



DASHBOARD ORDER

- 1 Inbox (core engine)
- 2 Contacts
- 3 Sales Pipeline
- 4 Automation
- 5 Broadcast
- 6 Templates
- 7 Analytics
- 8 Billing

Because:

Inbox powers everything else.

#Schema

FINAL REFACTORED SAAS SCHEMA

🔧 ENUM TYPES (data safety first)

```
CREATE TYPE pipeline_stage AS ENUM  
('New','Engaged','Interested','Paid','Lost');
```

```
CREATE TYPE message_direction AS ENUM  
('in','out');
```

```
CREATE TYPE message_status AS ENUM  
('sent','delivered','failed','read');
```

```
CREATE TYPE subscription_status AS ENUM  
('active','expired','cancelled');
```

🏢 BUSINESSES (CRM customers)

```
CREATE TABLE businesses (  
id SERIAL PRIMARY KEY,  
business_name TEXT NOT NULL,  
email TEXT UNIQUE NOT NULL,  
created_at TIMESTAMPTZ DEFAULT NOW(),  
status TEXT DEFAULT 'active'  
);
```

👤 USERS (team inside business)

```
CREATE TABLE users (  
id SERIAL PRIMARY KEY,  
business_id INT NOT NULL REFERENCES businesses(id) ON DELETE CASCADE,  
name TEXT NOT NULL,  
email TEXT NOT NULL,  
password_hash TEXT NOT NULL,
```

```
role TEXT DEFAULT 'owner',
created_at TIMESTAMPTZ DEFAULT NOW(),
UNIQUE(business_id, email)
);
```

WHATSAPP ACCOUNTS

```
CREATE TABLE whatsapp_accounts (
id SERIAL PRIMARY KEY,
business_id INT NOT NULL REFERENCES businesses(id) ON DELETE CASCADE,
phone_number TEXT UNIQUE NOT NULL,
api_token TEXT,
status TEXT DEFAULT 'active',
connected_at TIMESTAMPTZ DEFAULT NOW()
);
```

CONTACTS

```
CREATE TABLE contacts (
id SERIAL PRIMARY KEY,
business_id INT NOT NULL REFERENCES businesses(id) ON DELETE CASCADE,
phone TEXT NOT NULL,
name TEXT,
stage pipeline_stage DEFAULT 'New',
created_at TIMESTAMPTZ DEFAULT NOW(),
last_active TIMESTAMPTZ,
UNIQUE(business_id, phone)
);
```

MESSAGES (Inbox core)

```
CREATE TABLE messages (
id SERIAL PRIMARY KEY,
business_id INT NOT NULL REFERENCES businesses(id) ON DELETE CASCADE,
whatsapp_account_id INT REFERENCES whatsapp_accounts(id),
contact_id INT NOT NULL REFERENCES contacts(id) ON DELETE CASCADE,
direction message_direction NOT NULL,
content TEXT NOT NULL,
status message_status DEFAULT 'sent',
sent_at TIMESTAMPTZ DEFAULT NOW()
);
```

PIPELINE HISTORY

```
CREATE TABLE pipeline_history (
id SERIAL PRIMARY KEY,
business_id INT NOT NULL REFERENCES businesses(id) ON DELETE CASCADE,
contact_id INT NOT NULL REFERENCES contacts(id) ON DELETE CASCADE,
from_stage pipeline_stage,
to_stage pipeline_stage,
changed_at TIMESTAMPTZ DEFAULT NOW()
);
```

TAGS

```
CREATE TABLE contact_tags (
```

```
id SERIAL PRIMARY KEY,  
business_id INT NOT NULL REFERENCES businesses(id) ON DELETE CASCADE,  
contact_id INT NOT NULL REFERENCES contacts(id) ON DELETE CASCADE,  
tag TEXT NOT NULL  
);
```

TEMPLATE MANAGER

```
CREATE TABLE message_templates (  
id SERIAL PRIMARY KEY,  
business_id INT NOT NULL REFERENCES businesses(id) ON DELETE CASCADE,  
name TEXT NOT NULL,  
content TEXT NOT NULL,  
status TEXT DEFAULT 'pending'  
);
```

BROADCAST CAMPAIGNS

```
CREATE TABLE campaigns (  
id SERIAL PRIMARY KEY,  
business_id INT NOT NULL REFERENCES businesses(id) ON DELETE CASCADE,  
template_id INT NOT NULL REFERENCES message_templates(id),  
name TEXT NOT NULL,  
scheduled_at TIMESTAMPTZ,  
status TEXT  
);
```

```
CREATE TABLE campaign_logs (  
id SERIAL PRIMARY KEY,  
campaign_id INT NOT NULL REFERENCES campaigns(id) ON DELETE CASCADE,  
contact_id INT NOT NULL REFERENCES contacts(id) ON DELETE CASCADE,  
delivered BOOLEAN DEFAULT false,  
replied BOOLEAN DEFAULT false,  
sent_at TIMESTAMPTZ DEFAULT NOW()  
);
```

