Gateway v2.0

A two-part system:

- Frontend (Vite + React + Tailwind) for the UI and chat-driven controls
- Backend (Node.js + Express) as a controller/collector that converts natural language into actions on a Linux gateway over SSH

This README is split into two parts:

- Part A — App Usage (how to install, run, and use the app)
- Part B — Architecture & Gateway Write-up (the detailed network and system design you provided)

Part A — App Usage

Prerequisites

- Node.js 18 or newer
- Git (optional, for cloning)
- The backend and frontend run locally by default:
  - Backend: http://127.0.0.1:8081
  - Frontend: http://127.0.0.1:5173

1) Install & Run

From the repository root, install dependencies for both apps:

```
# Backend
cd backend
npm install
npm run dev   # starts Express on 127.0.0.1:8081

# In another terminal — Frontend
cd ../frontend
npm install
npm run dev   # starts Vite dev server on 5173
```

The frontend expects the API at `/api` and in dev mode Vite should proxy to `http://127.0.0.1:8081`. If you serve frontend directly without proxy, set `CORS_ORIGIN` on the backend to match your frontend origin.

2) Frontend overview

- Tech: React 18, Vite 5, Tailwind CSS, Framer Motion, Lottie
- Routes:
  - `/` Landing (two sections: Blocker and Guardian)

    - `/blocker` Chat for scheduling domain blocks
    - `/guardian` Chat for monitoring and mitigation insights
    - `/docs` FAQ (animated, searchable)
- Components are under `frontend/src/components` and pages under `frontend/src/pages`.

3) Backend overview

- Tech: Node.js (Express, Helmet, Morgan, node-fetch@2)
- Entrypoint: `backend/app.js`
- Endpoints (mounted under `/api`):
    - `GET /health` — health check
    - `POST /chat` — converts natural language into an intent, then performs action via SSH
    - `POST /direct-intent` — bypass intent and send an action directly
    - `GET /trace?domain=...` — diagnostics via SSH to the gateway

Environment variables (backend):

- `HOST` (default `127.0.0.1`)
- `PORT` (default `8081`)
- `CORS_ORIGIN` (default `http://127.0.0.1:5173`)
- SSH/gateway details are expected to be configured in your backend core (see `controllers` and `core` dirs in backend).

4) Typical flow

1. User submits a chat request from the frontend.
2. Backend `/api/chat` resolves it to an intent with action/target/duration.
3. Backend connects to the gateway via SSH and runs the proper script(s).
4. Results are streamed back to the UI and rendered.

5) Production build

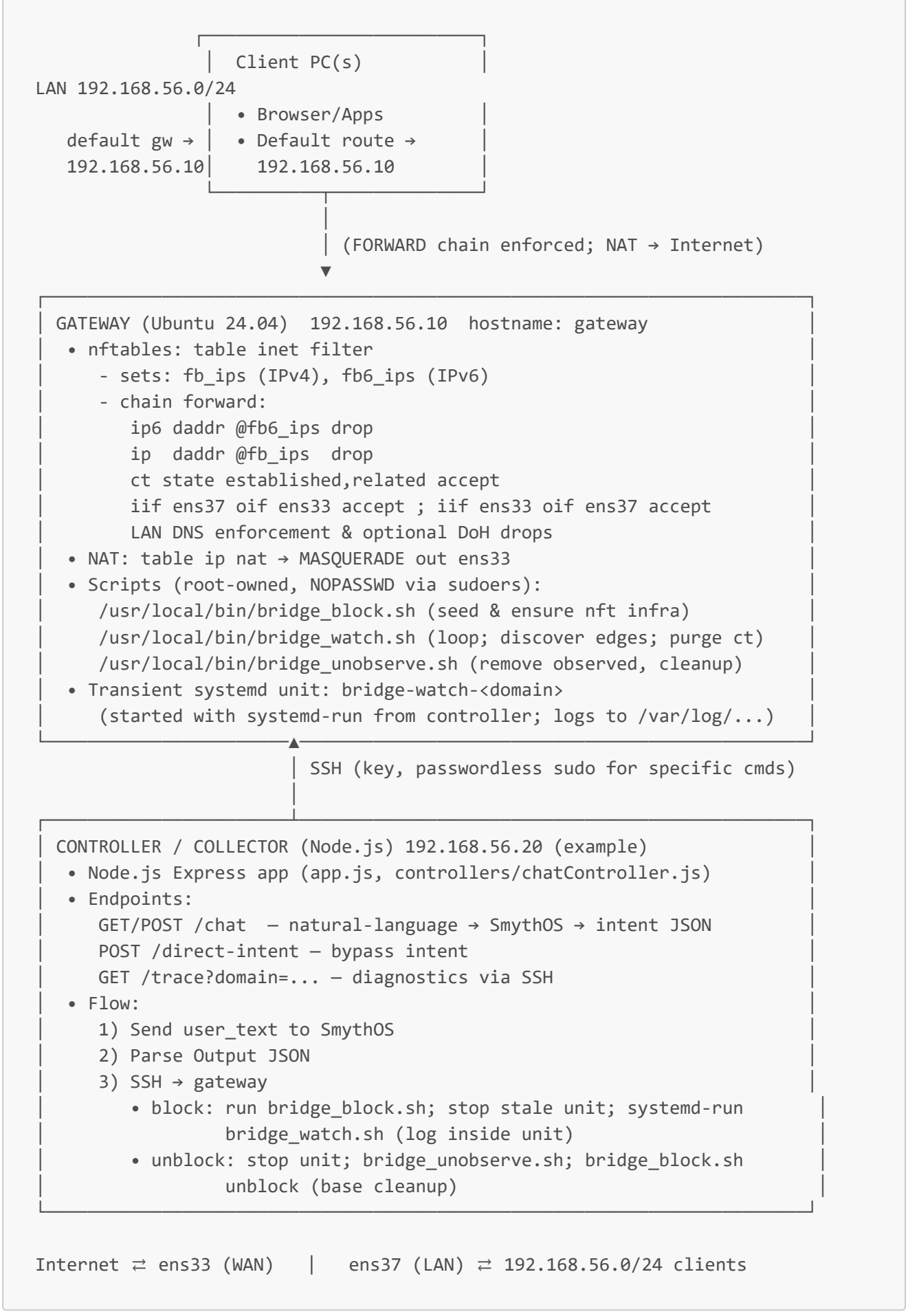
```
# Build frontend
cd frontend
npm run build
# Preview build locally (optional)
npm run preview
```

