate to a human if confidence is low or the request is outside policy.
7. Log tool errors and avoid pretending an action succeeded.

Before writing code:
- propose the architecture
- list required environment variables
- define the webhook flow
- define the memory schema
- define tool interfaces
- list failure cases and tests
```

WHY THIS WORKS

It forces architecture, boundaries and failure handling to be designed before implementation.



Connect WhatsApp

Create a Meta Developer App, add WhatsApp, and begin with a test number. Your backend needs the identifiers and secrets required to receive and reply to messages.

PHONE_NUMBER_ID

ACCESS_TOKEN

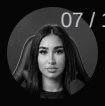
APP_SECRET

VERIFY_TOKEN

WEBHOOK_CALLBACK_URL

DEVELOPMENT

Use the test number until inbound messages, verification and outbound replies are stable. Then move to the real business number and production callback URL.



Understand the webhook flow

The webhook is the bridge between Meta and your application. Treat it as a small event-processing pipeline, not as the AI agent itself.

1. Customer message

2. Meta sends webhook

3. Verify + parse event

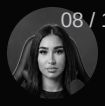
4. Load context

5. Call model / tools

6. Send reply

PRODUCTION RULE

Acknowledge events quickly, make processing idempotent where possible, and log failures so duplicate or retried events do not create duplicate actions.



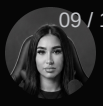
Connect the AI model

Use an AI gateway or provider adapter so model choice is not hard-wired into the WhatsApp integration.

OPENROUTER_API_KEY

MODEL LAYER

Keep model selection, retries and parsing in one module. This makes later model changes much safer.



Design the system prompt

ROLE

Who the agent is

GOAL

What success looks like

KNOWLEDGE

Approved facts only

RULES

Tone, boundaries, escalation

TOOLS

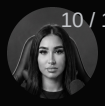
What actions it may call

OUTPUT

How responses should be structured

KEY IDEA

Separate instructions from business facts. A policy or price change should not require changing application code.



Build the knowledge layer

Your agent needs one approved source of truth. Start simple, then move to retrieval only when the knowledge base is too large for a compact system context.

- Business details
- Services
- Pricing
- Hours
- Locations
- Policies
- FAQs
- Brand voice
- Booking rules

GROUNDING

If an answer is not supported by approved knowledge, the agent should say it is unsure or hand off - not invent.

Give the agent memory

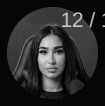
Conversation memory reduces repetition, but more memory is not always better. Save only what improves the next turn.

USEFUL STRUCTURED STATE

- intent
- selected_service
- preferred_time
- booking_status
- handoff_needed

PRIVACY

Define what you store, why you store it and how long you keep it. Avoid retaining unnecessary sensitive data.



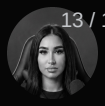
Connect real actions

An assistant becomes useful when it can safely take action. Start with one narrow tool such as booking.

- Check availability
- Show slots
- Collect details
- Create booking
- Cancel / reschedule
- Notify team

TOOL CONTRACT

Each tool should validate inputs and return a clear success/error result. The model must never claim an action succeeded if the tool returned an error.



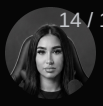
Add human handoff

