Aplikacija za izdelavo načrtov za fitnes vadbe in beleženje napredka (FitPlan)

Končno poročilo projekta pri predmetu Računalniške storitve v oblaku

- **Člani skupine:** Nik Jukič in Gašper Kolbezen
- **Projektna skupina:** 25
- **Ime skupine:** GiN

Povezava do GitHub organizacije: https://github.com/FitPlanGiN
Povezava do GitHub repozitorija z izvorno kodo: https://github.com/FitPlanGiN/FitPlan

Opis projekta:

Cilj projekta je bil razviti aplikacijo, ki uporabnikom pomaga pri načrtovanju fitnes vadbe in beleženju napredka. Zaradi tehničnih težav in omejitev smo uspeli implementirati osnovne funkcionalnosti, ki omogočajo:

- Kreacijo vaj,
- Dodajanje vadbenih načrtov,
- Pošiljanje obvestil o zaključeni vadbi na e-pošto,
- Pridobivanje povratnih informacij o vadbi preko AI modela.

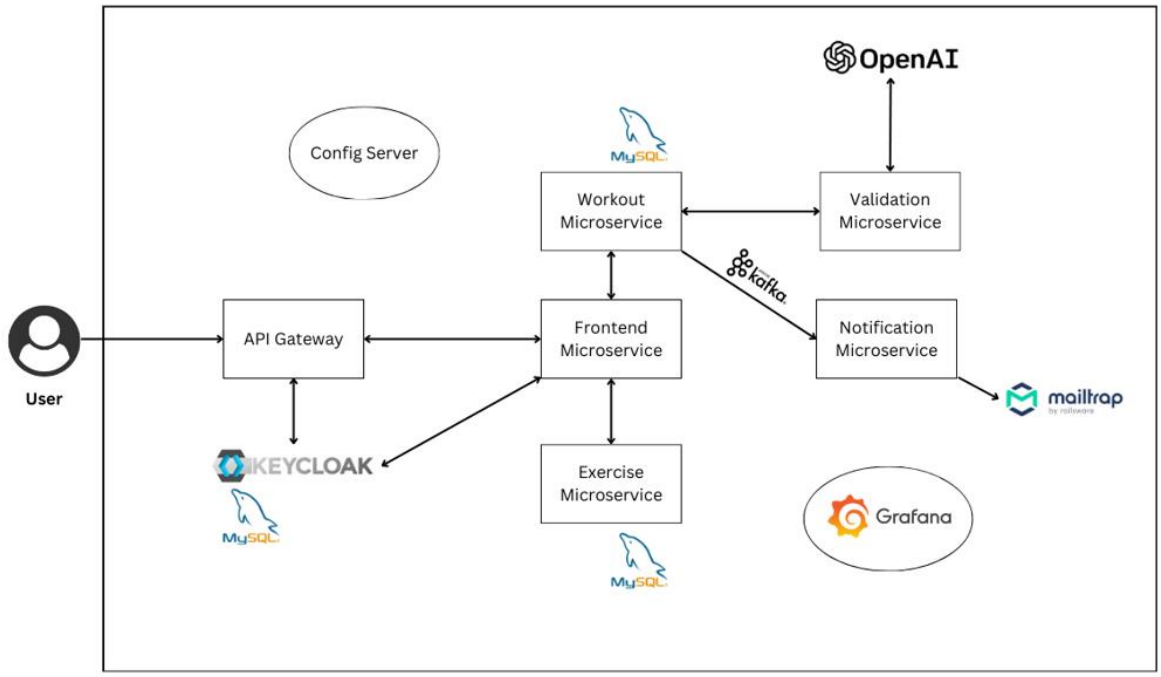
Ogrodje in razvojno okolje:

Pri razvoju projekta sva uporabila naslednje tehnologije:

- **Java**: Glavni programski jezik za razvoj aplikacije.
- **Spring Boot**: Ogrodje za implementacijo mikrostoritev in API-jev.
- **MySQL**: Relacijska podatkovna baza za shranjevanje podatkov o uporabnikih in ustvarjenih vadbenih načrtih (workout).
- **MongoDB**: NoSQL podatkovna baza za shranjevanje ustvarjenih vadb (exercise).
- **TypeScript**: Uporabljen za razvoj čelnega dela aplikacije.
- **Angular**: Ogrodje za razvoj čelnega dela aplikacije
- **HTML in Tailwind CSS**: Uporabljena za strukturiranje in stiliranje uporabniškega vmesnika.
- **Apache Maven**: Orodje za upravljanje odvisnosti in gradnjo projekta.
- **Docker**: Uporabljen za kontejnerizacijo aplikacije in poenostavitev nameščanja.
- **Kubernetes**: Aplikacija je bila uspešno nameščena v lokalni Kubernetes gruči (Kind Kubernetes), vendar delovanje v produkcijskem okolju ni bilo vzpostavljeno zaradi tehničnih omejitev.

