System Design - API Rate Limiter

https://www.youtube.com/@15minutesystemdesign

Functional requirements:

Functional requirements:

- Limit requests: Limit the number of requests an entity can send to an API within a time window.

Functional requirements:

- Limit requests: Limit the number of requests an entity can send to an API within a time window.
- Exception handling: The user should get an error message when the user excess the threshold.

Functional requirements:

- Limit requests: Limit the number of requests an entity can send to an API within a time window.
- Exception handling: The user should get an error message when the user excess the threshold.
- Distribution: The rate limiter can monitor requests among multiple servers.

Functional requirements:

- Limit requests: Limit the number of requests an entity can send to an API within a time window.
- Exception handling: The user should get an error message when the user excess the threshold.
- Distribution: The rate limiter can monitor requests among multiple servers.

Non Functional requirements:

Functional requirements:

- Limit requests: Limit the number of requests an entity can send to an API within a time window.
- Exception handling: The user should get an error message when the user excess the threshold.
- Distribution: The rate limiter can monitor requests among multiple servers.

Non Functional requirements:

- Low Latency

https://www.youtube.com/@15minutesystemdesign

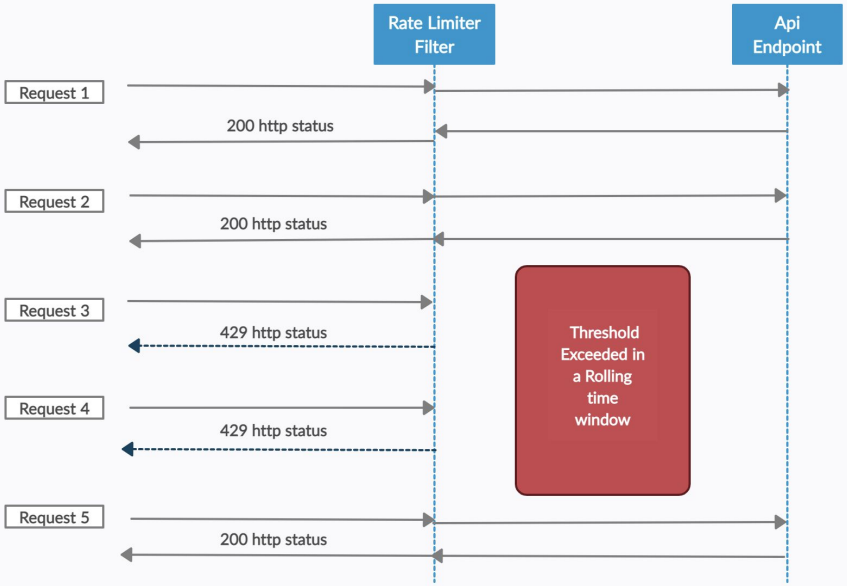
Functional requirements:

- Limit requests: Limit the number of requests an entity can send to an API within a time window.
- Exception handling: The user should get an error message when the user excess the threshold.
- Distribution: The rate limiter can monitor requests among multiple servers.

Non Functional requirements:

- Low Latency
- High Availability

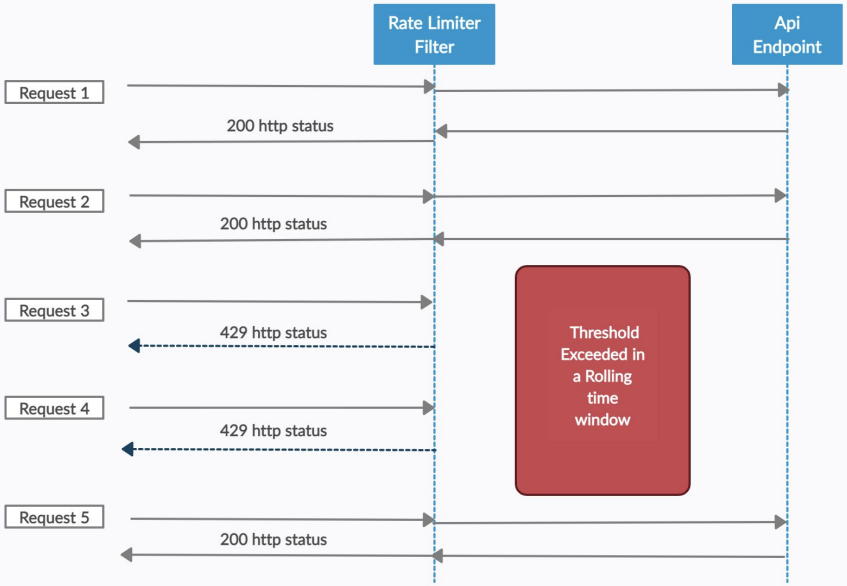
https://www.youtube.com/@15minutesystemdesign