AUTOMATION ENGINE (scalable rules)

```
CREATE TABLE automation_rules (  
id SERIAL PRIMARY KEY,  
business_id INT NOT NULL REFERENCES businesses(id) ON DELETE CASCADE,  
trigger TEXT NOT NULL,  
condition JSONB NOT NULL,  
action JSONB NOT NULL,  
delay_minutes INT DEFAULT 0  
);
```

BILLING SYSTEM

```
CREATE TABLE plans (  
id SERIAL PRIMARY KEY,  
name TEXT NOT NULL,  
conversation_limit INT NOT NULL,  
price NUMERIC NOT NULL  
);
```

```
CREATE TABLE subscriptions (
    id SERIAL PRIMARY KEY,
    business_id INT NOT NULL REFERENCES businesses(id) ON DELETE CASCADE,
    plan_id INT NOT NULL REFERENCES plans(id),
    renews_at TIMESTAMPTZ,
    status subscription_status DEFAULT 'active'
);
```

```
CREATE TABLE usage_logs (
    id SERIAL PRIMARY KEY,
    business_id INT NOT NULL REFERENCES businesses(id) ON DELETE CASCADE,
    conversation_id TEXT NOT NULL,
    cost NUMERIC NOT NULL,
    timestamp TIMESTAMPTZ DEFAULT NOW()
);
```

⚡ PERFORMANCE INDEXES (VERY IMPORTANT)

```
CREATE INDEX idx_contacts_business ON contacts(business_id);
CREATE INDEX idx_messages_business ON messages(business_id);
CREATE INDEX idx_messages_contact ON messages(contact_id);
CREATE INDEX idx_pipeline_business ON pipeline_history(business_id);
CREATE INDEX idx_campaigns_business ON campaigns(business_id);
CREATE INDEX idx_usage_business ON usage_logs(business_id);
CREATE INDEX idx_messages_time ON messages(sent_at DESC);
```

1. businesses
2. users
3. whatsapp_accounts
4. contacts
5. messages
6. pipeline_history
7. contact_tags
8. message_templates
9. campaigns
10. campaign_logs
11. automation_rules
12. plans
13. subscriptions
14. usage_logs

👉 Plus 4 ENUM types (not tables, but schema objects):

- pipeline_stage
- message_direction
- message_status
- subscription_status

✓ Final count:

14 tables + 4 enums

INBOX DASHBOARD — FULL BACKEND LOGIC (END-TO-END)

The Inbox has 4 major backend flows:

- 1 Receiving WhatsApp messages (webhook)
- 2 Storing & linking data properly
- 3 Showing inbox conversations
- 4 Sending messages from dashboard

Each triggers automation + billing.

Let's go one by one.

FLOW 1 — INCOMING WHATSAPP MESSAGE (MOST IMPORTANT)

This runs every time a customer messages a business.

Step 1 — WhatsApp webhook hits your server

Payload includes:

```
scss Copy code  
  
to_number  (your business WhatsApp)  
from_number (customer phone)  
message_text  
status (delivered/read etc sometimes)
```

Step 2 — Identify which BUSINESS owns this number

Query:

```
sql Copy code  
  
SELECT * FROM whatsapp_accounts  
WHERE phone_number = $to_number AND status='active';
```

Result gives:

- 👉 whatsapp_accounts.id
- 👉 business_id

Now you know which CRM customer owns this chat.

Step 3 — Upsert CONTACT (no duplicates)

sql

 Copy code

```
SELECT * FROM contacts  
WHERE business_id=$business_id AND phone=$from_number;
```

If exists:

sql

 Copy code

```
UPDATE contacts SET last_active=NOW() WHERE id=$contact_id;
```

If not exists:

sql

 Copy code

```
INSERT INTO contacts(business_id,phone,last_active)  
VALUES($business_id,$from_number,NOW())  
RETURNING id;
```

Default stage = `New` (your enum handles this)

Step 4 — Save MESSAGE

sql

 Copy code

```
INSERT INTO messages(  
business_id,  
whatsapp_account_id,  
contact_id,  
direction,  
content,  
status  
)  
VALUES(  
$business_id,  
$whatsapp_account_id,  
$contact_id,  
'in',  
$message_text,  
'sent'  
);
```

This populates Inbox.

