# Uvod

Cesto prilikom ovih system design intervjua ume da se desi I da nema tacnog odgovora nego se testira kompunikacija I problem solving skills pa vole da ih ubacuju u intervjue.

Bices ocenjen na osnovu toga koliko dobro analiziras nejasne probleme I kako resaves te probleme korak po korak.   
Takodje testira se i kako objasnjavas ideje, razgovaras sa drugima, evaluiras i optimizujes sistem.

The system design questions are open-ended.

Smer intervjua zavisi od osobe koja te intervjuise. Neki zele high level arhitekturu da se pokriju svi apsekti dok neki zele fokus na odredjenu sekciju.

Tipicno, system requirements, constraints & bottlenecks sistema koji “osmisljavas” treba dobro da se razumeju kako bi se oblikovao smer intervjuera i intervjuisanog.

## Chapter 1: Scale from zero to millions of users

Ovde cemo napraviti sistem koji podrzava jednog usera i lagano ga scaleovati na milione usera.

### Single server setup

Ovo je prvi korak.

Da bi zapoceli sa necim jednostavnim, prvomoramo imati jelte jednostavno.   
Slika ispod predstavlja single server setup gde je sve na jednom server; web app, db, cache i slicno.

A diagram of a computer system

AI-generated content may be incorrect.

Sta se desava:

1. Prvo user posalje DNS(Domain Name Server)-u upit za dobijanje IP adrese naseg servera. DNS je paid service kog daje neki 3rd party i ne hostuje se nasim serverima.
2. IP(Internet Protocol) adresa se vraca uredjaju.
3. Jednom kad je IP adresa pridobijena, salje se HTTP(Hyper Text Transfer Protocol) request direktno na tvoj server
4. Web server vraca HTML stranicu/JSON response za renderovanje

Pogledajmo traffic source. To su web application & mobile application.

* Web Application koristi kombinaciju server side jezika da resava “biznis logiku”, skladistenje i client side jezika(HTML & JS) za prezentaciju.
* Mobile application: HTTP protocol je komunikacioni protokol izmedju mobilne aplikacije i web servera. JSON je cesto koristeni API response format.

### Database

Recimo rastu ti korisnici. Jedan server nije dovoljan i sad nam treba vise servera. Jedan za web/mobile traffic, a drugi za bazu. Razdvajanjem web/mobile traffic(web tier) & baze(data tier), serveri sada mogu nezavisno da skaliraju.

A diagram of a computer network

AI-generated content may be incorrect.

### Koju bazu koristiti

Relaciona & nerelaciona.

Relacione znas vec sta su.   
Nerelacione(NoSQL)(CouchDB, Neo4j, Cassandra, HBase, Amazon DynamoDB) se dele u x4 kategorije:

1. Key-value stores
2. Graph stores
3. Column stores
4. Document Stores

Join operacije ovde cesto nisu podrzane.

Razlozi da se odlucis za nerelacione baze:  
- Treba ti super low latency  
- Podaci su ti nestruktuisani ili nemas “relacione” podatke  
- Trebas samo da serializujes i deserializujes podatke(JSON/XAML/YAML etc..)  
- Trebas da skladistis velike(massive) kolicine podataka

### Vertical vs Horizontal scale

Vertical scaling(scale up) znaci dodavanje snage procesorom/ramo memorijom na serveru.

Horizontalno(scale out) znaci dodavanje servera u tvoj pool resursa.

Kad je saobracaj nizak, vertikalno skaliranje je skroz okej i jednostavnost ove opcije je njena glavna prednost. Nazalost, ima ogranicenja:  
- Ima hard limit. Nemoguce je dodati “bezbrojno” CPU-a i RAM-a na single server  
- Vertical scale nema failover i redundanost. Ako jedan server pukne, puca sve.

Horizontalno skaliranje je pozeljnije za vece aplikacije upravo zbog ovih ogranicenja.

Znaci u gore primeru su se korisnici kacili na x1 server. Ako je taj server nedostupan, ne mozes mu pristupiti. Ako se vise korisnika prikaci na server istovremeno i dostignes web serverov load limit, korisnici iskuse generalno sporiji response ili ne mogu da se povezu na server.

Za taj problem, resenje je **LOAD BALANCER.**

### Load balancer

Load balancer podjednako rasporedjuje dolazeci saobracaj izmedju web servera koji su definisani u “load-balanced set”.

A diagram of a network

AI-generated content may be incorrect.

U gore slici je definisano kako load balancer izgleda.

Korisnici se povezuju na public IP load balancera direktno. Na taj nacin, web serverima se vise ne pristupa direktno od strane klijenata. Za bolji security, private IP’s se koriste za komunikaciju izmedju servera.

Private IP je IP adresa kojoj mogu pristupati samo serveri unutar iste mreze, ali ne i van te mreze(e.g. internet).

Posle load balancera i dodavanja drugog servera, sada smo resili problem failovera i poboljsali dostupnost **web tier**-a.

* Ako server 1 ode offline, sav saobracaj usmerice se na server 2. Na taj nacin website nece biti offline. Takodje dodacemo novi zdravi server u server pool da bi se balansiralo opterecenje
* Ako website saobracaj naglo poraste i dva servera nisu dovoljna da izdrze taj saobracaj, load balancer moze ovaj problem lako da sredi dodavanjem novih servera u web server pool kojim ce LB kasnije balansirati.

To je nas **web tier**. Sta je sa **data tier**? Trenutni dizajn ima samo x1 database bez failovera i redundantnosti. Database replication je cesta tehnika za resavanje tih problema.

### Database replication