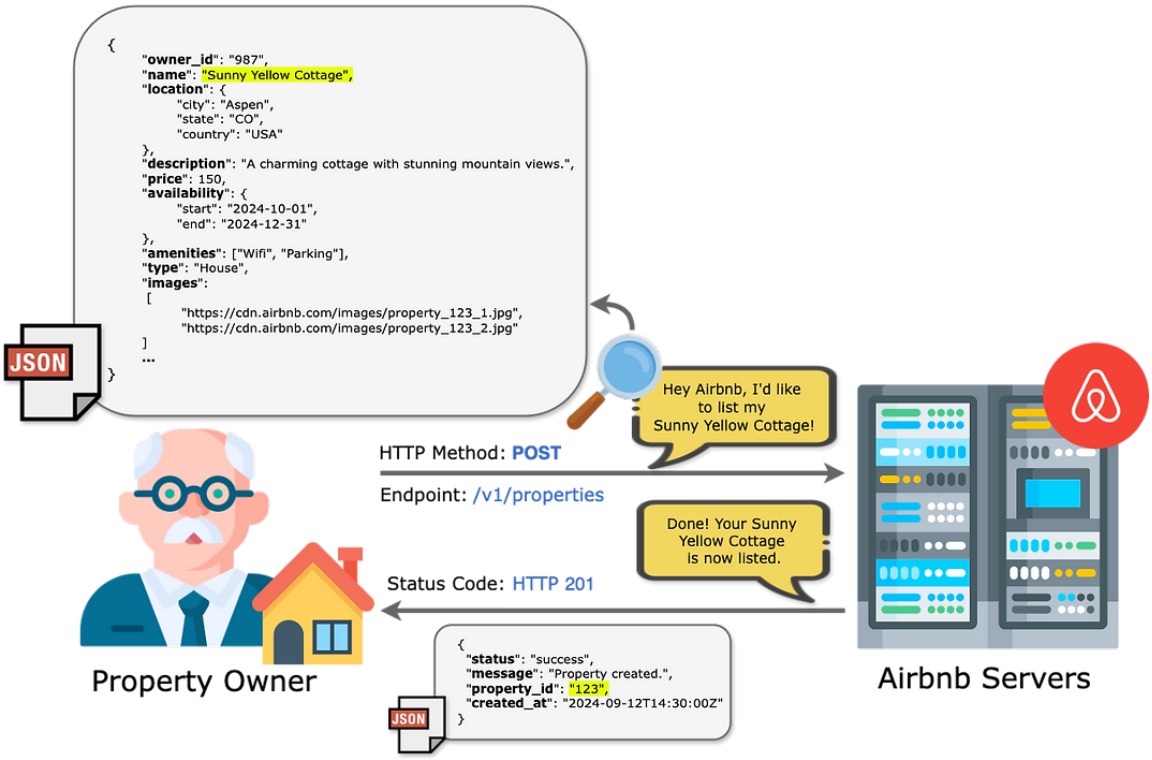
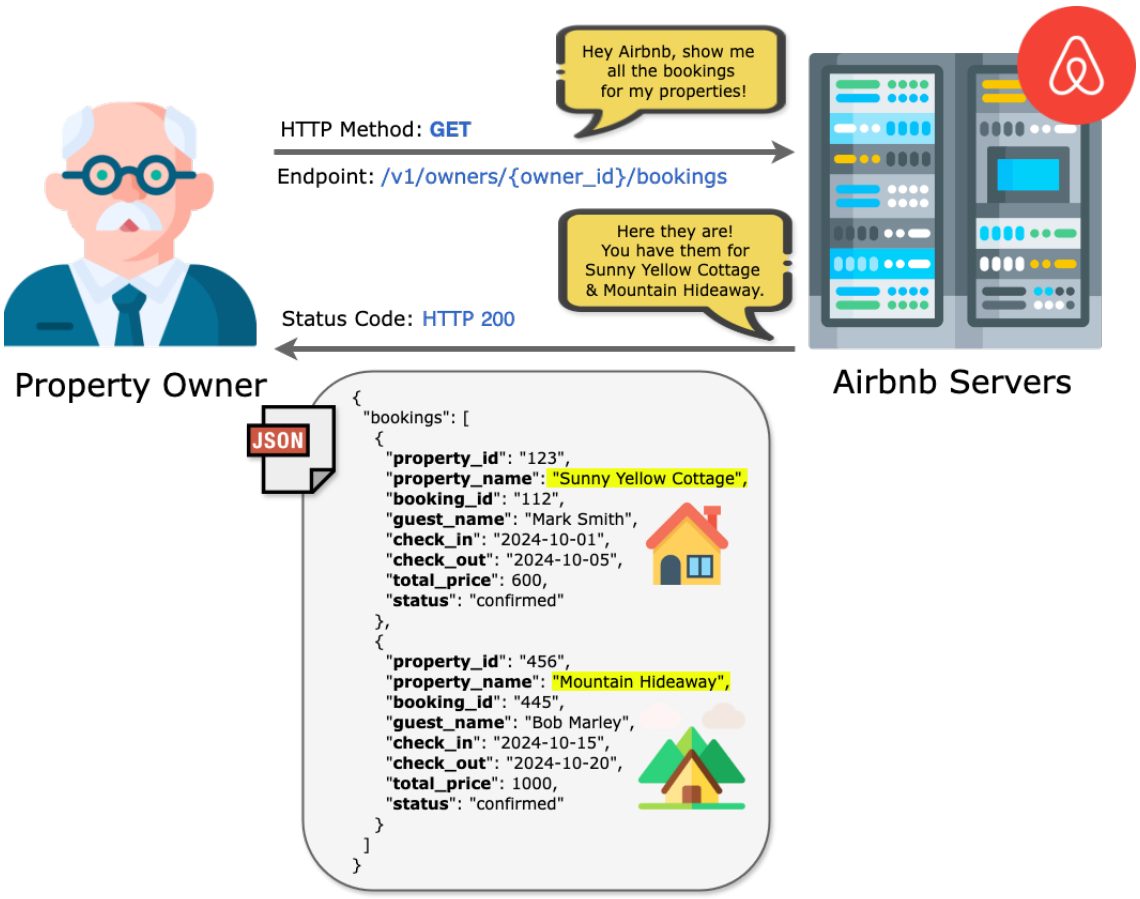
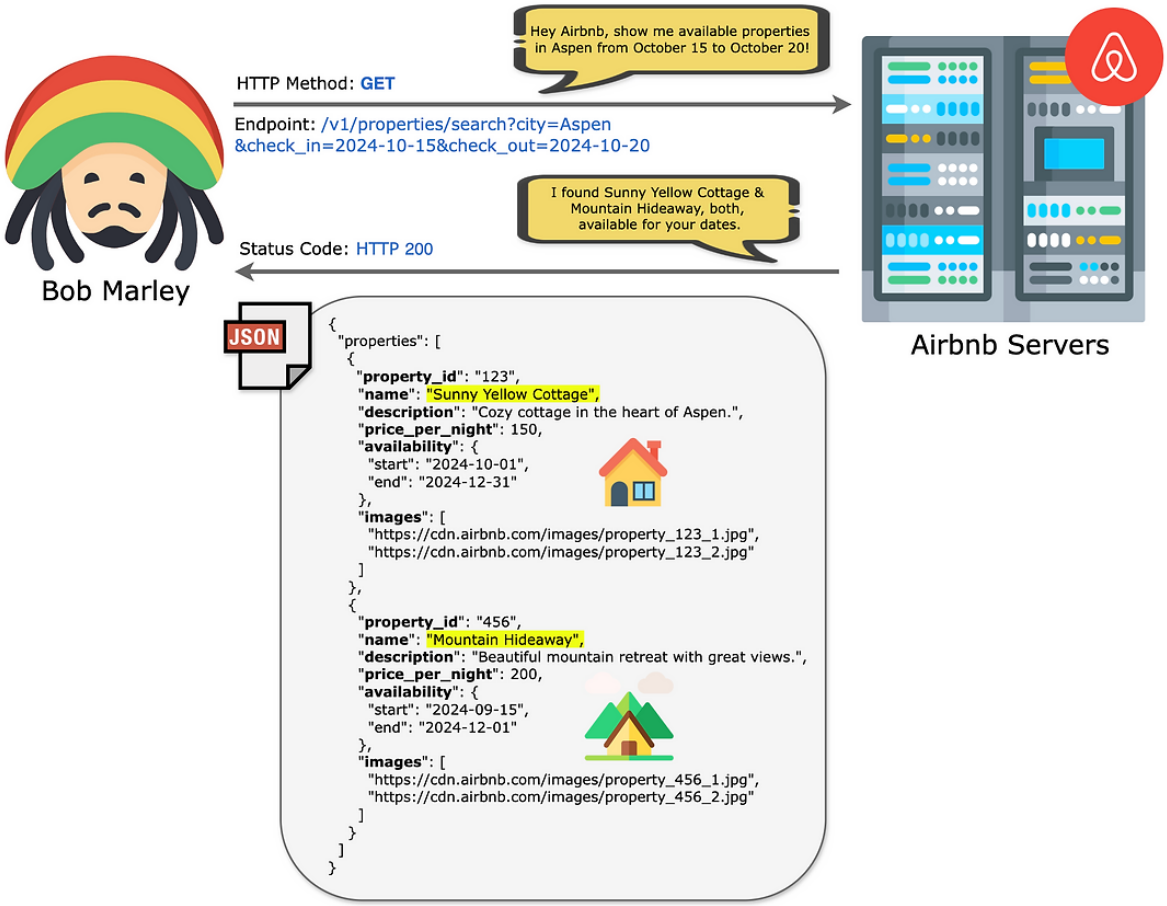
API Design-Add Property (Property Owner)



