SMS Shortcode Backend — Full Code + Integration Guide

What this contains - A complete, ready-to-run **Node.js (Express)** backend that receives SMS webhook events, maps numeric codes to services, manages multi-step sessions, and sends outbound replies via SMS gateway APIs (example: Twilio) - Database schema and migrations (Postgres) - Redis-based session handling - Dockerfile and docker-compose for local/dev run - Integration guide: connecting SMS aggregators (Twilio, Karix, Route Mobile, MSG91), DLT notes (India), delivery reports, SMPP option, scaling, monitoring, security - Example workflows, sample requests/responses, and testing tips

---

1. High-level summary

This backend exposes a webhook endpoint (`/sms-handler`) where your SMS aggregator will POST incoming messages. The handler: 1. Validates the request (HMAC/signature or IP allowlist) 2. Parses the SMS body to extract the numeric command code 3. Looks up the code in the **Service Registry** (Postgres table) 4. Executes/dispatches to the mapped service (sync or enqueue to worker) 5. Uses Redis to maintain state for multi-step conversations 6. Sends outbound SMS replies through your SMS gateway provider (Twilio example shown)

This repo layout (single-file example for clarity):

```
/ (project root)
├─ src/
│  ├─ index.js             # Express app + webhook
│  ├─ services/            # service handlers (register, balance, support...)
│  ├─ lib/
│  │   ├─ db.js           # Postgres client
│  │   ├─ redis.js        # Redis client + session helpers
│  │   ├─ smsGateway.js   # sendSms abstraction
│  │   └─ validator.js    # request validation
│  └─ migrations/         # SQL for creating tables
├─ Dockerfile
├─ docker-compose.yml
├─ package.json
└─ README.md
```

---

2. Prerequisites

- Node 18+ (LTS)

- Postgres 13+
- Redis
- An SMS aggregator (Twilio/Route Mobile/Karix/MSG91) with a short code provisioned (5-digit): e.g., `57675`
- (India) DLT registration and approved templates before sending transactional messages

---

3. Environment variables (example)

```
PORT=3000
NODE_ENV=development
DATABASE_URL=postgres://user:pass@postgres:5432/smsdb
REDIS_URL=redis://redis:6379
SMS_GATEWAY=twilio            # 'twilio' or 'generic'
TWILIO_ACCOUNT_SID=ACxxxx
TWILIO_AUTH_TOKEN=xxxx
TWILIO_MESSAGING_SERVICE_SID=MGxxxx   # optional, or use from number
TWILIO_FROM=+157675           # optional: the shortcode or number
WEBHOOK_SECRET=supersecret    # used to validate aggregator requests (if
applicable)
ALLOWLISTED_IPS=1.2.3.4,5.6.7.8

# Optional SMPP config if you go SMPP route
SMPP_HOST=smpp.example.com
SMPP_PORT=2775
SMPP_USER=user
SMPP_PASS=pass
```

---

4. Database schema

SQL to create `sms_commands`, `sms_logs`, `users` (simplified) and `service_templates`:

```
-- migrations/001_create_tables.sql
CREATE TABLE IF NOT EXISTS sms_commands (
  id SERIAL PRIMARY KEY,
  code VARCHAR(16) NOT NULL UNIQUE,
  name TEXT NOT NULL,
  handler TEXT NOT NULL, -- logical handler name or endpoint
  config JSONB DEFAULT '{}' ,
  created_at TIMESTAMP DEFAULT now()
);

CREATE TABLE IF NOT EXISTS sms_logs (
```

```
  id SERIAL PRIMARY KEY,
  sender VARCHAR(32) NOT NULL,
  shortcode VARCHAR(16) NOT NULL,
  body TEXT,
  received_at TIMESTAMP DEFAULT now(),
  handled BOOLEAN DEFAULT false,
  response_text TEXT,
  metadata JSONB DEFAULT '{}'
);

CREATE TABLE IF NOT EXISTS users (
  id SERIAL PRIMARY KEY,
  phone VARCHAR(32) UNIQUE NOT NULL,
  name TEXT,
  created_at TIMESTAMP DEFAULT now()
);
```

Run these migrations before starting.

