K26Group / **slick26-redirector**

Code Issues Pull requests 1 Agents Actions Projects Security Insights

Watch 0

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

Private repository

main **2** Branches **0** Tags Go to file t Go to file Code

Nicicalu Refresh token 8f47dc5 · 25 minutes ago

.vscode	Base	3 weeks ago
nginx	Fix coolify	4 days ago
scripts	Refresh token	25 minutes ago
src	Refresh token	25 minutes ago
static	Healthcheck and Favicon	3 days ago
.dockerignore	Base	2 weeks ago
.gitignore	Base	3 weeks ago
.npmrc	Base	3 weeks ago
Dockerfile	Update Dockerfile	6 hours ago
README.md	Fixes	last week
components.json	Base	3 weeks ago
docker-compose.coolify.yml	Update docker-compose.coolify.yml	2 hours ago
docker-compose.yml	Upgrade Postgres image to version 18-...	6 hours ago
docker-entrypoint.sh	Fix not both apps migrate at the same t...	last week
drizzle.config.ts	Base	2 weeks ago
package-lock.json	Update deps	2 hours ago
package.json	Update deps	2 hours ago
svelte.config.js	Fixes	last week
tsconfig.json	Base	3 weeks ago
vite.config.ts	Base	3 weeks ago

README

RaceNova Redirector

Quick Start (Docker)

A fresh clone + single command brings up the entire system:

```
git clone <repo-url>
cd slick26-redirector
docker compose up
```

- **App**: http://localhost (via Nginx)
- **Admin**: http://localhost/admin (login: `admin@example.com` / `admin123`)
- **Redirect example**: http://localhost/rrd/event123/rider456 → Rehm Race Days livetiming

Development

```
npm install
```

Create a `.env` file (or set env vars):

```
DATABASE_URL=postgresql://postgres:postgres@localhost:5432/racenova
JWT_SECRET=dev-secret
ADMIN_EMAIL=admin@example.com
ADMIN_PASSWORD=admin123
```

Start Postgres (e.g. `docker run -d -p 5432:5432 -e POSTGRES_PASSWORD=postgres postgres:16-alpine`), then:

```
npm run db:push
npm run db:seed
npm run dev
```

Tech Stack

- **Frontend**: SvelteKit, Tailwind, shadcn-svelte
- **Backend**: SvelteKit (Node adapter)
- **Database**: Postgres, Drizzle ORM
- **Auth**: JWT (jose)
- **Deploy**: Docker, Nginx load balancer, 2 app instances

URL Structure

Pattern	Auth	Description
`/{code}/{event_id}/{rider_id}`	No	Redirect + log
`/admin`	JWT	Admin dashboard
`/admin/login`	No	Login

Pattern	Auth	Description
`/admin/customers`	JWT	Customer CRUD
`/admin/stats`	JWT	Redirect counts

Environment Variables

Variable	Description
`DATABASE_URL`	Postgres connection string
`JWT_SECRET`	Secret for JWT signing
`ADMIN_EMAIL`	Admin login email (seed)
`ADMIN_PASSWORD`	Admin login password (seed)
`TRUSTED_ORIGINS`	Optional comma-separated CSRF allowlist (for example `http://localhost,http://127.0.0.1`)

Project Description

RaceNova Redirector is a small distributed web application that I am building as part of my Challenge Task. It is designed as a multi-tenant redirect service for the RaceNova platform.

RaceNova is a software platform for motorsport events that provides automated timing, session management, and live data for organizers and drivers. During events, QR codes are used to link directly to live timing pages for specific riders.

Right now, redirects are implemented only for one customer using a simple pattern like:

```
/(.*)/(.*)
→ https://live.rehmracedays.de/livetiming/{eventId}/rider/{riderId}
```

This works, but only for a single organizer. It is not flexible, not scalable, and does not provide any tracking or statistics.

The goal of this project is to generalize this idea into a reusable, multi-customer redirect system.

Main Idea

The system will support multiple customers.

Each redirect URL will follow this structure:

```
/{customerCode}/{eventId}/{riderId}
```

Example:

```
/rrd/123/456
```

This would redirect to:

```
https://live.rehmracedays.de/livetiming/123/rider/456
```

Each customer has:

- A unique short code (for example `rrd`)
- A configurable base URL
- A configurable path template
- Redirect tracking

This makes the system effectively a specialized URL shortener. The focus is to keep URLs as short as possible because they are used in QR codes. Shorter URLs result in less dense QR codes, which are easier to scan and more reliable in a racing environment.

Functionality

Public Redirects

The redirect endpoints are public and do not require authentication:

```
GET /{customerCode}/{eventId}/{riderId}
```

When a request is received:

1. The system looks up the customer using `customerCode`
2. It builds the final target URL using the stored template
3. It increments a redirect counter
4. It returns an HTTP redirect response

There is no public homepage. The application only works if a valid redirect URL is used.

