LInI911 / **MyTrip** 🔒

Code Issues Pull requests Agents Actions Projects Wiki Security Insights Settii

Watch 0

🇨🇭 A distributed trip planner powered by Swiss public transport data via the Open Journey Planner (OJP) API. Features JWT auth, load balancing, and service failover.

MIT license

0 stars **0** forks **0** watching Branches Activity
Tags

Private repository

... **1** Branch **0** Tags Go to file t Go to file Add file Code

LInI911 Bump Python to 3.14 and add token refresh 546fbe4 · now

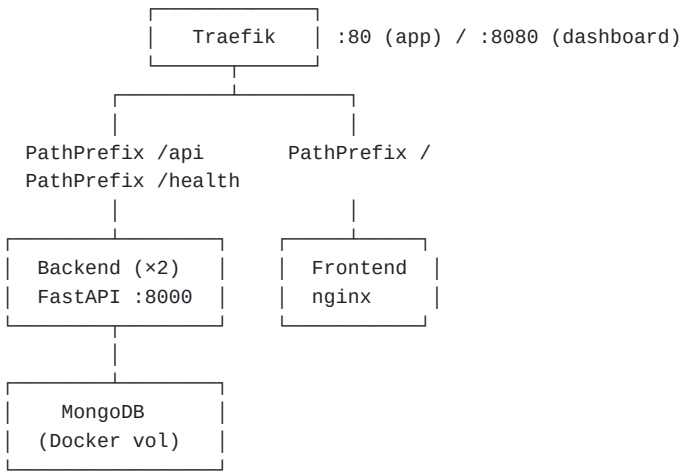
backend	Bump Python to 3.14 and add token refr...	now
frontend	Add JWT refresh token support	1 hour ago
k6	Add load testing via hey and k6	last week
.dockerignore	Initial commit	last month
.env.example	Add text search for origin and destination	2 weeks ago
.gitignore	Upgrade docker images and python de...	2 hours ago
LICENSE	Initial commit	last month
Makefile	Add load testing via hey and k6	last week
README.md	Bump Python to 3.14 and add token refr...	now
docker-compose.yml	Upgrade docker images and python de...	2 hours ago
pyrightconfig.json	Add Pyright configuration for Python 3....	last month
traefik.dynamic.yml	Bump Python to 3.14 and add token refr...	now

README MIT license

MyTrip — Distributed Trip Planner

A distributed trip planner powered by the Swiss OJP 2.0 API. The backend runs as multiple FastAPI replicas behind a Traefik reverse proxy, uses MongoDB for persistence, and serves a plain HTML/JS frontend — all launched with a single command.

Architecture



Service	Image / Build	Purpose
traefik	`traefik:v3.3`	Reverse proxy & load balancer (file provider)
backend	`./backend` (Dockerfile with `uv`)	FastAPI REST API — auth, trips, health
frontend	`nginx:alpine`	Static HTML/JS served as-is
mongo	`mongo:8`	User storage, persistent Docker volume

Quick Start

```
# 1. Clone
git clone <repo-url> && cd MyTrip

# 2. Configure
cp .env.example .env
# Edit .env — at minimum set JWT_SECRET and OJP_API_KEY

# 3. Run
make up
# or: docker compose up --build
```

Open **http://localhost** for the frontend, **http://localhost:8080** for the Traefik dashboard.

Environment Variables

Variable	Required	Default	Description
`JWT_SECRET`	Yes	`dev_secret`	Shared secret for signing access JWTs
`JWT_REFRESH_SECRET`	No	`dev_refresh_secret`	Shared secret for signing refresh tokens
`OJP_API_KEY`	Yes	—	Bearer token for the OJP API
`MONGO_URL`	No	`mongodb://mongo:27017/tripdb`	MongoDB connection string
`ADMIN_PASSWORD`	No	`admin123`	Password for the seeded `admin` user

API

Method	Path	Auth	Description
`GET`	`/health`	—	Health check (`{"status":"healthy"}`)
`POST`	`/api/auth/register`	—	Register a new user
`POST`	`/api/auth/login`	—	Authenticate and receive access + refresh tokens
`POST`	`/api/auth/refresh`	—	Exchange a refresh token for a new access token
`GET`	`/api/trips?origin=…&destination=…`	Bearer JWT	Fetch parsed OJP trip results
`POST`	`/api/saved-trips`	Bearer JWT	Save a trip for the current user
`GET`	`/api/saved-trips`	Bearer JWT	Get all saved trips for the current user
`DELETE`	`/api/saved-trips/{trip_id}`	Bearer JWT	Delete a saved trip
`PATCH`	`/api/saved-trips/{trip_id}`	Bearer JWT	Update trip details for a saved trip

Example: get a token & search trips

```
# Login
TOKEN=$(curl -s -X POST http://localhost/api/auth/login \
  -H 'Content-Type: application/json' \
  -d '{"username":"admin","password":"admin123"}' | jq -r .access_token)

# Search
curl -s "http://localhost/api/trips?origin=8503308&destination=8503424" \
  -H "Authorization: Bearer $TOKEN" | jq .
```

Makefile Targets

Target	Description
`make up`	Build & start entire stack
`make down`	Stop containers
`make restart`	`down` + `up`
`make build`	Build images only
`make logs`	Tail all service logs
`make clean`	Remove containers, volumes, caches
`make backend-install`	`uv sync` (local dev)
`make backend-dev`	Run backend locally with hot-reload
`make backend-login`	Get JWT from running backend
`make backend-trips`	Fetch trips with saved JWT

Variables can be overridden: `make backend-login USERNAME=alice PASSWORD=secret` .

