Requirement

1. User enter search here

2. Serach for cats in the backend here,

API handle user's search request here, a database for everysingle page on the internet

Designing a database here

1. How would this work

Logical

next question is how do we acquire a database with every single site on the

internet and the answer is that we need

some sort of crawler that can go out to

the internet find the HTML associated

with this page and download it to add it

to the database and we're going to call

that colada crawler so now the final

How does craller know which urls?

how does it know which URLs are associated with websites and to answer that question we really need to think about the web as a web a single page on the web has multiple URLs to point to other pages and each one of those pages has URLs that point to other Pages etc etc so

when we think about that we can think that once we download a single URL we can extract the page and then we can search the HTML content of the page for new URLs and we can add those URLs to our URL database so each one of these components presents its own unique challenges but now that we understand

**Database that needs to be stored here.**

**A black board with writing on it

Description automatically generated**

Introduce amazon blob storage

Use a hash to prevent the duplicate here

1. Get page by url here,

Instead of storing everythign in the database, we can store the content, and have an id that refs it over here.

And then sharding the database

A blackboard with writing on it

Description automatically generated

An algo can be run

Use url as the sharding key here,

**WHat should the databse do here?**

1. Be able to look up the url by the hash key

2. Be able to look up a word

What's a global index?

1. We can see what partition a data is on by looking up the shard key here, using shard key we can look up what data is on which index

2.

A diagram of a diagram

Description automatically generated with medium confidence

Part 2: for looking up the text in the index

A diagram of a diagram

Description automatically generated with medium confidence

Be able to look up the word here

A diagram of a server

Description automatically generated

**What's a robot txt fiel here? and why do we need to cache it here?**

need to make sure that we're filtering out URLs that are excluded by a sites robots.txt if you're unfamiliar every site has a robots.txt file that can determine what pages a crawler is allowed to crawl so we need to make sure that we're respecting this file

however if we're going to do that in this system we have to download that robots.txt file every single time we want to crawl a site

before we can fetch the actual page this is going to add a lot of overhead and is going to make our crawling significantly less efficient so in order to solve this problem we're going to introduce this robots.txt cache and the cache is just going to take the robots.txt file and hold on to it until our crawler needs it again and if the robots.txt file does not exist in the cache then

the crawler can go out to the internet download the file and place it in the cache so that it can be used later so looking at some math for how our crawlers can scale if we're doing 100 billion pages and crawling each one every 10 days that's 10 million crawls per day and if we assume that each crawl takes around 2 seconds which is the average page load time for a website we can see that we need 231 000 concurrent crawls that is a lot of print current crawls so in order to handle this let's say we can maybe run 20 or 30 crawls on a single computer we will still need 10 000 nodes in our system right however since the rest of our system that we're building is very scalable nothing else should should really be a bottleneck and scaling out our crawler infrastructure to that size the next thing to consider is how much bandwidth

A diagram of a computer network

Description automatically generated

We need keep the crawlers close to the page as said before

1. That's really improtant here, because of the band width page here

URL frontier

1.

A diagram of a data flow

Description automatically generated with medium confidence

what does politeness mean?

Like we said before 1 host -> 1 crawler at 1 time here,

A diagram of a computer

Description automatically generated

The above takes in prioirty up above and then absort those data and then do sth accordingly here

A diagram of a diagram

Description automatically generated

A couple of notes here

1. Only 1

2. Router selects url from priority q to the politeness q basically

**Fuzzy stuff here**

Solving the problem of politeness (not sending too many requests to a single server to crash it here)

here is going to randomly select one of these cues although it will be biased toward ones with higher priority so for example maybe there's a fifty percent chance of selecting this one thirty percent chance of selecting this one and twenty percent chance of selecting the last once it selects that it's going to again pull the URL down crawl the page and then send it back to the prioritizer where it will determine which queue it should belong in so this does solve our priority Problem right every single one of these URLs will get crawled

eventually however the higher priority ones up here will just get crawled more frequently however the last thing that we still need to focus on is politeness if all of these URLs are ordered and each one is from example.com.

here so every single URL in one of these queues has to be from the same host in this case race example1.com and if example1.com is assigned to this queue none of these these other cues are allowed to be containing URLs from example1.com then

**Adding a time-based heap problem here for the time constraint here**

we're also going to add this Heap over here that contains a reference to each queue along with a time stamp of the next time that we're allowed to fetch a URL from that website

[timestamp based on the amount of time ittook the page to load a good rule of](javascript:;)  
[thumb here is that you wait 10 times the](javascript:;) [page load time so for example if your](javascript:;)  
[page takes two seconds to load you'dwait 20 seconds before performing the](javascript:;)  
[next request at that website event](javascript:;)

Notice the time based heap is

0 t = 10: 01

0 t = 10: 05

0 t = 10: 06

Remember url 3 can only be assigned to queue 1 there in the first here

1.

once we've grabbed this element we can take the first URL call that URL and put it back to the prioritizer which will put it in the correct queue based on its priority just like before once we've finished scraping that URL we can then put this element back in the Heat and update the timestamp based on the amount of time it took the page to load a good rule of thumb here is that you wait 10 times the page load time so for example if your page takes two seconds to load you'd wait 20 seconds before performing the next request at that website event actually throughout this process one of these cues will become empty so let's say for example this top queue is no longer filled with items that's where this router comes into play the router will use the same algorithm before where it picks a random one of these cues and it'll start taking URLs out of those queues and assigning them to other cues when it takes a URL out of one of these queues it's first going to try to determine what the host of that URL is so for example let's assume that this URL here is for example3.com so the first step is to check whether any of these cues down here remember that this one is empty are assigned to example3.com if they are we'll go ahead and set that URL over to that queue otherwise we'll go ahead and put it in the empty queue this is how the cues will end up filling up over time as we continue scanning for more URLs from these queues as a quick recap we have two sets of cues one is designed for ensuring that we are taking care of priority and the other is designed for take making sure that we're taking care of politeness by sorting by host and claiming one of these elements whenever we're trying to scrape a site we can ensure that there's only one worker ever working on a site at once and by sorting these first cues By Priority we can ensure that our router selects cues with a higher priority more frequently so let's take a look at some math and see how big this URL Frontier really needs to be looking at 100 billion URLs times 50 bytes per URL we can see that we have five terabytes of URLs that's a lot just

What's ntxt

1. Fault tolerance

2 .Close duplicates

3. Digging into the index algorithm

4. Use page rank and personalizning results are also a hard problem here

5. And more here