Project information

Created on
February 16, 2026

Name	Last update
docs	1 month ago
gradle	22 minutes ago
http	5 hours ago
keycloak	22 minutes ago
load-test	22 minutes ago
src	5 hours ago
.gitignore	2 months ago
Dockerfile	1 month ago
Dockerfile.native	5 hours ago
README.md	4 hours ago
build.gradle.kts	5 hours ago
docker-compose.yml	4 hours ago
gradle.properties	2 months ago
gradlew	2 months ago
gradlew.bat	2 months ago
nginx.conf	1 month ago
settings.gradle.kts	2 months ago

README.md

idea-graveyard-service

Kotlin + Ktor REST API for Idea Graveyard. Clean/hexagonal architecture with PostgreSQL persistence and Keycloak JWT authentication.

Local development

```
./gradlew run           # dev server on port 8080
./gradlew test          # run tests
./gradlew build         # full build
./gradlew buildFatJar   # executable JAR
```

Database config: `src/main/resources/application.yaml` — PostgreSQL at `localhost:5432/idea_graveyard` (user: `admin`, password: `admin`).

Docker Compose

The stack is profile-based: choose `jvm` or `native` to select the app build. All other services (DB, Keycloak, nginx, web) start regardless of profile.

```
docker compose --profile jvm up -d      # JVM build (Dockerfile)
docker compose --profile native up -d   # Native build (Dockerfile.native)
```

Build images before starting (or rebuild after code changes):

```
docker compose --profile jvm build
docker compose --profile native build
```

Tear down and remove volumes:

```
docker compose --profile jvm down -v
```

Services

Service	Container	Port	Description
`idea-graveyard-db`	`idea-graveyard-db`	`5432`	PostgreSQL 18 — app database
`keycloak`	`idea-graveyard-keycloak`	`8180`	Keycloak 26 — JWT issuer, realm imported from `keycloak/realm-export.json`
`idea-graveyard-app-jvm-1/2`	—	—	Two JVM app instances (profile: `jvm`)
`idea-graveyard-app-native-1/2`	—	—	Two native app instances (profile: `native`)
`idea-graveyard-web`	`idea-graveyard-web`	—	Angular frontend (internal only)
`nginx`	`idea-graveyard-nginx`	`80`	Reverse proxy — routes `/api/*` round-robin to the two app instances, `/` to the frontend

nginx round-robins between `app-1` and `app-2` using the Docker network aliases defined per profile, so the load balancing is transparent to clients.

Load testing

k6 — full scenario

Requires a running stack. Ramps to 100 VUs over 2 minutes hitting `GET /api/ideas` with a Keycloak Bearer token. Threshold: p95 < 500 ms, error rate < 1%.

```
k6 run load-test/load-test.js
```

Generate an HTML report alongside the JSON summary:

```
K6_WEB_DASHBOARD=true \
K6_WEB_DASHBOARD_EXPORT=load-test/results/report.html \
k6 run load-test/load-test.js
```

k6 — JVM vs Native comparison

Starts each stack in turn, runs the full k6 scenario, tears it down, then prints a side-by-side summary. Results are saved to `load-test/results/`.

```
./load-test/compare.sh             # build + run both variants
./load-test/compare.sh --skip-build  # reuse already-built images
```

HTML reports are generated automatically at `load-test/results/jvm-report.html` and `load-test/results/native-report.html`.

hey — quick smoke test

Requires a running stack. 500 requests at 50 concurrency against `GET /api/ideas`. Fetches a Keycloak token automatically.

```
./load-test/hey-get-ideas.sh

# Override defaults
./load-test/hey-get-ideas.sh --requests=1000 --concurrency=100

# Target a different host
BASE_URL=http://staging.example.com ./load-test/hey-get-ideas.sh
```

Requires `hey` (`brew install hey`) and a running stack.