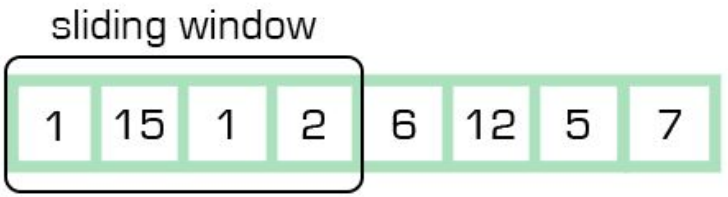


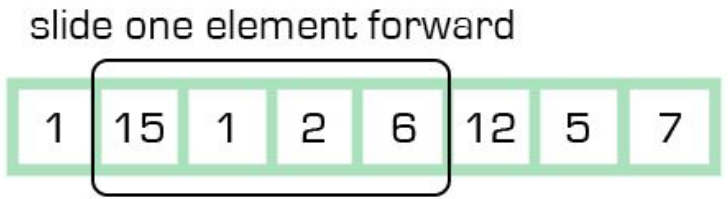
Rolling Window Rate Limiter

Only Y requests per user to be allowed over last Z minutes

https://www.youtube.com/@15minutesystemdesign



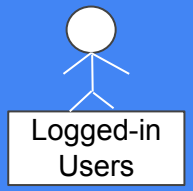


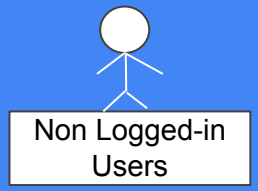


Rolling Window Rate Limiter

Only Y requests per user to be allowed over last Z minutes

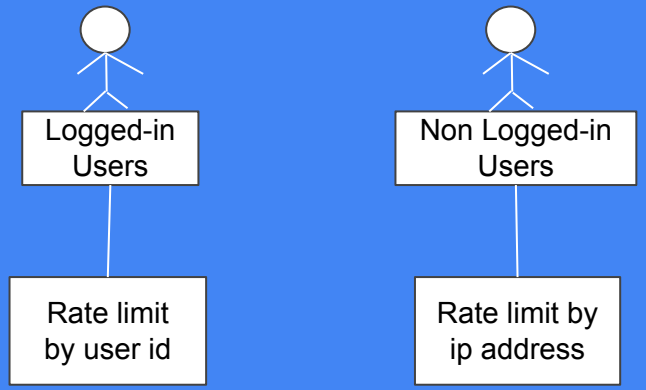
https://www.youtube.com/@15minutesystemdesign





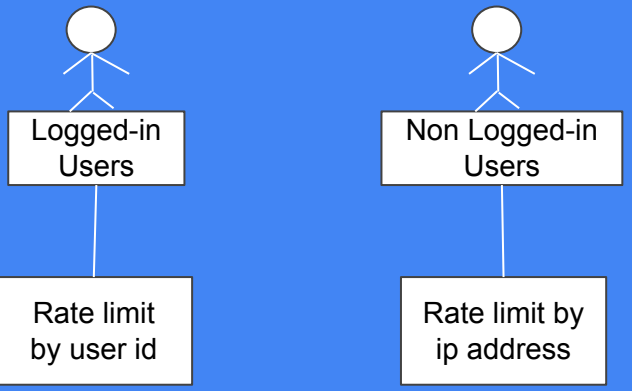
User Type	End Point id	End point	Method	Limit per 15 minute	Limit per minute
Logged-in	94	/api1	GET	900 req/ 15 min	200 req/ min
Logged-in	95	/api2	GET	900 req/ 15 min	200 req/ min
Logged-in	96	/api2	POST	500 req/ 15 min	100 req/ min
Logged-in	97	/api3	GET	800 req/ 15 min	150 req/ min
Non Logged-in	98	/*	GET	250 req/ 15 min	50 req/ min

https://www.youtube.com/@15minutesystemdesign



User Type	End Point id	End point	Method	Limit per 15 minute	Limit per minute
Logged-in	94	/api1	GET	900 req/ 15 min	200 req/ min
Logged-in	95	/api2	GET	900 req/ 15 min	200 req/ min
Logged-in	96	/api2	POST	500 req/ 15 min	100 req/ min
Logged-in	97	/api3	GET	800 req/ 15 min	150 req/ min
Non Logged-in	98	/*	GET	250 req/ 15 min	50 req/ min

https://www.youtube.com/@15minutesystemdesign



User Type	End Point id	End point	Method	Limit per 15 minute	Limit per minute
Logged-in	94	/api1	GET	900 req/ 15 min	200 req/ min
Logged-in	95	/api2	GET	900 req/ 15 min	200 req/ min
Logged-in	96	/api2	POST	500 req/ 15 min	100 req/ min
Logged-in	97	/api3	GET	800 req/ 15 min	150 req/ min
Non Logged-in	98	/*	GET	250 req/ 15 min	50 req/ min