Step 5 — Trigger AUTOMATION ENGINE

Fetch rules:

sql

```
SELECT * FROM automation_rules  
WHERE business_id=$business_id AND trigger='message_received';
```

Evaluate:

- keyword matches
- time rules
- stage rules

Execute actions:

- send reply
- update stage
- notify

(we'll deep dive this later)

Step 6 — BILLING USAGE CHECK

Conversation window logic:

perl

```
if no conversation_id in last 24h:  
    create new conversation_id  
    log cost in usage_logs
```

Insert:

sql

```
INSERT INTO usage_logs(business_id,conversation_id,cost)  
VALUES($business_id,$cid,$price);
```

Step 7 — Realtime push to frontend

Socket emit:

csharp

 Copy code

```
new_message event → inbox UI updates instantly
```

RESULT:

- ✓ Message appears in Inbox
- ✓ Contact auto-created
- ✓ Automations fired
- ✓ Billing tracked

All from one webhook.

FLOW 2 — LOAD INBOX CONVERSATION LIST

This shows chat list in dashboard.

Goal:

One row per contact with last message preview.

SQL (optimized by your indexes):

sql

 Copy code

```
SELECT DISTINCT ON (c.id)
  c.id,
  c.name,
  c.phone,
  c.stage,
  c.last_active,
  m.content AS last_message,
  m.sent_at
FROM contacts c
LEFT JOIN messages m
ON c.id=m.contact_id
WHERE c.business_id=$business_id
ORDER BY c.id, m.sent_at DESC;
```

Frontend shows:

- Name/phone
- Stage badge
- Last message
- Time
- Unread logic (later)

Flow 3 — Open Chat (Load Message History)

sql

 Copy code

```
SELECT *
FROM messages
WHERE business_id=$business_id
AND contact_id=$contact_id
ORDER BY sent_at ASC;
```

Simple & fast via indexes.

Flow 4 — Send Message from Dashboard

When business replies.

Step 1 — Billing limit middleware

Check:

sql

 Copy code

```
SELECT SUM(cost) FROM usage_logs
WHERE business_id=$business_id
AND timestamp >= start_of_month;
```

Compare with plan limit.

Step 2 — Send via WhatsApp API

Success/failure callback.

Step 3 — Save message

sql

 Copy code

```
INSERT INTO messages(
  business_id,
  whatsapp_account_id,
  contact_id,
  direction,
  content,
  status
)
VALUES(
  $business_id,
  $whatsapp_id,
  $contact_id,
  'out',
  $message,
  'sent'
);
```

Step 4 — Trigger automations

Rules like:

- after reply → stage move
- after payment text → mark Paid

EDGE CASES YOU MUST HANDLE

Case	Handling
Duplicate webhooks	idempotency check
Message failure	status='failed'
Deleted contact	cascade auto deletes messages
Inactive WhatsApp	reject webhook
Exceeded billing	block sends

CONTACTS DASHBOARD — FULL BACKEND LOGIC (END-TO-END)

Main backend responsibilities:

- 1 Create contacts (auto from inbox + manual)
- 2 Prevent duplicates per business
- 3 Update contact info
- 4 Tag system
- 5 Filtering & searching
- 6 Sync with pipeline & automation

All using your schema.

CORE TABLE (REFERENCE)

sql

 Copy code

```
contacts(  
    id,  
    business_id,  
    phone,  
    name,  
    stage,  
    created_at,  
    last_active,  
    UNIQUE(business_id, phone)  
)
```

FLOW 1 — AUTO CREATE FROM INBOX (already partially covered)

Triggered when webhook receives message.

Logic:

sql

 Copy code

```
IF contact exists (business_id + phone):  
    update last_active  
ELSE:  
    insert new contact
```

Handled by unique constraint → no duplicates.

FLOW 2 — MANUAL CONTACT CREATION (FROM UI)

Endpoint:

bash

 Copy code

POST /contacts

Backend steps:

1 Validate phone format

2 Check duplicate:

sql

 Copy code

```
SELECT 1 FROM contacts  
WHERE business_id=$business_id AND phone=$phone;
```

3 Insert:

sql

 Copy code

```
INSERT INTO contacts(  
  business_id, phone, name, stage  
)  
VALUES(  
  $business_id, $phone, $name, 'New'  
)
```

FLOW 3 — UPDATE CONTACT DETAILS

Example: rename, update phone, stage override.

Endpoint:

bash

 Copy code

PUT /contacts/:id

SQL:

sql

 Copy code

```
UPDATE contacts  
SET name=$name,  
    phone=$phone  
WHERE id=$id AND business_id=$business_id;
```

If phone changes → uniqueness still enforced per business.