- **Kind Kubernetes**: Lokalno Kubernetes okolje za razvoj in testiranje.
- **GitHub Actions**: Nastavljen za osnovno avtomatizacijo CI/CD procesov.
- **IntelliJ IDEA**: Glavno razvojno okolje za pisanje in testiranje kode zalednega dela aplikacije.
    - **Visual Studio Code**: Stransko razvojno okolje za pisanje in testiranje kode čelnega dela aplikacije.
- **Postman**: Uporabljen za testiranje API-jev med razvojem.
- **GPT-4o-mini:** Model uporabljen za generiranje povratnih informacij o vadbenem načrtu.

Shema arhitekture:



Seznam funkcionalnosti mikrostoritev

1. Upravljanje vaj

- Kreacija novih vaj z informacijami, kot so ime vaje, opis in kategorija.
- Pridobivanje seznamov obstoječih vaj iz podatkovne baze.

2. Vadbeni načrti

- Dodajanje novih vadbenih načrtov v račun uporabnika.

3. Obvestila o vadbi

- Pošiljanje e-poštnih obvestil uporabnikom, ko zaključijo vadbo.

4. AI povratne informacije

- Pošiljanje vadbenih načrtov AI modelu za generiranje povratnih informacij o strukturivadbe.
- Pridobivanje nasvetov za izboljšanje vadbenega programa.

Kompleksnejši primer uporabe

Dodajanje vadbenega načrta z AI povratnimi informacijami in obvestili

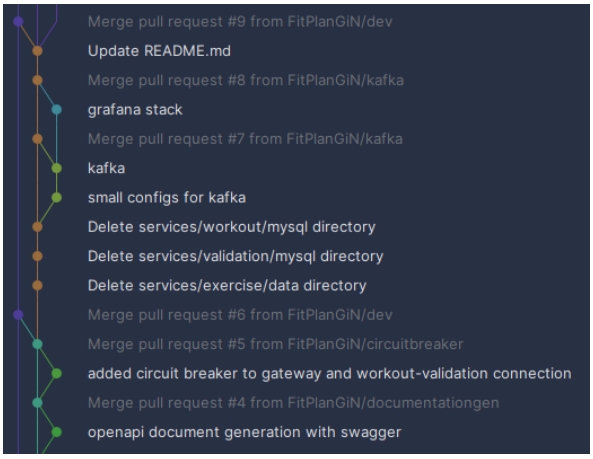
Uporabnik preko čelnega dela aplikacije (frontend mikrostoritve) doda vadbeni načrt v svoj osebni predal. Ta zahteva se pošlje na **gateway mikrostoritev**, ki usmerja promet zalednim storitvam. Gateway preusmeri zahtevo na **workout mikrostoritev**, ki obdeluje podatke o vadbenih načrtih. Ko je vadbeni načrt uspešno dodan:

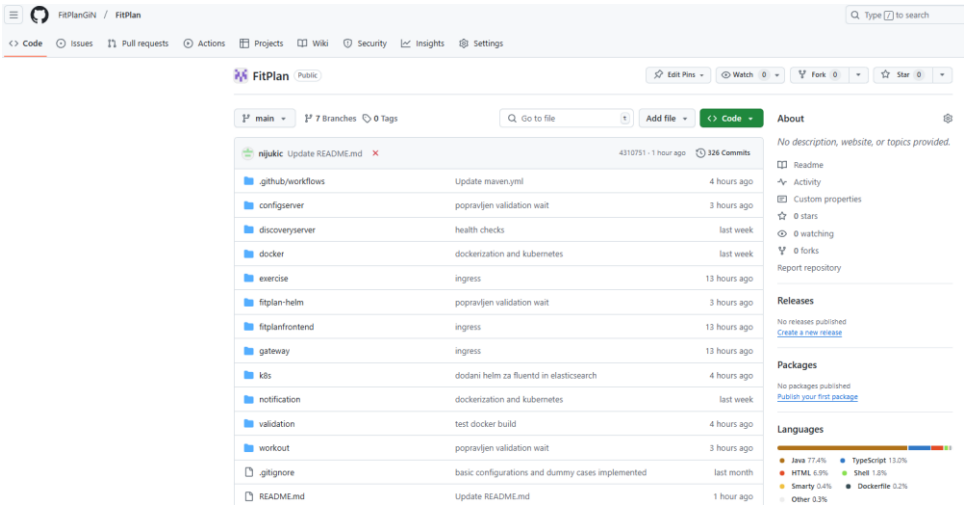
1. **Pošiljanje obvestila:**
   Workout mikrostoritev preko Kafka sporočilnega sistema pošlje informacijo notification mikrostoritvi, ki uporabniku pošlje e-poštno obvestilo o dodanem načrtu (mailtrap.io).
2. **Validacija načrta:**
   Hkrati workout mikrostoritev preko HTTP zahteve pokliče validation mikrostoritev, ki preveri skladnost vadbenega načrta.
3. **AI povratne informacije:**
   Workout mikrostoritev pošlje vadbeni načrt AI modelu za generiranje povratnih informacij. AI model analizira načrt in poda nasvete za njegovo izboljšanje. Povratne informacije se vrnejo in prikažejo uporabniku preko čelnega dela aplikacije.

Opravljene/vključene osnovne projektne zahteve:

Repozitorij

Ustvarila sva Git repozitorij na platformi GitHub, ki je dostopen preko povezave na začetku poročila. Na začetku sva ustvarila veji main in dev, tekom razvoja pa sva po potrebi ustvarjala in zapirala dodatne veje. V repozitorij je vključena README.md datoteka, ki vsebuje informacije o projektu.





Mikrostoritve in »cloud-native« aplikacija

Razvojno okolje sva nastavila z uporabo IntelliJ IDEA. Projekt je organiziran z ločenimi direktoriji za izvorno kodo, konfiguracijske datoteke in teste. Za upravljanje odvisnosti sva uporabila Maven, za nadzor različic pa Git.

Aplikacijo sva razdelila na mikrostoritve:

- Exercise: za ustvaritev vaje.
- Workout: upravljanje vadbenih načrtov.
- Notification: pošiljanje obvestil.
- Validation: interakcija z OpenAI API.

Komunikacija med mikrostoritvami poteka prek HTTP zahtev in asinhronih sporočil s pomočjo Kafka.

Vsaka mikrostoritev ima Kubernetes YAML datoteke za namestitve, storitve in konfiguracije. Uporabila sva ConfigMaps za nastavitve in Secrets za varne podatke.

Za podatkovne baze sva uporabila MySQL za vadbene načrte in MongoDB za vaje. Baze vsebujejo osnovne tabele s testnimi podatki. Navodila za nastavitev okolja in uporabo aplikacije so dokumentirana v README.md datoteki.