Admin Area

The system includes a simple admin interface.

Only authenticated users can access it. It allows:

- Creating and managing customers
- Configuring base URLs and path templates
- Activating or deactivating customers
- Viewing redirect statistics

All admin API endpoints are protected using JWT-based authentication.

Technical Stack

The application is implemented as a distributed system and fulfills all requirements of the challenge task.

Frontend

- SvelteKit

- Tailwind CSS
- shadcn-svelte

Backend

- SvelteKit server endpoints
- JWT authentication

Database

- PostgreSQL for persistent storage

Infrastructure

- Nginx as reverse proxy and load balancer
- Docker and Docker Compose
- Two identical service instances for failover

Distributed System Design

The system consists of:

- One Nginx load balancer
- Two identical application instances
- One PostgreSQL database

Nginx distributes incoming requests between the two service instances. If one instance fails during the presentation, the other instance continues to handle traffic.

The entire system can be started with a single command:

```
docker compose up
```

This will start:

- The database
- The two application instances
- The load balancer
- Database migrations and seed data

No manual setup steps are required.

Load Testing

As required by the assignment, a load test is performed against a JWT-protected admin endpoint.

What is tested

- Endpoint: `GET /api/customers` (JWT protected)
- The k6 script authenticates first (login), then repeatedly calls the endpoint using the issued JWT.
- Key metrics:

- Requests per second (throughput)
- Response time distribution (p90, p95)
- Error rate (`http_req_failed`)

How to run

Run from WSL:

```
npm run loadtest
```

Heavy profile:

```
npm run loadtest:heavy
```

Optional variables:

```
BASE_URL=http://localhost VUS=80 DURATION=120s SLEEP_SECONDS=0 npm run loadtest
```

Failover demo during presentation:

```
# terminal A
VUS=80 DURATION=120s npm run loadtest

# terminal B (while test is running)
docker compose stop app1
sleep 20
docker compose start app1
```

Results and interpretation

Both runs below used:

- `VUS=150`
- `DURATION=90s`
- `SLEEP_SECONDS=0`
- Thresholds:
  - `http_req_duration p(95) < 500ms`
  - `http_req_failed rate < 1%`

Run A, only one app container up

Observed results (k6 summary):

- `http_req_failed` : **96.94%**
- `http_req_duration p(95)` : **27.38ms**
- Throughput: **~10,250 req/s**
- Successful checks (`status is 200`): **~3%**

Interpretation:

- The p95 latency looks excellent, but the error rate shows the test was not actually succeeding for most requests.
- With only one app instance running while Nginx still balances across two upstreams, a large share of requests are routed to the down instance and fail immediately (fast failures), which explains:

  - Extremely high throughput, because failures return quickly and VUs can loop faster
  - Very low latency percentiles on the overall request duration
- Takeaway: this run demonstrates why health checks and failover aware load balancing matter, the load balancer must avoid sending traffic to unhealthy instances.

Suggested improvement (optional):

- Add container health checks and configure Nginx upstream behavior to stop routing to unhealthy backends (for example, active health checks or a setup where the upstream list is updated based on container health).

Run B, both app containers up

Observed results (k6 summary):

- `http_req_failed` : **0.00%**
- `http_req_duration` :
  - avg **154.76ms**
  - p(90) **355.86ms**
  - p(95) **401.92ms**
- Throughput: **~968 req/s**
- Successful checks (`status is 200`): **100%**

Interpretation:

- With both instances available, the system meets the thresholds:
  - p95 stays below 500ms
  - failure rate stays below 1%
- Throughput is lower than in Run A, because the system is doing real work for every request instead of failing fast.
- The results are consistent with a small Node based service that:
  - authenticates once and then serves a simple DB backed admin query
  - sits behind Nginx which distributes load across 2 instances

Summary

RaceNova Redirector is a small but realistic distributed system that solves a practical problem in the RaceNova ecosystem.

It extends a simple hardcoded redirect into a scalable, multi-tenant solution with authentication, persistent storage, load balancing, and failover.

At the same time, it remains focused and minimal, making it suitable for the scope of an individual semester project.

Releases

No releases published

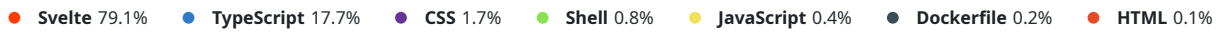
Packages

No packages published

Contributors 1

Nicicalu Nicolas Caluori

Languages


Svelte 79.1%
TypeScript 17.7%
CSS 1.7%
Shell 0.8%
JavaScript 0.4%
Dockerfile 0.2%
HTML 0.1%