You can serve the built frontend with any static server and run the backend as a service (pm2/systemd).

Part B — Architecture & Gateway Write-up

Below is the detailed architecture and operational guide you provided, formatted for quick reference.

Architecture (what runs where)



Control/data flow (block case)

User says “block facebook for 2 hours” → POST /chat.

Controller calls SmythOS → gets `{action:"block", target:"facebook.com", duration:"2h"}`.

Controller SSH → gateway:

- a) `bridge_block.sh block domain facebook.com 2h` (ensure nft infra, seed current A/AAAA to sets).
- b) `systemctl stop bridge-watch-facebook-com || true`.
- c) `systemd-run --unit=bridge-watch-facebook-com /bin/bash -lc "bridge_watch.sh facebook.com ... >>/var/log/bridge-watch-facebook-com.log 2>&1"`

Watcher (every 10s): curl with `-w %{remote_ip}` to apex & www (v4+v6), add IPs (and optionally /24 wideners), purge conntrack for immediate cut, append to `/var/lib/bridge/observed_facebook.com.txt`.

FORWARD chain drops any packets to those IPs for LAN clients (v4 & v6).

Browser gives `CODE=000` (timeout).

Unblock: controller stops unit, unobserves cached IPs, runs base unblock (optional), sets empty → flows restored.

Files & configs (final working versions)