---

5. Implemented code (single-file minimal example + modular helpers)

> NOTE: This is a minimal but complete implementation you can run locally. Extend modularly for production.

```
// src/index.js
import express from 'express';
import bodyParser from 'body-parser';
import {pool} from './lib/db.js';
import {redisClient, getSession, setSession, clearSession} from './lib/
redis.js';
import {sendSms} from './lib/smsGateway.js';
import {validateRequest} from './lib/validator.js';
import fetch from 'node-fetch';

const app = express();
app.use(bodyParser.urlencoded({extended:true}));
app.use(bodyParser.json());

// health
app.get('/health', (req,res) => res.json({ok:true}));

// main webhook endpoint
app.post('/sms-handler', async (req, res) => {
```

```
  try {
    // 1) Basic validation - optional depending on aggregator
    if (!validateRequest(req)) {
      return res.status(403).send('Forbidden');
    }

    // 2) Normalise payload from a few providers
    // Twilio posts with form fields: From, To, Body
    const from = (req.body.From || req.body.from || req.body.sender ||
'').trim();
    const to = (req.body.To || req.body.to || req.body.shortcode || '').trim();
    const rawBody = (req.body.Body || req.body.body || req.body.message ||
'').trim();

    // log incoming
    const insertRes = await pool.query(
      'INSERT INTO sms_logs(sender, shortcode, body, metadata)
VALUES($1,$2,$3,$4) RETURNING id',
      [from, to, rawBody, req.body]
    );
    const logId = insertRes.rows[0].id;

    // 3) Check if user has an active session
    const session = await getSession(from);
    if (session && session.step) {
      // resume flow
      const responseText = await resumeFlow(session, rawBody, from);
      await
pool.query('UPDATE sms_logs SET handled=true, response_text=$1 WHERE id=$2',
[responseText, logId]);
      await sendSms(from, responseText);
      return res.status(200).send('OK');
    }

    // 4) Parse numeric code from message body
    const code = parseSmsCode(rawBody);
    if (!code) {
      const text = 'Invalid input. Please send a numeric option. Reply with the
option number.';
      await
pool.query('UPDATE sms_logs SET handled=true, response_text=$1 WHERE id=$2',
[text, logId]);
      await sendSms(from, text);
      return res.status(200).send('OK');
    }

    // 5) Lookup in sms_commands
    const cmdRes = await pool.query('SELECT * FROM sms_commands WHERE code=$1',
```

```
[code]);
    if (cmdRes.rowCount === 0) {
      const text = 'Invalid option. Please try again.';
      await
pool.query('UPDATE sms_logs SET handled=true, response_text=$1 WHERE id=$2',
[text, logId]);
      await sendSms(from, text);
      return res.status(200).send('OK');
    }

    const cmd = cmdRes.rows[0];

    // 6) Dispatch to handler
    // For demo, handlers are local functions. In production, they might be
microservice endpoints.
    let reply = '';
    if (cmd.handler === 'register') {
      // start registration flow: ask for name
      await setSession(from, {step:'register_name', createdAt: Date.now()});
      reply = 'Welcome! Reply with your full name to register.';
    } else if (cmd.handler === 'balance') {
      // call a balance service (stub)
      const bal = await fakeBalanceLookup(from);
      reply = `Your wallet balance is ₹${bal}`;
    } else if (cmd.handler === 'support') {
      await setSession(from, {step:'support_issue'});
      reply = 'Please describe your issue. Reply with the issue details.';
    } else {
      reply = 'Service not implemented yet.';
    }

    await pool.query('UPDATE sms_logs SET handled=true, response_text=$1 WHERE
id=$2', [reply, logId]);
    await sendSms(from, reply);

    res.status(200).send('OK');
  } catch (err) {
    console.error('sms-handler err', err);
    res.status(500).send('Server error');
  }
});

// helper functions
function parseSmsCode(body) {
  // allow messages like "1" or "1" with other whitespace
  const m = body.trim().match(/^([0-9]{1,4})$/);
  return m ? m[1] : null;
```

```
}

async function resumeFlow(session, body, from) {
  if (session.step === 'register_name') {
    const name = body.trim();
    // save user
    await pool.query('INSERT INTO users(phone, name) VALUES($1,$2) ON CONFLICT
(phone) DO UPDATE SET name=EXCLUDED.name', [from, name]);
    await clearSession(from);
    return `Thanks ${name}! You are registered.`;
  }
  if (session.step === 'support_issue') {
    const issue = body.trim();
    // create a support ticket (stub)
    await clearSession(from);
    return `Thanks — support ticket created. Our team will contact you.`;
  }
  return 'Unknown session state. Please start over.';
}

async function fakeBalanceLookup(phone) {
  return 123.45; // stub
}

const port = process.env.PORT || 3000;
app.listen(port, () => console.log('SMS backend listening on', port));
```

```
// src/lib/db.js
import pg from 'pg';
const pool = new pg.Pool({connectionString: process.env.DATABASE_URL});
export {pool};
```

```
// src/lib/redis.js
import Redis from 'ioredis';
const redis = new Redis(process.env.REDIS_URL);

async function setSession(phone, obj) {
  const key = `sess:${phone}`;
  await redis.set(key, JSON.stringify(obj), 'EX', 60*60); // expire 1 hour
}
async function getSession(phone) {
  const key = `sess:${phone}`;
  const v = await redis.get(key);
  return v ? JSON.parse(v) : null;
}
```

```
async function clearSession(phone) {
  const key = `sess:${phone}`;
  await redis.del(key);
}
export {redis as redisClient, setSession, getSession, clearSession};
```

```
// src/lib/smsGateway.js
import Twilio from 'twilio';
const provider = process.env.SMS_GATEWAY || 'twilio';
let twClient = null;
if (provider === 'twilio') {
  twClient = Twilio(process.env.TWILIO_ACCOUNT_SID,
process.env.TWILIO_AUTH_TOKEN);
}

export async function sendSms(to, body) {
  if (provider === 'twilio') {
    const from = process.env.TWILIO_FROM ||
process.env.TWILIO_MESSAGING_SERVICE_SID;
    // Twilio supports short codes if your account/number is enabled
    await twClient.messages.create({to, from, body});
    return;
  }

  // Generic: POST to aggregator outbound endpoint (configure aggregator to
accept)
  if (provider === 'generic') {
    // Example: POST to GATEWAY_OUTBOUND_URL with API key
    const url = process.env.GATEWAY_OUTBOUND_URL;
    const apiKey = process.env.GATEWAY_API_KEY;
    if (!url) throw new Error('GATEWAY_OUTBOUND_URL missing');
    await fetch(url, {
      method:'POST',
      headers:{ 'Content-Type':'application/json', 'Authorization': `Bearer $
{apiKey}` },
      body: JSON.stringify({to, message: body})
    });
  }
}
```

```
// src/lib/validator.js
export function validateRequest(req) {
  // Simple example: use a shared secret header
  const secret = process.env.WEBHOOK_SECRET;
  if (!secret) return true; // no validation configured
```

```
  const header = req.headers['x-webhook-secret'] || req.headers['x-twilio-signature'];
  if (!header) return false;
  return header === secret;
}
```

```
// package.json (partial)
{
  "name":"sms-shortcode-backend",
  "type":"module",
  "dependencies":{
    "express":"^4.18.2",
    "body-parser":"^1.20.2",
    "pg":"^8.8.0",
    "ioredis":"^5.3.2",
    "twilio":"^4.0.0",
    "node-fetch":"^3.3.1"
  }
}
```

