# Izzivi pri prenovi spletne aplikacije za iskanje knjižničnega gradiva

Andrej Krajnc, Vojko Ambrožič, Gregor Štefanič, Bojan Štok

IZUM – Institut informacijskih znanosti, Maribor, Slovenija
andrej.krajnc@izum.si, vojko.ambrozic@izum.si, gregor.stefanic@izum.si, bojan.stok@izum.si

Zasnova obstoječe aplikacije za iskanje gradiva v knjižnicah, imenovane COBISS+, je stara že 10 let. Lani smo se odločili za arhitekturno prenovo aplikacije. Glavna arhitekturna sprememba je, da namesto JSF, jQuery in JavaScript za gradnjo uporabniškega vmesnika uporabljamo ogrodji React in Next.js, na strežniški strani pa smo zamenjali Java EE 8 z Jakarta EE 10. V prispevku smo opisali izzive pri prehodu iz JSF/JavaScript na React/Next.js ter migraciji iz Java EE na Jakarta EE. Največ pozornosti smo namenili temu, kako smo v novi generaciji aplikacije reševali probleme glede robustnosti aplikacije, da se lažje odzovemo v primeru povečanega števila zahtev ali napada in v primeru, da mikrostoritve, ki jih aplikacija uporablja, niso dostopne, so preobremenjene, ali pa ne delujejo pravilno. V ta namen smo uporabili knjižnico Fault Tolerance, ki je del specifikacije MicroProfile. Predstavili smo primere uporabe razredov Timeout, Bulkhead, Retry, CircuitBraker in Fallback. Predstavili smo tudi praktično uporabo knjižnice OpenTracing, ki je del specifikacije MicroProfile in nam omogoča lažje sledenje med klici različnih mikrostoritev.

**Ključne besede:**
COBISS
COBISS+
Next.js
spletne aplikacije
mikrostoritve

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-894-9
DOI https://doi.org/10.18690/um.feri.4.2024.21

# 1 Uvod

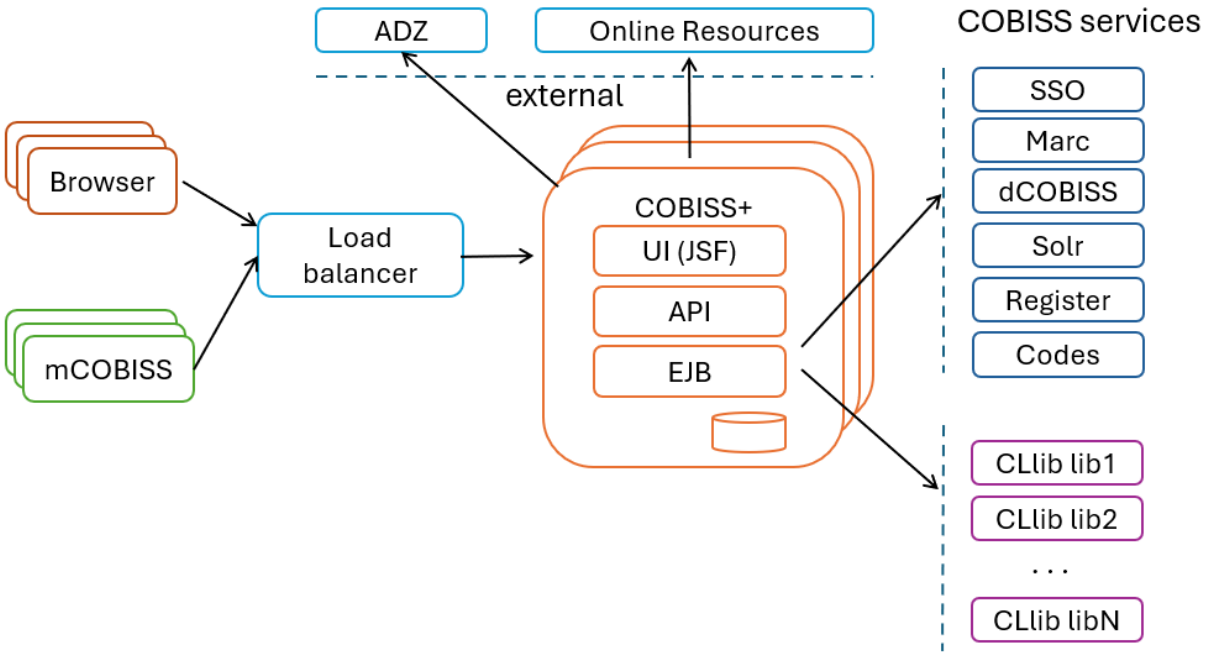
IZUM razvija sistem COBISS, v katerega je vključenih skoraj 1500 knjižnic. Sistem COBISS sestavljajo številne aplikacije, širši javnosti je najbolj poznana aplikacija COBISS+ [1]. COBISS+ je aplikacija za iskanje po knjižničnem gradivu slovenskih knjižnic in tudi knjižnic v državah jugovzhodne Evrope. Omogoča tako rezervacijo gradiva kot tudi izposojo e-knjig, pregled javno dostopnih člankov in dostop do gradiva tujih baz glede na uporabnika oz. pravila njegove knjižnice. Aplikacija COBISS+ deluje kot nekakšen integrator različnih spletnih storitev, ki jih sistem COBISS ponuja za knjižnice in so hkrati zanimive za širšo javnost.

