Opis arhitekture sustava

Za arhitekturu sustava bih izabrao da bude implementirana kao modularni monolit sa jasno definiranim granicama svakog modula. Klasični monolit bi imao prednosti bržeg inicijalnog razvoja, ali bi u budućnosti razvio probleme teškog skaliranja, tight coupling u sustavu i sporijeg developmenta zbog kompleksnosti sustava. S druge strane mikroservisna arhitektura bi pružila značajne prednosti u čistoj organizaciji koda te u budućnosti bi pružila jaku podršku za fleksibilno skaliranje. Nažalost ovo ne dolazi bez cijene. Kompleksnost takvog sustava bi značila da bi razvoj do inicijalnog proizvoda trajao duže nego kod monolita, a isto tako distribuirani sustavi traže velike resurse u vidu orkestracije I monitoring sustava.

Zbog svega ovoga modularni monolit bi pružio dobre karakteristike obje arhitekture. Zbog svoje jednostavnosti inicijalni razvoj do proizvoda bi bio relativno brz, dok bi s druge strane jaka disciplina oko modularnosti I jasnih granica u sustavu nudila mogućnost skaliranja, a po potrebi I separacije sustava u mikroservise u budućnosti. Granice između modula I biznis pravila bih podržao DDD-om.

Jedina situacija gdje bih razmišljao o inicijalnom razvoju projekta u mikroservisnoj arhitekturi jest da svi developeri u oba tima imaju puno iskustva u toj arhitekturi.

Sustav bi se sastojao od sljedećih slojeva:

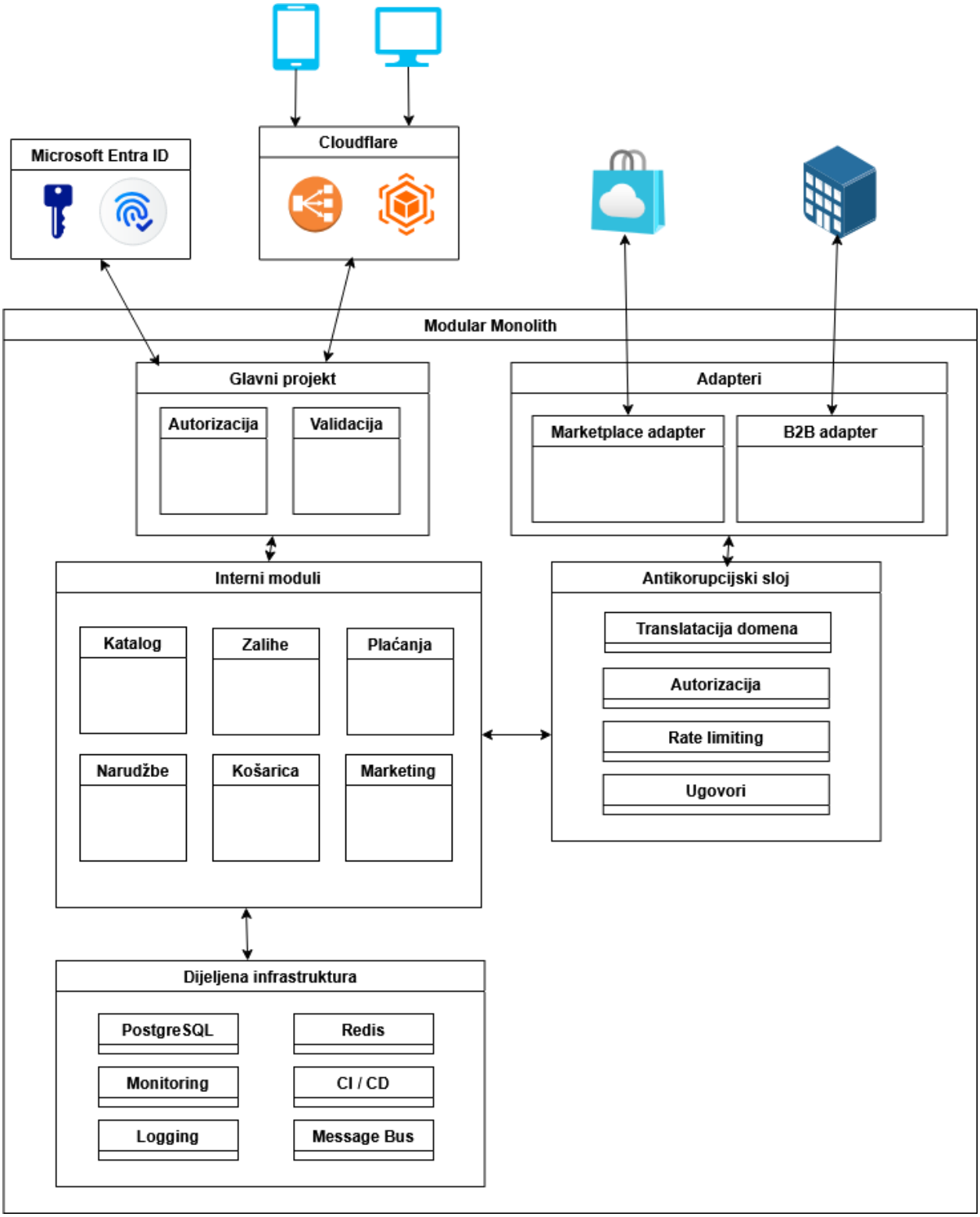
- Ulaz u system realiziran upotrebom Cloudflare ili sličnog sa podrškom za load balancing, prevenciju DDOS napada, CDN i WAF
- Glavni projekt koji učitava module i brine za stvari od značaja za sve module kao logiranje, autorizaciju, swagger dokumentaciju I slično
- Moduli – svaki od modula sadrži vlastite API endpoint-e te Application -> Domain -> Infrastructure DDD slojeve.
- Podaci – svaki od modula ima vlastitu shemu unutar PostgreSQL-a i zasebni ključ za keširanje u Redis-u