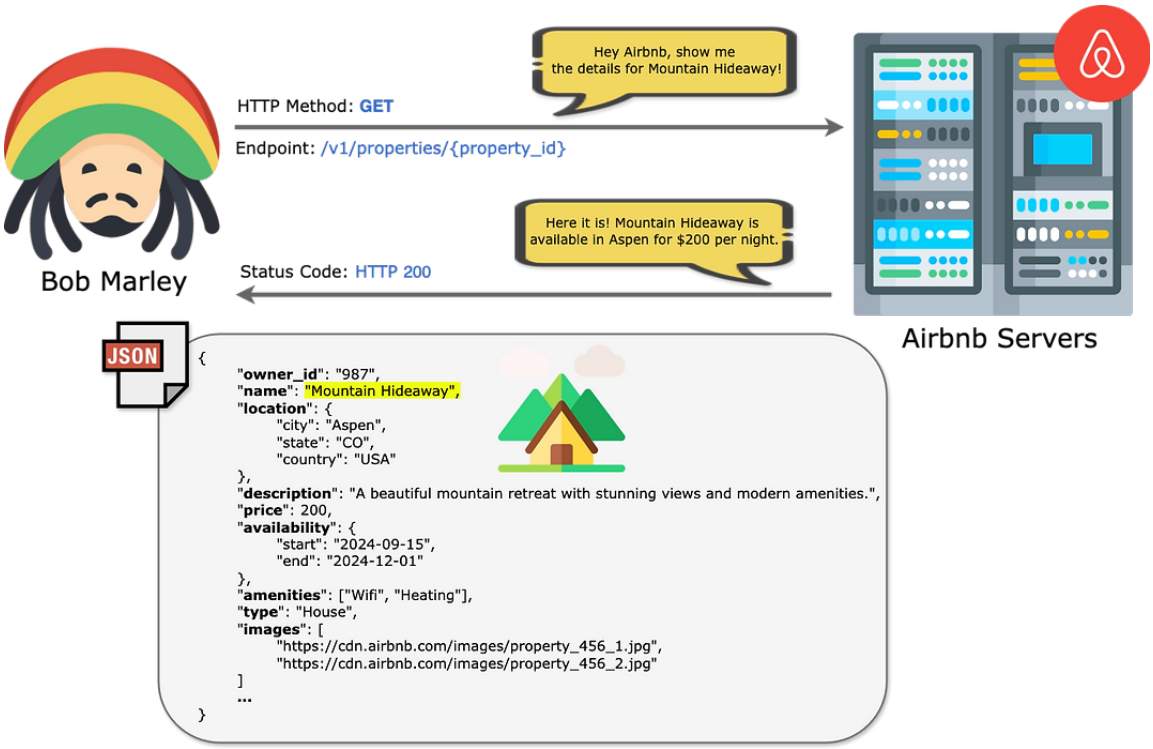
API Design-View Bookings (Property Owner)



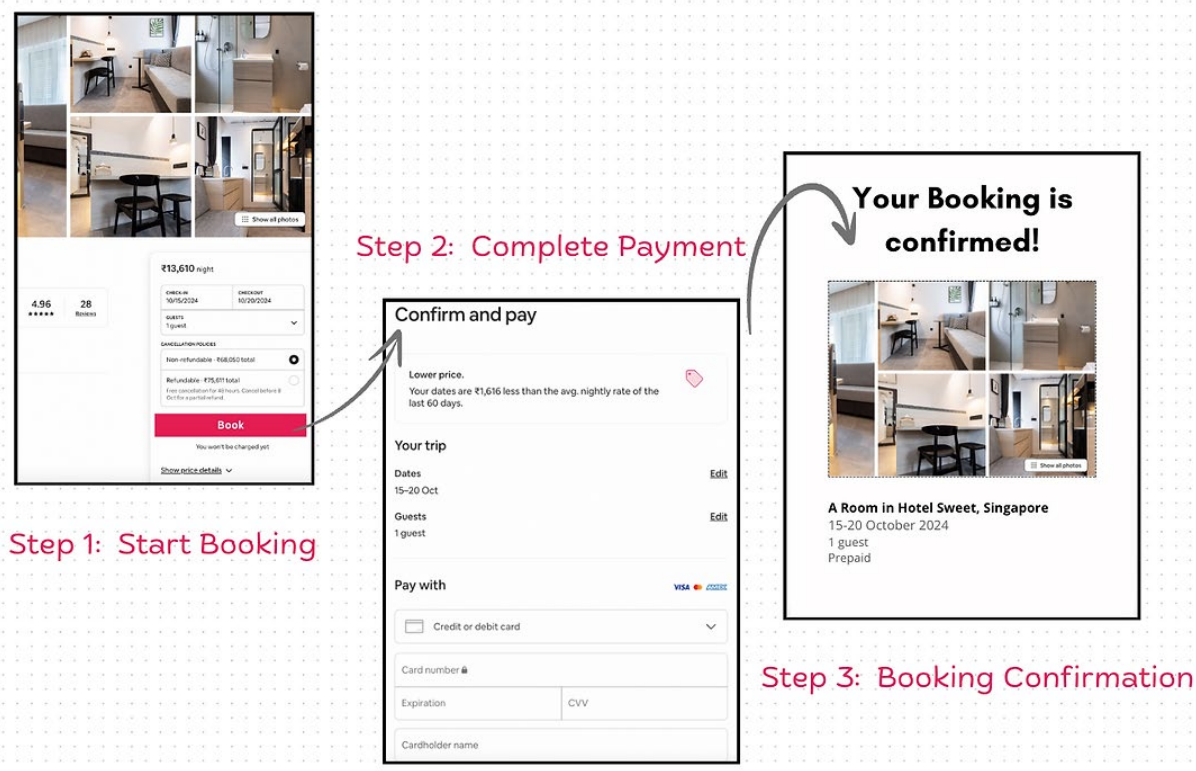
API Design-Search Properties (Guest)