Obstoječa aplikacija COBISS+ je bila razvita pred več kot 10 leti. V tem času smo jo dopolnjevali tako tehnološko (npr. prehod na Linux) kot tudi vsebinsko (nove funkcionalnosti ipd.). Obstoječa aplikacija COBISS+ je bila narejena v tehnologiji Java EE z uporabo ogrodja Spring. Na strežniku je zagnana v več instancah, preko uporabe metrik imamo dober vpogled v delovanje aplikacije. Odločili smo se za prenovo aplikacije, vendar velja poudariti, da obstoječa aplikacija dobro deluje, je odzivna in služi svojemu namenu.

Prvi problem obstoječe aplikacije je bil razvoj uporabniškega vmesnika. Obstoječa aplikacija za izris uporabniškega vmesnika uporablja JSF, za dodajanje interaktivnosti pa se je uporabljala velika količina knjižnic JavaScript, kot npr. jQuery in TomSelect. Zaradi tega je razvoj modernih funkcionalnosti v uporabniškem vmesniku zahteven, otežena je tudi uporaba novejših orodij in knjižnic JavaScript. Odločili smo se za ločitev razvoja aplikacije na uporabniški vmesnik in zaledje, pri tem pa smo uporabili obstoječo zaledno kodo Java EE.

Drugi problem je bila sama robustnost sistema. Število uporabniških zahtev je veliko (tudi do 1000 na sekundo) in sistem je praviloma zelo odziven, saj je zahteva v povprečju obdelana v 20 ms. Toda včasih se pojavijo problemi pri odvisnih servisih, ki so lahko nedosegljivi oz. preobremenjeni in imajo zato predolge odzivne čase. To smo poskušali omejiti z uporaba timeout-ov in predpomnilnika, toda sama rešitev ni bila celovita oz. konsistentna po celotnem sistemu. Zgodila se je tudi občasna preobremenjenost sistema zaradi prevelikega števila zahtev iz celega sveta (Kitajska, Rusija …).

Na sliki 1 je groba arhitektura obstoječe aplikacije COBISS+.