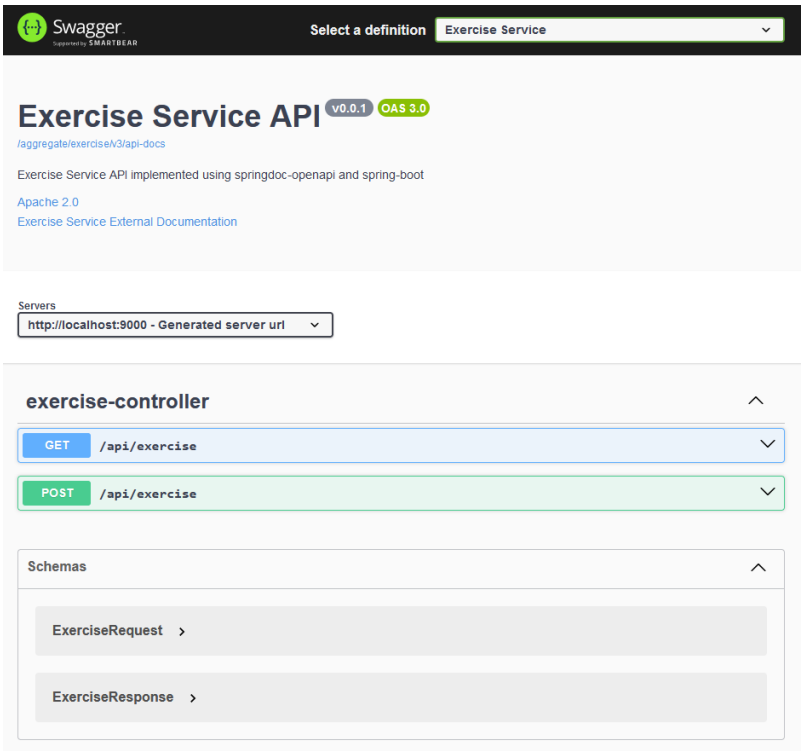
Dokumentacija API

Dokumentacija API: Za dokumentacijo API sva uporabila Swagger.

Omogočeni so klici na /api/exercise, /api/workout in /api/validation.

- Klic GET na /api/exercise ne potrebuje nobenih parametrov, odgovor s kodo 200; OK pa vsebuje json polje, definirano po shemi na sliki. Klic POST na /api/exercise potrebuje enako json polje v telesu in lahko vrne kodo 201; Created, ki prav tako vrne json polje ustvarjenega objekta.
- Na /api/workout je omogočen klic POST, ki zahteva json polje v telesu. Vrne kodo 201; Created.
- Prav tako je na /api/validation omogočen klic POST, ki zahteva polja »name« in »description«, vrača pa kodo 200; OK in polje z odgovorom zunanjega API v obliki string.

Readme.md datoteka vključuje del te zahteve.



Cevovod CI/CD

Za avtomatizacijo CI/CD procesa sva uporabila GitHub Actions. Namen cevovoda je bil zagotoviti, da se koda samodejno gradi, namešča in testira.

Nastavitev CI/CD

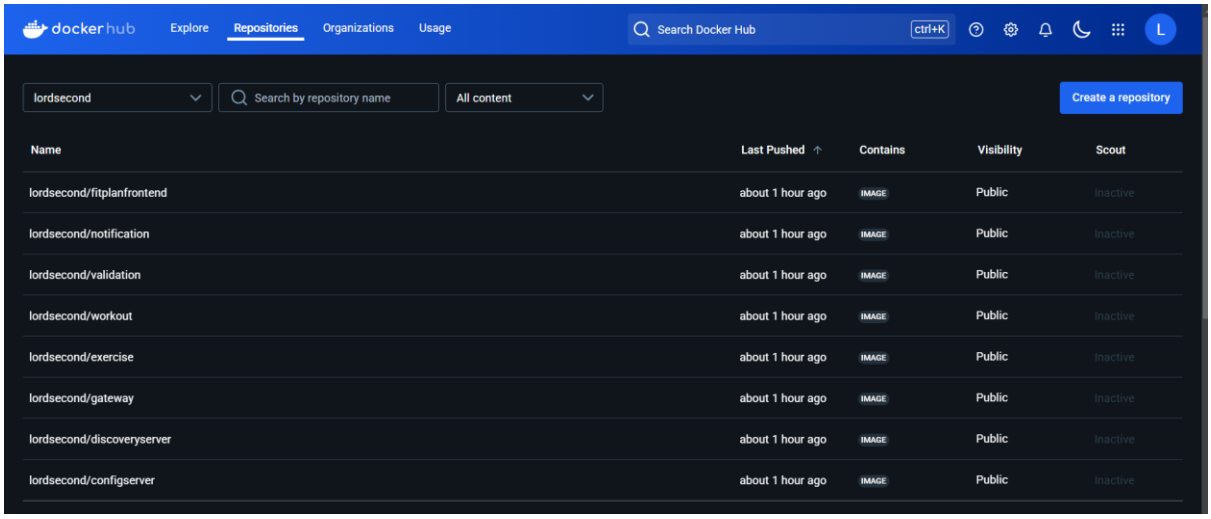
- Gradnja: Ob vsaki spremembi v veji main ali dev se koda samodejno gradi z uporabo Maven.
- Namestitev: Priprava Docker slik za vse mikrostoritve in nalaganje slik v Docker Hub. Ter nameščanje v kubernetes gruči pri ponudniku Google Cloud (GKE) z uporabo manifestov in kasneje tudi helm chartov.
- Testiranje: Testiranje če je aplikacija dostopna na statičnem IP naslovu.

