1) Create the project

```
mkdir ivr_flask && cd ivr_flask

python -m venv .venv && source .venv/bin/activate   # (Windows: .venv\Scripts\activate)

pip install Flask twilio python-dotenv flask-socketio eventlet
```

We’ll use Socket.IO for live updates (optional; you can skip and just poll).

---

2) Make the folders & files

```
mkdir -p templates static/js static/css data

touch app.py .env templates/index.html static/js/main.js static/css/style.css data/transactions.json
```

---

3) Put secrets in .env

```
TWILIO_SID=ACxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

TWILIO_AUTH_TOKEN=your_auth_token

TWILIO_NUMBER=+1XXXXXXXXXX

PUBLIC_URL=https://your-ngrok-or-host-url

FLASK_ENV=development
```

---

4) Seed 20 dummy transactions

Create data/transactions.json:

```
[

  {"id":"TXN001","client_name":"John Doe","card_number":"***1234","client_phone":"+14155550101","amount":2500.75,"bank_name":"Bank of America","merchant_name":"Amazon","transaction_date":"15/03/2025","action":"Not Answered"},

  {"id":"TXN002","client_name":"Jane Smith","card_number":"***5678","client_phone":"+14155550102","amount":520,"bank_name":"Chase Bank","merchant_name":"Walmart","transaction_date":"14/03/2025","action":"Not Answered"}
  // add up to TXN020 (copy/adjust)
]
```

---

5) Build the Flask app skeleton

Put this in app.py (minimal, working):

```
import json, os
from flask import Flask, render_template, request, jsonify
from twilio.rest import Client
from twilio.twiml.voice_response import VoiceResponse, Gather
from dotenv import load_dotenv

# --- setup
load_dotenv()
app = Flask(__name__)
DATA_PATH = "data/transactions.json"

def read_txns():
   with open(DATA_PATH, "r") as f: return json.load(f)
def write_txns(txns):
   with open(DATA_PATH, "w") as f: json.dump(txns, f, indent=2)

# Twilio client
twilio_client = Client(os.getenv("TWILIO_SID"), os.getenv("TWILIO_AUTH_TOKEN"))
TW_NUMBER = os.getenv("TWILIO_NUMBER")
PUBLIC_URL = os.getenv("PUBLIC_URL")  # e.g., https://abc123.ngrok.io

# --- web pages
@app.get("/")
```

```
def index():
    return render_template("index.html", transactions=read_txns())


# --- APIs for UI
@app.get("/transactions")
def api_transactions():
    return jsonify(read_txns())


@app.post("/update_phone/<txn_id>")
def update_phone(txn_id):
    phone = (request.json or {}).get("client_phone", "")
    txns = read_txns()
    for t in txns:
        if t["id"] == txn_id:
            t["client_phone"] = phone
            break
    write_txns(txns)
    return jsonify({"ok": True})


@app.post("/set_action/<txn_id>")
def set_action(txn_id):
    action = (request.json or {}).get("action", "")
    txns = read_txns()
    for t in txns:
        if t["id"] == txn_id:
            t["action"] = action
            break
    write_txns(txns)
    return jsonify({"ok": True})


# --- start a call
@app.post("/call/<txn_id>")
def call(txn_id):
    txns = read_txns()
    txn = next((t for t in txns if t["id"] == txn_id), None)
    if not txn: return jsonify({"error":"not found"}), 404
    # mark connecting
    txn["action"] = "Connecting..."
    write_txns(txns)

    twilio_client.calls.create(
        to=txn["client_phone"],
        from_=TW_NUMBER,
        url=f"{PUBLIC_URL}/voice/{txn_id}",  # Twilio will fetch TwiML here
        status_callback=f"{PUBLIC_URL}/status/{txn_id}",
        status_callback_event=["completed","no-answer","busy","failed"]
    )
    return jsonify({"ok": True})


# --- TwiML: first prompt
@app.post("/voice/<txn_id>")
def voice(txn_id):
    txns = read_txns()
    txn = next((t for t in txns if t["id"] == txn_id), None)
    resp = VoiceResponse()
```

```
  g = Gather(input="speech", action=f"/gather/{txn_id}", method="POST", timeout=5, 
hints="yes,no,fraud,call back")
  if txn:
    g.say(
      f"Hello. This is an automated call from your bank's fraud prevention team. "
      f"We detected a suspicious activity on the account of {txn['client_name']}, "
      f"card ending {txn['card_number'][-4:]}, at {txn['merchant_name']}, "
      f"for {txn['amount']} dollars through {txn['bank_name']}. "
      "Did you make this transaction? Please say: Yes I did, No I did not, or This is fraud."
    )
  else:
    g.say("Hello. We have a security alert. Did you make the transaction? Please say yes 
or no.")
  resp.append(g)
  resp.say("We did not receive your answer. Goodbye.")
  return str(resp)


# --- Twilio posts speech result here
@app.post("/gather/<txn_id>")
def gather(txn_id):
  speech = request.form.get("SpeechResult", "") or ""
  action = classify_action(speech)


  txns = read_txns()
  for t in txns:
    if t["id"] == txn_id:
      t["action"] = action
      break
  write_txns(txns)
```

```
    resp = VoiceResponse()
    resp.say(f"Thank you. We've marked this call as {action}. Goodbye.")
    resp.hangup()
    return str(resp)

# --- status (detects no-answer/busy/disconnected)
@app.post("/status/<txn_id>")
def status(txn_id):
    call_status = request.form.get("CallStatus", "")
    if call_status in ["no-answer", "busy", "failed"]:
        update(txn_id, "Not Answered")
    elif call_status in ["completed"]:
        pass  # already set in /gather
    return ("", 204)

def update(txn_id, action):
    txns = read_txns()
    for t in txns:
        if t["id"] == txn_id:
            t["action"] = action
            break
    write_txns(txns)

def classify_action(text):
    l = text.lower()
    if any(k in l for k in ["fraud","unauthoriz","chargeback","scam"]): return "Marked as
Fraud"
```

```
  if any(k in l for k in ["yes","i did","authorized"]): return "Resolved"
  if any(k in l for k in ["no","did not","not me","dispute"]): return "Connecting..."
  if not l.strip(): return "Not Answered"
  return "Disconnected"

if __name__ == "__main__":
  # if you use Socket.IO, run with eventlet; for plain Flask, just app.run
  app.run(host="0.0.0.0", port=5000, debug=True)
```

