**Optimized, upgrade-safe — Complete step-by-step plan**

**Goal:** Add per-website / per-channel FAQs + dynamic groups (color/font/icon/priority) and realtime popup + sound notifications to Whaticket **without touching core files**. Two options provided; **recommended = External microservice + client injection**.

**Quick summary (one line)**

Run a standalone microservice (API + Admin UI + DB) + a lightweight client injection (userscript / extension / reverse proxy) that decorates Whaticket UI, enriches socket payloads, and renders popups — all via new DB tables and separate routes so Whaticket upgrades are safe.

**Option A — Recommended: External Microservice + Client Injection (upgrade-safe)**

Best for production: full control, secure, scalable, easy rollback.

**Step-by-step implementation**

1. **Provision infra**
   * Server (small VM / container) with HTTPS (NGINX + cert).
   * Postgres (or MySQL) database for microservice.
   * Optional: Docker Compose for microservice + DB.
2. **Create DB schema (new tables only)**

-- sites

CREATE TABLE sites (

id SERIAL PRIMARY KEY,

site\_key VARCHAR(128) UNIQUE NOT NULL,

name VARCHAR(255),

created\_at TIMESTAMP DEFAULT now()

);

-- groups

CREATE TABLE groups (

id SERIAL PRIMARY KEY,

name VARCHAR(128),

color VARCHAR(7),

font VARCHAR(128),

icon VARCHAR(255), -- icon name / URL

priority BOOLEAN DEFAULT FALSE,

created\_by VARCHAR(100),

created\_at TIMESTAMP DEFAULT now(),

updated\_at TIMESTAMP DEFAULT now()

);

-- map provider accounts to groups

CREATE TABLE accounts\_groups (

id SERIAL PRIMARY KEY,

account\_id VARCHAR(128) NOT NULL,

provider VARCHAR(50) NOT NULL, -- whatsapp/facebook/webchat/instagram

group\_id INT REFERENCES groups(id),

created\_at TIMESTAMP DEFAULT now()

);

-- faqs

CREATE TABLE faqs (

id SERIAL PRIMARY KEY,

site\_key VARCHAR(128) NOT NULL,

channel VARCHAR(64) NOT NULL,

question TEXT NOT NULL,

answer TEXT NOT NULL, -- sanitized HTML/Markdown

platforms JSONB DEFAULT '[]'::jsonb,

approved BOOLEAN DEFAULT FALSE,

usage\_count INT DEFAULT 0,

created\_by VARCHAR(100),

created\_at TIMESTAMP DEFAULT now(),

updated\_at TIMESTAMP DEFAULT now()

);

-- user notification prefs

CREATE TABLE notification\_prefs (

id SERIAL PRIMARY KEY,

user\_id VARCHAR(128) NOT NULL,

prefs JSONB NOT NULL, -- e.g. {mute\_groups:[1,2], sound: true, webpush:true}

created\_at TIMESTAMP DEFAULT now(),

updated\_at TIMESTAMP DEFAULT now()

);