Key : user_id_endpoint_id

Value :
[
  epoch : request_count,
  epoch : request_count
]

Key : ip_address

Value :
[
  epoch : request_count,
  epoch : request_count
]

=> rolling window = total # of epoch entries
=> limit = sum (request_count for each epoch_entry)

https://www.youtube.com/@15minutesystemdesign

User_id_endpoint_id :

{

$epoch_{(t)}$: $request\_count_{(t)}$

$epoch_{(t-1)}$: $request\_count_{(t-1)}$

$epoch_{(t-2)}$: $request\_count_{(t-2)}$

.

.

.

.

$epoch_{(t-12)}$: $request\_count_{(t-12)}$

$epoch_{(t-13)}$: $request\_count_{(t-13)}$

$epoch_{(t-14)}$: $request\_count_{(t-14)}$

}

User Type	End Point id	End point	Method	Limit per 15 minute	Limit per minute
Logged-in	94	/api1	GET	900 req/ 15 min	200 req/ min
Logged-in	95	/api2	GET	900 req/ 15 min	200 req/ min
Logged-in	96	/api2	POST	500 req/ 15 min	100 req/ min
Logged-in	97	/api3	GET	800 req/ 15 min	150 req/ min
Non Logged-in	98	/*	GET	250 req/ 15 min	50 req/ min

Check if sum of requests for last 15 epochs (t) > 500 (rolling window limit)

Check if number of request at current epoch > 100 (unit level limit)

https://www.youtube.com/@15minutesystemdesign

User_id_endpoint_id :

{

$epoch_{(t)}$: $request\_count_{(t)}$

$epoch_{(t-1)}$: $request\_count_{(t-1)}$

$epoch_{(t-2)}$: $request\_count_{(t-2)}$

.

.

.

.

$epoch_{(t-12)}$: $request\_count_{(t-12)}$

$epoch_{(t-13)}$: $request\_count_{(t-13)}$

$epoch_{(t-14)}$: $request\_count_{(t-14)}$

}

User Type	End Point id	End point	Method	Limit per 15 minute	Limit per minute
Logged-in	94	/api1	GET	900 req/ 15 min	200 req/ min
Logged-in	95	/api2	GET	900 req/ 15 min	200 req/ min
Logged-in	96	/api2	POST	500 req/ 15 min	100 req/ min
Logged-in	97	/api3	GET	800 req/ 15 min	150 req/ min
Non Logged-in	98	/*	GET	250 req/ 15 min	50 req/ min

How can we do operations like insertion, deletion, count, sum operations in O(1)?

Solution => ?

https://www.youtube.com/@15minutesystemdesign

User_id_endpoint_id :

{

$epoch_{(t)}$: $request\_count_{(t)}$

$epoch_{(t-1)}$: $request\_count_{(t-1)}$

$epoch_{(t-2)}$: $request\_count_{(t-2)}$

.

.

.

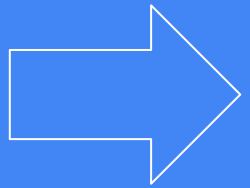
.

$epoch_{(t-12)}$: $request\_count_{(t-12)}$

$epoch_{(t-13)}$: $request\_count_{(t-13)}$

$epoch_{(t-14)}$: $request\_count_{(t-14)}$

}



User_id_endpoint_id :

{

$request_{n}$: $epoch_{(t)}$

$request_{n-1}$: $epoch_{(t)}$

.

.

.

.