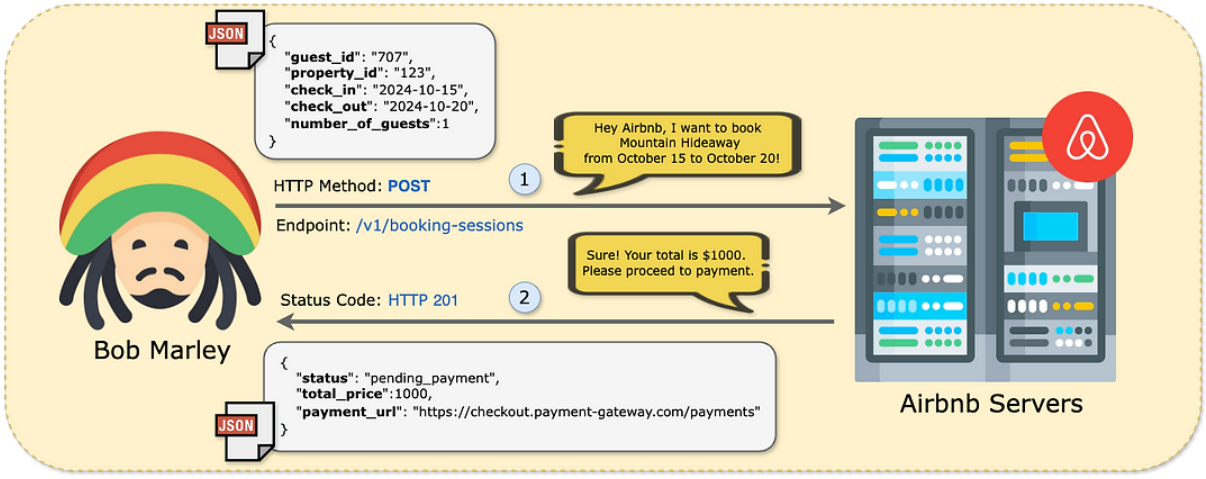


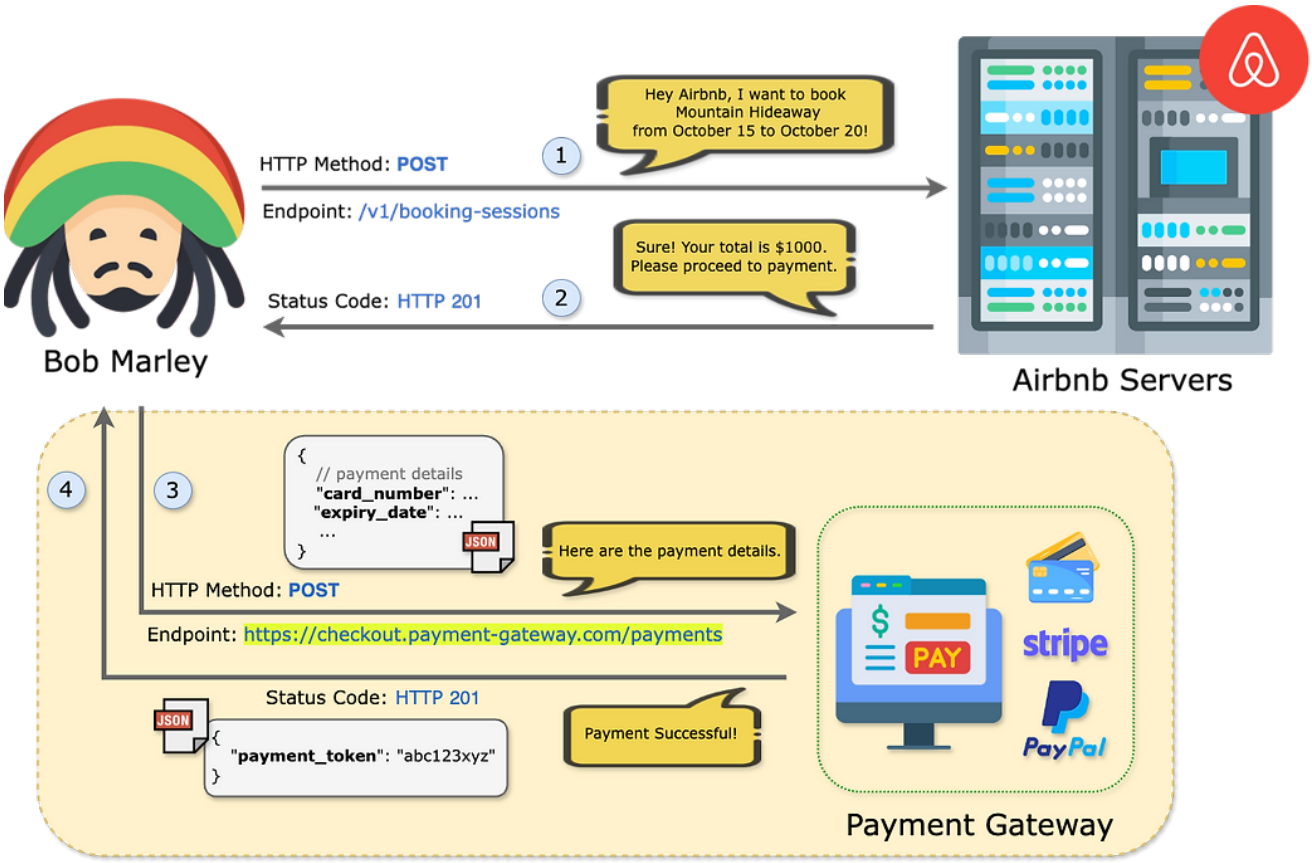
API Design-View Property (Guest)

