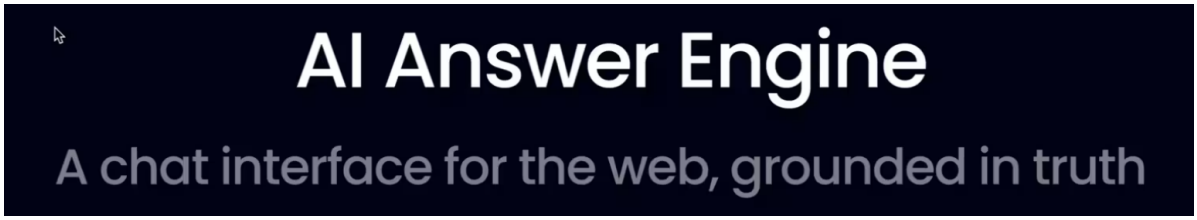
Project 7: AI Answer Engine

▼ info:

- build an AI answer engine w/ Next.js & Typescript that can scrap content from website & mitigate hallucinations by citing its sources when providing answers
- a chat interface for the web, grounded in truth
- topics:
  - caching
  - rate limiting
  - middleware
  - api design
  - ai engineering fundamentals (memory management & prompt design)

▼ intro:



AI Answer Engine

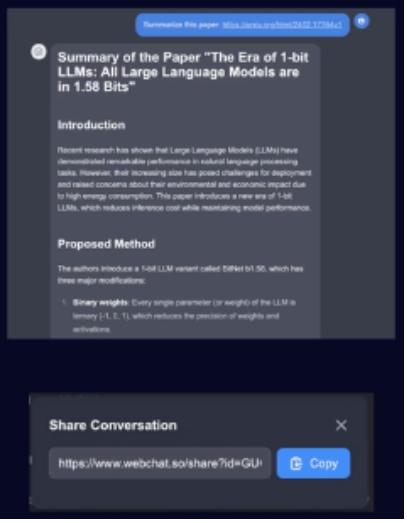
Scenario

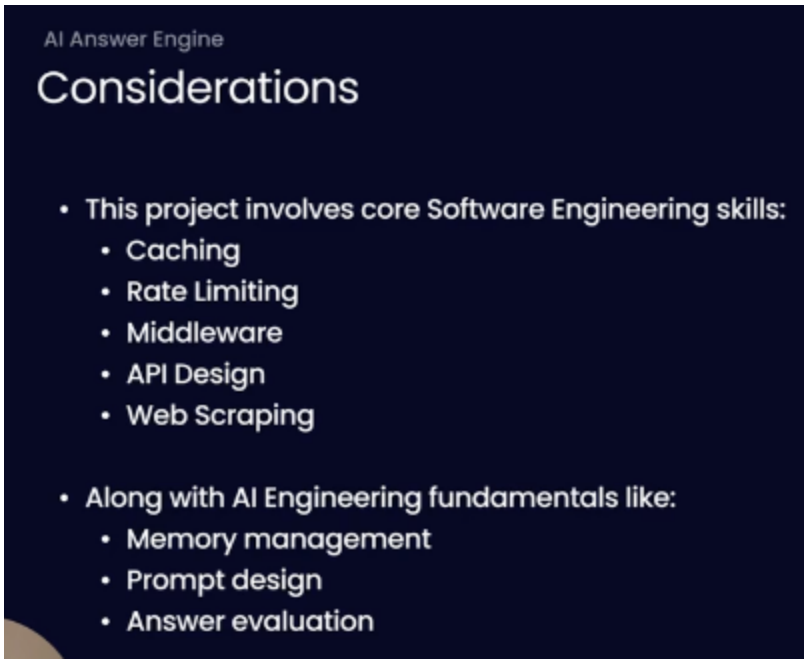
- Imagine you are AI researcher at Company X, where your work involves reviewing lots of academic papers, technical blogs, and news articles to stay informed about breakthroughs in AI.
- You are tasked with creating reports, summarizing trends, or compiling datasets for experiments.
- However, finding accurate, reliable information quickly is a challenge. Existing tools often provide incomplete or incorrect information.

