- **System specs**
  - let's assume we will start with a server has 4 core / 8 threads
  - let's assume the Database:
    - **Type**:
      - maria db, postgresql, mysql -we can use any type of database but my experience with these database make me capable of doing anything
    - **Structure**:
      - will be just one table, called shortened_links
      - id, actual_link, encoded_link, frequency, country, created_at

- first case : (**10 requests per second**)
  - **how system will work**
    - when user insert new link, we'll insert it at the database directly, we don't need multiple instances of the database at this case, we don't need replication or sharding

    - when user insert new link the encoded_link will consist of
      - we'll use base 62 encoding and decoding algorithm because it has less encoding characters to generate, so the shortened link will be good looking, we can any algorithm.
      - base62_encode(md5(actual_link).”-”.created_at)
      - example : google.com will be
        - base62_encode(md5(“https://google.com”).”-”.”2022-06-15 12:35:00”)
        - then new link will be
        - **https://newsystem.com/links/Oso3ApZGW8ZfqbcmrJKPj1VW71x0O9Y**

      - **why did I make this structure, i'll explain it at the reading from database section**

    - when user use shortened-link, the system will make a direct query to the database, but we'll use the data from encoded_link to add where condition also with the created_at column to make it's easy to the database because we'll make index at created_at and make another index encoded_link
      - I know a lot of indices will make writing is relatively-slow, but I can tolerate with that
      - why did I want to make an index at created_at, because index will be work as faster as possible when we use integers like timestamp
      - and in the scaling stage, we'll use replication or sharding and the main key i'll relay at this process will be the created_at column
    - when database returns the link, we'll update the frequency to make it increase by one, we'll use this key at the scaling process
  - if we have one instance of that server it won't make any performance issues at writing at database or at reading from database or at redirect to the actual link.

- **WOW, Congratulations, our system now is so popular, oh-god, we need to scale, no-worries we'll do it.**

- Second case (1000 RPS)
  - we'll do the scaling process step by step and keep an eye at the system logs, if the system has performance issue, then we'll do the next step to make it faster
  - **Cache : Redis –** we can use anything
    - if the frequency is greater that 500, then this link is popular, cache it for 1 week, and if it called after the week is passed, cache it again and so on.
    - We can change the value of the condition **500** to anything based on the system status.
    - I think this will not be enough, so we'll do the next step
  - **Partitioning**
    - at this stage of the system – no doubt – the reads from database will be more then the writes, so, let's help the database to to make queries faster
    - check system analytics and decide if we want the next step or not
  - **Replication**
    - we can use this method to make multiple instance for reading and just one for writing, so it will be a lot easier to the database to get queries, bacause it's multiple instances now
    - but every instance has all data, if we want to make it better, let's move to the next step
  - **Sharding**
    - let's assume if someone use our system to generate link, he'll share it at the same country he is from.
    - So let's shard the database based on country column
    - we can make another scenarios, but I guess that's is fine
  - **we've talked so much about the database, what if the server it-self has performance issues**
  - **Load-Balancer**
    - let's make a multiple instances of this server and make load-balancer and it'll work fine
    - of course we need to use docker (CI-CD) to make it easier for us

I didn't make capacity estimations because I assumed we're so rich, so we can buy anything, but I can do it if you want