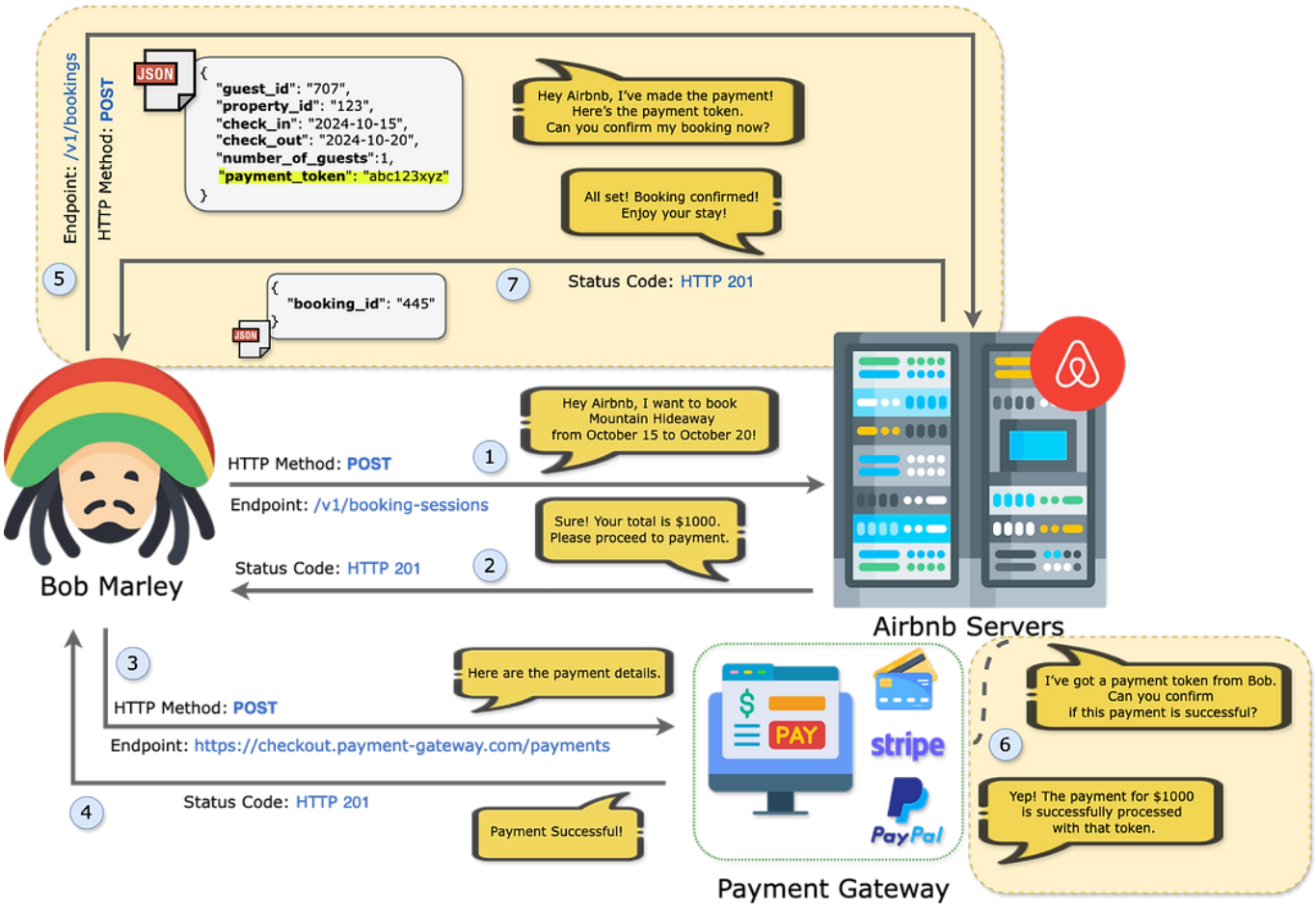


API Design-Book Property (Guest)

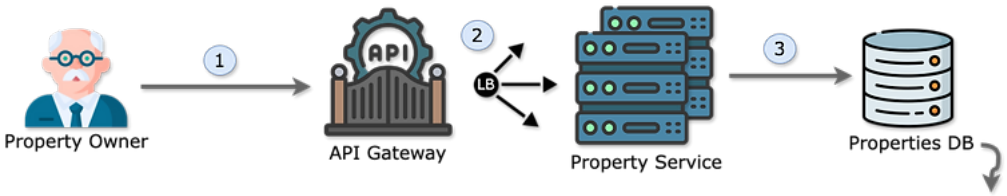




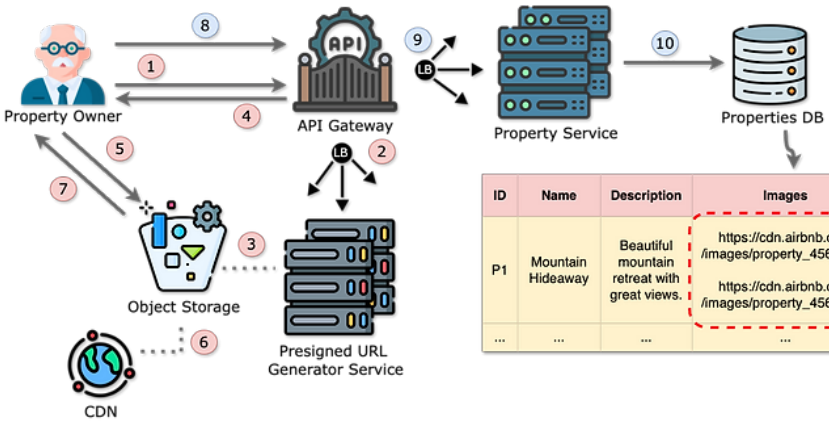

Hey Airbnb, I want to book Mountain Hideaway from October 15 to October 20!
1
HTTP Method: POST
Endpoint: /v1/booking-sessions
Sure! Your total is $1000. Please proceed to payment.
Status Code: HTTP 201
2
Bob Marley
Airbnb Servers
4
3
{
// payment details
"card_number": ...
"expiry_date": ...
...
}
JSON
Here are the payment details.
HTTP Method: POST
Endpoint: https://checkout.payment-gateway.com/payments
Status Code: HTTP 201
JSON
{
"payment_token": "abc123xyz"
}
Payment Successful!
PAY
stripe
PayPal
Payment Gateway


Endpoint: /v1/bookings
HTTP Method: POST
JSON
{
"guest_id": "707",
"property_id": "123",
"check_in": "2024-10-15",
"check_out": "2024-10-20",
"number_of_guests":1,
"payment_token": "abc123xyz"
}
Hey Airbnb, I've made the payment! Here's the payment token. Can you confirm my booking now?
All set! Booking confirmed! Enjoy your stay!
5
7
Status Code: HTTP 201
{
"booking_id": "445"
}
JSON
Hey Airbnb, I want to book Mountain Hideaway from October 15 to October 20!
1
HTTP Method: POST
Endpoint: /v1/booking-sessions
Sure! Your total is $1000. Please proceed to payment.
Status Code: HTTP 201
2
Bob Marley
Airbnb Servers
3
Here are the payment details.
HTTP Method: POST
Endpoint: https://checkout.payment-gateway.com/payments
4
Status Code: HTTP 201
Payment Successful!
PAY
stripe
PayPal
Payment Gateway
I've got a payment token from Bob. Can you confirm if this payment is successful?
6
Yep! The payment for $1000 is successfully processed with that token.