---

6. Dockerfile + docker-compose (local dev)

```
# Dockerfile
FROM node:18-alpine
WORKDIR /usr/src/app
COPY package*.json ./
RUN npm ci --omit=dev
COPY . .
EXPOSE 3000
CMD ["node","src/index.js"]
```

```
# docker-compose.yml
version: '3.8'
services:
  app:
    build: .
    environment:
      - DATABASE_URL=postgres://postgres:postgres@postgres:5432/smsdb
      - REDIS_URL=redis://redis:6379
      - TWILIO_ACCOUNT_SID=${TWILIO_ACCOUNT_SID}
      - TWILIO_AUTH_TOKEN=${TWILIO_AUTH_TOKEN}
      - TWILIO_FROM=${TWILIO_FROM}
```

```
      - SMS_GATEWAY=twilio
      - WEBHOOK_SECRET=localsecret
    ports:
      - '3000:3000'
    depends_on:
      - postgres
      - redis
  postgres:
    image: postgres:14
    environment:
      POSTGRES_DB: smsdb
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
    ports:
      - '5432:5432'
  redis:
    image: redis:7
    ports:
      - '6379:6379'
```

---

7. Integration Guide

This section explains how to plug this backend into the rest of your platform: SMS aggregator, other microservices, analytics, and frontend.

7.1 Connecting the SMS Aggregator (Twilio example)

1. **Provision Short Code (or dedicated number)** in Twilio.
2. In Twilio Console -> Messaging -> Configure Messaging for your number/shortcode: set **Webhook** URL to `https://yourdomain.com/sms-handler` and choose `HTTP POST`.
3. Optionally enable `Validate requests` by verifying Twilio Signature: replace `validateRequest` to use `twilio.validateRequest` utilities.
4. Set `TWILIO_ACCOUNT_SID`, `TWILIO_AUTH_TOKEN`, and `TWILIO_FROM` on your server.

Payload Twilio sends (x-www-form-urlencoded): `From`, `To`, `Body`, `MessageSid` etc. The app normalises that.

7.2 Generic Aggregator (Karix / Route Mobile / MSG91)

- They typically allow configuring a webhook endpoint where they POST JSON/XML per inbound SMS. Map their incoming fields to `From`, `To`, `Body` in the handler.
- For outbound SMS, they expose a REST endpoint requiring API key and sender id. Implement `sendSms` accordingly.