🏷️ FLOW 4 — TAG SYSTEM (IMPORTANT FOR SEGMENTATION)

➕ Add tag:

```
sql Copy code  
  
INSERT INTO contact_tags(  
    business_id,  
    contact_id,  
    tag  
)  
VALUES($business_id,$contact_id,$tag);
```

✖ Remove tag:

```
sql Copy code  
  
DELETE FROM contact_tags  
WHERE business_id=$business_id  
AND contact_id=$contact_id  
AND tag=$tag;
```

📥 Load tags for contact:

```
sql Copy code  
  
SELECT tag FROM contact_tags  
WHERE contact_id=$contact_id;
```

🔍 FLOW 5 — SEARCH + FILTER CONTACTS

Common filters:

- by name/phone
- by stage
- by tag
- by last active

Example: filter by stage

```
sql Copy code  
  
SELECT * FROM contacts  
WHERE business_id=$business_id  
AND stage='Interested'  
ORDER BY last_active DESC;
```

Example: filter by tag

```
sql Copy code  
  
SELECT c.*  
FROM contacts c  
JOIN contact_tags t ON c.id=t.contact_id  
WHERE c.business_id=$business_id  
AND t.tag='Hot Lead';
```

🔗 FLOW 6 — CONTACT STAGE CHANGE (PIPELINE LINK)

When contact moves stage manually or via automation.

Steps:

1 Fetch old stage:

```
sql Copy code  
  
SELECT stage FROM contacts WHERE id=$id;
```

2 Update:

```
sql Copy code  
  
UPDATE contacts SET stage=$new_stage WHERE id=$id;
```

3 Log history:

```
sql Copy code  
  
INSERT INTO pipeline_history(  
  business_id, contact_id, from_stage, to_stage  
)  
VALUES(  
  $business_id, $id, $old_stage, $new_stage  
)
```

4 Trigger automation rules (stage_changed)

📊 FLOW 7 — CONTACT ACTIVITY TRACKING

Each time message arrives:

```
sql Copy code  
  
UPDATE contacts SET last_active=NOW()  
WHERE id=$contact_id;
```

Used in:

- inbox sorting
- analytics
- follow-ups

⚠ EDGE CASE HANDLING

Problem	Fix
Duplicate phone	UNIQUE(business_id, phone)
Cross business access	always filter by business_id
Deleted contact	cascades messages & logs
Bad stage value	enum pipeline_stage prevents
Mass imports later	batch upsert

📊 SALES PIPELINE DASHBOARD — FULL BACKEND LOGIC

Your pipeline is powered by:

 Copy code

```
contacts.stage  
pipeline_history  
automation_rules
```

With enum safety:

 Copy code

```
pipeline_stage = New → Engaged → Interested → Paid → Lost
```

🎯 CORE GOALS OF PIPELINE

- Move leads through stages
- Track conversion flow
- Trigger automations
- Feed analytics

FLOW 1 — LOAD PIPELINE BOARD (KANBAN VIEW)

Frontend needs:

Columns = each pipeline stage

Cards = contacts inside each stage

Backend query:

sql

 Copy code

```
SELECT id, name, phone, stage, last_active
FROM contacts
WHERE business_id=$business_id
ORDER BY last_active DESC;
```

Frontend groups by `stage`.

Fast due to index.

FLOW 2 — MOVE LEAD BETWEEN STAGES (DRAG & DROP)

Endpoint:

ruby

 Copy code

```
PUT /pipeline/:contactId/move
```

Backend steps:

Step 1 — Fetch current stage

sql

 Copy code

```
SELECT stage FROM contacts
WHERE id=$contact_id AND business_id=$business_id;
```

Step 2 — Validate transition (optional but pro)

Example rules:

vbnet

 Copy code

```
New → Engaged ✓
Interested → Paid ✓
Lost → Paid ✗ (optional block)
```

This avoids bad flows.



Step 3 — Update contact stage

sql

 Copy code

```
UPDATE contacts
SET stage=$new_stage
WHERE id=$contact_id;
```

Step 4 — Log movement

sql

 Copy code

```
INSERT INTO pipeline_history(
  business_id,
  contact_id,
  from_stage,
  to_stage
)
VALUES(
  $business_id,
  $contact_id,
  $old_stage,
  $new_stage
);
```

Step 5 — Trigger automations

Fetch:

sql

 Copy code

```
SELECT * FROM automation_rules
WHERE business_id=$business_id
AND trigger='stage_changed';
```

Examples:

- Interested → send brochure
- Paid → thank you message
- Lost → feedback request

FLOW 3 — PIPELINE METRICS (FOR ANALYTICS)

Leads per stage:

sql

 Copy code

```
SELECT stage, COUNT(*)
FROM contacts
WHERE business_id=$business_id
GROUP BY stage;
```

Conversion tracking:

sql

 Copy code

```
SELECT to_stage, COUNT(*)  
FROM pipeline_history  
WHERE business_id=$business_id  
GROUP BY to_stage;
```

FLOW 4 — TIME SPENT IN EACH STAGE (ADVANCED)

Helps find bottlenecks.

