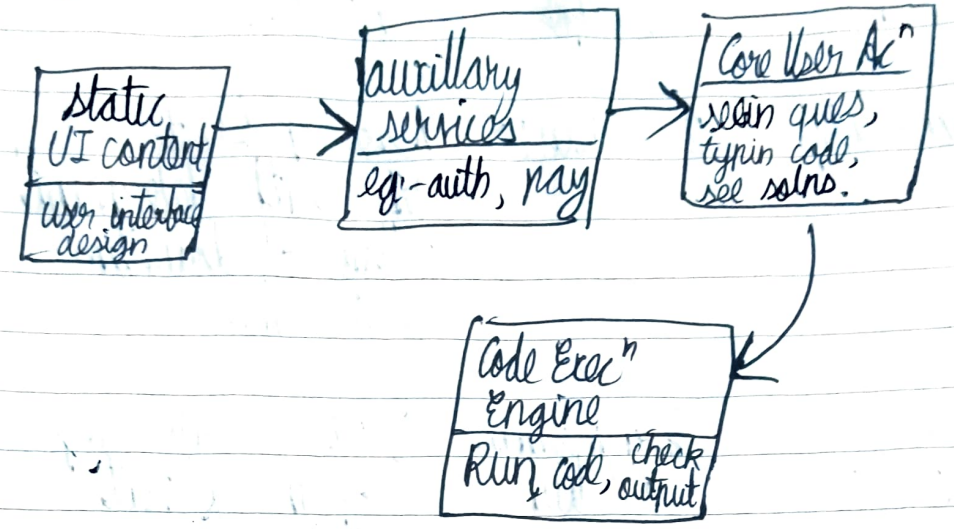
Q) Design core services of Algo Exp (no need for payments, security, auth^n, as 3rd party services like stripe, Oauth 2 used)

Design - Algoexpert

Decision	Info	Reason
Availability	2-3 nines	Algo exp isn't a super-critical life-death determinin, sys so 3 days of downtime / yr max, 8 hrs min.
Latency Thruput	~~key~~ medium medium	} not really imp for most part of the website
Latency for code exec^n engine	Very High	Code exec^n engine is highly interactive so 0.5-4 s of latency reqd.
Scale	10^4+ users	10^4 users on website at any pt. in time.
Optimize regionally	Optimize for US, India; available globally	U.S, India → top 2 markets
Robust API + dB to back API	Google Cloud store (GCS), for static content	website → static → homepage, JS bundle → dynamic → purchase, saved solns. of users.

Date:
M T W T F S S

Design Flow



	Decision	Info	Reason
1]	STATIC UI CONTENT		
	Use blob store for static UI content	GCS, Amazon S3	static UI content - homepage JS bundle, imgs, icons.
	CDN for static content	Google Cloud CDN, Cloudflare	CDN for US, India customers cuz homepage is imp. CDN would hv its own servers near US, India to serve static content really fast (reduce res time of home, purchase pg)
2]	AUXILLIARY SERVICES		
	No need to dev auth^n, pay		As 3rd party soln's used

NOTE: ACL (access control list) → a table that tells a comp. OS which access rights each user has to a particular sys. obj. (eg:- file, dir) usin User's security attribute

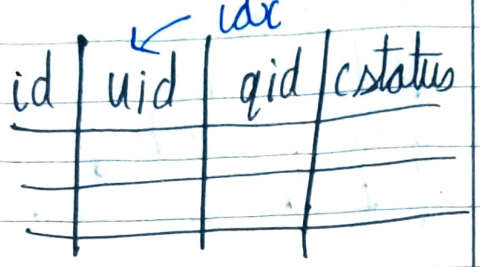
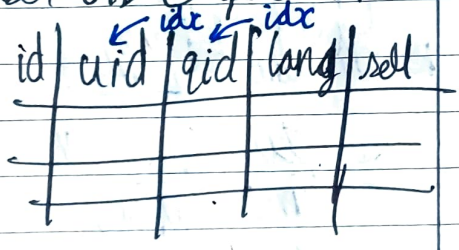
(LB)

Load Balancing at DNS level	Dependin of loca" of user, req. is rer-outed to specific cluster of servers by DNS LB	3 major hubs to serve req. fast → US, India, maybe Europe
Path based LBing,	Handle diff isolated services on website eg: multip servers handle auth, other multip handle code exec" etc.	Some services are completely isolated, indep of other services eg:- code exec" and payments
Applica" servers	At end of LB in to handle logic	
Access Control Implenta"	App servers make API call (internal) to see authriz" of user Use ACL and a tag/attr on user to identify his/her ACL	To serve / block some content dependin on wether user has permission (is authorized or not
② static content such as ques list in blob store (already decided before)	App servers receive req., they do the ACL check and req. somethin frm blob store	Same reason as above
Caching ques list in client side	Cached in client browser (front-end -code)	∵ clients will access ques. list a lot in 1 session to solve multip ques, so we wanna cache to minimize reqs to blob store to ↓ latency

