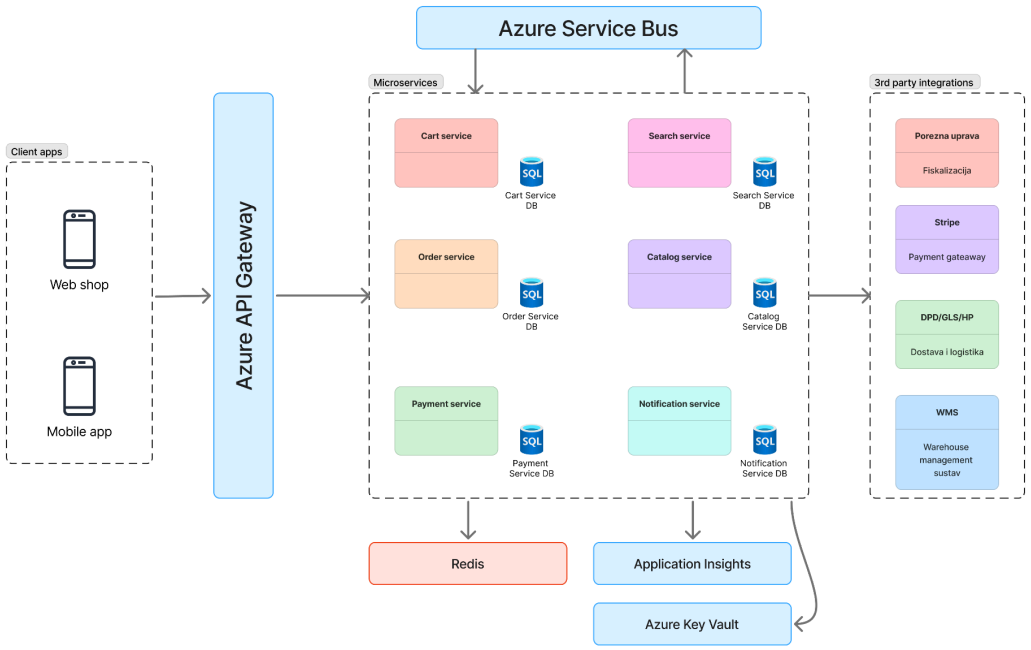
Online maloprodajna platforma

High-level arhitektura i strategija implementacije



1. Arhitekturalni pregled

Sustav sam zamislio kao skup manjih servisa koji međusobno komuniciraju putem API-ja. Glavni razlog za takav pristup je očekivani velik broj korisnika te činjenica da na projektu rade dva tima. Mikroservisni pristup omogućuje da timovi rade paralelno bez prevelikog međusobnog blokiranja.

Svaki servis je zasebna .NET 8 Web API aplikacija s vlastitom SQL Server bazom. Odlučio sam se za database-per-service pristup jer smanjuje ovisnosti između timova. Svjestan sam da to donosi kompleksnost kod eventualnih cross-service upita, ali smatram da je to prihvatljiv kompromis za skalabilnost i jasnu odgovornost nad podacima.

Komunikacija između servisa je kombinacija:

- REST poziva kada je potreban trenutni odgovor (npr. dohvat cijene proizvoda)
- Azure Service Bus eventova kada je moguće raditi asinkrono (npr. nakon kreiranja narudžbe)

Asinkroni pristup smanjuje međusobnu ovisnost servisa i povećava otpornost sustava ako neka komponenta privremeno nije dostupna.

Servisi se pokreću kao Docker kontejneri. Za orkestraciju koristim Azure Kubernetes Service jer omogućuje jednostavno horizontalno skaliranje i rolling deploy.

Za ulazni promet predviđen je API gateway (Azure API Management) koji centralizira autentifikaciju i routing. Time se pojednostavljuje logika unutar samih servisa.

---

2. Skaliranje

Svi servisi su dizajnirani kao stateless — stanje se čuva u SQL Serveru ili Redisu. To omogućuje jednostavno horizontalno skaliranje kroz Kubernetes Horizontal Pod Autoscaler.

Cache (Redis) koristim za:

- podatke o košarici
- čitanja iz kataloga proizvoda

Primijenjen je cache-aside pattern. TTL je postavljen razumno (npr. 15–30 minuta) kako bi se smanjilo opterećenje baze u peak situacijama.

Azure Service Bus omogućuje asinkrono procesiranje. Ako dođe do povećanja broja poruka, servisi se mogu skalirati horizontalno.

Nisam detaljno razrađivao multi-region strategiju ni disaster recovery, ali u stvarnom sustavu to bi bio sljedeći korak.

---

3. Sigurnost i autentifikacija

Autentifikacija je riješena putem OAuth 2.0 i OpenID Connect protokola. Kao identity provider predviđen je Azure AD B2C.

JWT token se validira na API gateway razini, a interni servisi vjeruju gatewayu kao ulaznoj točki.

