Raspodijeljeni sustavi: Razvoj aplikacije u mikroservisnoj arhitekturi (online studij)

Rok: 11.1.2026.

Fakultet informatike u Puli

Potrebno je implementirati aplikaciju po izboru u mikroservisnoj arhitekturi prema smjernicama iz skripti `RS4`, `RS5`, `RS6`.

Mikroservisna arhitektura podrazumijeva razdvajanje funkcionalnosti aplikacije u manje, neovisne servise koji komuniciraju putem API-ja ili poruka. Svaki mikroservis treba biti odgovoran za specifičnu funkcionalnost i može se razvijati, testirati i deployati (postaviti) neovisno o drugim servisima. Studenti su dužni proučiti teorijski dio skripte RS5 i primijeniti dobre prakse u dizajnu mikroservisne arhitekture za svoju aplikaciju.

Mikroservise je preporučljivo organizirati u skladu s poslovnom logikom aplikacije, ali i prema tehničkim odgovornostima sustava. Takav pristup olakšava održavanje, skaliranje i daljnji razvoj. Primjeri tehničkih aspekata uključuju autentifikaciju i autorizaciju, obradu big data skupova podataka te integraciju s vanjskim servisima (npr. integracija s vanjskim cloud-servisom za provedbu plaćanja, slanje e-mailova ili SMS poruka).

Napomena: Pod pojmom "vanjski servis" podrazumijeva se svaka platforma, SaaS-rješenje ili API koji koristite u cloudu, a koji niste sami razvili (primjerice *Infobip* za slanje SMS poruka, *Stripe* za obradu plaćanja, *AWS S3* za pohranu binarnih datoteka i slično). Preporučuje se implementacija vlastitog mikroservisa koji prilagođava poslovnu logiku specifičnim potrebama aplikacije i koji služi kao integracijski sloj prema tim vanjskim servisima (pogledati primjer dolje).

Primjer: Aplikacija za e-trgovinu može biti podijeljena na mikroservise kao što su:

- `order-service`: servis za provedbu, upravljanje i praćenje narudžbi (ima integraciju s bazom podataka narudžbi i komunicira s vanjskim servisom za slanje SMS-ova)
- `auth-service`: servis za provedbu autentifikacije i autorizacije korisnika (npr. prijava, registracija, upravljanje ulogama; komunicira s vanjskim IAM servisom poput Auth0 ili AWS Cognito)
- `payment-service`: servis za obradu plaćanja (komunicira s vanjskim servisom za plaćanja poput Stripea ili PayPala)
- `product-service`: servis za upravljanje katalogom i zalihama proizvoda (integracija s bazom podataka proizvoda, upravljanje zalihama, praćenje KPI-ja, demand forecasting)

Uz to, ova aplikacija može imati korisničko sučelje (*frontend*) namijenjeno krajnjim korisnicima (kupcima) i administrativno sučelje namijenjeno djelatnicima (npr. upravljanje narudžbama i proizvodima).

Napomena: Korisničko sučelje možete implementirati kao zasebnu aplikaciju (npr. SPA aplikaciju u *Reactu*, ili *Vue.js-u*) koja komunicira s mikroservisima putem definiranih API-ja. Korisničko sučelje ne računajte kao zaseban mikroservis, budući da je fokus ovog zadatka prvenstveno na transformaciji *backend* arhitekture.

> Ako nemate prethodnog iskustva u razvoju korisničkog sučelja, možete pogledati materijale iz kolegija Web aplikacije (lekcija: Razmjena podataka između klijenta i poslužitelja) i popratni video te materijale iz kolegija Programsko inženjerstvo gdje se prolazi razvoj jednostavne SPA aplikacije u *Vue.js*-u, ili možete koristiti jednostavno HTML/CSS/JavaScript sučelje koje komunicira s mikroservisima putem *fetch API*-ja.