$request_{2}$: $epoch_{(t-14)}$

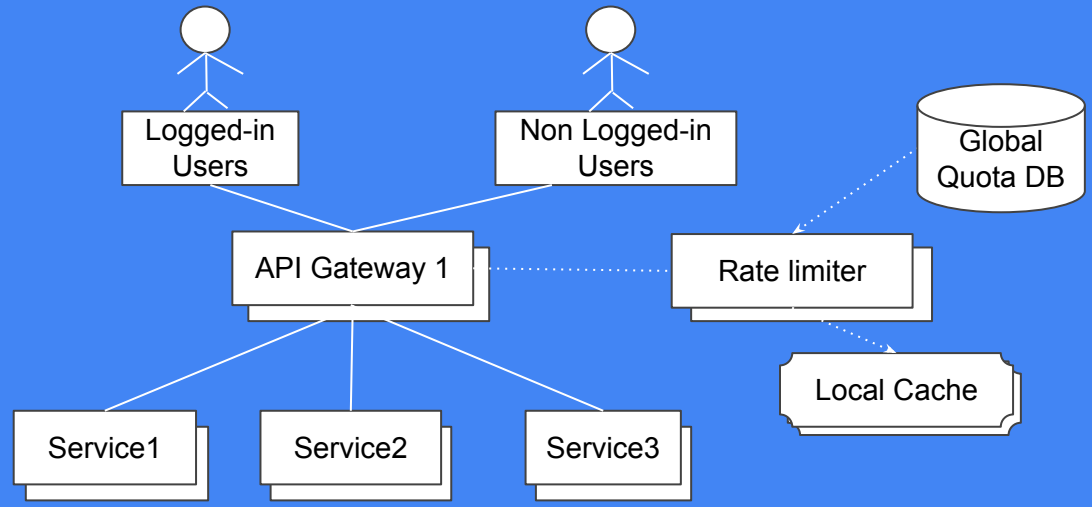
$request_{1}$: $epoch_{(t-14)}$

}

How can we do operations like insertion, deletion, count, sum operations in O(1)?

Solution => Yes, by doing reverse mapping and using redis sorted set!!

https://www.youtube.com/@15minutesystemdesign



User_id_endpoint_id : { $request_n$: $epoch_{(t)}$,

.

$request_2$: $epoch_{(t-14)}$,

$request_1$: $epoch_{(t-14)}$

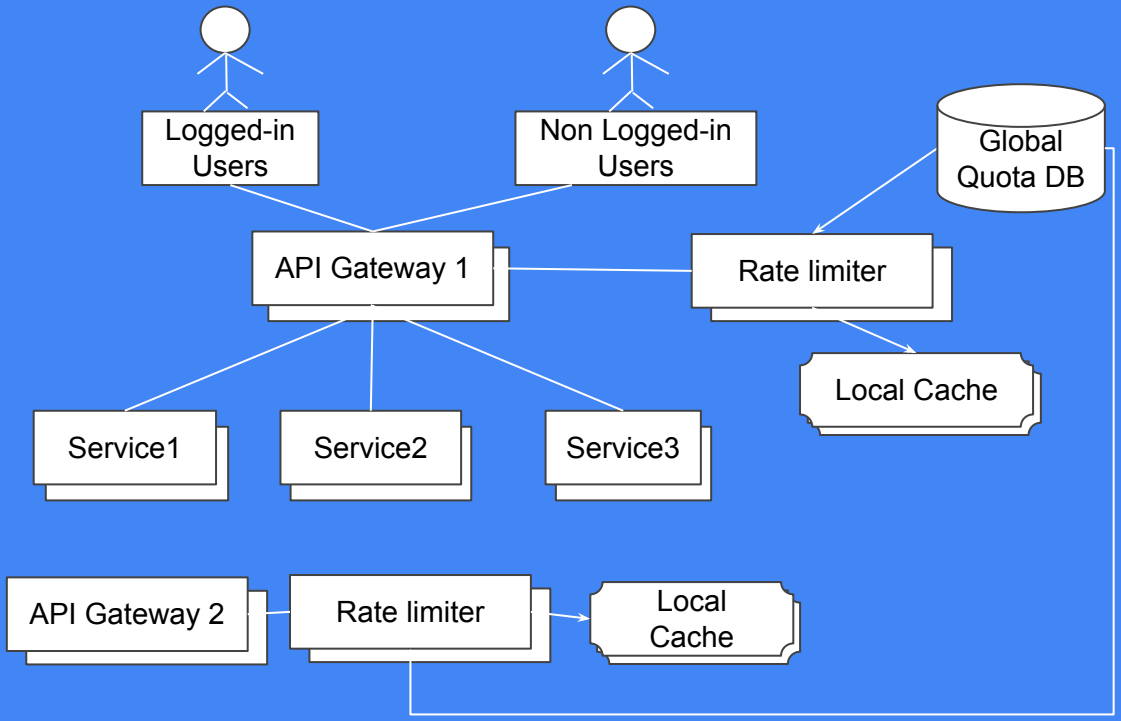
}

Rate Limiter Algorithm (using redis sorted set)