Osjetljivi podaci (connection stringovi, API ključevi) pohranjeni su u Azure Key Vault. Time se izbjegava hardkodiranje tajni u aplikaciji.

Na razini baze koristi se Transparent Data Encryption. PII podaci minimalizirani su i čuvaju se samo gdje je to nužno.

Ovdje sam se fokusirao na osnovne sigurnosne principe; dodatne mjere poput detaljne revizije pristupa i fine-grained autorizacije razradio bih u kasnijoj fazi.

---

4. Ključne komponente

Sustav je podijeljen prema domenama:

Transakcijski servisi

Cart Service
Upravlja košaricom korisnika i izračunom ukupnog iznosa. Redis se koristi za brza čitanja, SQL je source of truth.

Order Service
Upravlja životnim ciklusom narudžbe. Nakon kreiranja objavljuje event.

Payment Service
Izolira logiku plaćanja. Ovaj servis komunicira s vanjskim payment providerom i minimalizira izloženost osjetljivih podataka.

Platform servisi

Catalog Service
Upravlja proizvodima i cijenama.

Search Service
Omogućuje full-text pretraživanje (npr. kroz Elasticsearch).

Notification Service
Konzumira evente i šalje email ili SMS poruke.

Identity je vanjska komponenta (Azure AD B2C).

Podjela servisa je pojednostavljena radi jasnog razgraničenja odgovornosti. U stvarnom sustavu granice bi se dodatno validirale kroz stvarne use-caseove.

---

5. Vanjske integracije

Integracije s vanjskim sustavima izolirane su kroz zasebne adapter komponente.
Cilj ovog pristupa je smanjiti ovisnost ostatka sustava o promjenama u vanjskim API-jima.

Svaka integracija ima jasno definiranu odgovornost i komunicira s ostatkom sustava putem REST poziva ili eventa.

Fiskalizacija (Porezna uprava)

Fiskalizacija je implementirana kroz zaseban Fiscalisation Adapter.

Nakon potvrde narudžbe, sustav šalje zahtjev prema sučelju Porezne uprave i dohvaća JIR broj koji se pohranjuje uz račun.

Ako je vanjski servis privremeno nedostupan, zahtjev se može ponoviti kroz jednostavan retry mehanizam. Time se izbjegava blokiranje korisničkog checkout procesa zbog kratkotrajne greške vanjskog sustava.

Svi odgovori fiskalizacijskog sustava pohranjuju se uz narudžbu radi sljedivosti.

Payment Gateway

Integracija s payment providerom (npr. Stripe) realizirana je putem službenog .NET SDK-a.

Payment Service je jedina komponenta koja komunicira s vanjskim sustavom naplate.
Zaštita od duple naplate rješava se korištenjem idempotency ključa pri svakom zahtjevu za naplatu.

U slučaju greške, sustav evidentira status transakcije te omogućuje ponovni pokušaj ili ručnu intervenciju.

Dostava (Shipping)

Integracija s dostavnim službama (npr. DPD, GLS) implementirana je kroz Shipping Adapter.

Nakon što je narudžba spremna za slanje, sustav šalje zahtjev za kreiranje pošiljke i dohvaća tracking broj.

Status isporuke može se ažurirati putem webhooka ili periodičkim dohvatom podataka.
Ažuriranja statusa se pohranjuju uz narudžbu.

Ako je API dostavne službe nedostupan, zahtjev se ponovno pokušava u razumnim vremenskim intervalima.

Warehouse Management System (WMS)

Integracija sa skladišnim sustavom implementirana je kroz WMS Adapter.

Nakon kreiranja narudžbe, sustav šalje zahtjev za rezervaciju zaliha.
Ako rezervacija uspije, narudžba prelazi u sljedeću fazu obrade.

Sinkronizacija stanja zaliha može biti:

- event-driven (ako WMS podržava notifikacije)
- ili periodička (batch sinkronizacija)

Time se osigurava da stanje zaliha u sustavu ostane usklađeno sa skladištem.

---

6. Monitoring

Za monitoring koristim Application Insights.

Praćenje uključuje:

- latenciju
- error rate
- ovisnosti (SQL, vanjski API-ji)

Svaki servis ima:

- /health/live
- /health/ready

Alerting bi se konfigurirao na osnovne metrike poput povećanog error ratea ili visoke latencije.

Nisam ulazio u detalje distributed tracinga i naprednog observability stacka jer smatram da je za početnu fazu dovoljno centralizirano logiranje i osnovni monitoring.

---

7. Plan isporuke

Koristi se GitHub Flow.

- Feature grane

- Pull request review
- Main je uvijek deployable

CI pipeline uključuje:

- build
- testove
- Docker build
- push u registry

CD pipeline deploya na dev automatski, dok staging i production zahtijevaju manualnu potvrdu.

Za deploy koristim rolling update strategiju. Blue-green i canary pristupi mogu se uvesti kasnije ako promet i kompleksnost to zahtijevaju.

Database migracije su verzionirane SQL skripte u repozitoriju.