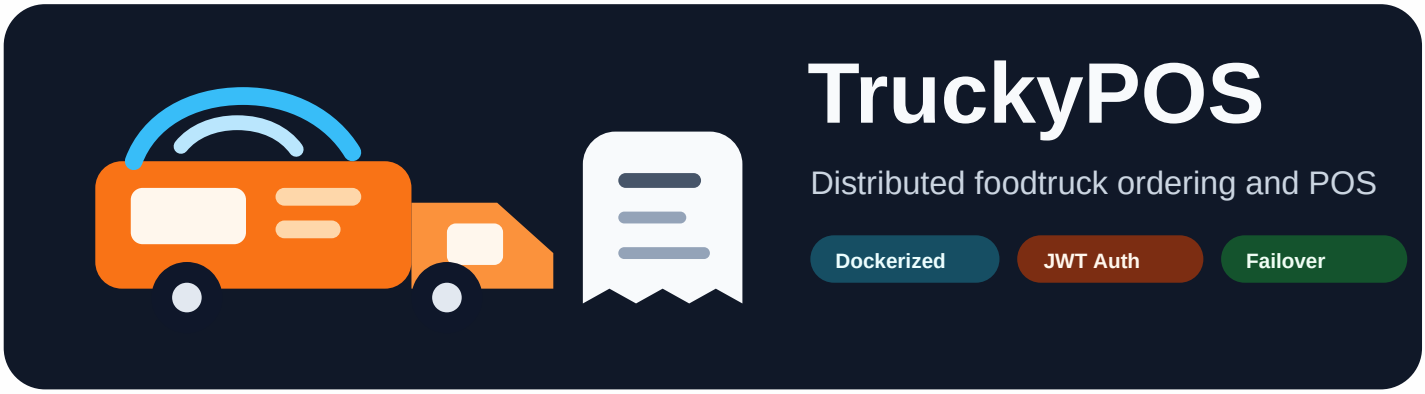

TruckyPOS

TruckyPOS is a Dockerized foodtruck point-of-sale and ordering system. It serves a customer order page, POS/kitchen/admin views, a public pickup display, JWT authentication, persistent storage, and a load-balanced backend with two API instances for failover demos.

The main goal of the project is to provide foodtruck operators with the simplest possible POS setup that still covers the required operational needs: taking customer orders, handling kitchen and pickup workflows, managing menu and inventory data, supporting staff/admin access, and offering a practical public ordering option through QR codes.

The project was built for the FS 2026 distributed systems challenge task, but the deployment scripts are structured so a foodtruck operator can run it on a Raspberry Pi, Linux laptop, or small local server.

Quick Install

Use this one-liner on a fresh apt-based machine such as Raspberry Pi OS, Debian, or Ubuntu when the goal is to get a running TruckyPOS instance with as little manual setup as possible:

```
curl -fsSL https://raw.githubusercontent.com/GoniMcColly/truckyPOS/main/bootstrap.
```

The bootstrap script installs missing host dependencies, including Git and Docker, clones or updates the repository, runs the installer, starts the Docker Compose stack, and prints a public `trycloudflare.com` URL for QR-code ordering. The `public-qr` mode uses Cloudflare Quick Tunnel, a free Cloudflare service that does not require a Cloudflare account or domain.

If Git and Docker are already installed, prefer the normal repository workflow:

```
git pull
./install.sh -public-qr
```

Default local login:

```
Username: admin
Password: password
```

The installer lets you set a new admin password during setup. Change the default before using the system outside a local demo.

Requirements

For normal operation the host only needs:

```
Docker Engine
Docker Compose v2
Git
Bash
```

The application services are provided by Docker images. You do not need to install Go, Node.js, PostgreSQL, Caddy, or cloudflared manually.

`bootstrap.sh` currently supports apt-based Linux distributions and installs Git, Curl, and Docker when missing. If Git and Docker are already installed, or on other operating systems, use the repository directly with `git pull` and `./install.sh`.

Install Modes

Mode	Command	Use case
Local only	`./install.sh -local`	Staff devices use `http://localhost` or the local network. No public URL.
Public QR	`./install.sh -public-qr`	Creates a temporary public `trycloudflare.com` URL for QR-code ordering through Cloudflare Quick Tunnel. This currently uses Cloudflare's free quick tunnel service and does not require a Cloudflare account or domain.
Public domain	`./install.sh -public-domain`	Uses a named Cloudflare Tunnel with a stable custom domain or subdomain. Requires a Cloudflare tunnel token.