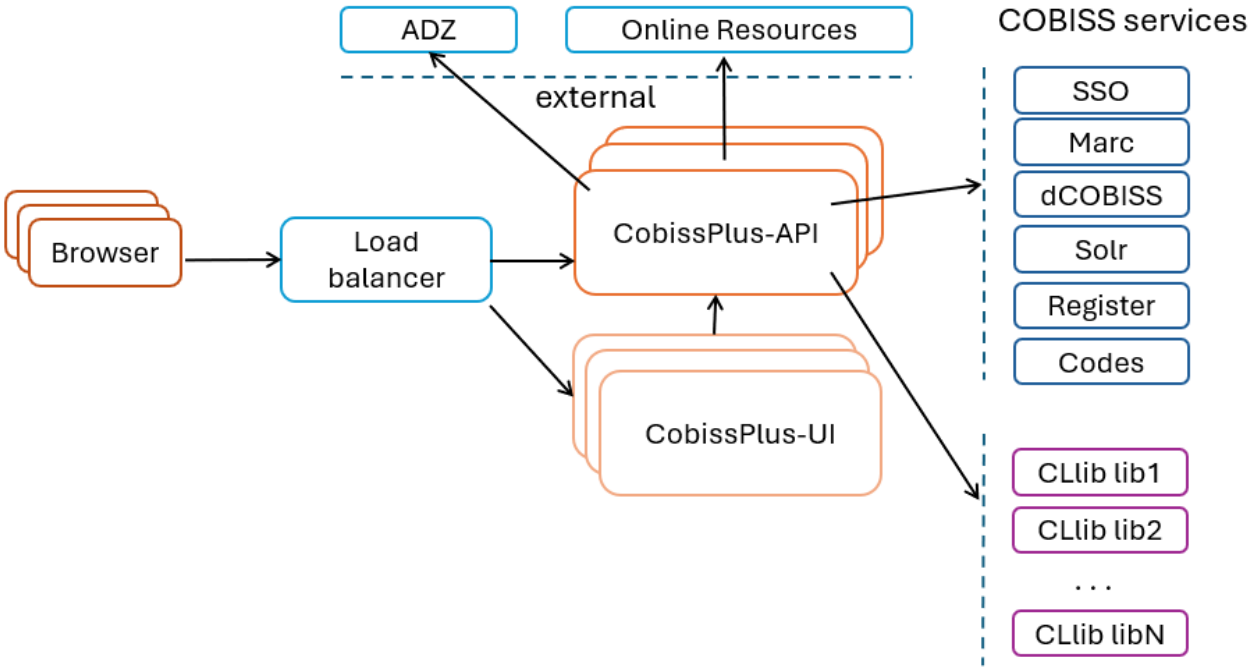



**Slika 1: Arhitektura obstoječe aplikacije COBISS+.**

## 2 Nova arhitektura COBISS+

V okviru prenove smo aplikacijo COBISS+ razdelili na dva dela. Prvi del je zaledna aplikacija CobissPlus-API, ki je napisana v Javi in teče na aplikacijskem strežniku WildFly ter skrbi za celotno poslovno logiko ter povezovanje različnih mikroservisov v enoten servis, ki je dostopen prek vmesnika REST. Drugi del aplikacije je CobissPlus-UI, ki je odjemalska aplikacija, napisana v TypeScript z uporabo ogrodja Next.js [2]. Vse interakcije med uporabnikom in aplikacijo potekajo na relaciji brskalnik – CobissPlus-UI, ta pa do podatkov dostopa prek vmesnika CobissPlus-API. Strežnik CobissPlus-UI se neposredno povezuje na CobissPlus-API za pridobivanje podatkov, ki so skupni za vse uporabnike aplikacije (npr. konfiguracija strežnika, seznam knjižnic, podatki o posameznem gradivu). Na strežniški strani smo Java EE 8 zamenjali z Jakarta EE 10.

Na sliki 2 je prikazana poenostavljena arhitektura delovanja aplikacije COBISS+.




**Slika 2: Nova arhitektura aplikacije COBISS+.**

## 3 Uporabniški vmesnik CobissPlus-UI

CobissPlus-UI je aplikacija, ki izrisuje uporabniški vmesnik. Za izris smo uporabili ogrodje Next.js, ki v ozadju uporablja knjižnico React, omogoča pa tudi izris HTML na strežniku. Podatki, ki so za vse uporabnike enaki, se pridobijo na strežniku z direktnim klicem na CobissPlus-API, preostali podatki pa se pridobivajo s klici iz odjemalca (brskalnika) na CobissPlus-API. V aplikaciji smo za usmerjevalnik (ang. router) uporabili Next.js app router, za upravljanje z asinhrono pridobljenimi podatki pa smo uporabili knjižnico Tanstack Query. Uporabniški vmesnik aplikacije je izdelan z ogrodjem Tailwind CSS, generične komponente pa temeljijo na zbirki komponent shadcn/ui. Implementacije teh komponent smo vključili v projekt že na začetku razvoja aplikacije, med razvojem pa smo jih še spreminjali in dopolnjevali.