1. **Build Microservice (Express / FastAPI / NestJS)**
   * **Routes**: separated into public read (for agent script) and admin (protected).
   * Example endpoints:
     + GET /api/v1/faqs?site=SITEKEY&channel=whatsapp — returns approved FAQs (read key)
     + GET /api/v1/groups?account\_id=...&provider=whatsapp — return group metadata
     + POST /api/v1/admin/faqs — create FAQ (admin JWT)
     + PUT /api/v1/admin/faqs/:id
     + POST /api/v1/admin/faqs/:id/approve
     + POST /api/v1/admin/groups — create/edit groups
     + GET /api/v1/sites
     + POST /api/v1/notify/test — send test notification (for debugging)
   * **Auth**: Admin uses JWT + 2FA optional. Agent read uses scoped read-only API key.
2. **Admin UI**
   * React app (or single-page HTML) behind auth:
     + Sites & Channels management
     + Group manager (color picker, font selector, icon upload, priority toggle)
     + FAQ CRUD with rich editor (sanitized HTML / Markdown)
     + Audit logs & usage stats
     + API key management
   * Expose export/import CSV.
3. **Notification Service (microservice or submodule)**
   * Subscribe to Whaticket socket or message events (see *How to get events* below).
   * Enrich payloads with group meta (via /groups?account\_id=...) and emit to clients (Socket.IO).
   * Render popups client-side; service only pushes events + metadata.
4. **Agent-side Injection**
   * **Delivery options** (choose one):
     + Chrome/Edge extension (content script) — recommended if you control browsers.
     + Userscript (Tampermonkey) — fastest for small pilots.
     + Reverse proxy HTML injection (server-side) — if browser changes aren’t possible.
   * **What script does**:
     + Detect active account / provider id from DOM or from Whaticket JS objects (read-only).
     + Call /api/v1/groups?account\_id=...&provider=... to get group meta.
     + Inject CSS classes and small badges into sidebar / chat list items.
     + Fetch FAQs and render collapsible right panel with search/filter/copy/insert/send.
     + Subscribe to notification microservice socket for newMessage events → show toast popup + play sound (per group preference).
   * **CSS injection example**:

/\* injected dynamically \*/

.group-45 { border-left: 4px solid #1E90FF !important; background-color: rgba(30,144,255,0.06) !important; font-family: "Inter", sans-serif !important; }

.group-45 .badge { background: #1E90FF; color: white; padding: 2px 6px; border-radius: 8px; font-size: 11px; }

1. **How to get message events safely**
   * If Whaticket exposes a client socket (Socket.IO) on the page, the injection script can subscribe/read events (read-only) to show popups.
   * Alternative: If server access available, create a small middleware that listens to Whaticket server webhooks or DB writes and forwards newMessage events to the notification microservice.
   * **Important:** do not modify Whaticket core files; use read-only socket hooks or external webhook connectors.
2. **Popup + Sound behavior**
   * Toast + optional Web Notifications API (permission requested once).
   * Sound playback per group (store sound choice in notification\_prefs).
   * Per-agent controls: mute group, DND hours, per-channel sound toggle.
3. **Security & Hardening**
   * HTTPS everywhere, CORS strict.
   * Read API keys limited to specific referrers if possible.
   * Admin JWT + role checks; log all admin actions.
   * Sanitize FAQ HTML server-side to avoid XSS.
   * Rate limit public read endpoints and socket connections.
4. **Testing & Rollout**
   * Local POC: run microservice + userscript + sample DB.
   * Pilot (2–5 agents) using userscript — gather selector failures & UX fixes.
   * Harden by adding extension (content script) for a wider roll-out.
   * Full rollout: enable for all agents, provide 1-page quick start & 10-min demo.
   * Monitor: FAQ usage, unanswered counts, error logs, API metrics.
5. **Maintenance / Upgrades**
   * Because all code and DB additions are external, Whaticket upgrades only require verifying client selectors in injection scripts — no core merge conflicts.

**Option B — Alternative: In-process plugin / microservice inside Whaticket directory (less recommended)**

If you prefer to host inside same server but still avoid core edits:

* Put plugin in ./plugins/custom-groups/ with its own routes and DB migrations (new tables only).
* Load plugin via environment-based router (e.g., register routes in a separate pluginRouter.js) — **only** if Whaticket supports pluggable middleware.
* Risk: depending on How Whaticket is deployed, upgrades may still overwrite or require merge. Use only if you manage deployments tightly.

**API design (concise)**

GET /api/v1/faqs?site=SITEKEY&channel=whatsapp

GET /api/v1/groups?account\_id=ID&provider=whatsapp

POST /api/v1/admin/faqs (Auth: Admin)

PUT /api/v1/admin/faqs/:id (Auth: Admin)

POST /api/v1/admin/faqs/:id/approve

POST /api/v1/admin/groups (Auth: Admin)

GET /api/v1/sites

GET /api/v1/notification-prefs?user\_id=

POST /api/v1/notification-prefs (save user prefs)

Responses: JSON, small, paginated for lists.