Repeat for each incoming api request from user X:
(a) remove all entries older than rolling window time for user X - O(log n)
(b) add request entry for current epoch - O(1)
(c) check if sum of number of requests across all epochs is less than primary threshold - O(1)
(d) if not, block the request and send 429 http status
(e) else, check if sum of number of requests for current epoch is less than secondary threshold - O(1)
(f) if not, return false else return true

Note: We are first writing then reading. This is primarily to avoid race condition. We may delay unblocking user if limit is exceeded but that should be fine.

https://www.youtube.com/@15minutesystemdesign

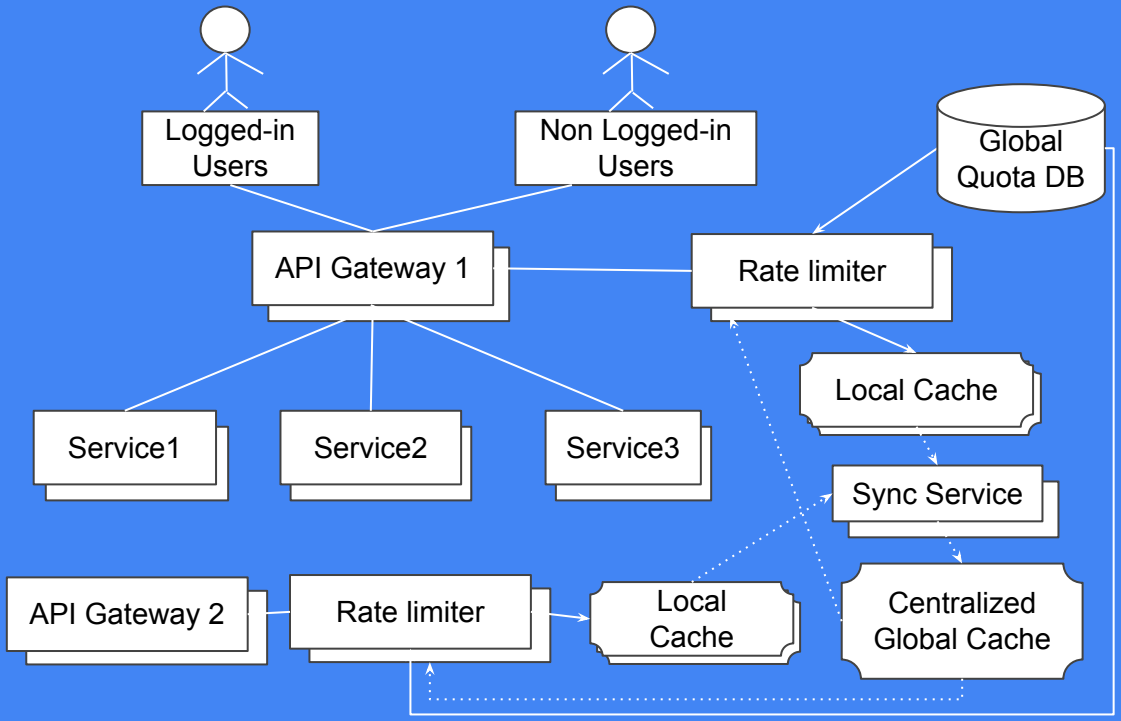


Rate Limiter Algorithm (using redis sorted set)

Repeat for each incoming api request from user X:
(a) remove all entries older than rolling window time for user X - O(log n)
(b) add request entry for current epoch - O(1)
(c) check if sum of number of requests across all epochs is less than primary threshold - O(1)
(d) if not, block the request and send 429 http status
(e) else, check if sum of number of requests for current epoch is less than secondary threshold - O(1)
(f) if not, return false else return true

Note: We are first writing then reading. This is primarily to avoid race condition. We may delay unblocking user if limit is exceeded but that should be fine.