Za lažji razvoj in sinhronizacijo med CobissPlus-UI in CobissPlus-API smo vse končne točke (ang. end point) na zalednem delu aplikacije opremili z anotacijami Swagger, kar nam omogoča avtomatsko izdelavo specifikacije OpenAPI 3.1 za celotno zaledno aplikacijo. V CobissPlus-UI na podlagi tega z uporabo knjižnice OpenAPI TypeScript generiramo definicije tipov TypeScript.

# 4 CobissPlus-API

CobissPlus-API omogoča dostop do skoraj 1500 knjižnic (servisi CLib). Prek teh servisov so omogočeni pregled knjižničnega gradiva, rezervacije gradiva, izposoja e-knjig itd. Servisi so dostopni prek lastnega protokola, ki je ovit znotraj HTTPS-ja. Združuje različne mikroservise, in sicer:

- Za vsako knjižnico teče aplikativni strežnik **CLib.**
- Storitev **Marc** omogoča tudi dostop do bibliografskih, normativnih in drugih vrst zapisov iz vzajemne baze in lokalnih baz knjižnic.
- Aplikacija **dCOBISS** je digitalni repozitorij, ki vsebuje polna besedila, slike, videoposnetke in druge digitalne vsebine.
- Jedro sistema je odprtokodni iskalnik **Apache Solr**, v katerem so indeksirane vse bibliografske in druge baze, ki so shranjene v MARC-obliki.
- Storitev **Register** omogoča dostop do metapodatkov same infrastrukture, storitev, podatkov o knjižnicah itd.
- Storitev **Codes** omogoča prevajanje sporočil v različne jezike in šifrante.
- **Online resources** so zunanji viri, kot so e-knjige, zvočne knjige, video posnetki.
- **ADZ (Akademska digitalna zbirka Slovenije)** omogoča integrirano iskanje podatkov po več milijonih različnih elektronskih virov svetovnih založnikov, digitalnih repozitorijev in tiskanih virov, ki so naročeni v slovenskih knjižnicah. Omogočen je dostop do celotnih besedil vsebin v odprtem dostopu, kot tudi licenčnih vsebin, za katere se zahteva avtoriziran dostop na osnovi IP-naslova institucije ali prijave AAI.
- Servis **SSO** omogoča enotno prijavo med vsemi servisi ter omogoča tudi avtentikacijo Shibboleth.

# 5 Robustnost aplikacije COBISS+

Problem pri takšni arhitekturi je, da lahko servis CobissPlus-API hitro postane nestabilen oz. performančno problematičen zaradi morebitnih problemov na odvisnih mikroservisih. Tudi sam vmesnik je lahko preobremenjen zaradi prevelikega števila zahtev uporabnikov oz. raznih zlonamernih botov. Dodaten problem je iskanje problemov na tako razvejani arhitekturi. Težavo smo poskušali rešiti oz. omiliti z uporabo ogrodja MicroProfile, ki je vgrajeno v aplikacijski strežnik WildFly.

## 5.1. Ogrodje MicroProfile

Java MicroProfile [3] je odprtokodna iniciativa, namenjena razvoju in izboljšavi mikroservisne arhitekture v okolju Java. Ponuja sklop specifikacij, ki omogočajo enostavno in učinkovito razvijanje ter upravljanje mikroservisov:

- **MicroProfile Config** omogoča dinamično konfiguriranje aplikacij brez ponovnega zagona. Konfiguracijske nastavitve so lahko v različnih podatkovnih virih.
- **MicroProfile Health** omogoča spremljanje stanja mikroservisov, kar olajša odkrivanje in odpravljanje težav.
- **MicroProfile Metrics** zagotavlja zbiranje in objavo podatkov o zmogljivosti in obnašanju aplikacij, kar pomaga pri spremljanju in optimizaciji delovanja.

- **MicroProfile JWT Propagation** omogoča varen dostop do mikroservisov z uporabo žetonov za preverjanje identitete in avtorizacijo.
- **MicroProfile Fault Tolerance** omogoča razvoj zanesljivih aplikacij z uporabo tehnik, kot so ponovni poskusi, časovni izklopi, izolacija krogotokov in razmiki med ponovnimi poskusi.
- **MicroProfile OpenAPI** omogoča generiranje in dokumentiranje API-jev, kar olajša integracijo in uporabo mikroservisov.
- **MicroProfile OpenTracing** omogoča sledljivost klicev znotraj mikroservisov in med njimi, kar pomaga pri diagnostiki in optimizaciji delovanja.
- **MicroProfile REST Client** omogoča enostavno klicanje storitev RESTful znotraj mikroservisov.