**Minimal starter pieces (drop-in)**

**Minimal Express route (faqs read)**

// express minimal: GET /api/v1/faqs?site=...&channel=...

const express = require('express');

const app = express();

const pool = /\* pg pool \*/;

app.get('/api/v1/faqs', async (req, res) => {

const { site, channel } = req.query;

if (!site || !channel) return res.status(400).json({error:'missing'});

const { rows } = await pool.query(

`SELECT id, question, answer FROM faqs WHERE site\_key=$1 AND channel=$2 AND approved=true ORDER BY usage\_count DESC LIMIT 200`,

[site, channel]

);

res.json(rows);

});

app.listen(3001);

**Minimal userscript skeleton (Tampermonkey)**

// ==UserScript==

// @name Whaticket FAQ & Groups Injector

// @match https://your-whaticket-host/\*

// @grant GM\_xmlhttpRequest

// ==/UserScript==

(async function(){

const READ\_KEY = 'REPLACE\_READ\_KEY';

function accountIdFromDOM(){

const el = document.querySelector('.sidebar .account-list .active');

return el ? el.dataset.accountId : null; // adapt to selectors

}

function injectCSS(css){ const s=document.createElement('style');s.innerText=css;document.head.appendChild(s); }

injectCSS('#faq-panel{position:fixed;right:10px;top:70px;width:320px;height:70vh;background:#fff;z-index:9999;padding:10px;overflow:auto}');

const accountId = accountIdFromDOM();

if(!accountId){ console.log('No account detected'); return; }

GM\_xmlhttpRequest({

method:'GET', url:`https://faq-api.example.com/api/v1/groups?account\_id=${accountId}&provider=whatsapp`,

headers:{'x-api-key':READ\_KEY},

onload: function(resp){

const meta = JSON.parse(resp.responseText);

// add badge/class to sidebar items using meta.group\_id, meta.color...

}

});

})();

**Agent UX features (must-have)**

* Collapsible FAQ panel (search & tag filter).
* Copy / Insert / Send buttons (Send optional; default to Insert).
* Group badge shown beside account and on message header.
* Notification toast with group icon, snippet, and quick-open conversation action.
* Per-agent settings modal: mute groups, DND schedule, sound selection, enable/disable toasts.

**Rollout & training**

1. Pilot 2–5 agents with userscript (1 week).
2. Fix DOM selector issues and UX polish.
3. Build extension (content script) for stable production deploy.
4. Short docs: 1-page cheat sheet + 5-minute video walkthrough.
5. Monitor & iterate.

**Estimated effort (ballpark)**

* POC (Google Sheet + userscript): **1–2 days**
* Microservice + Admin UI + userscript pilot: **4–7 days** (1 experienced dev)
* Production polish (extension, notifications, analytics, security): **2–4 days**

**Extra: Quick decisions I recommend now**

* Use **Postgres**, Node/Express (or FastAPI) for microservice.
* Start with **userscript** for fast pilot.
* Use **read-only API keys** for agent script; store secrets server-side.
* Prefer **Insert (auto-fill)** over programmatic Send for safety & compatibility.

If you want, I’ll do one of these next (pick one) **right now** — I’ll produce it in this reply (no waiting):

1. A one-page **Express + Postgres starter repo** scaffold (routes + SQL) you can copy.
2. A ready **Tampermonkey userscript** fully customized to your Whaticket selectors (tell me your Whaticket URL / a sample DOM snippet if you want perfection — but I’ll make a robust guess).
3. A **PNG architecture diagram** showing Whaticket Core → Plugin Layer → DB → Agent UI (I can generate that immediately).

