Step‑by‑Step: Installation von Backend & Frontend (Asia Restaurant)

Dieses Dokument führt dich detailliert durch die Installation und Inbetriebnahme von Backend (FastAPI) und Frontend (Vite/React) für restaurant.bibabau.ch – sowohl für lokale Entwicklung als auch für den Produktivbetrieb mit Docker Compose. Mail/Backups/Nginx sind hier nicht Thema; der Fokus liegt auf API & Web‑App.

# 0) Voraussetzungen

• Debian 13 Server mit SSH und sudo (für Produktion).

• Node.js ≥ 18.x und npm (für lokale Frontend‑Entwicklung).

• Docker & Docker Compose (für Produktion oder containerisierte Entwicklung).

• Git (zum Übertragen/Versionieren des Projekts).

# A) Backend (FastAPI) installieren

Du kannst das Backend entweder lokal (Python venv) oder direkt mit Docker betreiben. Empfohlen wird Docker/Compose.

## A1) Projektstruktur vorbereiten

Wechsle in dein Projektverzeichnis und stelle sicher, dass der Ordner /api die folgenden Dateien enthält:

• main.py – FastAPI App mit Endpunkten

• models.py – SQLAlchemy Modelle (User, Order, etc.)

• database.py – DB‑Engine/Session

• utils.py – Zeitzone/Zeitfunktionen

• requirements.txt – Python‑Abhängigkeiten

cd ~/asia-restaurant  
ls api/

## A2) Umgebungsvariablen (.env)

Lege im Projektroot eine .env mit DB‑URL & API‑Parametern an (für Dev/Prod identisch nutzbar):

TZ=Europe/Zurich  
API\_HOST=0.0.0.0  
API\_PORT=8000  
  
# PostgreSQL (für Dev kannst du SQLite verwenden, dann DB\_URL auf sqlite setzen)  
POSTGRES\_USER=asiar  
POSTGRES\_PASSWORD=STRONG\_DB\_PASSWORD  
POSTGRES\_DB=asiar  
POSTGRES\_HOST=db  
POSTGRES\_PORT=5432  
DB\_URL=postgresql+psycopg://asiar:STRONG\_DB\_PASSWORD@db:5432/asiar

## A3) Backend lokal (Option 1: Python venv)

Für einen schnellen lokalen Start ohne Docker (z. B. wenn Postgres extern läuft):

sudo apt -y install python3 python3-venv  
cd ~/asia-restaurant/api  
python3 -m venv .venv  
source .venv/bin/activate  
pip install --upgrade pip  
pip install -r requirements.txt  
  
# Starte die API (liest DB\_URL aus Umgebung, ggf. exportieren)  
export DB\_URL="sqlite:///./app.db" # Alternative: deine Postgres‑URL  
uvicorn main:app --host 0.0.0.0 --port 8000 --reload

• Aufruf im Browser: http://localhost:8000/docs (Swagger UI).

## A4) Backend mit Docker (Option 2: empfohlen)

Baue das Backend‑Image und starte den Container (alle Variablen kommen aus .env):

# Im Projektroot  
docker compose -f docker-compose.yml up -d --build api  
  
# Prüfen  
curl http://localhost:8000/

Wenn du Postgres in Compose nutzt, stelle sicher, dass der Dienst db definiert und gestartet ist.

## A5) Datenbank initialisieren & Demodaten

Die Tabellen werden beim Start per SQLAlchemy erzeugt (Base.metadata.create\_all). Für Demodaten:

# Menü-Demo-Daten anlegen  
curl -X POST http://localhost:8000/dev/seed  
  
# Liste der Menü-Items  
curl http://localhost:8000/menu/items

## A6) Auth testen (optional, falls in API aktiviert)

# Registrierung  
curl -X POST http://localhost:8000/auth/register -H "Content-Type: application/json" -d '{"email":"test@example.com","password":"secret","name":"Test"}'  
  
# Login  
curl -X POST http://localhost:8000/auth/login -H "Content-Type: application/json" -d '{"email":"test@example.com","password":"secret"}'

Der Login liefert ein JWT‑Token. Dieses im Frontend als Bearer Token verwenden.