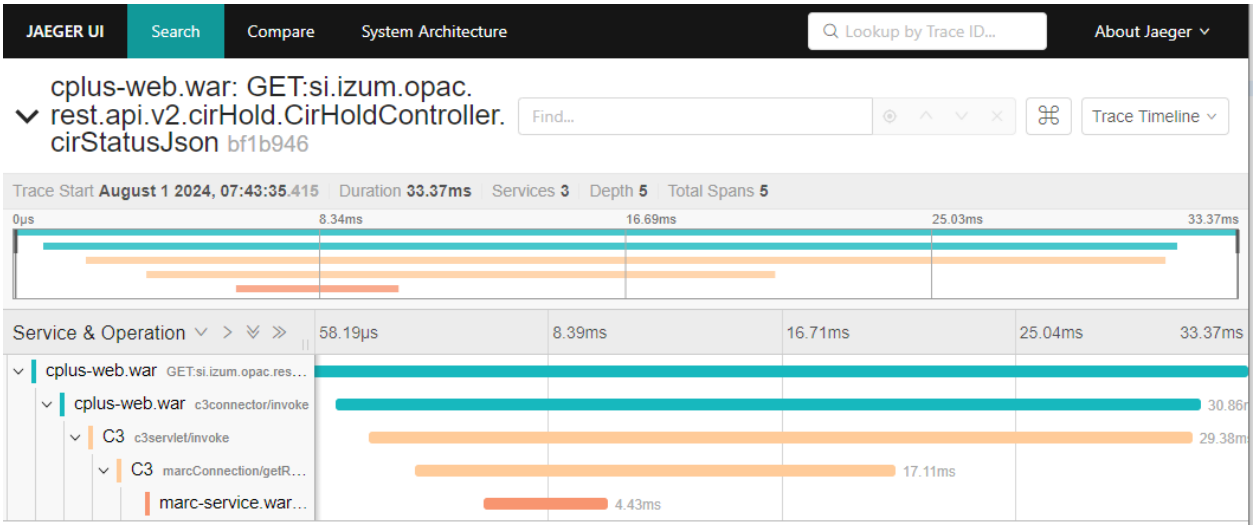
Nekatere od teh specifikacij smo v aplikaciji COBISS+ implementirali oz. nadgradili z lastnimi rešitvami. Rešitve MicroProfile kombiniramo z našimi lastnimi rešitvami za spremljanje in upravljanje infrastrukture COBISS (Monitor manager).

Za opozarjanje na napake oz. probleme uporabljamo odprtokodni sistem Icinga, ki omogoča spremljanje razpoložljivosti in zmogljivosti omrežnih storitev, strežnikov, aplikacij in drugih virov. Periodično se izvaja zdravstveni pregled (ang. Health Check) storitve COBISS+. Izvajamo sistemske in poslovne teste.

Sistemske in poslovne metrične podatke (ang. Metrics) zbiramo z odprtokodnim sistemom Prometheus, ki je bil zasnovan za monitoring in opozarjanje. Prikaz teh podatkov nam omogoča odprtokodno orodje Grafana, ki je namenjeno analitiki in vizualizaciji teh podatkov. Prometheus nam lahko prek Alert Managerja pošilja opozorila o morebitnih problemih aplikacije.

Specifikacija MicroProfile OpenTracing je implementirana z uporabo odprtokodnega programa Jaeger. Program je namenjen sledenju zahtevkov v porazdeljenih sistemih, kar razvijalcem omogoča analizo in odpravljanje težav v mikroservisni arhitekturi. Uporablja se za spremljanje in diagnosticiranje učinkovitosti ter zaznavanje napak v kompleksnih aplikacijah. Podatki se črpajo iz podatkovne baze Elasticsearch. To je zmogljiv in skalabilen odprtokodni iskalnik, ki omogoča shranjevanje, iskanje in analizo velikih količin podatkov v skoraj realnem času.

Na sliki 3 sta prikazana grafični vmesnik Jaeger in primer klica metode na servis CLib in preko tega na servis Marc. Klici metod na različnih strežnikih so prikazani v različnih barvah.