AI Answer Engine

Requirements

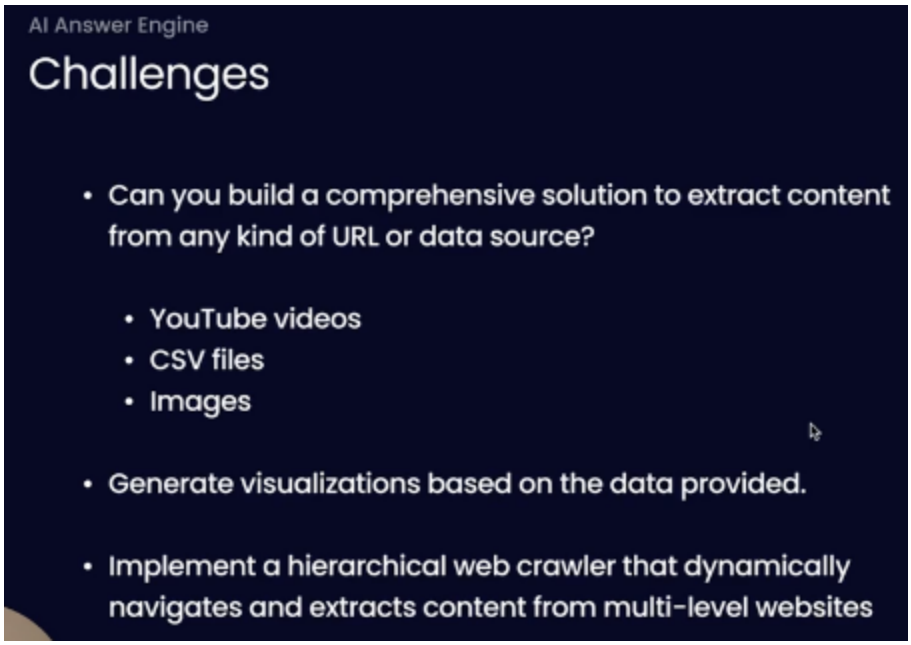
- A chat interface where a user can:
  - Paste in a set of URLs and get a response back with the context of all the URLs through an LLM
  - Ask a question and get an answer with sources cited
  - Share their conversation with others, and let them continue with their conversation



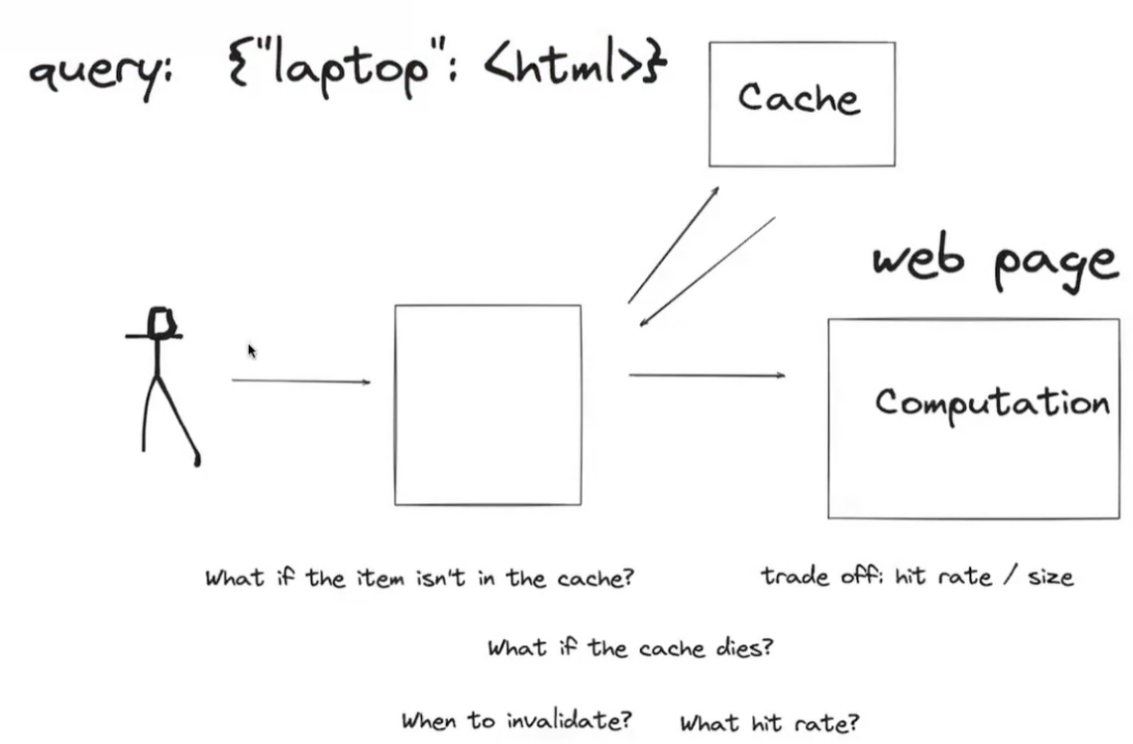


- caching:
  - if a website has already been scraped, its content should be read from the cache instead of being scraped again
- rate limiting:
  - someone can not abuse the system by sending a lot of messages at a time
  - certain amount of messages/hour (20-30 per hour)
- middleware:
  - full stack applications
  - middleware allows you to modify and redirect requests as they come into your application - gateway between client and server
- api design:
  - multiple api routes to handle chat messages, loading conversations, and sharing conversations
- web scraping:

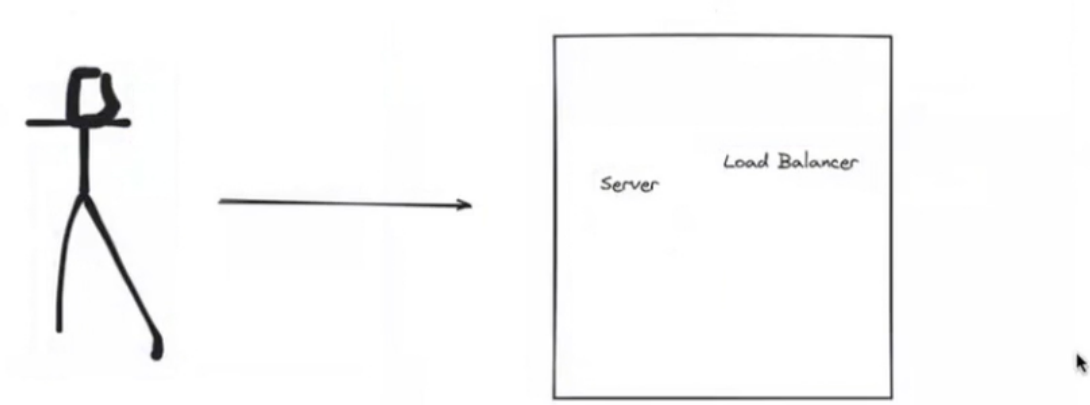
    - extract text from url can be challenging (some websites have preventative measures that prevent you from scraping them)
    - headless browser to extract text form them
- ai engineering fundamentals:
    - memory management:
        - how you manage the context you feed the llms
    - prompt design:
        - just a couple of words added or removed can mean alot
    - answer evaluation:
        - can you use another llm to determine if the llm's is correct
-



▼ hassan:

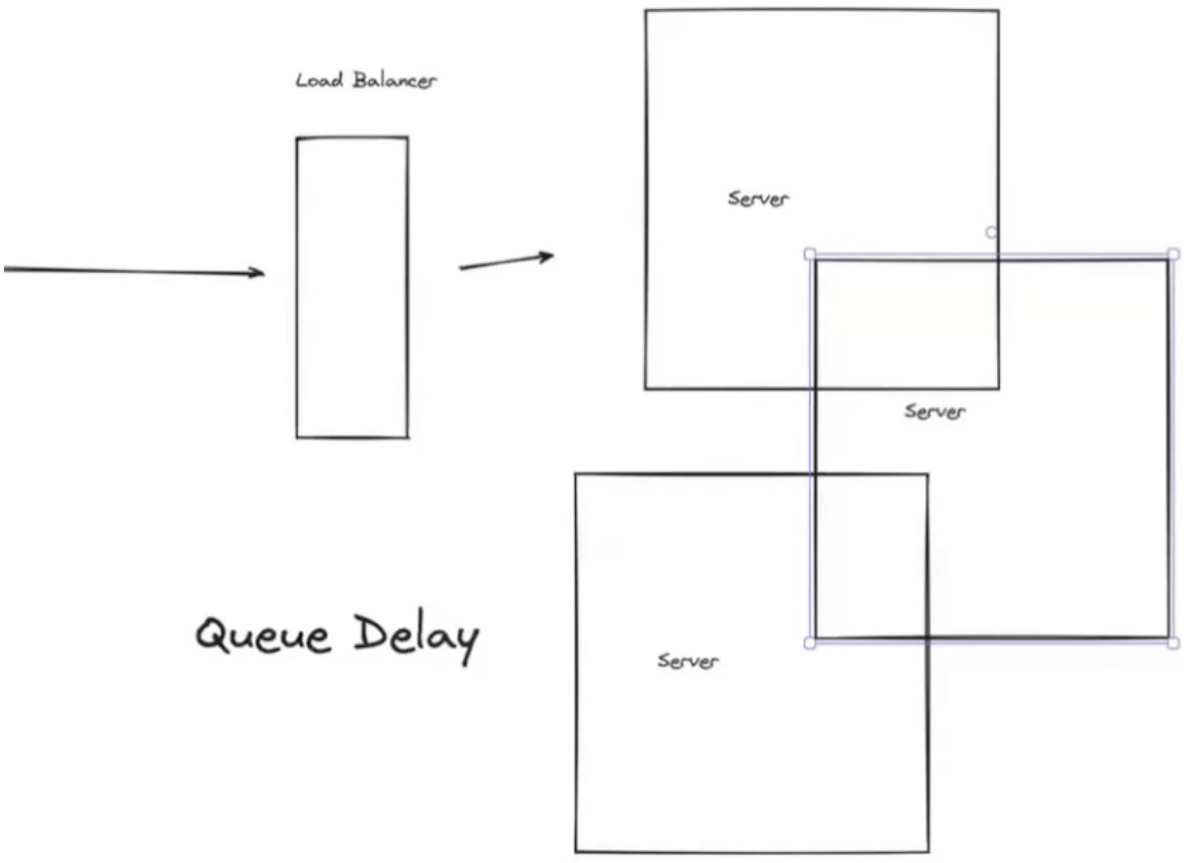


- smaller latency
- how big is this going to cache
- usually 80% of searches come from 20% of the requests (we want to go for that 20%)
- hit rate: how many go to cache and how many go to other
- this data is stored in memory and not rate it in disk (we want to be extremely fast)
- when to invalidate?
- ex. mr beast just uploaded a video to youtube (we know he is going to get 100m views)
    - they go to the edge (data center closest to you)



Round Robin Queue Delay

Round Robin

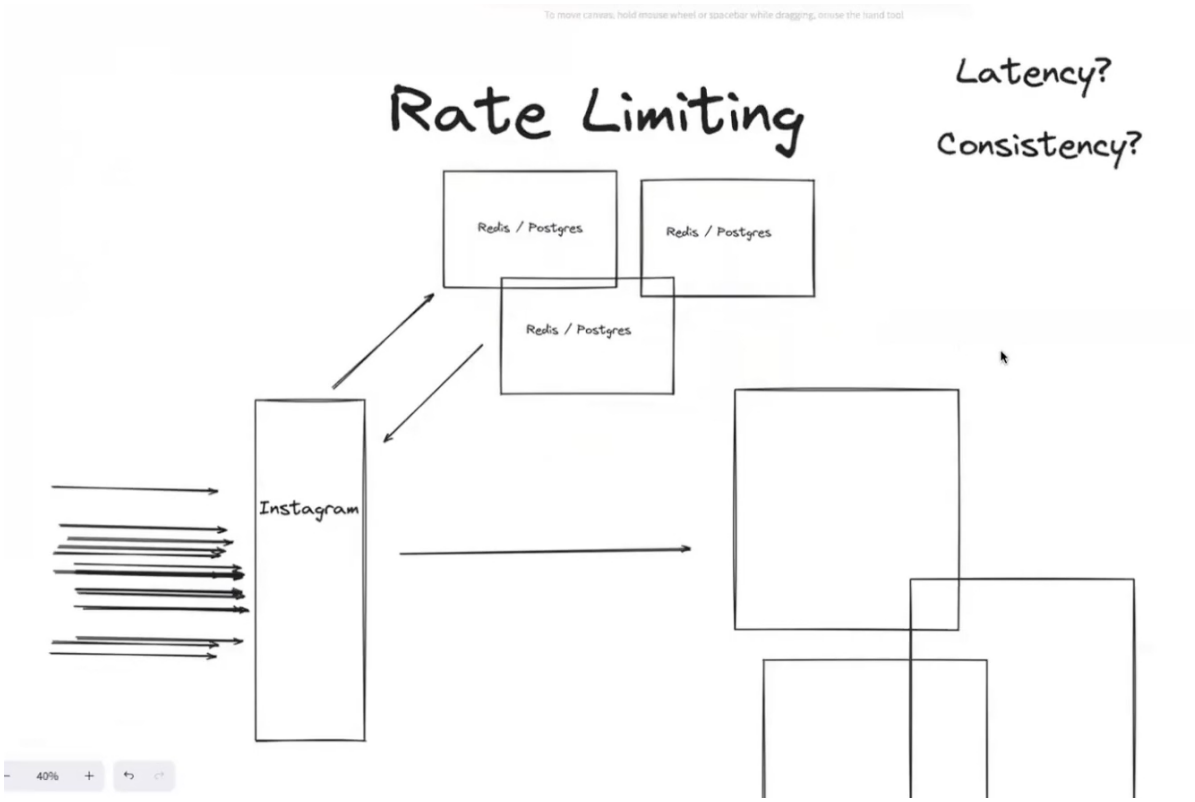


Queue Delay

- server: the thing that is doing work (ex. the thing that is actually calling open ai and returning a response
- load balancer: sits in front of your server and route request to distribute the load
- strategies:
  - round robin: one at a time
  - queue delay: stacking up servers