Feasibility of caching if ques list is updated

(100 ques) × (7 langs) × (2 solns/ques provided by creator) × (5000 bytes for storing 10^4 bytes of chars in code) ≈ 10MB small cache

Caching in app. server of ques list.	Cachin in mem. on server Approx 10MB cache for entire ques. list (QL)	To further reduce latency for a bunch of rpd accessin QL
Cache evic" policy	Evict data frm cache every 30 mins. 10MB evicted Next req. will be to blob store and now cache starts refilling	To give good 'refresh rate' each time a new ques is added or a bug is fixed in QL
3] DYNAMIC CONTENT/CORE USER AC" eg:- user written solns. storage		
A structured RDB used → 2 tables used	Can use PostreSQL	Cuz we gonna be queryin this data (eg- user written solns.) a lot
→ Table 1: Ques comple" status	Every row is a ques. so it'll have ① row ID ② ques ID and the ③ user's UID (frm auth service), ④ comple" status as attr.s/ cols.	Comple"n status is an enum → complete, not attempted ← in progress. Every user has no. of rows in table = no. of ques. in QL. Each row → 1 ques of a user

Idx table 1 on user UID	Indexin done on basis of user UID 	~~Idxing~~ We need to query a lot and need to do it quickly Won't be writin too much to dB so replication all writes in dB idx is not a prob.
Table 2: Soln's table	Rows ⇒ per ques of a user. they have cols → ① Row ID ② ques ID ③ user UID ④ lang ⑤ soln.	No. of rows/user = (No. of ques. in qz) × (No. of langs of soln in ques)
Idx table 2	Idxin done on ① user UID ② ques ID 	we gonna be goin on indiv ques. a bunch of times. If no. of langs ↑ by a lot → ① idx lang col ② client side cachin of ques ③ no server side cachin here as no sharin of user written solns. across users
Debouncin UI	Write reqs. for a soln are made every 2-3 s 10000 users → (8000 typin / 3s debounce) → 2666 write req/s Postgres can handle this	Aus we don't wanna constantly keep writin to dB, savin user soln each time user types a char in code (i.e. issue write req. every 0.5s)

User Data Storage
For user data, we have to design the storage of question completion status and of user solutions to questions. Since this data will have to be queried a lot, a SQL database like **Postgres or MySQL** seems like a good choice.

We can have 2 tables. The first table might be **question_completion_status**, which would probably have the following columns:

- id: *integer*, primary key (an auto-incremented integer for instance)
- user_id: *string*, references the id of the user (can be obtained from auth)
- question_id: *string*, references the id of the question
- completion_status: *string*, enum to represent the completion status of the question

We could have a uniqueness constraint on (user_id, question_id) and an index on user_id for fast querying.

The second table might be **user_solutions**:

- id: *integer*, primary key (an auto-incremented integer for instance)
- user_id: *string*, references the id of the user (can be obtained from auth)
- question_id: *string*,references the id of the question
- language: *string*, references the language of the solution
- solution: *string*, contains the user's solution

We could have a uniqueness constraint on (**user_id, question_id, language**) and an index on **user_id** as well as one on **question_id**. If the number of languages goes up significantly, we might also want to index on language to allow for fast per-language querying so that the UI doesn't fetch all of a user's solutions at the same time (this might be a lot of data for slow connections).

	Async. replica" b/w diff regional dB	eg:- US dB replicates data from India dB every 4 hrs and vice versa Base a backup if 1 dB goes down	User of site travellin from US to India has data (their soln.) access to in both regions Async cuz this doesn't happen a lot + flight time is > 4 hrs
4]	CODE EXEC" ENGINE		
	Security aspect already taken care of		Given prob. is not a distri. sys. prob, its an OS prob.
	Route code servers ↳ to handle traffic	An endpt. of path based LB in Route codde to "worker servers"	It's an isolated service to it has its own path
	Given based rate-limitin at run-code servers	eg:- twice over 10s, 5 times/min, 20 times/hr.	To prev. malicious client frm bringin server down by issuin too many req-s/sec to overload server
		K-V store (eg: Redis) used to implement rate limitin	
	Run code servers	Receive routed code frm route code servers and run code (queue based sys → workers given req. by FIFO) 2s → to run code 1s → to route code (3s) → met our goal for	Route code servers handle traffic and hv a way to know which worker server is curr. by free to run code Worker server 'resets' itself after runnin code - deletes zombie processes extra generated files