A) Gateway scripts

`/usr/local/bin/bridge_block.sh`

```
#!/usr/bin/env bash
set -euo pipefail

ACTION="${1:-}"; shift || true
TTYPE="${1:-}"; shift || true
TARGET="${1:-}"; shift || true
DURATION="${1:-}"

ensure_nft() {
  nft list table inet filter >/dev/null 2>&1 || nft add table inet filter
  nft list chain inet filter forward >/dev/null 2>&1 || \
    nft add chain inet filter forward '{ type filter hook forward priority 0;
policy accept; }'

  nft list set inet filter fb_ips  >/dev/null 2>&1 || \
    nft add set inet filter fb_ips  '{ type ipv4_addr; flags interval; timeout
2h; }'
  nft list set inet filter fb6_ips >/dev/null 2>&1 || \
    nft add set inet filter fb6_ips '{ type ipv6_addr; flags interval; timeout
2h; }'

  # forward drops for sets (idempotent)
```

```
  nft list chain inet filter forward | grep -q 'ip daddr @fb_ips drop'  || \
    nft insert rule inet filter forward ip daddr @fb_ips drop
  nft list chain inet filter forward | grep -q 'ip6 daddr @fb6_ips drop' || \
    nft insert rule inet filter forward ip6 daddr @fb6_ips drop

  # NOTE: no OUTPUT-chain drops (watcher must curl out)
}

resolve_v4() { getent ahostsv4 "$1" 2>/dev/null | awk '{print $1}' | sort -u; }
resolve_v6() { getent ahostsv6 "$1" 2>/dev/null | awk '{print $1}' | sort -u; }

block_domain() {
  local domain="$1" dur="$2"
  ensure_nft
  mapfile -t V4 < <(resolve_v4 "$domain" || true)
  mapfile -t V6 < <(resolve_v6 "$domain" || true)
  if [[ ${#V4[@]} -eq 0 && ${#V6[@]} -eq 0 ]]; then
    echo "WARN: no IPs resolved for ${domain}" >&2
  fi
  for ip in "${V4[@]:-}"; do nft add element inet filter fb_ips  "{ ${ip}
timeout ${dur} }" 2>/dev/null || true; done
  for ip in "${V6[@]:-}"; do nft add element inet filter fb6_ips "{ ${ip}
timeout ${dur} }" 2>/dev/null || true; done
  echo "Blocked ${domain} for ${dur}."
}

unblock_domain() {
  local domain="$1"
  mapfile -t V4 < <(resolve_v4 "$domain" || true)
  mapfile -t V6 < <(resolve_v6 "$domain" || true)
  for ip in "${V4[@]:-}"; do nft delete element inet filter fb_ips  "{ ${ip} }"
2>/dev/null || true; done
  for ip in "${V6[@]:-}"; do nft delete element inet filter fb6_ips "{ ${ip} }"
2>/dev/null || true; done
  echo "Unblock requested for ${domain}."
}

main() {
  case "${ACTION}:${TTYPE}" in
    block:domain)
      [[ -n "${TARGET}" ]] || { echo "missing domain" >&2; exit 2; }
      [[ -n "${DURATION}" ]] || DURATION="2h"
      [[ "${DURATION}" =~ ^[1-9][0-9]*(m|h|d)$ ]] || DURATION="2h"
      block_domain "${TARGET}" "${DURATION}"
      ;;
    unblock:domain)
      [[ -n "${TARGET}" ]] || { echo "missing domain" >&2; exit 2; }
      ensure_nft
      unblock_domain "${TARGET}"
      ;;
    *)
      echo "Usage: $0 <block|unblock> domain <name> [duration]" >&2
      exit 2
```

```
      ;;
  esac
}
main "$@"
```

/usr/local/bin/bridge_watch.sh

```
#!/usr/bin/env bash
set -u -o pipefail

domain="${1:?domain}"
window="${2:-7200}"
interval="${3:-10}"
ttl="${4:-10m}"

end=$(( $(date +%s) + window ))
cache="/var/lib/bridge/observed_${domain}.txt"
mkdir -p /var/lib/bridge
touch "$cache"

NFT=/usr/sbin/nft
CT=/usr/sbin/conntrack

log() { printf '[%s] %s\n' "$(date -Is)" "$*"; }

add_v4() {
  local ip="$1"
  $NFT add element inet filter fb_ips "{ $ip timeout $ttl }" 2>/dev/null ||
true
  echo "$ip v4 $(date -Is)" >> "$cache"
  $CT -D -d "$ip" 2>/dev/null || true
  $CT -D -q -d "$ip" 2>/dev/null || true
  log "v4 add $ip  ttl=$ttl"
}

add_v6() {
  local ip="$1"
  $NFT add element inet filter fb6_ips "{ $ip timeout $ttl }" 2>/dev/null ||
true
  echo "$ip v6 $(date -Is)" >> "$cache"
  $CT -D -f ipv6 -d "$ip" 2>/dev/null || true
  log "v6 add $ip  ttl=$ttl"
}

log "watch start domain=$domain window=${window}s interval=${interval}s
ttl=$ttl"
while [ "$(date +%s)" -lt "$end" ]; do
  ip4=$(curl -4 -sS -o /dev/null -w '%{remote_ip}' "https://${domain}" -m 6 ||
true)
  [ -n "${ip4:-}" ] && add_v4 "$ip4"
```

```
  ip4w=$(curl -4 -sS -o /dev/null -w '%{remote_ip}' "https://www.${domain}" -m
6 || true)
  [ -n "${ip4w:-}" ] && add_v4 "$ip4w"

  ip6=$(curl -6 -sS -o /dev/null -w '%{remote_ip}' "https://${domain}" -m 6 ||
true)
  [ -n "${ip6:-}" ] && add_v6 "$ip6"

  ip6w=$(curl -6 -sS -o /dev/null -w '%{remote_ip}' "https://www.${domain}" -m
6 || true)
  [ -n "${ip6w:-}" ] && add_v6 "$ip6w"

  sleep "$interval"
done
log "watch end"
exit 0
```