Logic:

sql

 Copy code

```
difference between stage change timestamps
```

Query example:

sql

 Copy code

```
SELECT contact_id,  
       from_stage,  
       to_stage,  
       changed_at  
  FROM pipeline_history  
 WHERE business_id=$business_id  
 ORDER BY changed_at;
```

Process in backend.

🔔 FLOW 5 — AUTOMATIC STAGE UPDATES (FROM INBOX)

Examples:

Message	Stage
"interested"	Interested
"paid"	Paid
"not now"	Lost

Automation engine executes:

sql Copy code

```
UPDATE contacts SET stage='Interested'  
WHERE id=$contact_id;
```

Then logs pipeline_history.

⚠ EDGE CASES HANDLED

Issue	Fix
Invalid stage	ENUM prevents
Cross business tampering	business_id check
Duplicate history	only log on change
Deleted contact	cascade cleanup
Out-of-order automations	timestamp based

AUTOMATION DASHBOARD — FULL BACKEND LOGIC

Powered by:

```
java  
  
automation_rules  
messages  
contacts  
pipeline_history  
campaigns (later)
```

 Copy code

With JSON conditions & actions.

AUTOMATION RULE STRUCTURE (IMPORTANT)

Your table:

```
sql  
  
automation_rules(  
  id,  
  business_id,  
  trigger,  
  condition JSONB,  
  action JSONB,  
  delay_minutes  
)
```

 Copy code

This allows infinite flexibility.

Example rule stored in DB

Trigger: message_received

```
json  
  
{  
  "contains": ["price", "cost"]  
}
```

 Copy code

Action:

```
json  
  
{  
  "send_message": "pricing_template_id",  
  "move_stage": "Interested"  
}
```

 Copy code

AUTOMATION EXECUTION FLOW (CORE ENGINE)

This runs on:

- new incoming message
- stage change
- time delay

MASTER FUNCTION

scss

 Copy code

```
onEvent(eventType, payload):  
  fetch matching rules  
  evaluate conditions  
  execute actions  
  schedule delayed jobs
```

FLOW 1 — MESSAGE RECEIVED AUTOMATION

Triggered after inbox webhook saves message.

Step 1 — Load rules

sql

 Copy code

```
SELECT * FROM automation_rules  
WHERE business_id=$business_id  
AND trigger='message_received';
```

Step 2 — Evaluate condition

Example logic:

sql

 Copy code

```
if message.content contains any keyword in condition.contains
```

JSON evaluation in backend.

Step 3 — Execute actions

Send reply:

sql

 Copy code

```
SELECT content FROM message_templates  
WHERE id=$template_id;
```

Send via WhatsApp API.

Save to **messages** as outgoing.

Move pipeline stage:

sql

 Copy code

```
UPDATE contacts SET stage='Interested'  
WHERE id=$contact_id;
```

Log pipeline_history.

FLOW 2 — STAGE CHANGED AUTOMATION

Trigger after pipeline move.

Load:

sql

 Copy code

```
SELECT * FROM automation_rules  
WHERE business_id=$business_id  
AND trigger='stage_changed';
```

Condition example:

json

 Copy code

```
{ "from": "Engaged", "to": "Interested" }
```

Actions:

- send brochure
 - notify owner
 - tag contact
-

FLOW 3 — TIME DELAY AUTOMATION (FOLLOW UPS)

Example:

If no reply in 24 hours → send reminder

Logic:

When message arrives:

```
nginx
```

 Copy code

```
schedule job after delay_minutes
```

Later job checks:

```
sql
```

 Copy code

```
has new message arrived?  
if not → execute action
```

Uses:

 Copy code

```
messages.sent_at  
contacts.last_active
```

ACTION TYPES YOU SHOULD SUPPORT

Action	Backend operation
Send message	WhatsApp API + messages insert
Move stage	contacts + pipeline_history
Add tag	contact_tags insert
Notify user	email/web push
Create campaign	campaigns insert

SAFETY & EDGE CASES

Issue	Fix
Automation loops	detect repeated triggers
Message flood	rate limit
Deleted contact	cancel jobs
Invalid JSON	schema validation
Conflicting rules	priority system

WHY JSON RULES ARE POWERFUL

You can later support:

- AND/OR logic
- multiple keywords
- sentiment
- time windows

Without DB changes.