https://www.youtube.com/@15minutesystemdesign

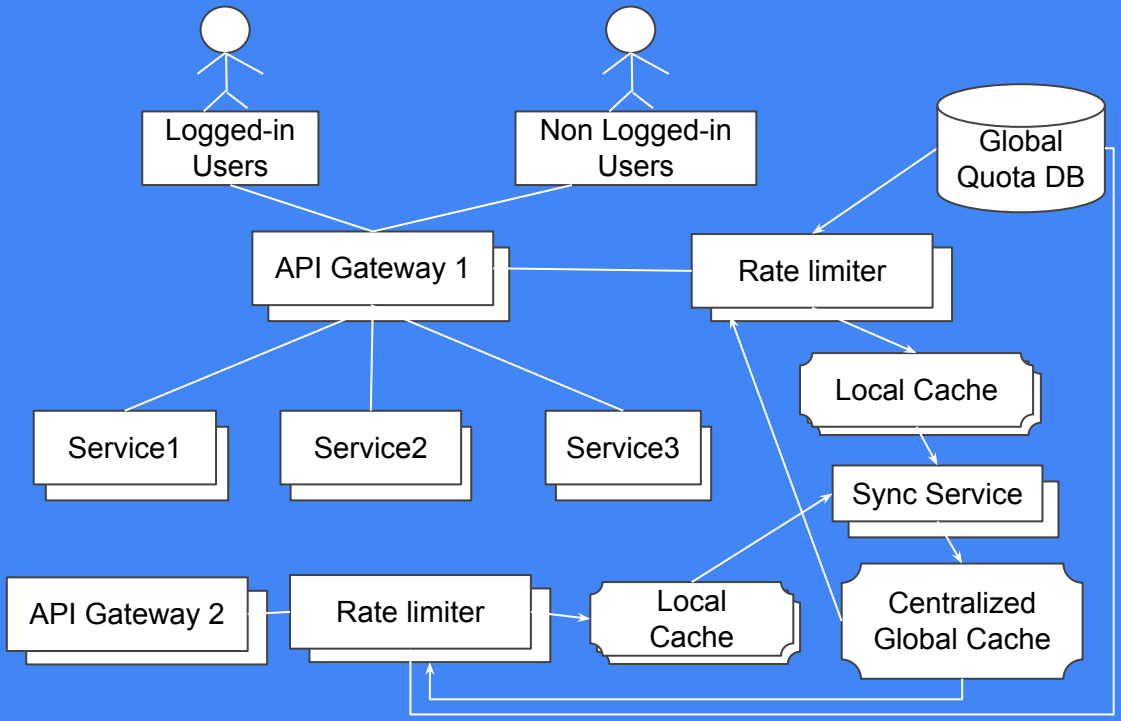


Rate Limiter Algorithm (using redis sorted set)

Repeat for each incoming api request from user X:
(a) remove all entries older than rolling window time for user X - O(log n)
(b) add request entry for current epoch - O(1)
(c) check if sum of number of requests across all epochs is less than primary threshold - O(1)
(d) if not, block the request and send 429 http status
(e) else, check if sum of number of requests for current epoch is less than secondary threshold - O(1)
(f) if not, return false else return true

Note: We are first writing then reading. This is primarily to avoid race condition. We may delay unblocking user if limit is exceeded but that should be fine.

https://www.youtube.com/@15minutesystemdesign



Rate Limiter Algorithm (using redis sorted set)

Repeat for each incoming api request from user X:
(a) remove all entries older than rolling window time for user X - O(log n)
(b) add request entry for current epoch - O(1)
(c) check if sum of number of requests across all epochs is less than primary threshold - O(1)
(d) if not, block the request and send 429 http status
(e) else, check if sum of number of requests for current epoch is less than secondary threshold - O(1)
(f) if not, return false else return true

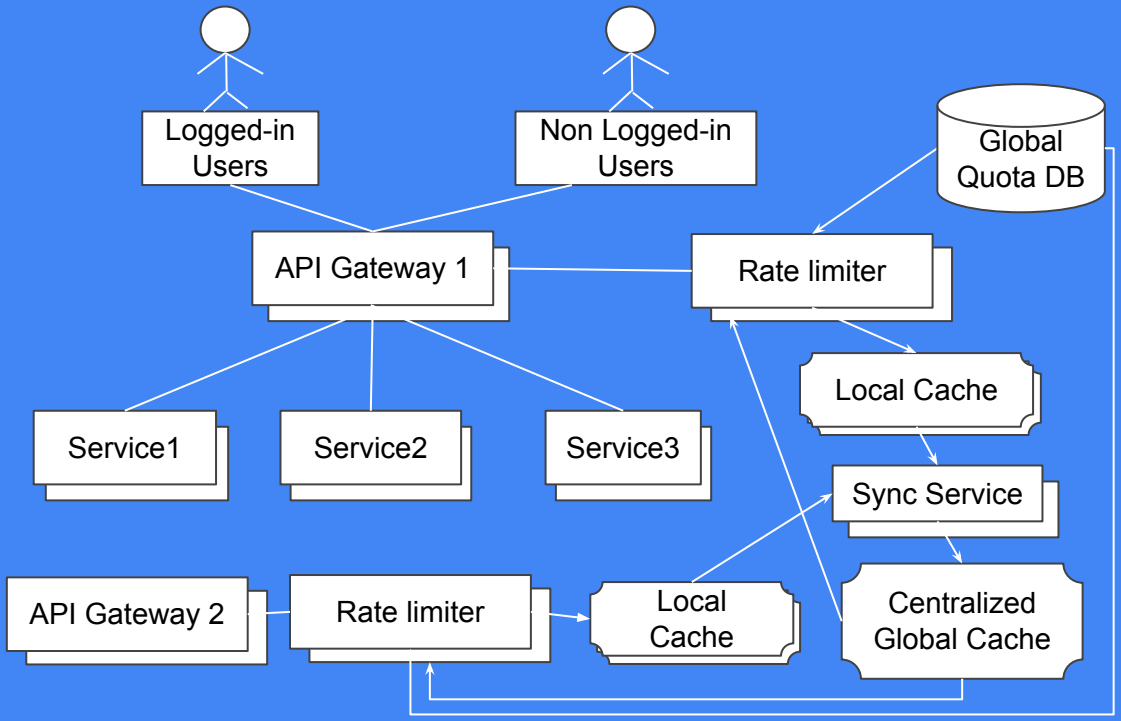
Note: We are first writing then reading. This is primarily to avoid race condition. We may delay unblocking user if limit is exceeded but that should be fine.