High Level Design-Add/Update property

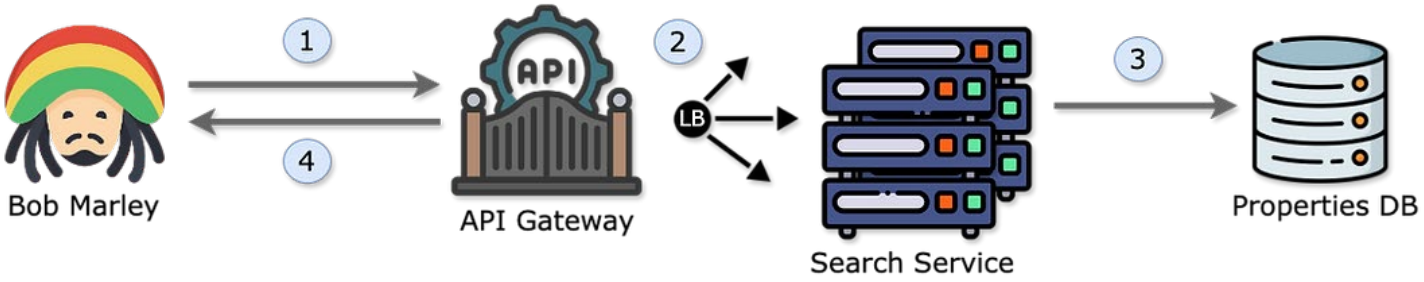


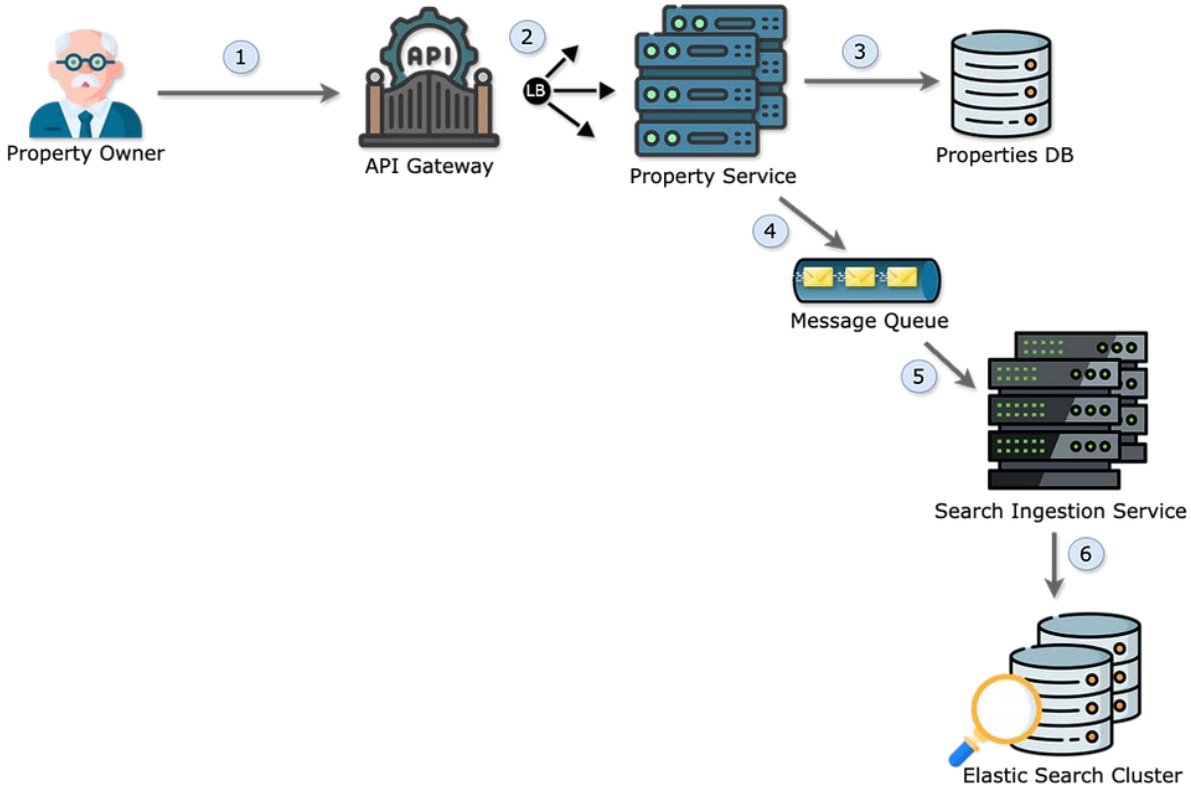
ID	Name	Description	Images	Price	Available Start Dates	Available End Dates	OwnerID	City	State	Country	...
P1	Mountain Hideaway	Beautiful mountain retreat with great views.	N/A	200	2024-09-15	2024-12-01	987	Aspen	CO	USA	...
...	...	...	...	...	...	...	...	...	...	...	...



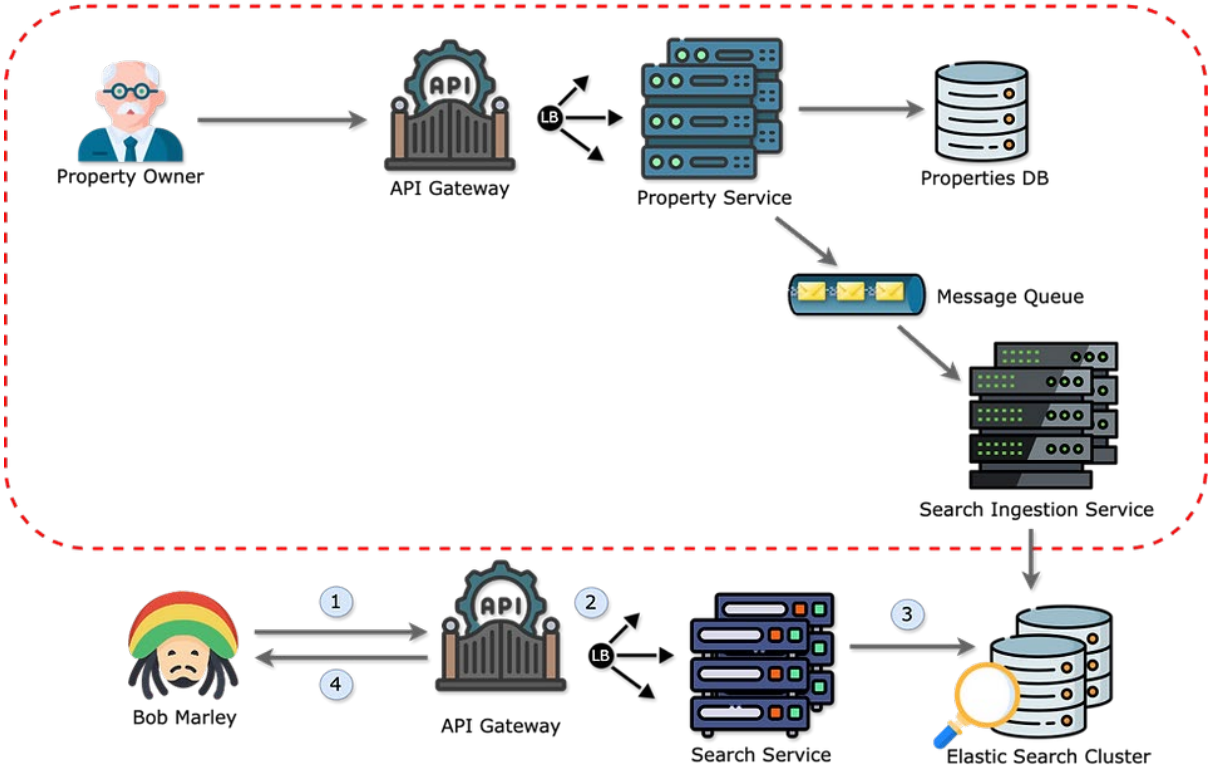
ID	Name	Description	Images	Price	Available Start Dates	Available End Dates	OwnerID	City	State	Country	...
P1	Mountain Hideaway	Beautiful mountain retreat with great views.	https://cdn.airbnb.com /images/property_456_1.jpg https://cdn.airbnb.com /images/property_456_2.jpg	200	2024-09-15	2024-12-01	987	Aspen	CO	USA	...
...	...	...	...	...	...	...	...	...	...	...	...