Zahtjevi:

1. **Dijagram arhitekture i komunikacije**: Pripremite dijagram mikroservisne arhitekture koji jasno prikazuje sve mikroservise, njihovu međusobnu (*service-to-service*) komunikaciju te integracije s vanjskim servisima. Dijagram treba biti jasan i lako razumljiv, u stilu dijagrama *e-commerce* aplikacije iz skripte RS5. Preporuka je koristiti alate poput *Excalidraw*, *Lucidchart* ili *diagrams.net*. Osim međusobne komunikacije, potrebno je prikazati i komunikaciju između mikroservisa za ključne poslovne funkcionalnosti aplikacije (npr. obrada narudžbe). Ovdje morate prikazati koji mikroservisi međusobno komuniciraju, na koji način (npr. REST API, gRPC, poruke) te znati objasniti zašto je odabrana ta vrsta komunikacije.e.
2. **Struktura repozitorija**: Svaki mikroservis definirajte u zasebnom direktoriju unutar repozitorija projekta. Mikroservis ne mora nužno biti implementiran u Pythonu; može biti i u drugom okruženju (npr. *Node.js, Java, Go*). Međutim, preporučuje se korištenje Pythona radi konzistentnosti s kolegijem. Moguće je kombinirati različite tehnologije (npr. *Node.js* servis za izradu PDF računa, Python servis za autentifikaciju korisnika) - **važno je znati objasniti zašto je odabrana određena tehnologija za pojedini mikroservis**. Svaki mikroservis mora imati polaznu datoteku (npr. `app.py`, `server.js`) gdje je definirano sučelje mikroservisa (npr. REST API endpoint kroz `aiohttp` ili `FastAPI`). Poslovne funkcionalnosti servisa trebale bi biti podijeljene u zasebne pakete/module. Također, svaki mikroservis treba imati vlastitu datoteku za upravljanje ovisnostima (npr. `requirements.txt`, `package.json`), upute za lokalno pokretanje i konfiguraciju (npr. `README.`md) te predložak varijabli okruženja (npr. `.env.example`) u kojem ćete navesti sve potrebne varijable za pokretanje mikroservisa (npr. baza podataka, API ključevi za vanjske servise). **Mikroservis koji se ne može pokrenuti lokalno ili nema navedenu dokumentaciju neće biti bodovan**. Ako postoji potreba za vanjskim servisom ili bazom podataka, potrebno je navesti upute kako to podesiti lokalno.
3. **Minimalni broj mikroservisa**: Implementirajte **barem tri mikroservisa (plus frontend)** koji zajedno pokrivaju ključne poslovne funkcionalnosti vaše aplikacije. **Iznimno je moguće implementirati minimalno dva mikroservisa (plus frontend)** ako arhitektura aplikacije to opravdava - u tom slučaju trebate znati argumentirano objasniti zašto je takav izbor prihvatljiv. Svaki mikroservis treba imati definirano sučelje (npr. REST API poslužitelj putem HTTP-a, poruke putem message brokera poput RabbitMQ-a ili Apache Karfe, ili real-time komunikaciju putem WebSocket-a), ili kombinaciju navedenog prema potrebama mikroservisa. Dakle, jedan mikroservis mora imati najmanje jedan API endpoint za komunikaciju s drugim mikroservisima ili korisničkim sučeljem. Svaki mikroservis treba sadržavati implementirane osnovne poslovne funkcionalnosti relevantne za njegovu ulogu u sustavu (npr. `order-service` treba imati funkcionalnosti za kreiranje, ažuriranje i dohvat narudžbi). Mikroservisi trebaju biti neovisni - promjene u jednom mikroservisu ne smiju utjecati na druge servise. Također, pad jednog mikroservisa ne smije onemogućiti rad drugih mikroservisa (npr. ako `payment-service` nije dostupan, `order-service` i dalje može primati narudžbe i spremati ih u bazu za kasniju obradu). Dobra praksa je da svaki mikroservis ima vlastitu *lightweight* bazu podataka (npr. SQLite, PostgreSQL, MongoDB) kako bi se osigurala neovisnost podataka i spriječile race condition situacije.

4. **Service-to-service komunikacija**: Mora postojati komunikacija između mikroservisa radi ostvarivanja zajedničkih poslovnih funkcionalnosti (npr. `order-service` komunicira s `payment-service` radi obrade plaćanja prilikom kreiranja narudžbe). Komunikacija može biti sinkrona (npr. REST API pozivi) ili asinkrona (npr. poruke putem message brokera). Važno je znati objasniti zašto je odabrana određena vrsta komunikacije za svaki konkretan slučaj. Taj dio morate jasno prikazati i u dijagramu arhitekture (točka 1).
5. **Lokalna razvojna razina**: Za sada ostajete na lokalnoj, razvojnoj razini aplikacije. Vaša aplikacija implementirana mikroservisnom arhitekturom ne mora biti deployana negdje u produkciju; vrlo je važno da se svi mikroservisi mogu pokrenuti lokalno uz jasno definirane upute za pokretanje svakog mikroservisa. **U README.md definirajte upute za izradu virtualnog okruženja** (**1 mikroservis = 1 Python virtualno okruženje**) te **verziju Pythona** koju je potrebno instalirati kako bi se ovisnosti iz `requirements.txt` uspješno instalirale i mikroservis ispravno radio.
6. **Dokumentacija svih API-ja**: Potrebno je dokumentirati sve API endpoint-e mikroservisa koristeći alate poput *Swaggera* (FastAPI) ili *Postmana*. Ako koristite Postman, izradite novo radno okruženje i kolekcije u kojima ćete definirati sve API endpoint-e za svaki mikroservis, kao i varijable okruženja (npr. `payment-service` radi na `localhost:8001`, `order-service` na `localhost:8002` i sl.). Ako koristite Postman, izradite javno radno okruženje i **podijelite link na taj workspace u glavnom README.md repozitorija**. Ukoliko koristite FastAPI, osigurajte da je automatska dokumentacija dostupna na /docs endpointu svakog mikroservisa.

