Proof of Concept

Skalabilnost sistema za rezervaciju letova/hotelskog smeštaja/vozila zavisi u najvećem delu od arhitekture sistema. Optimalno rešenje za veće sisteme koji imaju jasno odvojene celine (podsisteme) je mikroservisna arhitektura.

Jedan od prvih koraka kod projektovanje takve arhitekture je određivanje nivoa granularnosti sistema. Prilikom određivanja nivoa granularnosti izuzetno je važno pronaći optimalno rešenje koje će sistem ubrzati do zadatih granica, ali u isto vreme učiniti (zadržati) pouzdanim. Ukoliko previše usitnimo sistem i kreiramo veći broj izuzetno malih mikroservisa, možemo otežati održavanje i nadogradnju jednog takvog sistema.

U slučaju ovog projekta mikroservisi koje u ovom nacrtu predlažemo su:
Reservation*, Search*, Rating*, Auth, Flight, Hotel, RentACar

Da bi upotpunili mikroservisnu arhitekturu, potrebno je impementirati *discovery server*, kao i *gateway (proxy server)* koji će sakriti URL-ove naših mikroservisa i izložiti u javnosti samo jedan jedinstven URL. Discovery server nam je neophodan zbog međusobne komunikacije mikroservisa, ali i zbog implementacije *load balancer*-a koji bi koristeći neki od poznatih algoritama ravnomerno raspoređivao pristigle zahteva na određene instance mikroservisa.

Ovakav sistem nam omogućava skaliranje prema zahtevima koji bi mogli da se menjaju iz dana u dan, a u nekim slučajevima i iz sata u sat. Uz danas dostupne alate bi pratili opterećenost sistema i po potrebi uz adekvatan alat za orkestraciju podizali (ili gasili) dodatne instance određenih mikroservisa. Na primer, u periodu kada se očekuje povećan broj posetilaca koji žele da kupe avionske karte, možemo podići dodatne instance *Search Flight* i *Flight Reservation* mikroservisa. Skaliramo prema trenutnom/očekivanom stanju na tržištu.

Nakon ispravno implementirane mikroservisne arhitekture, pažnju treba posvetiti asinhronosti sistema. Sve funkcije koje sistem može da obavi asinhrono bez posledica po poslovnu logiku treba i implementirati na asinhron nacin. Operacije koje sistem treba da obavi se mogu ubacivati u redove i onda kada se skupi određena količina operacije izvršiti ih zajedno. Dakle, odlažemo izvršavanje onih operacija koje ne moraju biti izvršene u realnom vremenu. Na taj način rasterećujemo sistem i povećavamo brzinu odziva sistema.

Na mestima gde učitavamo veliku količinu podataka u jednom prolazu uvesti *lazy load* i na taj način podatke učitavati parcijalno i samo onda kada su oni neophodni za interakciju sa sistemom (angular/spring lazy load).

Uvesti keširanje podataka i time znatno ubrzati rad aplikacije.

*napraviti za svaki modul (flight, hotel, rentacar)