FitPlan / .github / workflows / **maven.yml**

nijukic Update maven.yml

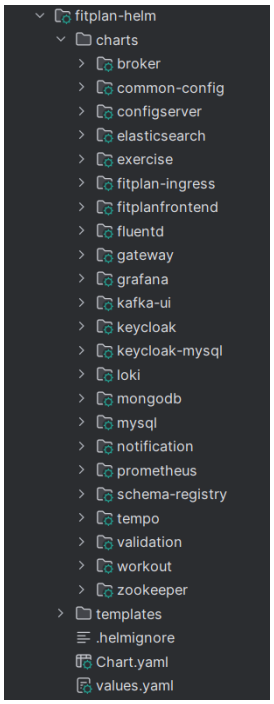
Code | Blame | 297 lines (227 loc) · 9.86 KB

```yaml
name: SpringBoot CI/CD Pipeline (Helm)

on:
  push:
    branches: [ "main" ]
  workflow_dispatch:

jobs:
  build-and-deploy:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Set up JDK 21
        uses: actions/setup-java@v4
        with:
          java-version: '21'
          distribution: 'temurin'
          cache: maven

      - name: Build and push images using spring-boot:build-image
        env:
          DOCKER_PASSWORD: ${{ secrets.DOCKER_PASSWORD }}
        run: |
```



Helm charts:
Za vsako mikrostoritev sva pripravila Helm Chart (mikrostoritve so razdeljene na applications in na infrastructure), ki vključuje vse potrebne Kubernetes vire, kot so Deployment, Service, ConfigMap, in Secret, DeamonSet in StatefulSet.

Helm Chart nama je omogočil centralizirano upravljanje vseh konfiguracij in namestitev. V mapi fitplan-helm se lahko jasno vidi Helm datotečna struktura, kjer sva tudi vključila -dev in -test value datoteki, ki omogočata uporabo različnih konfiguracij v različnih okoljih.



Ustvarila sva ločene datoteke values-dev.yaml in values-prod.yaml, ki omogočajo prilagoditev ključnih parametrov za posamezno okolje, kot so:

- Število replik (replicaCount),
- Dodeljeni viri (resources),
- Naslovi zunanjih storitev in baze podatkov.
  Na ta način sva zagotovila enostavno preklapljanje med okolji brez ročnih sprememb v manifestih.

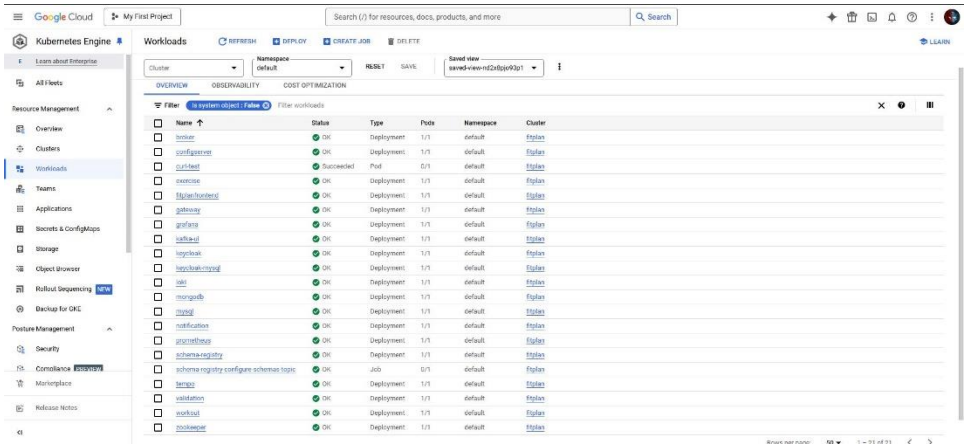
Za namestitev aplikacije sva uporabila Helm ukaze, ki omogočajo hitro inicializacijo aplikacije:

- helm install za inicialno namestitev,
- helm upgrade za posodobitev konfiguracij ali verzij mikrostoritev. Helm Chart omogoča enostavno vzdrževanje aplikacije, saj je mogoče vse spremembe uvesti z enim ukazom, pri čemer se posodobijo le spremenjeni deli.