High Level Design- Search Property (Guest)

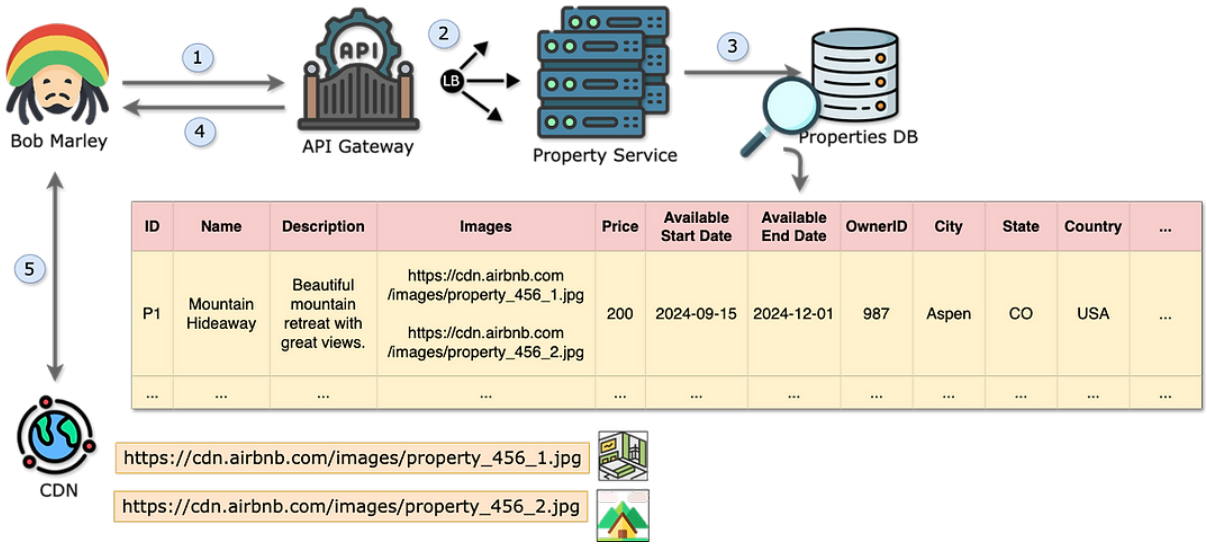




Search Data 'Pre'-Processing



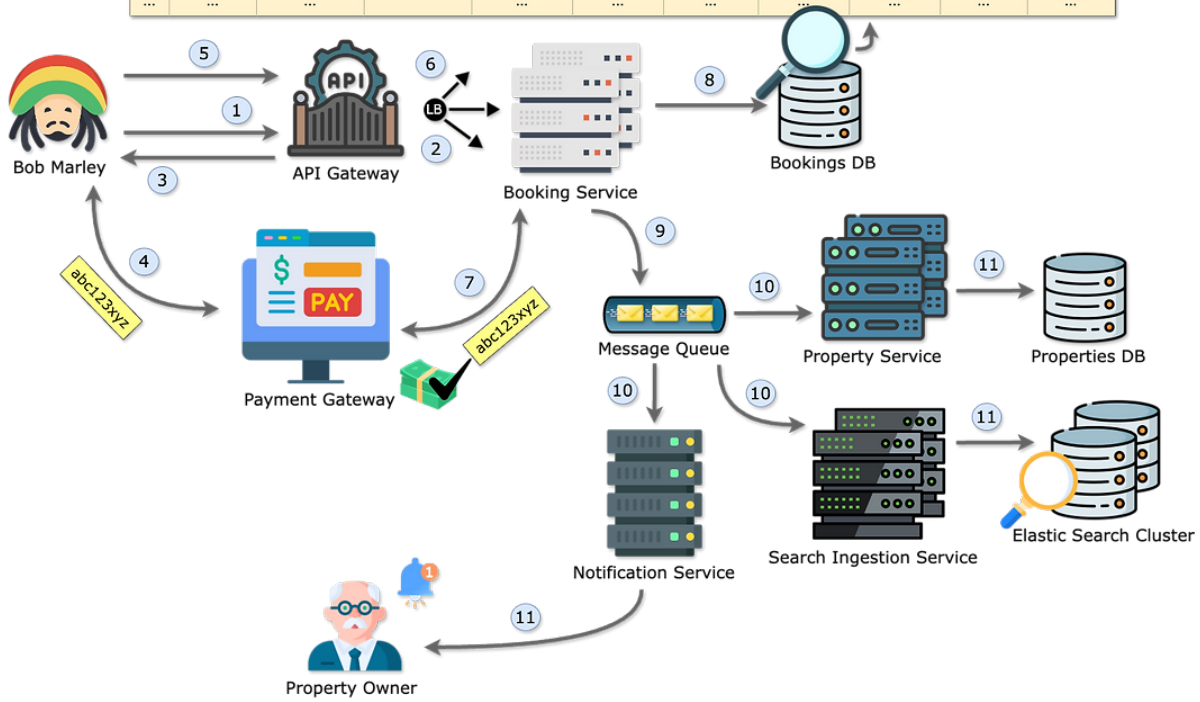
High Level Design-View Property (Guest)



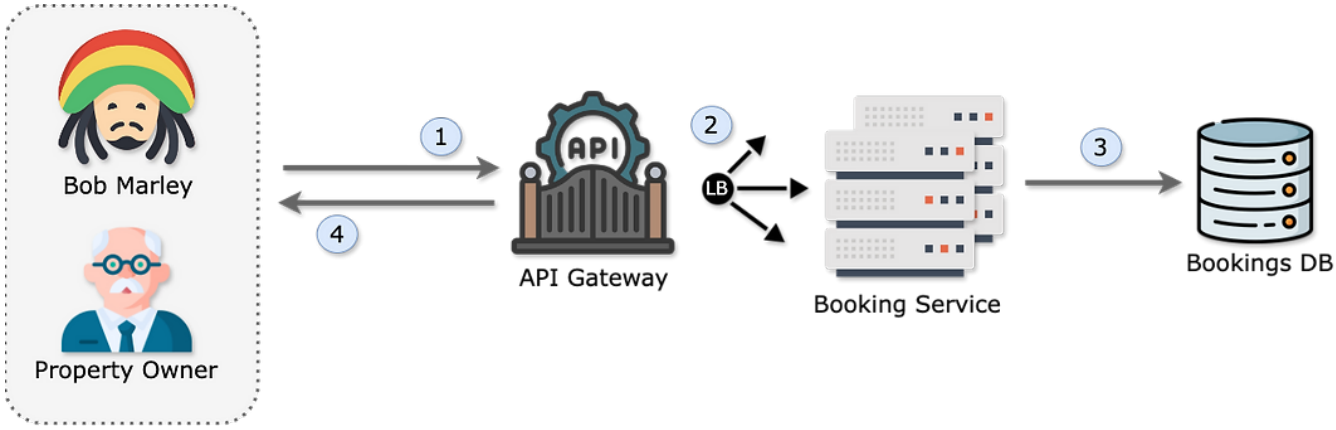
ID	Name	Description	Images	Price	Available Start Date	Available End Date	OwnerID	City	State	Country	...
P1	Mountain Hideaway	Beautiful mountain retreat with great views.	https://cdn.airbnb.com /images/property_456_1.jpg https://cdn.airbnb.com /images/property_456_2.jpg	200	2024-09-15	2024-12-01	987	Aspen	CO	USA	...
...	...	...	...	...	...	...	...	...	...	...	...