---

6) Build the HTML (dashboard)

templates/index.html (simple, editable phones + actions + call button):

```
<!doctype html>
<html>
<head>
 <meta charset="utf-8" />
 <title>IVR Agent</title>
 <link rel="stylesheet" href="/static/css/style.css" />
</head>
<body>
 <div class="container">
  <h1>IVR Agent</h1>
  <input id="search" placeholder="Search..." />
  <div class="table-wrap">
   <table id="txnTable">
    <thead>
     <tr>
      <th>Transaction ID</th><th>Client Name</th><th>Card Number</th>
      <th>Client Phone</th><th>Amount</th><th>Bank Name</th>
      <th>Merchant Name</th><th>Transaction Date</th><th>Action</th><th></th>
     </tr>
    </thead>
    <tbody>
     {% for t in transactions %}
     <tr data-id="{{t.id}}">
      <td>{{t.id}}</td>
      <td>{{t.client_name}}</td>
      <td>{{t.card_number}}</td>
      <td><input class="phone" value="{{t.client_phone}}" /></td>
      <td>{{"%.2f"|format(t.amount)}}</td>
      <td>{{t.bank_name}}</td>
      <td>{{t.merchant_name}}</td>
      <td>{{t.transaction_date}}</td>
      <td><span class="badge" data-action="{{t.action}}">{{t.action}}</span></td>
      <td>
       <button class="call">Call</button>
       <select class="quick">
        <option value="">Quick set</option>
        <option>Resolved</option>
        <option>Connecting...</option>
        <option>Marked as Fraud</option>
        <option>Not Answered</option>
        <option>Disconnected</option>
       </select>
      </td>
     </tr>
     {% endfor %}
```

```
      </tbody>
     </table>
   </div>
  </div>
  <script src="/static/js/main.js"></script>
</body>
</html>
```

---

7) Add basic JS (edit phone, call, quick set, polling)