A production agent needs an escape hatch. Define explicit conditions that stop automation and route the conversation to a person.

- Customer asks for a human
- Request is outside policy
- Repeated tool failure
- Low confidence / missing facts
- Complaint or sensitive issue
- High-value exception

HANDOFF CONTEXT

When you hand off, pass the human a short summary, customer details already collected and the last relevant messages.



Protect secrets and customer data

01 Keep API keys in environment variables.

02 Never expose App Secret or access tokens in client-side code.

03 Log identifiers and errors, not full sensitive message bodies by default.

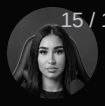
04 Use HTTPS in production.

05 Limit stored memory and define retention.

06 Rotate compromised credentials immediately.

SCOPE

This playbook is an implementation guide, not a substitute for legal/privacy review for regulated or sensitive use cases.



Test like a real customer

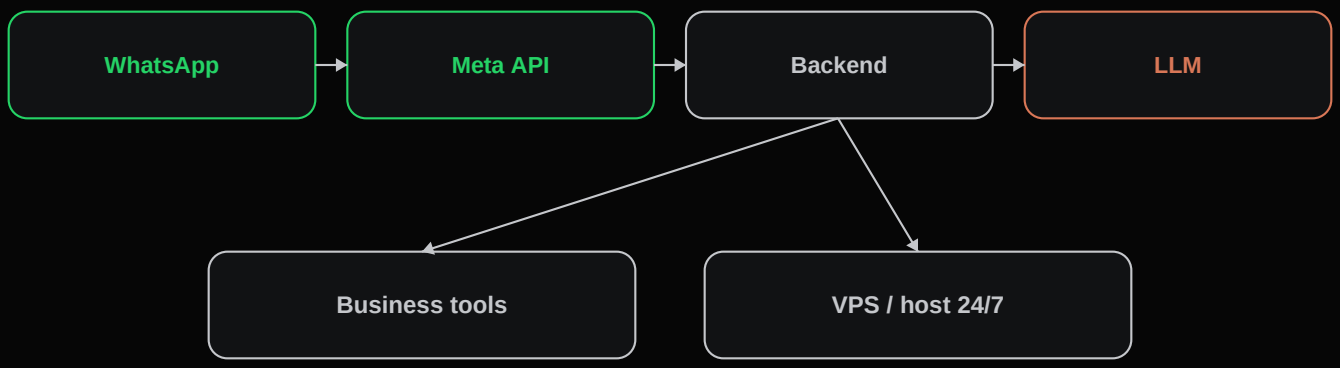
NORMAL	price • hours • book
AMBIGUOUS	'something tomorrow' • unclear service
EDGE	timezone • no slots • duplicate
FAILURE	API timeout • invalid token • tool error
HANDOFF	complaint • human request

ACCEPTANCE CRITERIA

The agent should be accurate, concise, non-repetitive, honest about uncertainty, and able to recover from tool errors without pretending success.

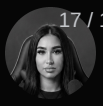
Deploy it 24/7

Your laptop should not be the server. Deploy the backend to a production host with a stable HTTPS URL.



AFTER DEPLOY

Update the production callback URL, verify the webhook again, then run an end-to-end test using the real business number.



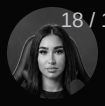
Add logs and observability

When the bot fails, you need to know whether the problem was Meta, your code, the model or a business tool.

Webhook events	received / failed
Model calls	latency / errors
Tool calls	success / failure
Bookings	created / duplicate
Handoffs	reason / volume

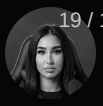
DEBUGGING

Attach a correlation ID to each conversation/event so you can trace one message across webhook, model and tool calls.



Copy-paste system prompt

```
ROLE
You are the WhatsApp assistant for [BUSINESS].

GOAL
Resolve customer questions and complete allowed actions quickly and accurately.

KNOWLEDGE
Use only the approved business information provided to you.

RULES
- Understand the goal before recommending.
- Ask only the minimum questions needed.
- Never invent price, availability or policy.
- If unsure, say so and offer human help.
- Never claim a tool action succeeded unless the tool confirms success.
- Keep replies concise and on-brand.

TOOLS
Use booking / CRM tools only when their required inputs are present.

MEMORY
Use recent conversation context and structured state to avoid repeating questions.

HANDOFF
Escalate when the customer requests a person, policy is unclear, confidence is low,
 or tools repeatedly fail.
```



Production checklist

- [ ] Meta webhook verifies successfully
- [ ] Real inbound and outbound message test passes
- [ ] Secrets are environment variables
- [ ] Approved knowledge is current
- [ ] Memory schema and retention are defined
- [ ] Tool failures do not produce false confirmations
- [ ] Human handoff works
- [ ] Duplicate booking scenario is tested
- [ ] HTTPS production URL is live
- [ ] Logs and alerts are enabled
- [ ] End-to-end test passes on the real number

SHYRAFORGE

Built with intent. Designed to last.