AUTOMATION METRICS (for analytics)

Track:

- replies generated
- conversions after automation
- time saved

All derivable from:

```
nginx
```

 Copy code

```
messages
```

```
pipeline_history
```

```
automation_rules
```

BROADCAST DASHBOARD — FULL BACKEND LOGIC

Powered by:

nginx

 Copy code

```
campaigns
campaign_logs
message_templates
contacts
messages
automation_rules
```

BROADCAST GOALS

- Send approved templates in bulk
- Target specific contacts
- Track delivery & engagement
- Trigger automation on replies
- Respect billing limits

FLOW 1 — CREATE CAMPAIGN

Endpoint

bash

 Copy code

```
POST /campaigns
```

Payload example:

json

 Copy code

```
{
  "name": "Festive Offer",
  "template_id": 5,
  "filters": {
    "stage": "Interested",
    "tags": ["Hot Lead"]
  },
  "scheduled_at": "2026-02-05T10:00:00Z"
}
```

(You may store filters in memory or extend campaigns table later)

Backend steps:

Step 1 — Validate template ownership

sql

 Copy code

```
SELECT id FROM message_templates  
WHERE id=$template_id AND business_id=$business_id;
```

Step 2 — Insert campaign

sql

 Copy code

```
INSERT INTO campaigns(  
  business_id, template_id, name, scheduled_at, status  
)  
VALUES($business_id,$template_id,$name,$scheduled_at,'scheduled');
```

FLOW 2 — SELECT TARGET CONTACTS

Based on filters.

Example: stage + tag

sql

 Copy code

```
SELECT c.id  
FROM contacts c  
JOIN contact_tags t ON c.id=t.contact_id  
WHERE c.business_id=$business_id  
AND c.stage='Interested'  
AND t.tag='Hot Lead';
```

Result = broadcast list.

FLOW 3 — SCHEDULER TRIGGERS CAMPAIGN

Cron job runs every minute:

pgsql

 Copy code

```
SELECT * FROM campaigns  
WHERE status='scheduled' AND scheduled_at <= NOW();
```

For each campaign:

→ begin sending loop

FLOW 4 — SEND MESSAGES (CONTROLLED BATCH)

Never blast all at once (WhatsApp may block).

Recommended:

sql

 Copy code

```
send 50-100 messages per minute
```

For each contact:

1 Send via WhatsApp API using template

2 Log result

sql

 Copy code

```
INSERT INTO campaign_logs(
    campaign_id,
    contact_id,
    delivered,
    sent_at
)
VALUES($campaign_id,$contact_id,false,NOW());
```

3 Save message to inbox

sql

 Copy code

```
INSERT INTO messages(
    business_id,
    whatsapp_account_id,
    contact_id,
    direction,
    content,
    status
)
VALUES(
    $business_id,
    $whatsapp_id,
    $contact_id,
    'out',
    $template_content,
    'sent'
);
```

4 Billing usage check

If new conversation window → log in `usage_logs`.

FLOW 5 — DELIVERY STATUS WEBHOOK

WhatsApp sends:

```
arduino
```

[Copy code](#)

```
delivered / read / failed
```

Update:

```
sql
```

[Copy code](#)

```
UPDATE campaign_logs  
SET delivered=true  
WHERE campaign_id=$cid AND contact_id=$contact_id;
```

Also update:

```
sql
```

[Copy code](#)

```
UPDATE messages SET status='delivered'  
WHERE contact_id=$contact_id AND sent_at=$time;
```

FLOW 6 — REPLY TRACKING (ENGAGEMENT)

When contact replies:

Inbox webhook runs (already built).

Now mark:

```
sql
```

[Copy code](#)

```
UPDATE campaign_logs  
SET replied=true  
WHERE contact_id=$contact_id  
AND campaign_id=$campaign_id;
```

(Detect via sent time window)

FLOW 7 — AUTOMATION FROM BROADCAST

Example rules:

- If replied to campaign → move stage to Engaged
- If clicked link → tag Interested

Automation engine handles this using trigger:

nginx

 Copy code

```
message_received_after_campaign
```

FLOW 8 — CAMPAIGN STATS (ANALYTICS)

Delivered:

sql

 Copy code

```
SELECT COUNT(*) FROM campaign_logs  
WHERE campaign_id=$id AND delivered=true;
```

Replies:

sql

 Copy code

```
SELECT COUNT(*) FROM campaign_logs  
WHERE campaign_id=$id AND replied=true;
```

Conversion lift:

Join with pipeline_history.



IMPORTANT EDGE CASES

Issue	Fix
Spam risk	rate limit
Wrong template	validate ownership
Exceeded billing	block sends
Deleted contact	cascade skip
API failure	retry queue

WHY YOUR SCHEMA FITS BROADCAST PERFECTLY

Need	Table
Campaign metadata	campaigns
Delivery tracking	campaign_logs
Inbox sync	messages
Automation	automation_rules
Billing	usage_logs
Segmentation	contacts + tags

ANALYTICS DASHBOARD — FULL BACKEND LOGIC

Analytics is powered mainly by:

```
nginx  
  
contacts  
messages  
pipeline_history  
campaign_logs  
usage_logs
```

 Copy code

No extra tables needed.