## A7) Bestellung anlegen (Smoke‑Test)

# Beispiel-Order (Pickup in 45 Minuten)  
NOW=$(date -u +"%Y-%m-%dT%H:%M:%SZ")  
IN45=$(date -u -d "+45 minutes" +"%Y-%m-%dT%H:%M:%SZ")  
curl -X POST http://localhost:8000/orders -H "Content-Type: application/json" -d "{  
 "items": [{"menu\_item\_id": 1, "qty": 2}],   
 "pickup\_at": "${IN45}",  
 "notes": "ohne Koriander"  
}"

# B) Frontend (Vite/React) installieren

Das Frontend ist eine Single‑Page‑App, die per fetch gegen die API spricht. Für lokale Entwicklung nutzt du den Vite Dev‑Server; für Produktion wird gebaut und via Nginx ausgeliefert.

## B1) Node/NPM prüfen & installieren

node -v  
npm -v  
# Falls nicht vorhanden: sudo apt -y install nodejs npm (oder via NodeSource für neuere Versionen)

## B2) Abhängigkeiten installieren

cd ~/asia-restaurant/frontend  
npm install

## B3) Frontend‑Konfiguration (.env für Vite)

Für die lokale Entwicklung setze die API‑URL auf http://localhost:8000 und WS auf ws://localhost:8000:

# Datei: frontend/.env.local  
VITE\_API\_URL=http://localhost:8000  
VITE\_WS\_URL=ws://localhost:8000  
VITE\_MATOMO\_URL=  
VITE\_MATOMO\_SITE\_ID=

## B4) Dev‑Server starten (lokal)

npm run dev  
# Ausgabe zeigt die lokale URL, z. B. http://localhost:5173  
# Öffne diese URL im Browser.

## B5) API‑Konnektivität testen

• Öffne die Seite „Menü“ und lade ggf. Demodaten per Button (dev/seed).

• Lege 1–2 Artikel in den Warenkorb, gehe zu „Checkout“, stelle Abholzeit ≥ +30 Min ein, Bestellen.

• Erfolgsmeldung erscheint, Bestellung ist in der API sichtbar (und in der Kitchen‑UI).

## B6) Produktionsbuild erstellen

# im Ordner frontend  
npm run build  
# erzeugt das Verzeichnis dist/ mit statischen Dateien

Für die Produktion im Compose‑Setup wird der Build über das Dockerfile.prod erledigt und in Nginx eingebunden (web‑Service).

# C) Kurz: Produktion mit Docker Compose

Wenn Backend/Frontend sauber lokal funktionieren, kannst du beides in Compose als Produktion starten (Nginx/TLS sind hier ausgeklammert):

# Backend im Prod-Compose starten  
docker compose -f docker-compose.yml up -d --build api  
  
# Frontend dev (separat) oder via prod-Build + Nginx (siehe Gesamt-Guide)  
docker compose -f docker-compose.yml up -d frontend

Im vollständigen Produktions‑Stack (siehe Gesamt‑Handbuch) werden Frontend‑Dateien gebaut und über Nginx unter / ausgeliefert; die API liegt unter /api/.

# D) Troubleshooting (Backend/Frontend)

• API 404/500: Logs prüfen:

docker compose logs -f api  
# oder bei venv: im uvicorn-Output nach Tracebacks schauen

• CORS‑Probleme lokal: Stelle sicher, dass die API CORS erlaubt, wenn Frontend/Backend auf verschiedenen Hosts/Ports laufen (FastAPI CORSMiddleware).

• WebSocket nicht erreichbar: Prüfe VITE\_WS\_URL (ws://localhost:8000 im Dev).

• Build‑Fehler Frontend: Node‑Version ≥ 18, npm cache leeren: npm ci / npm cache verify.