**Slika 3: Prikaz klica metode čez različne servise.**

V tem prispevku smo se osredotočili predvsem na implementacijo specifikacije MicroProfile Fault Tolerance (Toleranca napak).

### 5.2. MicroProfile Fault Tolerance

Specifikacija Fault Tolerance [4] zagotavlja nabor programskih modelov in vzorcev, ki razvijalcem pomagajo graditi odporne mikrostoritve, ki lahko obvladujejo napake na eleganten način. Uporablja anotacije za poenostavitev implementacije teh vzorcev. Specifikacija vključuje naslednje funkcije:

- **Timeout**: Funkcija omogoča nastavitev časovne omejitve za izvajanje metode. Če metoda traja dlje od določenega časa, se prekine in vrže TimeoutException.

```
@Timeout(500)
public String doSomething() {
// implementacija metode
}
```

- **Retry**: Funkcija omogoča samodejno ponavljanje izvajanja metode, ko ta ne uspe. Razvijalci lahko določijo število ponovitev, zamik med ponovitvami in pogoje, pod katerimi naj se ponovitve izvedejo.

```
@Retry(maxRetries = 3, delay = 1000)
public String callService() {
// implementacija metode
}
```

- **Circuit Breaker** (varovalka): Vzorec preprečuje storitvi, da bi večkrat poskušala poklicati neuspešno operacijo. Ko doseže določen prag napak, se odklopnik odpre in nadaljnji klici takoj ne uspejo brez poskusa operacije. Po določenem zamiku se lahko odklopnik premakne v pol-odprto stanje, da preizkusi, ali je operacija ponovno na voljo.

```
@CircuitBreaker(requestVolumeThreshold = 4, failureRatio = 0.75, delay = 1000)
public String getData() {
// implementacija metode
}
```

- **Bulkhead** (pregrada): Vzorec omejuje število sočasnih klicev metode, s čimer zagotavlja, da preobremenjeni del aplikacije ne blokira celotnega sistema.

```
@Bulkhead(value = 5)
public String processRequest() {
// implementacija metode
}
```

- **Fallback**: Funkcija omogoča definiranje alternativne metode, ki se izvede, ko glavna metoda ne uspe.

```
@Fallback(fallbackMethod = "fallback")
public String fetchResource() {
// implementacija metode
}
public String fallback() {
return "rezervni odgovor";
}
```

Vse te mehanizme smo vključili v klice naših storitev REST. Pri tem smo morali biti pozorni tudi na nastavitve aplikacijskega strežnika WildFly, saj ima lastne nastavitve za maksimalno število EJB-zrn.

```
<bean-instance-pools>
<strict-max-pool name="mdb-strict-max-pool" derive-size="from-cpu-count" instance-acquisition-timeout="5" instance-acquisition-timeout-unit="MINUTES"/>
<strict-max-pool name="slsb-strict-max-pool" max-pool-size="50" instance-acquisition-timeout="5" instance-acquisition-timeout-unit="MINUTES"/>
</bean-instance-pools>
```

Za klic do servisov naših knjižnic (CLib), ki ne poteka po protokolu REST, temveč prek našega lastnega protokola, nismo uporabili mehanizma MicroProfile Fault Tolerance, temveč smo razvili lasten mehanizem, ki deluje na podoben način. Pri tem smo izkoristili možnost specifikacije EJB, ki omogoča klic prestreznikov (ang. interceptor):

```
@AroundInvoke
public Object invoke(final InvocationContext context) throws Exception {
  String className = context.getTarget().getClass().getSimpleName();
  String methodName = context.getMethod().getName();
```

Ko sta MicroProfile Fault Tolerance in MicroProfile Metrics [5] uporabljena skupaj, se metrike samodejno dodajo za vsako metodo, ki je označena z anotacijo @Retry, @Timeout, @CircuitBreaker, @Bulkhead ali @Fallback. Imena za samodejno dodane metrike sledijo doslednemu vzorcu, ki vključuje polno kvalificirano ime označene metode.