Fresh apt-based machines can use the same modes through bootstrap:

```
curl -fsSL https://raw.githubusercontent.com/GoniMcColly/truckyPOS/main/bootstrap.
curl -fsSL https://raw.githubusercontent.com/GoniMcColly/truckyPOS/main/bootstrap.
curl -fsSL https://raw.githubusercontent.com/GoniMcColly/truckyPOS/main/bootstrap.
```

If the repository is already cloned and Git/Docker are installed, run:

```
git pull
./install.sh -local
./install.sh -public-qr
./install.sh -public-domain
```

Public QR Mode

`public-qr` is the simplest mode for small operators who want to print or display a QR code

and let customers order without joining the foodtruck WLAN.

Start:

```
./install.sh -public-qr
```

Show the current public URL again:

```
./install.sh -public-url
```

Cloudflare Quick Tunnel limitations:

```
Random URL changes when the tunnel restarts
200 concurrent request limit
No Server-Sent Events support
Not intended as a production replacement for a named tunnel
```

TruckyPOS uses polling for live updates, so the missing SSE support does not affect the current pickup/order update flow.

Public Domain Mode

Use this mode when the operator wants a stable URL such as:

```
https://order.example-foodtruck.ch
```

Create a Cloudflare Tunnel in Cloudflare Zero Trust, configure the public hostname, and point the tunnel service to:

```
http://caddy:80
```

Then run:

```
./install.sh -public-domain
```

The installer asks for `CLOUDFLARED_TOKEN` only in this mode.

Manual Docker Usage

Local stack:

```
docker compose up -d --build
```

Public QR stack:

```
docker compose --profile public-qr up -d --build
./install.sh -public-url
```

Public domain stack:

```
docker compose --profile public-domain up -d --build
```

Stop without deleting data:

```
docker compose --profile public-qr --profile public-domain down --remove-orphans
```

Remove through the interactive helper:

```
./uninstall.sh
```

`uninstall.sh` asks before deleting the database volume, locally built images, or the `.env` file.

Architecture

The default Compose stack contains:

```
Caddy
  Static frontend server and API load balancer

api-1 / api-2
  Two identical Go API instances

PostgreSQL
  Persistent database for users, refresh-token sessions, menu, inventory,
  orders, and order items

cloudflared-qr / cloudflared-domain
  Optional public access through Cloudflare
```

Key properties:

- frontend: Svelte/Vite single-page application
- backend: Go `net/http`
- auth: short-lived JWT access token plus HttpOnly refresh-token cookie
- storage: PostgreSQL with Docker volume persistence
- load balancing: Caddy round-robin over two API instances
- failover: stopping one API instance should not interrupt authenticated usage
- live updates: polling instead of SSE, which works with Cloudflare Quick Tunnel

Detailed diagrams are available in docs/c4-architecture.md.

Application Areas

Area	Access	Purpose
`/`	Public	Customer ordering page
`/pickup`	Public	Pickup display for ready/preparing orders
`/login`	Public	Staff/admin login
`/pos`	Authenticated	Point-of-sale workflow
`/kitchen`	Authenticated	Kitchen order handling
`/dashboard`	Authenticated	Operational overview
`/admin`	Admin only	Menu, inventory, and employee user management

Public API endpoints include menu loading, order creation, order status, and ready-order pickup data. Admin-only endpoints are enforced in the backend, not only hidden in the frontend.

Configuration

The installer writes a local `.env` file. For manual setup, copy the example:

```
cp .env.example .env
```

Important variables:

Variable	Purpose
`POSTGRES_DB`	Database name
`POSTGRES_USER`	Database user
`POSTGRES_PASSWORD`	Database password
`JWT_SECRET`	HMAC secret for JWT access tokens
`AUTH_COOKIE_SECURE`	Set `true` for public HTTPS access through Cloudflare
`CORS_ALLOWED_ORIGIN`	Allowed browser origin for API requests
`CLOUDFLARED_TOKEN`	Cloudflare named tunnel token for `public-domain` mode

Do not commit `.env`. It contains local secrets.

Development

Run backend tests:

```
cd backend
go test ./...
```

Run frontend tests and build:

```
cd frontend
npm test
npm run build
```

Validate Compose:

```
docker compose config --quiet
```

Load Testing

The project includes a load-test helper for a JWT-protected endpoint:

```
./scripts/load-test.sh
```

Current documented results are in docs/load-test.md.

Failover Demo

The authentication/failover demo script forces login through one API instance, then refreshes the session through the other instance:

```
./scripts/failover-auth-test.sh
```

This demonstrates that refresh-token state is stored in PostgreSQL and survives backend instance changes.

Documentation

- Cloudflare public access
- C4 architecture diagrams
- Load-test results