/usr/local/bin/bridge_unobserve.sh

```
#!/usr/bin/env bash
set -u -o pipefail
domain="${1:?domain}"
cache="/var/lib/bridge/observed_${domain}.txt"

# delete elements listed in cache (first field is the IP/prefix)
if [ -f "$cache" ]; then
  awk '{print $1}' "$cache" | while read -r x; do
    [ -z "$x" ] && continue
    /usr/sbin/nft delete element inet filter fb_ips  "{ $x }" 2>/dev/null ||
true
    /usr/sbin/nft delete element inet filter fb6_ips "{ $x }" 2>/dev/null ||
true
    # if you later store widened v4 (/24), this will remove it too
    cidr="$(echo "$x" | sed -n 's/^\([0-9]\+\.[0-9]\+\.[0-9]\+\)\.[0-
9]\+/\1.0\/24/p')"
    [ -n "${cidr:-}" ] && /usr/sbin/nft delete element inet filter fb_ips "{
$cidr }" 2>/dev/null || true
  done
  rm -f "$cache"
fi

# clean temp sets if they exist
/usr/sbin/nft list set inet filter fb_ips_temp  >/dev/null 2>&1 &&
/usr/sbin/nft flush set inet filter fb_ips_temp  || true
/usr/sbin/nft list set inet filter fb6_ips_temp >/dev/null 2>&1 &&
/usr/sbin/nft flush set inet filter fb6_ips_temp || true

echo "unobserved ${domain}"
```

Permissions (gateway):

```
sudo chmod 0755 /usr/local/bin/bridge_*.sh
```

Sudoers (gateway): `/etc/sudoers.d/bridge_scripts`

```
lab ALL=(root) NOPASSWD: /usr/local/bin/bridge_block.sh
lab ALL=(root) NOPASSWD: /usr/local/bin/bridge_watch.sh
lab ALL=(root) NOPASSWD: /usr/local/bin/bridge_unobserve.sh
lab ALL=(root) NOPASSWD: /usr/bin/systemd-run, /usr/bin/systemctl
Defaults!(/usr/bin/systemd-run) !requiretty
```

Validate:

```
sudo visudo -c
```

Forwarding (gateway): `/etc/sysctl.d/99-forwarding.conf`

```
net.ipv4.ip_forward=1
net.ipv6.conf.all.forwarding=1
```

Apply:

```
sudo sysctl --system
```

Persist nftables (gateway):

```
sudo bash -lc 'nft list ruleset > /etc/nftables.conf && systemctl enable --now
nftables'
```

Controller service (Node.js)

- `controllers/chatController.js` — use the patched builder shown above (replace `buildRemoteCmdFromIntent`).
- `app.js` — your existing version is fine; `/trace` pulls unit status, forward flags, NAT/filter chains, set contents, watcher log tail, and quick curl probes.
- Dependencies: Node.js ≥ 18; `express`, `helmet`, `morgan`, `node-fetch@2`.

From-scratch setup (step-by-step)

1. Network plan

- Gateway (Ubuntu 24.04): LAN `ens37` (192.168.56.10/24), WAN `ens33` (DHCP)
- Controller (Node.js): 192.168.56.20 (example)
- Client(s): 192.168.56.x; default gw = 192.168.56.10

2. Gateway provisioning (192.168.56.10)

- Install packages: `nftables conntrack curl jq`
- Enable nftables; set forwarding flags (IPv4 + IPv6)
- Configure NAT (ip nat table postrouting) and filter table/sets/forward chain rules
- Optional: DNS enforcement & DoH drop rules
- Persist nftables rules to `/etc/nftables.conf`
- Install the 3 scripts and sudoers file; verify with `visudo -c`

3. Controller provisioning (192.168.56.20)

- Install Node ≥18 and basic tools
- Create the app directory and drop in your working `app.js` and `controllers/chatController.js` (with the patched builder)
- Install dependencies and start the app

SSH access from controller → gateway:

- Controller user `lab` has private key at `/home/lab/.ssh/sre_agent_key` (chmod 600)
- Authorized key exists on gateway for user `lab`
- Use `ssh -i /home/lab/.ssh/sre_agent_key -o BatchMode=yes -o StrictHostKeyChecking=accept-new lab@192.168.56.10`