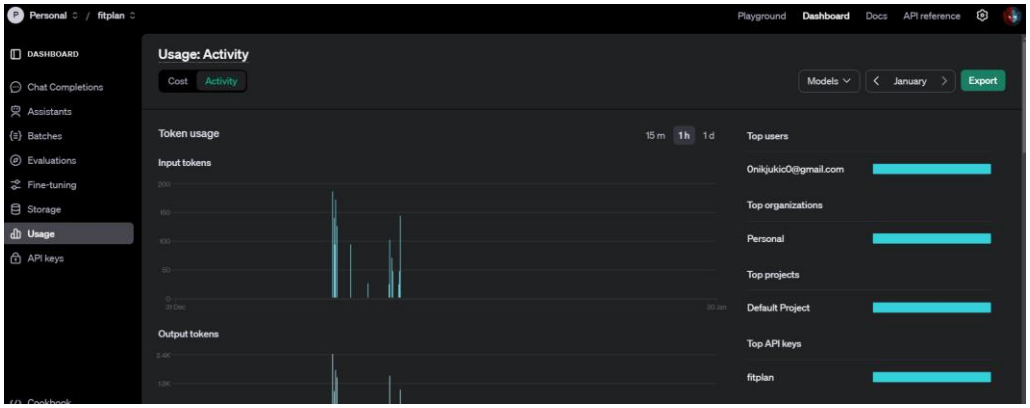
Namestitev v oblak
Za namestitev aplikacije sva uporabila Google Cloud Platform (GCP). Kubernetes gruča je bila vzpostavljena z uporabo Google Kubernetes Engine (GKE).

Dostop do aplikacije je omogočen preko zunanjega IP naslova z uporabo LoadBalancer servisa.

Aplikacija je dostopna na javnem naslovu preko statičnega IP naslova.



Zunanji API
V mikrostoritvi *Validation* sva integrirala OpenAI API za pridobivanje povratnih informacij o vadbenih načrtih. API analizira načrt in vrne svoje mnenje o načrtu ter morebitne nasvete za izboljšavo. Za dostop do OpenAI API sva uporabila API ključ, ki je varno shranjen v Kubernetes *Secret*. Mikrostoritev do ključa dostopa preko okoljske spremenljivke.

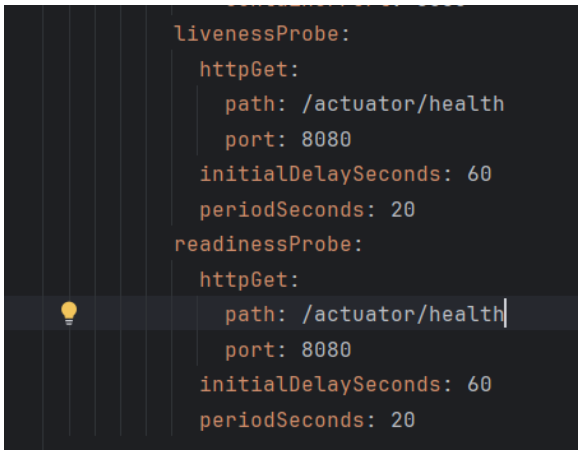


Preverjanje zdravja:
V mikrostoritvah *Workout*, *Exercise*, *Validation* in *Notification* sva implementirala /health končne točke, ki vračajo stanje storitev.
Kubernetes Liveness in Readiness Probes uporabljajo te končne točke za spremljanje stanja storitev.

- Liveness Probe: preverja, ali storitev deluje.

- Readiness Probe: preverja, ali je storitev pripravljena na obdelavo zahtev.

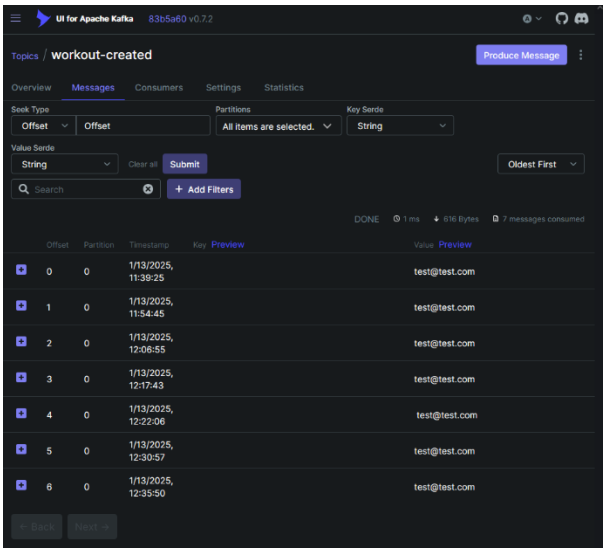
Preverjanje zdravja omogoča samodejno ponovno zagon okvarjenih storitev in zagotavlja stabilno delovanje aplikacije.