High Level Design-Book Property (Guest)

ID	Property ID	Guest ID	Guest Name	Owner ID	Check In	Check Out	Total Price	Payment Token	Booking Status	...
B1	P1	707	Bob Marley	987	2024-10-15	2024-10-20	1000	abc123xyz	Confirmed	...
...	...	...	...	...	...	...	...	...	...	...



High Level Design-View Bookings (Property Owner/Guest)



Properties DB Schema

{

PropertyID: A unique ID for each property listing.
Name: The name of the property.
Description: A brief description of the property.
Images: URLs to the property images.
Price: Price per night.
AvailableStartDates: List of start dates when the property is available.
AvailableEndDates: List of end dates when the property is no longer available.
OwnerID: ID of the property owner.
City: City where the property is located.
State: State where the property is located.
Country: Country where the property is located.
Amenities: ...
...

}

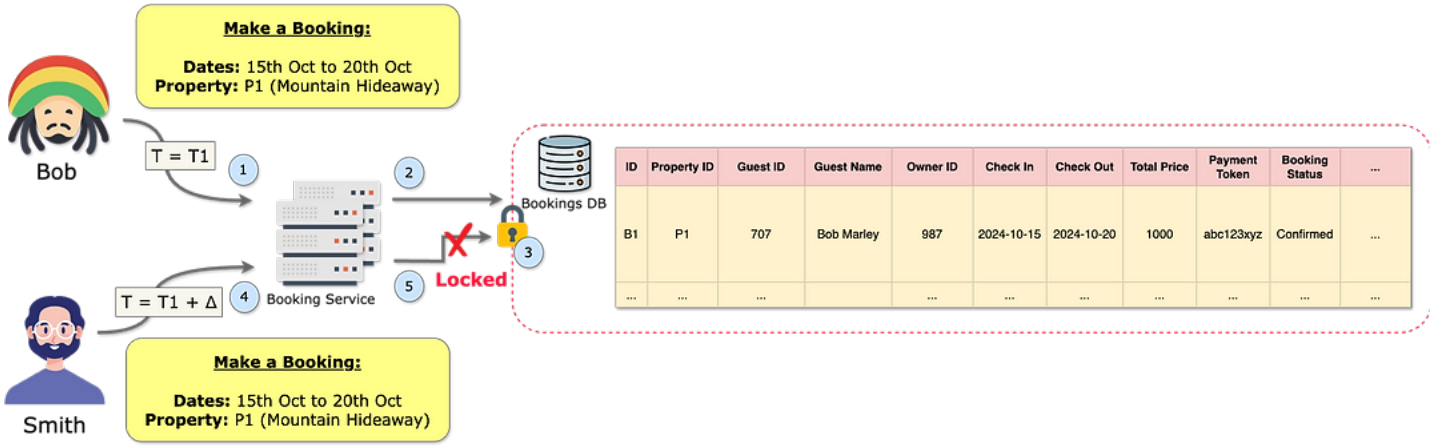
Bookings DB Schema

{

BookingID: A unique identifier for each group.
PropertyID: The ID of the property being booked.
GuestID: The ID of the guest making the booking.
GuestName: Name of the guest.
OwnerID: The ID of the property owner.
CheckInDate: Date when the guest will check in.
CheckOutDate: Date when the guest will check out.
TotalPrice: Total price for the stay.
PaymentToken: Token received after successful payment.
BookingStatus: Status of the booking (e.g., "Confirmed", "Completed", "Cancelled")
...