static/js/main.js:

```
const $ = (s, d=document)=>d.querySelector(s);
const $$ = (s, d=document)=>Array.from(d.querySelectorAll(s));

async function post(url, data){
  return fetch(url, {method:'POST', headers:{'Content-Type':'application/json'}, body: JSON.stringify(data||{})});
}

function attach() {
  // edit phone
  $$('.phone').forEach(inp=>{
    inp.addEventListener('change', async e=>{
      const tr = e.target.closest('tr');
      const id = tr.dataset.id;
      await post(`/update_phone/${id}`, {client_phone: e.target.value});
    });
  });
  });

  // quick action
  $$('.quick').forEach(sel=>{
    sel.addEventListener('change', async e=>{
      const tr = e.target.closest('tr'); const id = tr.dataset.id;
      if(!e.target.value) return;
      await post(`/set_action/${id}`, {action: e.target.value});
      refreshRow(id);
    });
  });

  // call button
  $$('.call').forEach(btn=>{
    btn.addEventListener('click', async e=>{
      const tr = e.target.closest('tr'); const id = tr.dataset.id;
      await post(`/call/${id}`);
      refreshRow(id);
    });
  });

  // search
  $('#search').addEventListener('input', e=>{
    const q = e.target.value.toLowerCase();
    $$('#txnTable tbody tr').forEach(r=>{
      r.style.display = r.innerText.toLowerCase().includes(q) ? '' : 'none';
    });
  });
}
```

```
async function refreshRow(id){
  const res = await fetch('/transactions'); const all = await res.json();
  const t = all.find(x=>x.id===id);
  if(!t) return;
  const tr = document.querySelector(`tr[data-id="${id}"]`);
  tr.querySelector('.phone').value = t.client_phone;
  const badge = tr.querySelector('.badge');
  badge.textContent = t.action; badge.dataset.action = t.action;
}

async function poll(){
  const res = await fetch('/transactions'); const all = await res.json();
  all.forEach(t=>refreshRow(t.id));
}
attach();
setInterval(poll, 5000);
```

---

8) Simple CSS

static/css/style.css:

```
body { font-family: system-ui, sans-serif; background:#fafafa; }
.container { padding: 20px; }
.table-wrap { overflow:auto; background:#fff; border-radius:12px; border:1px solid #eee;
}
table { width:100%; border-collapse:collapse; }
th, td { padding:12px 14px; border-bottom:1px solid #f0f0f0; font-size:14px; }
input.phone { width:170px; padding:6px 8px; }
button.call { padding:6px 10px; border:0; background:#4f46e5; color:#fff; border-
radius:8px; cursor:pointer; }
.badge { padding:6px 10px; border-radius:999px; background:#e5e7eb; }
```

```
.badge[data-action="Resolved"] { background:#bbf7d0; color:#065f46; }

.badge[data-action="Connecting..."] { background:#bfdbfe; color:#1e3a8a; }

.badge[data-action="Marked as Fraud"] { background:#fecaca; color:#7f1d1d; }

.badge[data-action="Not Answered"] { background:#ef4444; color:#fff; }

.badge[data-action="Disconnected"] { background:#fb923c; color:#fff; }

#search { margin:12px 0; padding:8px 10px; width:260px; }
```

---

9) Run it locally

```
flask --app app run  # http://127.0.0.1:5000
```

Open the dashboard and verify you see your 20 rows.

---

10) Expose webhooks for Twilio (dev)

```
# in another terminal

ngrok http 5000
```

- Copy the https://xxxxx.ngrok.io URL into your .env as PUBLIC_URL=...
- Restart Flask.

---

11) Configure Twilio number

- Buy/choose a Twilio number.
- You **don't** have to set a default Voice webhook on the number (we pass url when creating the call).
- Ensure outbound calls are allowed for your country.

---

12) Test a call end-to-end

- In the dashboard, edit a row's phone to your phone.
- Click **Call**.
- Answer → say **"Yes I did"**, **"No I did not"**, or **"This is fraud"**.
- Watch **Action** change within a few seconds.

---

13) Handle edge cases (already wired)

- **No answer / busy / failed** → /status/<id> sets **Not Answered**.
- **Silence / weird input** → /gather/<id> defaults to **Disconnected** or **Not Answered**.

---

14) (Optional) Persist beyond JSON

- Swap JSON for SQLite + SQLAlchemy (same fields).
- Replace read_txns/write_txns with DB queries.

---

15) (Optional) Hardening

- Add basic auth for the dashboard (Flask-Login).
- Validate/format phone numbers (use phonenumbers lib).
- Rate-limit endpoints (Flask-Limiter).

---

16) (Optional) Better NLP

Replace classify_action with an LLM call to classify free-form text into:
Resolved | Connecting... | Marked as Fraud | Not Answered | Disconnected.

---

17) Deploy

- Render/Railway/Azure Web App/AWS Lightsail.
- Set environment variables there (TWILIO_*, PUBLIC_URL).
- Update PUBLIC_URL to your deployed HTTPS domain.

You're set!

Follow steps 1→12 and you'll have a working Flask IVR dashboard that looks like your reference and updates actions from real calls.

Want me to zip these files into a starter project (so you can just download and run)?