goal of code engine latency to $\in [0.5, 4]$s

No. of workers

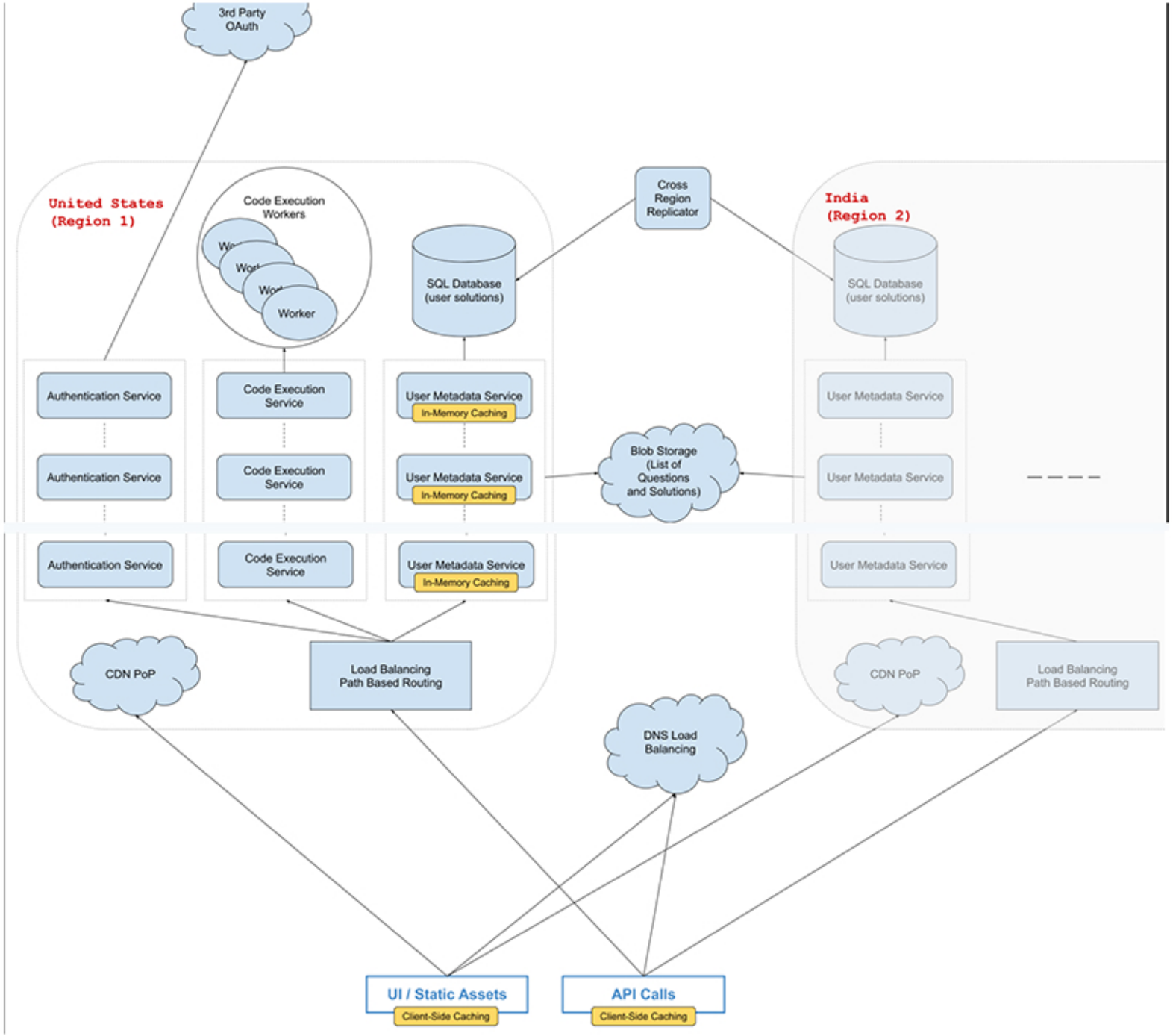
at any given instant we hv 10^4 users, 8000 actually coding, only few run code req (say 10/hr) by 1 user $\Rightarrow 8000\left(\frac{10}{60}\right)$ req/s

$= 163$ req/s so we'll keep around 200–300 worker machines

Can scale no. of workers horizontally, vertically (8 core machines)

log, monitor entire code exec^n part (route & run code servers) to keep optimizing sys based on collected data.

eg:- separate workers based on langs., qtes, etc


3rd Party
OAuth
United States
(Region 1)
Code Execution
Workers
Worker
SQL Database
(user solutions)
Cross
Region
Replicator
India
(Region 2)
SQL Database
(user solutions)
Authentication Service
Authentication Service
Authentication Service
Code Execution
Service
Code Execution
Service
Code Execution
Service
User Metadata Service
In-Memory Caching
User Metadata Service
In-Memory Caching
User Metadata Service
In-Memory Caching
Blob Storage
(List of
Questions
and Solutions)
User Metadata Service
User Metadata Service
User Metadata Service
CDN PoP
Load Balancing
Path Based Routing
CDN PoP
Load Balancing
Path Based Routing
DNS Load
Balancing
UI / Static Assets
Client-Side Caching
API Calls
Client-Side Caching