If a user sends api requests via different Api gateway, global quota will not breach

---

Solution => ?

https://www.youtube.com/@15minutesystemdesign



Rate Limiter Algorithm (using redis sorted set)

Repeat for each incoming api request from user X:
(a) remove all entries older than rolling window time for user X - O(log n)
(b) add request entry for current epoch - O(1)
(c) check if sum of number of requests across all epochs is less than primary threshold - O(1)
(d) if not, block the request and send 429 http status
(e) else, check if sum of number of requests for current epoch is less than secondary threshold - O(1)
(f) if not, return false else return true

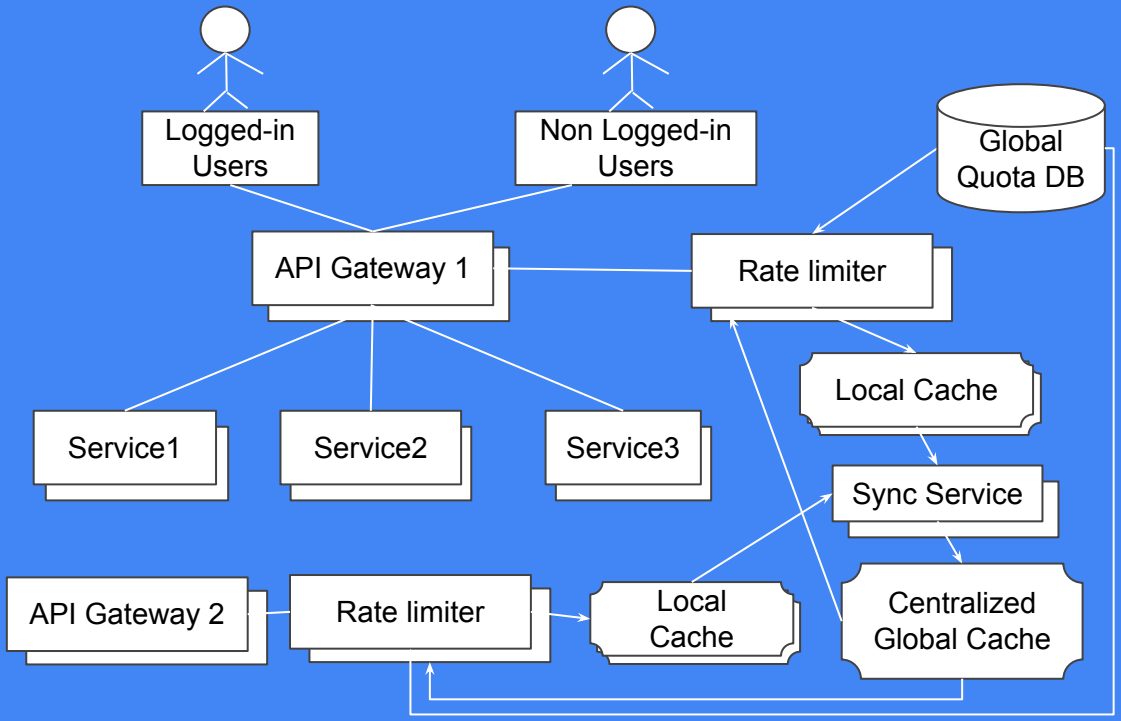
Note: We are first writing then reading. This is primarily to avoid race condition. We may delay unblocking user if limit is exceeded but that should be fine.