Project Structure

```
MyTrip/
├── backend/
│   ├── Dockerfile          # uv-based image
│   ├── main.py             # FastAPI application
│   ├── const.py
│   ├── pyproject.toml
│   └── uv.lock
├── frontend/
│   └── index.html          # Plain HTML/JS (no build step)
├── docker-compose.yml      # Orchestrates all services
├── traefik.dynamic.yml     # Traefik file-provider routing config
├── Makefile                # Convenience targets
├── .env.example            # Template for secrets
└── README.md
```

Key Design Decisions

- **MongoDB** — document store fits the flexible trip/user data; no schema migrations needed.
- **Argon2 password hashing** — avoids the bcrypt 72-byte limit issue on Python 3.14.
- **OJP XML built with `xml.etree.ElementTree`** — no string templating; safe and maintainable.
- **OJP response parsing** — XML is parsed server-side into structured JSON (trip legs, times, modes, intermediate stops) so the frontend receives clean data.
- **In-memory response cache** (60 s TTL) — prevents hammering the OJP API during load tests.
- **JWT Refresh Tokens** — implements a dual-token system (short-lived access token, long-lived refresh token) with "invisible" frontend refresh to enhance security and user experience.
- **Traefik file provider** — portable across macOS/Linux without requiring Docker socket access.
- **Admin seed** — an `admin` user is created on first startup if the `users` collection is empty.
- **Trip saving** — users can save trips with names and details, and retrieve them later via the `/api/saved-trips` endpoints.

Load Testing

Tested with `hey` (simple) or `k6` (advanced):

Quick Start (Docker Stack)

```
# Simple load test with hey
make loadtest

# Advanced load test with k6
make loadtest-k6
```

Local Development

```
# Test against local backend
make loadtest-local

# k6 test against local backend
make loadtest-k6-local
```

Customization

```
# Custom parameters for Docker stack
make loadtest REQUESTS=500 CONCURRENCY=50 \
  LOADTEST_USERNAME=alice LOADTEST_PASSWORD=secret \
  LOADTEST_ORIGIN="Geneva" LOADTEST_DEST="Lausanne"

# Custom parameters for local backend
make loadtest-local REQUESTS=100 CONCURRENCY=5 \
  BACKEND_URL=http://localhost:8000 \
  USERNAME=test PASSWORD=test123
```

Defaults

- **Docker stack**: `http://localhost` , `admin/admin` , `Zürich → Bern`
- **Local backend**: `http://localhost:8000` , `admin/admin123` , numeric codes

Features

k6 version includes:

- JavaScript scripting for complex scenarios
- Custom thresholds (P95 latency <500ms, error rate <1%)
- Docker-based execution (no local install needed)
- Health checks and automatic retries

hey version is simpler:

- Basic HTTP load testing
- No dependencies beyond curl/jq
- Quick iteration

Load Test Results

hey (Simple HTTP Load Tester)

Metric	Result
RPS	3936.6954
P95 latency	0.0020 secs
Error rate	100%
Average latency	0.0013 secs
Fastest request	0.0004 secs
Slowest request	0.0702 secs

Test conditions: 50000 requests, 5 concurrent users, local backend **Route**: 8503308 → 8503424 (cached responses)
Environment: macOS, Docker Desktop, M3 Pro

k6 (Advanced Load Testing)

Metric	Result
RPS	25.20
P95 latency	647.45ms

Metric	Result
Error rate	0%
Average latency	266.98ms
Requests	400 (200 iterations × 2 requests each)
Duration	15.9s

Test conditions: 200 iterations, 20 VUs, local backend **Route**: Zürich → Bern (with authentication) **Environment**: macOS, Docker Desktop, M3 Pro

Threshold results:

- ✓ Error rate < 1%: PASSED (0%)
- ✗ P95 latency < 500ms: FAILED (647.45ms)

Analysis:

- k6 performs full authentication flow (login + trips request) for each iteration
- Slower than hey due to authentication overhead and more realistic workload
- P95 threshold failure indicates some requests exceed 500ms (likely cache misses)
- 100% success rate shows system is stable under load

> Most requests are served from the 60 s in-memory cache.

License

See LICENSE.

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Contributors 1

LInI911

Languages

Python 32.6% HTML 31.9% CSS 12.5% Makefile 12.4% JavaScript 10.0% Dockerfile 0.6%

Suggested workflows

Based on your tech stack

Publish Node.js Package Configure

Publishes a Node.js package to npm.

SLSA Generic generator Configure

Generate SLSA3 provenance for your existing release workflows

Publish Python Package Configure

Publish a Python Package to PyPI on release.

More workflows

Dismiss suggestions