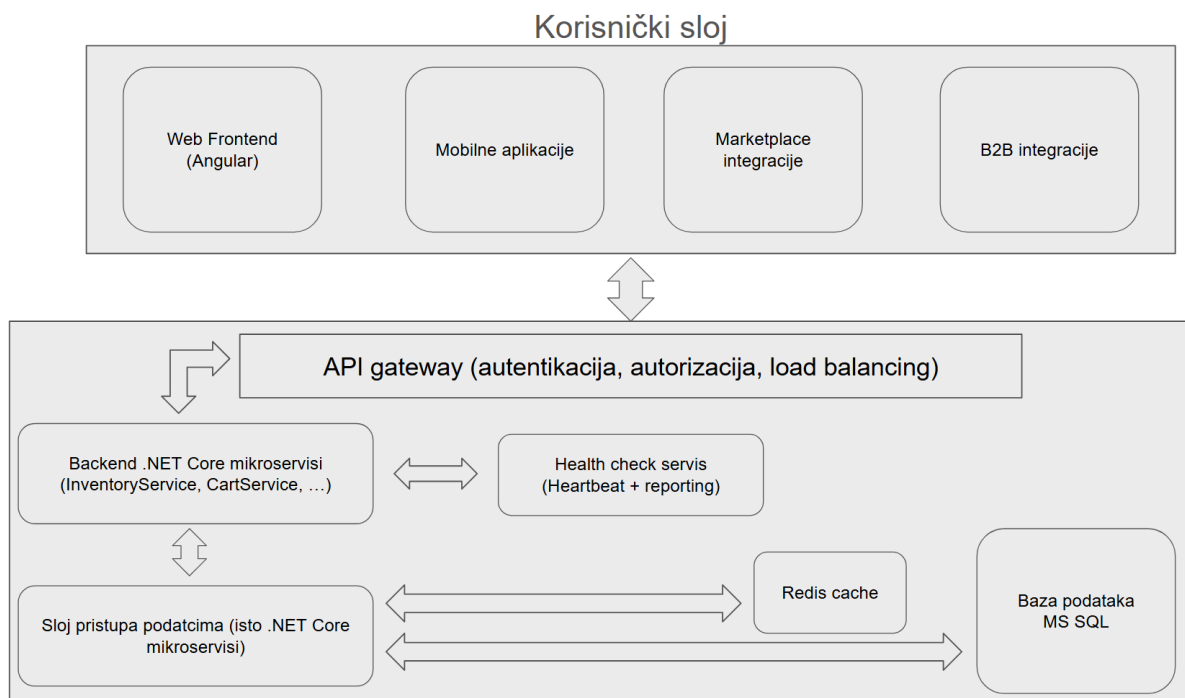


## Tehničko testiranje - dokumentacija rješenja

High-level skica sustava:



### Strategija implementacije:

Korisnički sloj biti će implementiran u različitim tehnologijama (npr. Angular web app, React Native mobilna aplikacija, ...), a sve od njih komunicirati će s backendom preko unificiranog web API-ja kroz HTTP zahtjeve.

Svaki od tih zahtjeva trebao bi sadržavati autentikaciju (JWT) radi povezivanja informacija korisnikovih zahtjeva s njihovim identitetom, ali i za autorizaciju koja je potrebna za neke akcije (na primjer, ažuriranje baze proizvoda).

Budući da je očekivan visoki promet, sam će backend trebati biti implementiran na distribuirani način, kako bi zahtjevi koji pristižu mogli biti preusmjeravani mnogim instancama backend mikroservisa.

Za perzistenciju će biti korištena MS SQL baza podataka povezana s backendom kroz data access mikroservise kojima je jedina dužnost koristiti Entity Framework Core za komunikaciju s bazom.

Međutim, kako baza ne bi bila preopterećena, biti će implementirana i Redis priručna memorija, u koju će se upisivati relevantna preslika stanja baze podataka bez potrebe za stalnim upitima prema bazi, što bi trebalo ubrzati tok podataka kroz cijeli sustav.

Za provjeru statusa sustava upogonit će se servisi za periodičko slanje heartbeat poruka servisima, kako bi se ustanovilo je li neki servis kompromitiran ili u kvarnom stanju. U slučaju stanja sustava koje ne odgovara željenom, ti servisi će poslati poruku na za to predviđenu e-mail adresu koju nadgleda osoblje, i također, ako je to potrebno, poslati signal za pokretanje nove instance servisa u kvaru.

Komunikacija između komponenti unutar sustava biti će implementirana kroz event streaming (Kafka) ili message queue protokol (AMQP) kako bi se ispoštovali ključni zahtjevi sustava (niska latencija, visoki promet) bez narušavanja integriteta sustava.

Integracija s vanjskim sustavima biti će implementirana kroz zasebne servise za komunikaciju unaprijed dogovorenim protokolima - mnoge takve integracije nije moguće uglaviti u ranije navedeni unificirani web API jer ne podržavaju HTTP kao protokol prijenosa podataka. Biti će proučene i popisane specifikacije za integraciju s tim servisima i implementirani podatkovni modeli koji tim specifikacijama odgovaraju (npr. UBL e-računi za fiskalizaciju 2.0).

Sam sustav naplate na webshopu biti će implementiran upravo kroz jednu takvu integraciju s vanjskim povjerljivim sustavom za naplatu, što omogućava sigurnost transakcija kroz već implementirani sigurni sustav (npr. CorvusPay, WSPay) koji, iako nije u našim rukama, predstavlja bolje rješenje nego in-house servis za naplatu (osim ako se firma bavi time kao glavni proizvod)

Grane na Git repozitoriju biti će posložene na način da postoji main grana, to jest, produkcijska grana, koja služi isključivo za produkcijski kod. Kod u toj grani je podložen CI/CD procesu kroz pipelines (npr. kroz Azure DevOps) kroz koje se kod builda i deploja automatski ili uz autorizaciju razvojnog tima. Druge grane uključuju "develop", to jest, razvojnu granu, koja sadrži kod koji nije namijenjen za produkciju - barem ne dok se kroz autorizirane pull requestove ne spoji u main. Grana develop služi za ujedinjavanje i testiranje implementiranih značajki sustava, svaka od kojih se za vrijeme razvoja nalazi u svojoj zasebnoj feature/feature-name grani. Nakon završetka razvoja, svaka značajka se kroz autorizirani pull request spaja s granom develop.

Rješenja za upogonjavanje svih navedenih dijelova sustava mogu se u unificiranom obliku naći u jednom od online ekosustava kao što su AWS ili Microsoft Azure. Koristeći te sustave moguće je na jednom mjestu dobiti hosting, kontejnerizaciju, message queue servise, event hub servise, monitoring servise, automatsko horizontalno i vertikalno skaliranje sustava, različite pristupe pohrani podataka i još mnogo toga.