If a user sends api requests via different Api gateway, global quota will not breach

---

Solution => To avoid rate limit data inconsistency, when request comes from a new user (there would be no entry in local cache for this user). Rate limiter tries to check and load entry from global cache to local cache for that user, if exists.

https://www.youtube.com/@15minutesystemdesign



If a user sends bulk api requests at same time, race condition may occur

---

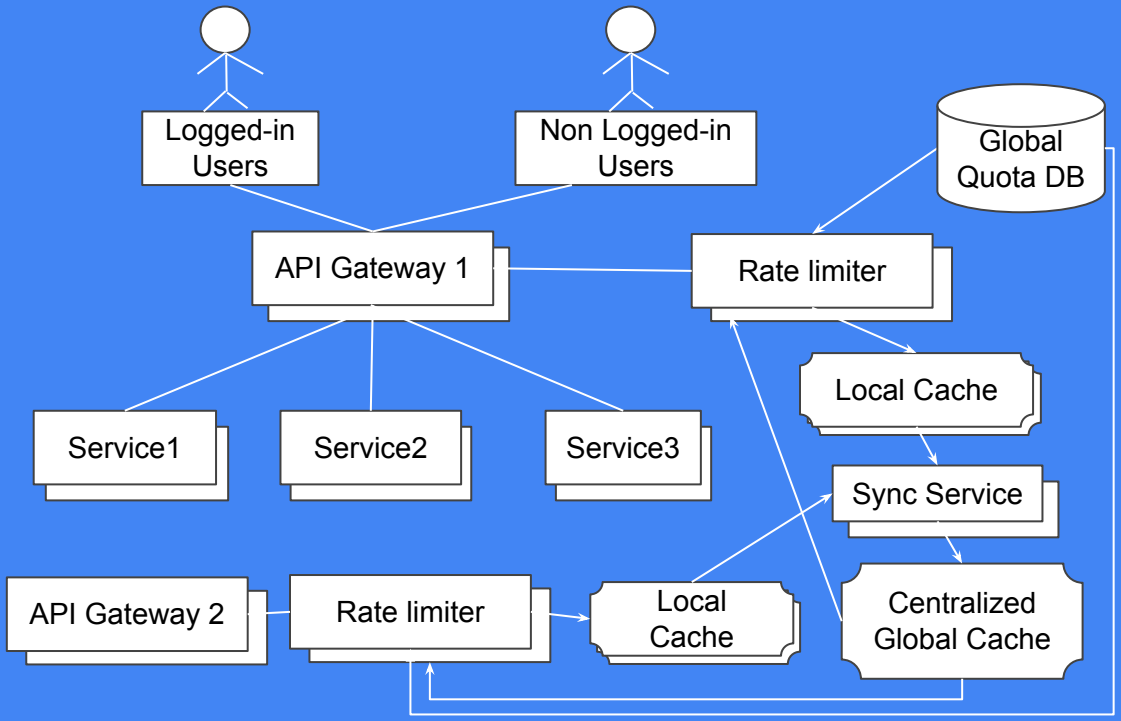
Solution => ?

Rate Limiter Algorithm (using redis sorted set)

Repeat for each incoming api request from user X:
(a) remove all entries older than rolling window time for user X - O(log n)
(b) add request entry for current epoch - O(1)
(c) check if sum of number of requests across all epochs is less than primary threshold - O(1)
(d) if not, block the request and send 429 http status
(e) else, check if sum of number of requests for current epoch is less than secondary threshold - O(1)
(f) if not, return false else return true

Note: We are first writing then reading. This is primarily to avoid race condition. We may delay unblocking user if limit is exceeded but that should be fine.

https://www.youtube.com/@15minutesystemdesign



Rate Limiter Algorithm (using redis sorted set)

Repeat for each incoming api request from user X:
(a) remove all entries older than rolling window time for user X - O(log n)
(b) add request entry for current epoch - O(1)
(c) check if sum of number of requests across all epochs is less than primary threshold - O(1)
(d) if not, block the request and send 429 http status
(e) else, check if sum of number of requests for current epoch is less than secondary threshold - O(1)
(f) if not, return false else return true

Note: We are first writing then reading. This is primarily to avoid race condition. We may delay unblocking user if limit is exceeded but that should be fine.

If a user sends bulk api requests at same time, race condition may occur

---

Solution => Rate limiter deletes all epochs older than 15 minute from now, locks the user record, updates request count for current epoch then checks if per_minute and per_15_minute limits breached

https://www.youtube.com/@15minutesystemdesign

Like, share, Subscribe

https://www.youtube.com/@15minutesystemdesign