---

rate limiting:



- uses caching strategy (user.id or ip) and put in account or ttl (time to live) and count amount of requests
- say we have more than 1 redis instances
- rate limiter is on the server side

Strongly Consistent

Advantages:
- Always consistent view of data
- Predictable behavior
- Easier to reason about

Disadvantages:
- Higher latency
- Lower availability
- Cannot function during network partitions

Weakly Consistent

Advantages:
- Lower latency
- Higher availability
- Can function during network issues

Disadvantages:
- May see stale data
- Complex conflict resolution
- Hard to reason about

▼ faizan:

middleware:

- gateway between client and server
- will be implementing rate limiting here using sliding window

▼ live tutorial:

-

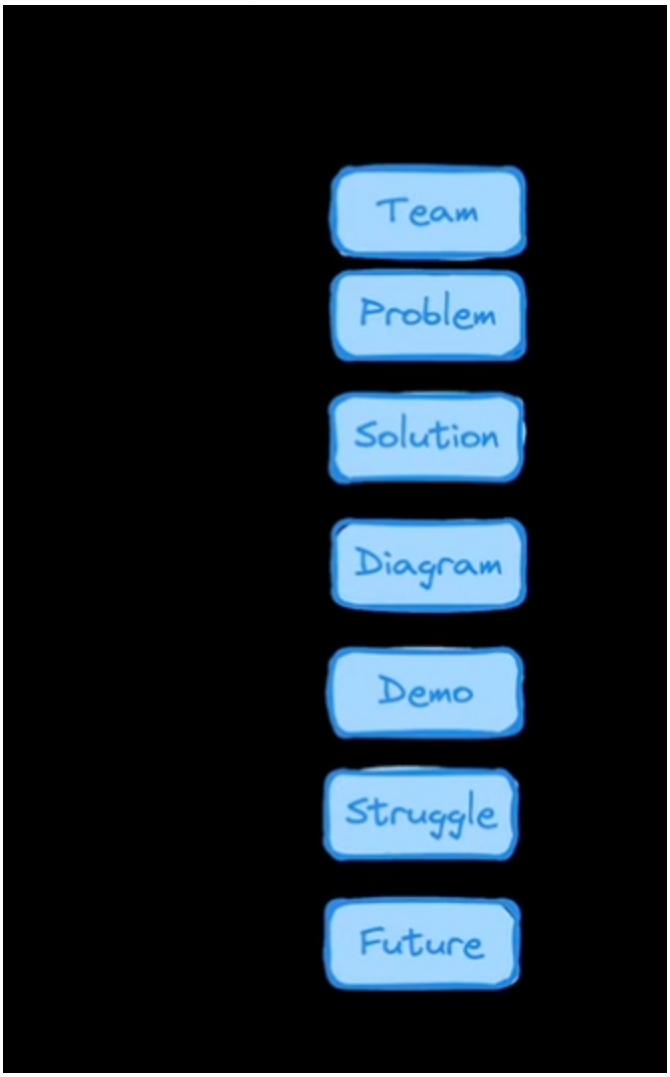
▼ yasin how to make a 2min demo:

- the demos did not look impressive (can’t send demo to hiring manager)
- most demos look same (no uniqueness)
- lot of missing components in the demo pitch

- what is your definition of cracked?
  - speed
    - if you hear something, you act on it quick and submit on time
  - feedback
    - after you get it done, your open to feedback
  - action
    - you get that feedback and you take action (speed)
    - are you updating your resume after you submit your project

---

- how to make a killer 2-min demo
  - ask a TON of questions
    - always talk to sponsors, hackathon organizers, and judges
    - has to be evident that we done research in our demo



- team: (quick synposis)
  - where you go to school
  - where your interviewing
  - how many places you've applied
  - how many leetcodes have you done
- problem: 10 sec
  - what are you doing
  - who is your customer
  - who is your stakeholder

    - who are you solving for
- solution: 20 sec
    - what was your approach
    - what is the research you did (what kind of research did i do outside of what was given to me)
    - showcase proactiveness
- diagram
    - excilidraw: 3-4 words per box
    - logos, api requests, etc
- demo: 30-40 sec
    - showcase executable demo or final result
- struggle:
    - where did i struggle in the code
- future:
    - what do i plan to do in the future