Sanity checks:

```
curl http://127.0.0.1:8081/health
curl 'http://127.0.0.1:8081/trace?domain=facebook.com'
```

4. Client PC(s)

- Default route must point to the gateway: `192.168.56.10`
- Disable VPN/split-tunnel during tests
- Quick test:

```
curl -sS -o /dev/null -w 'CODE=%{http_code}\n' https://www.facebook.com -m 6
```

Operations (normal use)

- Block (NL input via controller):

```
curl -X POST 'http://127.0.0.1:8081/chat' -d 'message=block facebook for 2
hours'
```

- Unblock:

```
curl -X POST 'http://127.0.0.1:8081/chat' -d 'message=unblock facebook'
```

- Diagnostics:

```
curl 'http://127.0.0.1:8081/trace?domain=facebook.com' | jq -r '.stdout'
```

Manual verification on gateway

```
systemctl status bridge-watch-facebook-com --no-pager || true
sudo tail -n 80 /var/log/bridge-watch-facebook-com.log || true
sudo nft list set inet filter fb_ips  | sed -n '1,120p'
sudo nft list set inet filter fb6_ips | sed -n '1,120p'
```

After reboot (what persists / what to re-run)

- nftables rules persist via `/etc/nftables.conf` + `systemctl enable nftables`
- Forwarding flags persist via `/etc/sysctl.d/99-forwarding.conf`
- Scripts persist in `/usr/local/bin`
- Watcher is a transient unit; it will not auto-restart after reboot (by design)
- To re-block, call the controller again (`POST /chat` with your command)

Full “clean-room” quickstart

Gateway (Ubuntu 24.04):

```
apt-get update
apt-get install -y nftables conntrack curl jq
systemctl enable --now nftables
echo -e "net.ipv4.ip_forward=1\nnet.ipv6.conf.all.forwarding=1" >
/etc/sysctl.d/99-forwarding.conf
sysctl --system
nft add table ip nat
nft add chain ip nat postrouting '{ type nat hook postrouting priority srcnat;
policy accept; }'
nft add rule ip nat postrouting oif "ens33" ip saddr 192.168.56.0/24 masquerade
nft add table inet filter
nft add set inet filter fb_ips  '{ type ipv4_addr; flags interval,timeout;
```

```
timeout 2h; }'
nft add set inet filter fb6_ips '{ type ipv6_addr; flags interval,timeout;
timeout 2h; }'
nft add chain inet filter forward '{ type filter hook forward priority filter;
policy accept; }'
nft add rule inet filter forward ip6 daddr @fb6_ips drop
nft add rule inet filter forward ip  daddr @fb_ips  drop
nft add rule inet filter forward ct state established,related accept
nft add rule inet filter forward iif "ens37" oif "ens33" accept
nft add rule inet filter forward iif "ens33" oif "ens37" accept
nft add rule inet filter forward ip saddr 192.168.56.0/24 udp dport 53 ip daddr
!= 192.168.56.10 drop
nft add rule inet filter forward ip saddr 192.168.56.0/24 tcp dport 53 ip daddr
!= 192.168.56.10 drop
# (optional DoH rule using a @doh_ips set)
# install the 3 scripts (as above), chmod 0755
# write /etc/sudoers.d/bridge_scripts (as above), visudo -c
nft list ruleset > /etc/nftables.conf
```

Controller (Node.js):

```
# install Node ≥18 and deps
npm i express helmet morgan node-fetch@2
# drop in app.js + controllers/chatController.js (patched builder)
node app.js
```

Client:

```
# ensure default route → 192.168.56.10
curl -sS -o /dev/null -w 'CODE=%{http_code}\n' https://www.facebook.com -m 6
```

Troubleshooting (quick map)

- Watcher service failing immediately?
  - Ensure OUTPUT chain has no fb drops.
  - Ensure sudoers allows systemd-run and systemctl (NOPASSWD).
  - Ensure `/usr/local/bin/bridge_watch.sh` tolerates curl failures (no `set -e`).
- Browser still loads briefly after block?
  - Ensure watcher logs show new edge IPs being added.
  - Purge conntrack for those IPs (v4 + QUIC and v6).
- Unblock leaves stale IPs?
  - Check `/var/lib/bridge/observed_<domain>.txt` existed; update `bridge_unobserve.sh`.

    - Manually delete set elements or let TTL expire.

- Clients unaffected?
    - Client default route must be the gateway.
    - No VPN/split tunnel bypassing LAN.