Sporočilni sistemi
Za asinhrono komunikacijo sva uporabila Apache Kafka kot sporočilni posrednik.

- Mikrostoritev *Workout* pošlje obvestilo na Kafka temo, ko uporabnik doda ali zaključi vadbeni načrt.
- Mikrostoritev *Notification* prejme sporočilo iz teme in pošlje e-poštno obvestilo uporabniku.

Kafka omogoča zanesljivo in razširljivo asinhrono komunikacijo med storitvama, kar izboljša zmogljivost in modularnost aplikacije.

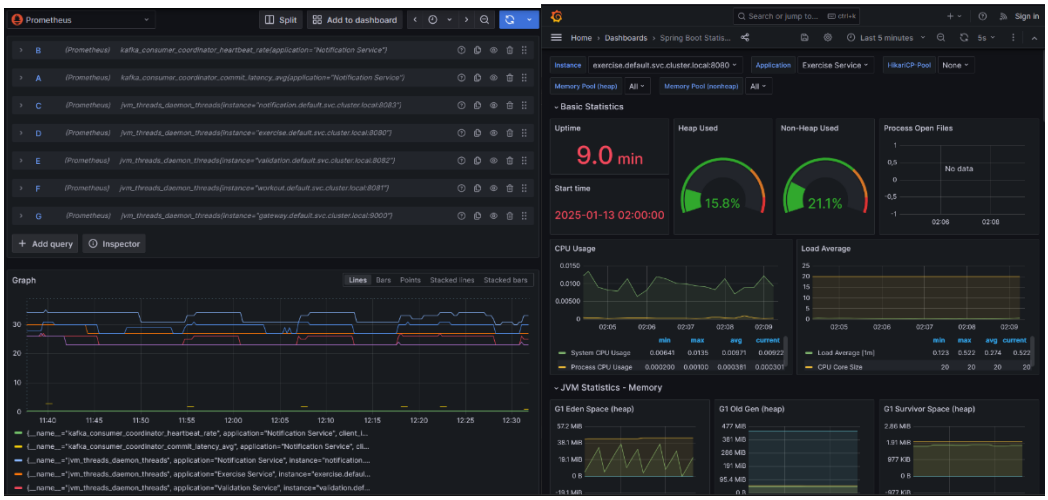


Centralizirano beleženje dnevnikov in zbiranje matrik
Za centralizirano beleženje in analizo dnevnikov ter zbiranje metrik sva uporabila:

- **Grafana**: Vizualizacija metrik in dnevnikov.

- **Loki in Tempo**: Za zbiranje in iskanje dnevnikov ter sledenje zahtevam (tracing).
- **Prometheus** zbira ključne metrike aplikacije, kot so odzivnost, zakasnitve in obremenitve.
- **Elasticsearch in Fluentd**: Fluentd preusmerja dnevnike vseh mikrostoritev v Elasticsearch za shranjevanje in analizo.

Nastavila sva opozorila v Grafani za zaznane vzorce v dnevnikih in za preseganje mej metrik.



Izolacija in toleranca napak

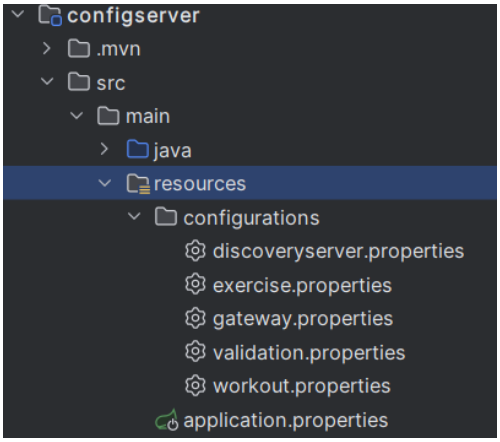
Uporabila sva *Resilience4j* v mikrostoritve za implementacijo vzorca Circuit Breaker. Pri zaznanih napakah storitev prekine klice do okvarjene komponente in omogoča njeno obnovitev.
Če ena storitev (npr. *Validation*) postane nedostopna:

1. Circuit Breaker preusmeri promet ali vrne privzete odgovore, da ohrani osnovno funkcionalnost.
2. Ostale mikrostoritve (npr. *Workout* in *Notification*) še naprej delujejo normalno.