Cjelokupni *codebase* mora biti objavljen na GitHubu/GitLabu. Zadatak možete definirati u zasebnom repozitoriju ili u repozitoriju iz vježbi. Ako koristite repozitorij iz vježbi, preporuka je izraditi zaseban direktorij gdje ćete pohraniti ukupnu mikroservisnu arhitekturu (uključujući frontend sučelje/sučelja, ako postoje). Važno je da svaki mikroservis bude u svom zasebnom direktoriju unutar repozitorija. **Git je obvezan sustav za verzioniranje**.

Primjer datotečne strukture:

```
.
├── auth-service
│   ├── README.md
│   ├── get_auth0_access_token.py
│   ├── helpers
│   ├── logging_setup.py
│   ├── requirements.txt
│   ├── server.py
│   ├── static
│   └── test.py
├── catalog-service
│   ├── README.md
│   ├── db.py
│   ├── fill_products.py
│   ├── helpers
│   ├── logging_setup.py
│   ├── products.db
│   ├── requirements.txt
│   └── server.py
├── frontend-app
│   ├── README.md
```

```
│   ├── dist
│   ├── index.html
│   ├── jsconfig.json
│   ├── node_modules
│   ├── package-lock.json
│   ├── package.json
│   ├── public
│   ├── src
│   └── vite.config.js
├── order-service
│   ├── README.md
│   ├── db.py
│   ├── helpers
│   ├── logging_setup.py
│   ├── orders.db
│   ├── requirements.txt
│   ├── server.py
│   └── websocket_handlers.py
└── user-service
    ├── README.md
    ├── db.py
    ├── helpers
    ├── logging_setup.py
    ├── requirements.txt
    ├── server.py
    └── users.db
```

Primjer možete pronaći na GitHub repozitoriju kolegija.

Studenti u sklopu ovog zadatka mogu ostvariti maksimalno 4 dodatna boda iz dijela vježbi.

Boduje se sljedeće:

- opseg zadatka, kvaliteta definiranih mikroservisa i poznavanje principa mikroservisne arhitekture
- kvaliteta dijagrama arhitekture i razumijevanje odabrane arhitekture te međusobne komunikacije mikroservisa
- kvaliteta dokumentacije API-ja mikroservisa
- kvaliteta programske implementacije mikroservisa, čitljivost i organizacija koda
- video prezentacija zadatka (detalji u nastavku)

Studenti moraju zadatak prezentirati **snimanjem kratke video prezentacije** (preporučeno trajanje: ~15 minuta) u kojoj trebaju:

- pokazati kako se aplikacija pokreće: sve potrebne korake za pokretanje i postavljanje pojedinog mikroservisa na lokalnoj razini
- prikazati dijagram definirane arhitekture i međusobnu komunikaciju izrađenih mikroservisa

- paralelno prolaziti kroz programski kod, skiciranu komunikaciju i demonstraciju rada aplikacije za glavne dijelove poslovne logike
- objasniti zašto su odabrane određene tehnologije, obrasci dizajna i vrste komunikacije između mikroservisa

Dodatne napomene:

- **Prezentaciju je potrebno učitati na YouTube te priložiti poveznicu u glavnom README.md repozitorija.**
- **Možete slobodno koristiti AI u svrhu bržeg razvoja aplikacije, međutim morate razumjeti i objasniti u video prezentaciji ključne dijelove programskog koda**
- **Izrazito neuredan AI-slop kod kažnjava se najstrože, uključujući gubitak mogućnosti polaganja kontinuiranim praćenjem te negativnim bodovima iz vježbi.**