CORE ANALYTICS CATEGORIES

- 1 Lead growth
- 2 Pipeline performance
- 3 Messaging performance
- 4 Campaign ROI
- 5 Automation impact
- 6 Usage & cost

1. LEAD GROWTH METRICS

New leads per day/week/month

sql

 Copy code

```
SELECT DATE(created_at) AS day, COUNT(*)  
FROM contacts  
WHERE business_id=$business_id  
GROUP BY day  
ORDER BY day;
```

Active leads

sql

 Copy code

```
SELECT COUNT(*)  
FROM contacts  
WHERE business_id=$business_id  
AND last_active >= NOW() - INTERVAL '7 days';
```

2. PIPELINE PERFORMANCE

Leads per stage

sql

 Copy code

```
SELECT stage, COUNT(*)
FROM contacts
WHERE business_id=$business_id
GROUP BY stage;
```

Conversion funnel

sql

 Copy code

```
SELECT to_stage, COUNT(*)
FROM pipeline_history
WHERE business_id=$business_id
GROUP BY to_stage;
```

Win rate

sql

 Copy code

Paid / Total leads

Derived in backend.

3. MESSAGING PERFORMANCE

Total messages sent/received

sql

 Copy code

```
SELECT direction, COUNT(*)
FROM messages
WHERE business_id=$business_id
GROUP BY direction;
```

Avg response time

Logic:

lua

 Copy code

```
time between inbound msg and next outbound msg
```

Backend calculation using message timestamps.

4. CAMPAIGN PERFORMANCE

Delivery rate

sql

 Copy code

```
delivered / total sent
```

Using `campaign_logs`.

Reply rate

sql

 Copy code

```
replied / delivered
```

Leads generated from campaigns

Join replies → pipeline stage changes.

5. AUTOMATION IMPACT

Examples:

- messages sent automatically
- stages changed by automation
- conversions after automation

Derived from:

nginx

 Copy code

```
messages + automation_rules + pipeline_history
```

6. BILLING & USAGE ANALYTICS

Conversations this month

sql

 Copy code

```
SELECT COUNT(DISTINCT conversation_id)
FROM usage_logs
WHERE business_id=$business_id
AND timestamp >= start_of_month;
```

Total cost

sql

 Copy code

```
SELECT SUM(cost)
FROM usage_logs
WHERE business_id=$business_id;
```

REAL-WORLD ANALYTICS FLOW

sql

 Copy code

raw activity → SQL aggregates → API response → charts

No heavy processing needed initially.

ANALYTICS API DESIGN

pgsql

```
GET /analytics/leads
GET /analytics/pipeline
GET /analytics/messages
GET /analytics/campaigns
GET /analytics/usage
```

Each returns computed metrics.

EDGE CASE HANDLING

Problem	Fix
Empty data	return zeros
Deleted contacts	cascade handled
Timezone	TIMESTAMPTZ
Huge data	paginate + cache
Slow queries	indexes already added

TEMPLATE MANAGER — FULL BACKEND LOGIC

Powered by:

nginx

 Copy code

```
message_templates  
messages  
campaigns  
automation_rules
```

GOALS OF TEMPLATE SYSTEM

- Create reusable message formats
- Submit for WhatsApp approval
- Track status
- Use everywhere safely
- Support variables later

FLOW 1 — CREATE TEMPLATE (FROM DASHBOARD)

Endpoint:

bash

 Copy code

```
POST /templates
```

Backend steps:

 Step 1 — Validate ownership

Attach `business_id` from auth.

Step 2 — Insert template

sql

 Copy code

```
INSERT INTO message_templates(  
    business_id, name, content, status  
)  
VALUES($business_id,$name,$content,'pending');
```

Step 3 — Send to WhatsApp API for approval

Payload:

pgsql

 Copy code

```
template name  
template content  
language  
category
```

FLOW 2 — WHATSAPP APPROVAL WEBHOOK

WhatsApp later responds:

approved / rejected.

Update:

sql

 Copy code

```
UPDATE message_templates  
SET status='approved'  
WHERE id=$template_id AND business_id=$business_id;
```

(or rejected)

Flow 3 — List Templates

sql

 Copy code

```
SELECT *
FROM message_templates
WHERE business_id=$business_id
ORDER BY id DESC;
```

Frontend shows status badges.

Flow 4 — Edit Template

Allowed only if:

lua

 Copy code

```
status != approved
```

(WhatsApp blocks editing approved ones)

sql

 Copy code

```
UPDATE message_templates
SET name=$name, content=$content
WHERE id=$id AND business_id=$business_id;
```

Resubmit for approval.