Komunikacija između komponenti može biti sljedeća:

Sinhrona (in-process pozivi) – za brzi dohvat podataka između modula na koje se ne može čekati. Za ovakvu komunikaciju se koriste Service klase (npr. ICatalogService) koje su implementirane unutar pojedinog modula. Nema poziva baze ili privatnih klasa modula.

Asinhrona (event-driven) – nešto se desilo u sustavu i drugi moduli bi trebali reagirati. Za ovu vrstu komunikacije koristimo Mass Transit s in-memory shemom za početak te opcijom za bezbolni prelazak na message brokere po potrebi kasnije. Ovakva komunikacija je podesna za duže taskove ili kad želimo slabije veze između modula.

Dijagram arhitekture sustava



Odabrane tehnologije

Tehnologije za inicijalni razvoj:

- .NET 8 ili 10
- PostgreSQL kao primarna SQL baza
- Redis kao distribuirani keš
- MassTransit za messaging

Tehnologije za uvođenje u kasnijim etapama:

- RabbitMQ za potrebe messaging-a
- ElasticSearch za potrebe pretraživanja

Strategija skaliranja

Skaliranje modularnog monolita podrazumijeva dizanje novih instanci monolita koje se sve nalaze iza load balacer-a s mogučnošću automatskog skaliranja. Za ovakav sustav je nužno da je stateless, a za snimanje podataka vezanih za sesiju koristimo snimanje u distribuirani keš Redis.

Globalnu strategiju keširanja provodimo u dva sloja. Za slike, video, datoteke ili statičke resurse koristimo keširanje u CDN sustavima dok za distribuirani keš I oterećenje baze koristimo Redis.

Strategija skaliranja baze podataka je od velike važnosti jer će baza prva potpasti pod veliko opterećenje. Dostupne su nam sljedeće operacije:

- Prva linija obrane se sastoji od odvajanja read i write operacija upotrebom primarne baze za operacije pisanja i dizanja višestrukih read replica kojećemo koristiti za operacije pisanja.
- Budući da su baze separirane u zasebne sheme za svaki pojedini modul, shemu koja je pod najvećem opterećenjem možemo izdvojiti na zasebnu mašinu ili raditi sharding.
- Za mogućnost simultanog velikog broja konekcija možemo koristiti connection-pooling alate kao PgBouncer.

Za podršku asinhronih operacija bi se koristio Mass Transit zbog fleksibilnosti. On omogućava apstraktni nivo asinhronih poruka koje mogu biti u praksi realizirane in-memory ili preko message queue-ova, a također omogućava i garantiranu dobavu poruka preko implementacije transactional-outbox pattern-a preko baze podataka. Uz to omogućava ponovne pokušaje i prekid komunikacije prema potrebama.

Sigurnost

Za autentikaciju i autorizaciju u sistemu koristio bih kombinaciju OAuth 2.0 + OpenID Connect zajedno sa JWT tokenima. Koristio bih uslugu vanjskog identity providera zbog:

- Povjerenja u provjerene sigurnosne platforme
- Jednostavna skalabilnost
- Podrška za razne tipove klijenata (B2C, B2B, Marketplace)
- Out-of-the-box compliance

Provjereni vanjski identity provideri su: Microsoft Entra ID, Okta i Auth0.