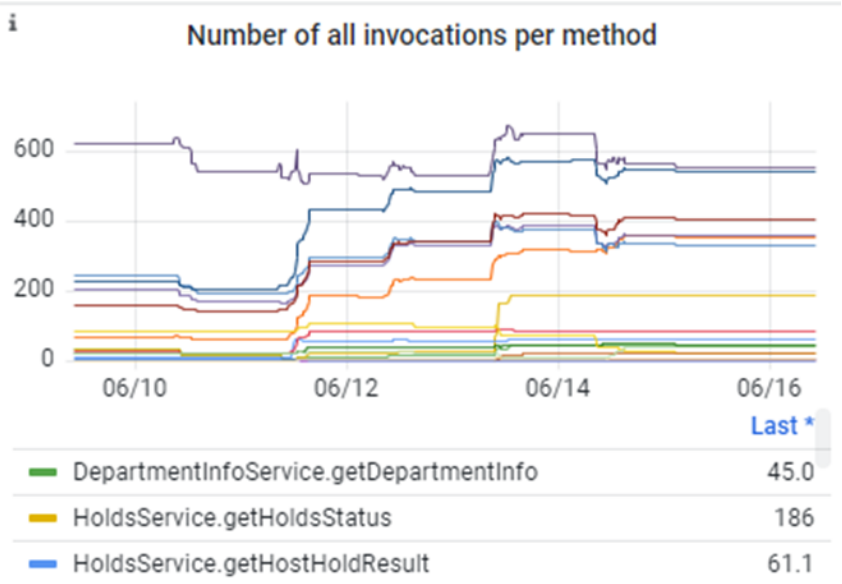
Primer metode in avtomatsko kreiranih metrik:

```
package com.exmaple;
@Timeout(1000)
public class MyClass {
@Retry
public void doWork() {
// work
}
}
```

Avtomatsko kreirane metrike:

```
ft.com.example.MyClass.doWork.invocations.total
ft.com.example.MyClass.doWork.invocations.failed
ft.com.example.MyClass.doWork.retry.callsSucceededNotRetried.total
ft.com.example.MyClass.doWork.retry.callsSucceededRetried.total
ft.com.example.MyClass.doWork.retry.callsFailed.total
ft.com.example.MyClass.doWork.retry.retries.total
ft.com.example.MyClass.doWork.timeout.executionDuration
ft.com.example.MyClass.doWork.timeout.callsTimedOut.total
ft.com.example.MyClass.doWork.timeout.callsNotTimedOut.total
```

Na osnovi teh metrik lahko s pomočjo Prometheusa in Grafane spremljamo odzivnost naše aplikacije oz. morebitne težave v njej. Slika 4 predstavlja primer prikaza matrik.




**Slika 4: Prikaz metrik v orodju Grafana.**

## 6 Zaključek

Pri prenovi spletne aplikacije za iskanje knjižničnega gradiva smo želeli predvsem prenoviti uporabniški vmesnik in izboljšati robustnost aplikacije.

Izkušnje pri razvoju uporabniškega vmesnika ločeno od zaledja so pozitivne, predvsem zaradi hitrosti iteriranja. Uporaba ogrodja Next.js pri izdelavi CobissPlus-UI nam je poenostavila razvoj interaktivne spletne aplikacije, težave pa so se pojavljale predvsem pri uporabi Next.js App Router, ki po naših izkušnjah še nima vseh funkcionalnosti, ki bi si jih pri razvoju takšne aplikacije želeli. Zaradi tega moramo včasih najti alternativne poti do rešitve.

Ogrodje MicroProfile Fault Tolerance lahko bistveno pripomore k boljši robustnosti aplikacij, zato nameravamo to ogrodje v prihodnje uporabiti tudi pri drugih aplikacijah v okviru sistema COBISS.

## 7 Literatura

[1] COBISS+, https://plus.cobiss.net/cobiss/si/sl/bib/search, obiskano 1.8.2024

[2] Next.js, https://nextjs.org/, obiskano 1.8.2024

[3] MicroProfile, https://microprofile.io/, obiskano 1.8.2024

[4] MicroProfile Fault Tolerance, https://download.eclipse.org/microprofile/microprofile-fault-tolerance-4.0/microprofile-fault-tolerance-spec-4.0.html, obiskano 1.8.2024

[5] MicroProfile Metrics, https://microprofile.io/specifications/microprofile-metrics/, obiskano 1.8.2024