X Flow 5 — Delete Template

Only if:

python

 Copy code

```
not used in campaigns or automations
```

Check:

sql

 Copy code

```
SELECT 1 FROM campaigns WHERE template_id=$id;
```

If safe:

sql

 Copy code

```
DELETE FROM message_templates WHERE id=$id;
```

📍 FLOW 6 — USING TEMPLATE (EVERYWHERE)

Whenever template is used:

Automation:

sql

 Copy code

```
SELECT content FROM message_templates  
WHERE id=$template_id AND status='approved';
```

Broadcast:

same query.

Save outgoing message to `messages`.

🧠 VARIABLE SUPPORT (OPTIONAL BUT POWERFUL)

Template example:

arduino

 Copy code

```
Hi {{name}}, your order is ready!
```

Backend replaces:

 Copy code

```
 {{name}} → contacts.name
```

Before sending.

No schema change needed.

⚠️ IMPORTANT SAFETY RULES

Rule	Reason
Only approved templates can broadcast	WhatsApp compliance
Lock approved edits	API restriction
Business isolation	SaaS safety
Delete checks	prevent broken campaigns

📈 TEMPLATE ANALYTICS (from existing data)

Track:

- usage count
- replies per template
- conversion per template

Derived via:

nginx

 Copy code

```
messages + campaign_logs
```

BILLING DASHBOARD — FULL BACKEND LOGIC

Powered by:

nginx

 Copy code

```
plans  
subscriptions  
usage_logs  
businesses
```

BILLING MODEL YOU'RE USING (BEST ONE)

WhatsApp-style usage-based pricing:

- ✓ Charge per conversation window
- ✓ Limit by monthly plan
- ✓ Easy upsell
- ✓ Fair pricing

Exactly what real CRMs use.

BILLING DATA REFERENCE

plans

bash

 Copy code

```
id | name | conversation_limit | price
```

subscriptions

lua

 Copy code

```
business_id | plan_id | renews_at | status
```

usage_logs

pgsql

 Copy code

```
business_id | conversation_id | cost | timestamp
```

FLOW 1 — BUSINESS SUBSCRIPTION CREATION

When business signs up.

Steps:

1 Assign default plan (free/trial)

```
sql Copy code  
  
INSERT INTO subscriptions(  
  business_id, plan_id, renews_at, status  
)  
VALUES($business_id,$plan_id,NOW() + INTERVAL '30 days', 'active');
```

FLOW 2 — TRACK CONVERSATION USAGE (REAL TIME)

Runs during inbox + send message.

Conversation window logic:

```
pgsql Copy code  
  
Find if conversation_id exists in last 24h  
IF NOT → create new conversation_id  
log cost
```

SQL:

```
sql Copy code  
  
INSERT INTO usage_logs(  
  business_id, conversation_id, cost  
)  
VALUES($business_id,$cid,$price);
```

FLOW 3 — ENFORCE PLAN LIMIT (CRITICAL)

Middleware before sending messages:

Step 1 — Fetch plan

```
sql Copy code  
  
SELECT p.conversation_limit  
FROM subscriptions s  
JOIN plans p ON s.plan_id=p.id  
WHERE s.business_id=$business_id  
AND s.status='active';
```

Step 2 — Count current month usage

sql

 Copy code

```
SELECT COUNT(DISTINCT conversation_id)
FROM usage_logs
WHERE business_id=$business_id
AND timestamp >= date_trunc('month',NOW());
```

Step 3 — Block or allow

bash

 Copy code

```
if count >= limit → reject send
else → allow
```

FLOW 4 — BILLING DASHBOARD METRICS

Conversations used:

sql

 Copy code

```
SELECT COUNT(DISTINCT conversation_id)
FROM usage_logs
WHERE business_id=$business_id
AND timestamp >= date_trunc('month',NOW());
```

Total spend:

sql

 Copy code

```
SELECT SUM(cost)
FROM usage_logs
WHERE business_id=$business_id;
```

Days until renewal:

scss

 Copy code

```
subscriptions.renews_at - NOW()
```

⌚ FLOW 5 — PLAN UPGRADE

When business upgrades:

Steps:

1 Update subscription

sql

 Copy code

```
UPDATE subscriptions
SET plan_id=$new_plan
WHERE business_id=$business_id;
```

2 Immediately unlock higher limit.

No data loss.

📋 FLOW 6 — MONTHLY RESET (AUTO)

You do NOT delete usage_logs.

You just query by month window.

Clean & safe.

⚠ EDGE CASES YOU MUST HANDLE

Issue	Fix
Expired subscription	block sends
Plan downgrade	enforce new limit
Payment failure	status='expired'
Timezone	TIMESTAMPTZ
Free trial abuse	cap