Ako kompanija ima uhodani interni identity provider onda bi zbog manjih troškova to bio optimalniji izbor.

Autorizacijski model bi uključivao kombinaciju Role-Based, Scope-Based i Policy-Based Access Control za pokrivanje svih scenarija te bolji decoupling sigurnosti od posovne logike.

Komponente unutar sistema bi bile zadužene za validaciju ispravnosti tokena, provjeru identiteta te primjene autorizacije na modulima u sistemu.

Svaki API modul bi implementirao vlastitu provjeru autorizacije (nema povjerenja između modula), striktnu provjeru i validaciju ulaznih podataka za sprječavanje injection napada te rate limiting.

Za transport bi se koristili isključivo TLS, HTTPS i secure headers (HSTS i CSP).

Za vanjske integracije bi se za svakog klijenta dodatno radio IP whitelisting i potpisivanje sa simetričnim ključem.

Za B2B korisnike bi se koristili specifični tokeni koji omogućavaju pregled proizvoda i povlaštenih cijena koju su dogovoreni trazličitim ugovorima.

Centralizirano audit logiranje bi uključivalo najvažnije akcije na sistemu kao što su pokušaji logiranja na sistem, promjena prava pristupa, akcije admin korisnika, stvaranje narudžbe i izvršavanje plaćanja.

Glavne komponente sustava

Prijedlog modula je sljedeći:

- Identitet i prava pristup – koristi se za mapiranje korisnika i prava pristupa sa vanjskog identity provider-a na naš sustav.
- Katalog – barata proizvodima, kategorijama i atributima. Jako je orijentiran na čitanje podataka. Jaki kandidat za kasniju optimizaciju upotrebom NoSQL-a ili ElasticSearch-a.
- Zalihe – upravlja trenutnim zalihama proizvoda, rezervacijom kod narudžbi te rasporedom proizvoda po lokacijama.
- Košarica za kupovinu – barata privremenim podacima o proizvodima koje se planiraju kupiti. Mijenja se brzo I često I zahtjeva veliku brzinu pisanja podataka. Najbolji je kandidat za selidbu implementacije u distribuirani keš Redis.
- Narudžbe – bavi se kompleksnim sustavom narudžbe koja prolazi kroz različite faze u kojima treba surađivati s drugim modulima.
- Plaćanja i fiskalizacija – barata sa vanjskim servisima za plaćanje i integracijom s poreznom upravom. Intenzivan po pitanju sigurnosti i usklađenosti za različitim zakonskim regulativama.
- Marketing i promocije - barata s popustima, trenutnim akcijama te skupljanjem bodova lojalnosti. Izazov leži u kompleksnim biznis pravilima.

Monitoring i alerting

Monitoring bih bazirao na sljedećem:

- Strukturirano logiranje u JSON-u upotrebom Serilog-a uz obavezno korištenje podataka kao moduleId, correlationId i userId.
- OpenTelemetry za prikupljanje podataka o broju upita, postotku upita sa greškama te podacima o korištenju resursa sustava.
- Distribuirani tracing za praćenje pojedine akcije u sutavu I vremena izvršavanja.

Health-check bi na razini modula proveo provjeravanjem života procesa uz dodatne provjere da li su povezani sustavi dostupni (Redis cache, API pšorezne uprave I slično).

Alerting bi se bazirao na informacijama o latenciji sistema, opterećenju sustava, prometu I količini grešaka. Nužno ga je organizirati tako da se uzima u obzir koliko je stvarno ozbiljan problem u sutavu. Za kritične probleme se šalje poziv/SMS, dok se za manje kritične stvari mogu slati poruke u Teams/Slack kanal.

Provjereni alati koji bi pokrili zahtjeve ove domene su OpenTellemetry, Grafana i Prometheus.

Plan isporuke koda

Za branching strategiju bih koristio Trunk-Based Development uz korištenje Feature Flag-ova. Ovakav pristup olakšava isporuke I izbjegava problem kod izazovnih merge-anja. Koristio bih jedan git repozitorij za cijeli project (sve module).

Continuos Integration (CI) bi sadržavao sljedeće korake:

- Linting i formatiranje
- Unit testiranje
- Dodatni unit testovi za testiranje arhitekture sustava (NetArchTest) i da li su narušena pravila modularnosti
- Integracijsko testiranje
- Provjeru vulnerabilities kod Nuget paketa i ostalih dependency-ja

Za Continuos Delivery (CD) bih koristio Blue-Green deployment.