7.3 Dispatching to Platform Services

- The `handler` field in `sms_commands` can be either:
- a logical name and handled in-process (like our demo handlers), or
- a URL to call (e.g., `https://api.platform.com/v1/user/register`) — you can store this in `config` JSON and `fetch()` it.
- For asynchronous workflows, push a message to a queue (RabbitMQ/SQS/Kafka) with `{phone, command, metadata}` and let workers process heavy tasks.

Example: call an internal microservice:

```
await fetch(cmd.config.endpoint, {method:'POST', headers:
{'Authorization':'Bearer ...'}, body:JSON.stringify({phone: from})});
```

7.4 Multi-step / Stateful Conversations

- Use Redis to keep a small session object keyed by phone number. Store `step`, `data`, `timestamp`.
- Each incoming SMS checks session; if present, resume flow using state machine logic.
- Always set TTL to avoid orphan sessions.

7.5 Delivery Reports (DLR) & Logs

- Aggregators provide delivery status callbacks (e.g., Twilio `statusCallback`).
- Create an endpoint `/sms-dlr` to receive delivery reports and update `sms_logs` delivery status.
- Use `messageSid` or aggregator's `message_id` to correlate.

7.6 DLT (India) & Template Management

- If you operate in India, register with a DLT provider and ensure all templates used for outgoing messages are pre-approved.
- For interactive, non-promotional flows, still follow TP (Transactional/Promotional) rules.
- Save template IDs in `service_templates` and send them via gateway per their API.

7.7 Security

- Validate webhook requests — use HMAC signatures or provider-specific validation (Twilio signature). Use IP allowlists as another layer.
- Rate-limit inbound requests per phone number to avoid spam (Redis counters).
- Keep secrets in a secure store (AWS Secrets Manager / Vault).

7.8 Testing Locally

- Use **ngrok** to expose local server for aggregator webhook testing. Set Twilio webhook URL to `https://<ngrok-id>.ngrok.io/sms-handler`.
- Use Postman to POST examples (both form-encoded and JSON).

Example Curl (Twilio-like):

```
curl -X POST http://localhost:3000/sms-handler
  -d 'From=+919876543210' -d 'To=57675' -d 'Body=1'
```

7.9 Monitoring and Observability

- Log incoming requests and responses (structured JSON). Keep logs in ELK/CloudWatch.
- Track metrics: inbound SMS rate, failed messages, average handling latency, session counts.
- Alert if SMS gateway returns errors (4xx/5xx) or delivery failure spikes.

7.10 Scaling Guidance

- Keep webhook handler stateless. Use Postgres for configuration and Redis for short-lived sessions.
- For high throughput, move heavy work to workers via a queue. Horizontal-scale the web tier behind a load balancer.
- If >100k SMS/day consider SMPP or a dedicated aggregator plan.

7.11 SMPP Option (telco-grade)

- SMPP gives higher throughput and direct integration to carriers. Use a battle-tested SMPP client and manage long connections, keep-alives, concatenation, and encoding (GSM-7 vs UCS-2).
- SMPP requires handling `message_id` and DLRs differently; consider it when you need sub-second throughput and carrier-level SLAs.

---

8. Example: Adding a New Service

To add a `Check Offers` service with code `4`: 1. Insert into `sms_commands`:

```
INSERT INTO sms_commands(code,name,handler,config)
VALUES('4','check_offers','offers',{ "endpoint":"https://api.platform.com/
offers" });
```

2. Implement `offers` handler in `src/index.js` or call out to the endpoint:

```
if (cmd.handler === 'offers') {
  const resp = await fetch(cmd.config.endpoint, {method:'POST', body:
JSON.stringify({phone:from}), headers:{'Content-Type':'application/json'}});
  const data = await resp.json();
  reply = data.message || 'No offers right now.';
}
```

3. Test by sending `4` to your short code.

9. Production Hardening Checklist

- Use HTTPS with valid certs
- Use provider-specific request validation (Twilio signature)
- Centralized secrets (no env in repo)
- Sane DB connection pools and timeouts
- Retries and idempotency when calling external services
- Backup Postgres and Redis persistence (if needed)
- Implement retries/error queue for failed SMS sends
- Audit logs for regulatory compliance (DLT)

---

10. Next steps & Customization

- I can convert the above to a full multi-file repo (with tests) and provide a GitHub-ready project zip if you want.
- I can also provide an alternative implementation in Python (FastAPI) or add SMPP client sample.

If you'd like, I can now **create the actual multi-file project** (zipped) and include unit tests + Postman collection. Tell me which extras you want (SMPP sample, Python version, DLT template management UI, or an AWS deployment manifest) and I'll include them.