Upravljanje s konfiguracijo
Konfiguracije, ki jih aplikacija potrebuje: Povezave na baze podatkov (URL, uporabniška imena, gesla, žetoni za dostop), zunanjimi API ključi, porti in nastavitve.
Konfiguracijo sva ločila od kode in omogočila njeno spremembo brez ponovnega prevajanja ali nameščanja z uporabo okoljskih spremenljivk in konfiguracijskega strežnika.

- **Okoljske spremenljivke**: Uporabljene za občutljive podatke (npr. API ključe), upravljane preko Kubernetes *Secrets*.
- **ConfigServer**: Na začetku sva uporabila Spring Cloud ConfigServer, ki je omogočal centralno upravljanje konfiguracije.

Med uporabo v gruči so se Kubernetes *ConfigMaps* izkazale za bolj učinkovite, kot konfiguracijski strežnik.

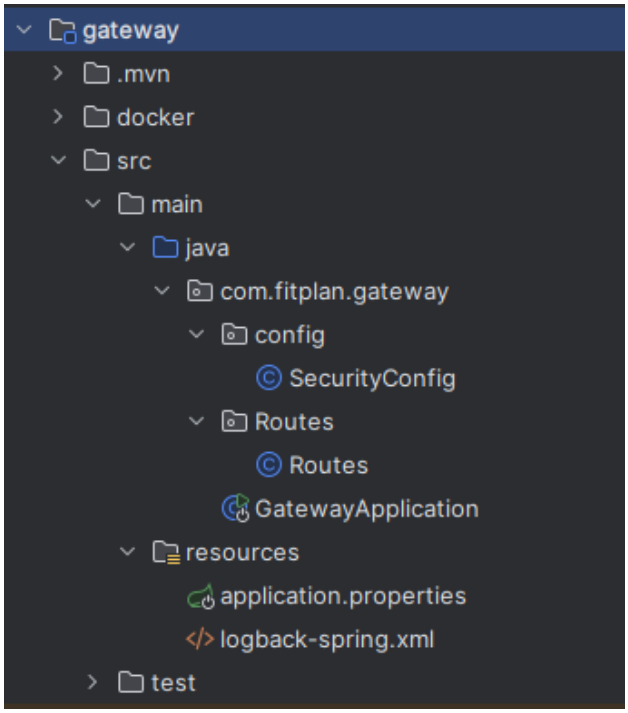


Grafični vmesnik

Grafični vmesnik: Za izdelavo grafičnega vmesnika sva uporabila ogrodje Angular. Za prijavo v aplikacijo sva implementirala povezavo s Keycloak strežnikom. Aplikacija dovoli prikaz »exercise«, le če je uporabnik prijavljen, kar je tudi vidno v grafičnem vmesniku. Preko grafičnega vmesnika je omogočeno ustvarjanje entitet »exercise«, ki se nato prikažejo v vmesniku. Iz ustvarjenih entitet »exercise« je nato mogoče ustvatiti entiteto »workout«, ki se ustvarijo preko REST klica na zaledni del.

API Gateway

Za upravljanje dostopa do mikrostoritev sva implementirala lastni API Gateway, ki služi kot centralna točka dostopa do drugih mikrostoritev. Dovoljuje dostop le preko avtentikacije s Keycloakom. Gateway sva implemetirala sama in je v bistvu tudi mikrostoritev.



OAuth2

OAuth2: Za upravljanje entitet in dostopa je implementirana rešitev Keycloak. Ob prvi uporabi aplikacije se mora uporabnik registrirati, nato se lahko prijavi z ustvarjenim uporabniškim računom. Pri REST klicih iz aplikacije se nanje avtomatsko doda avtentikacijski žeton, ki omogoča pravilno delovanje. Klic brez žetona ne uspe, s kodo 401; Unauthorized.