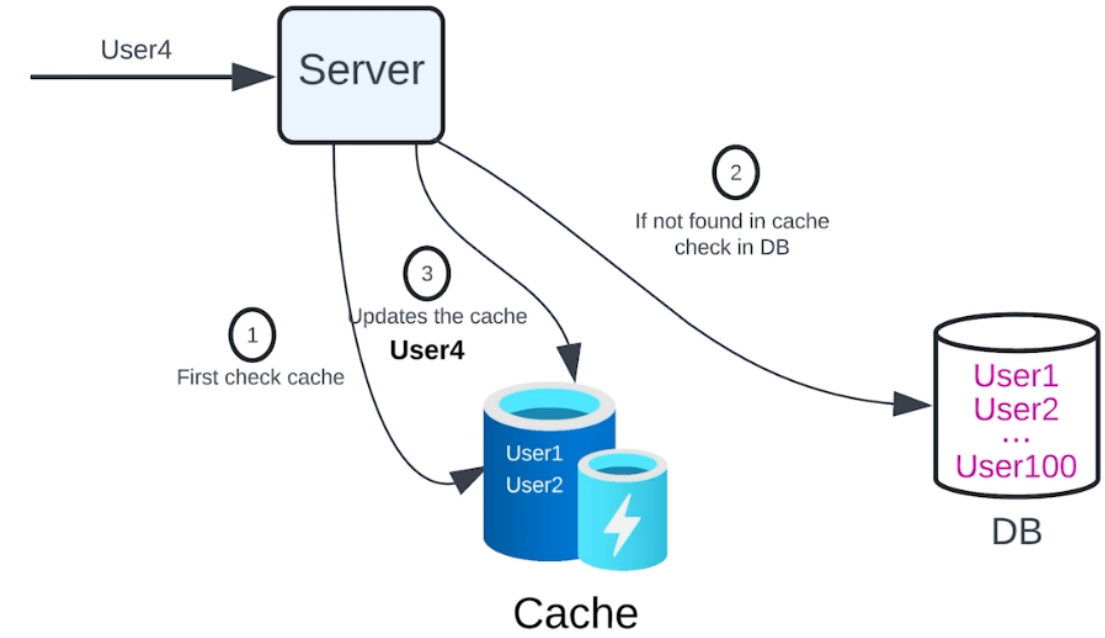
}

Handling Concurrent Bookings

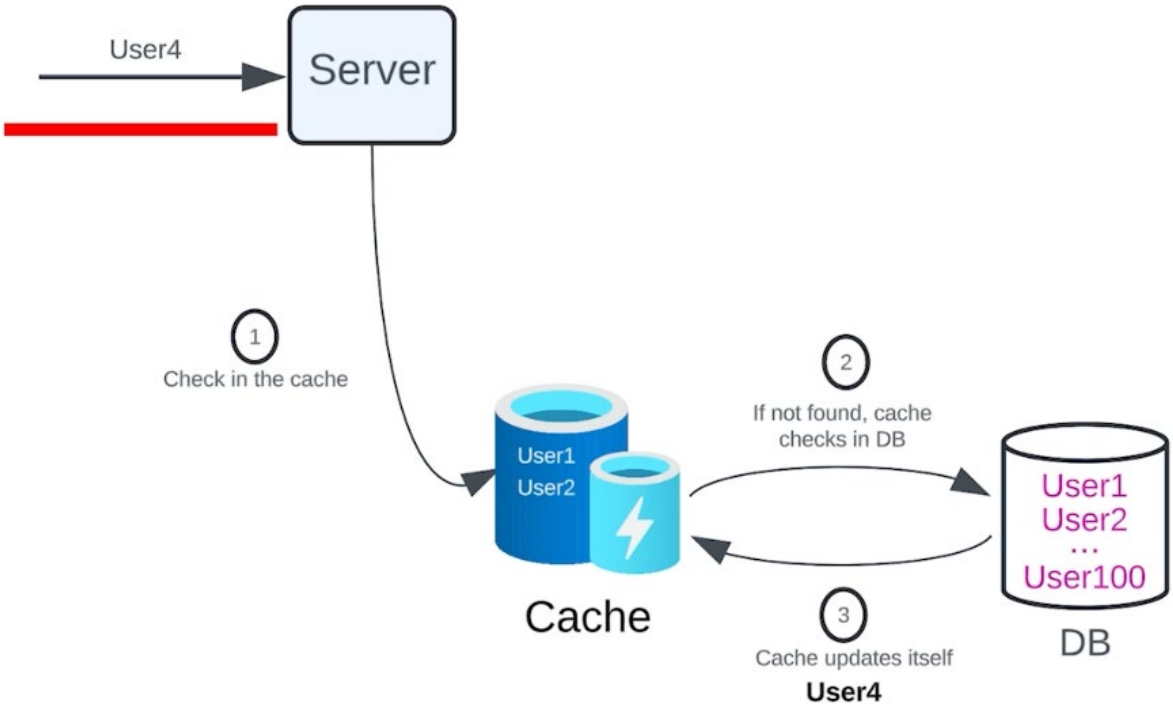


Caching Strategies (Read)

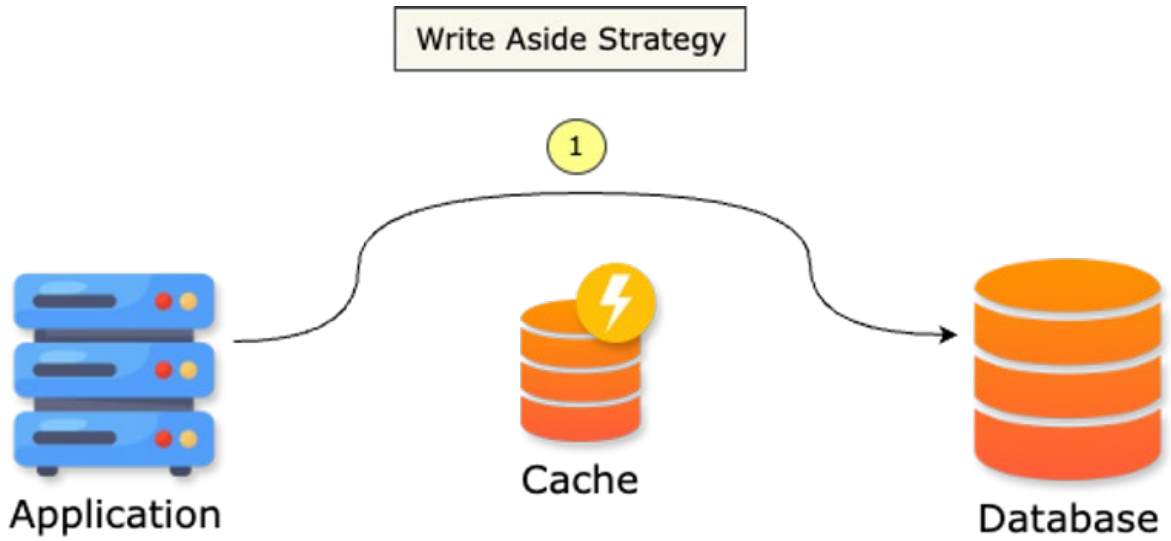
Cache Aside Strategy

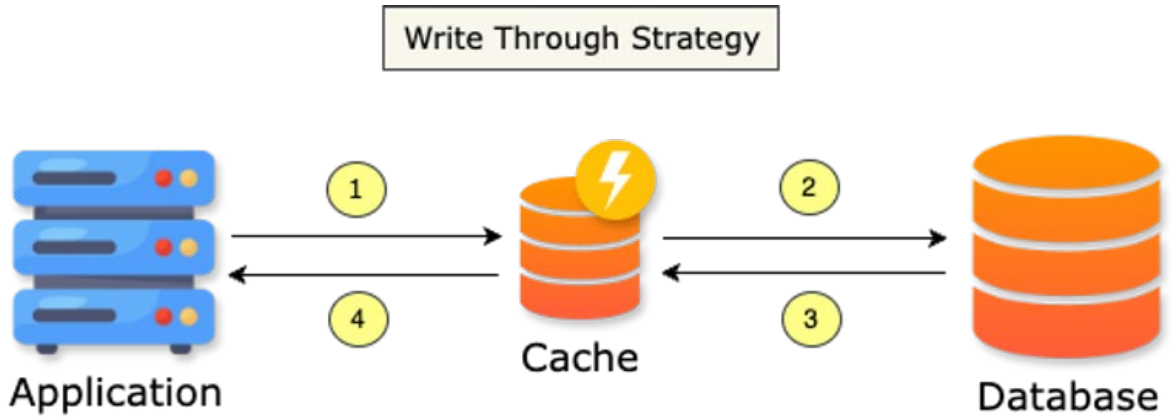


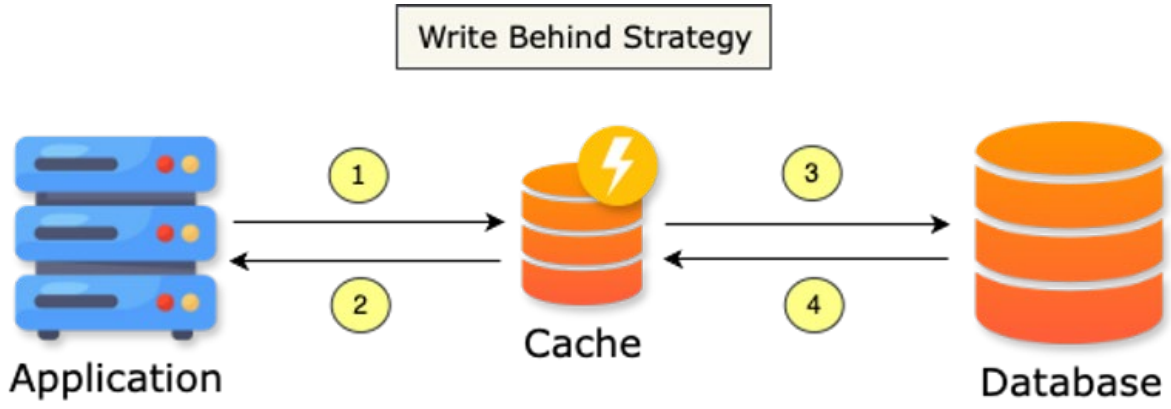
Read Through Strategy



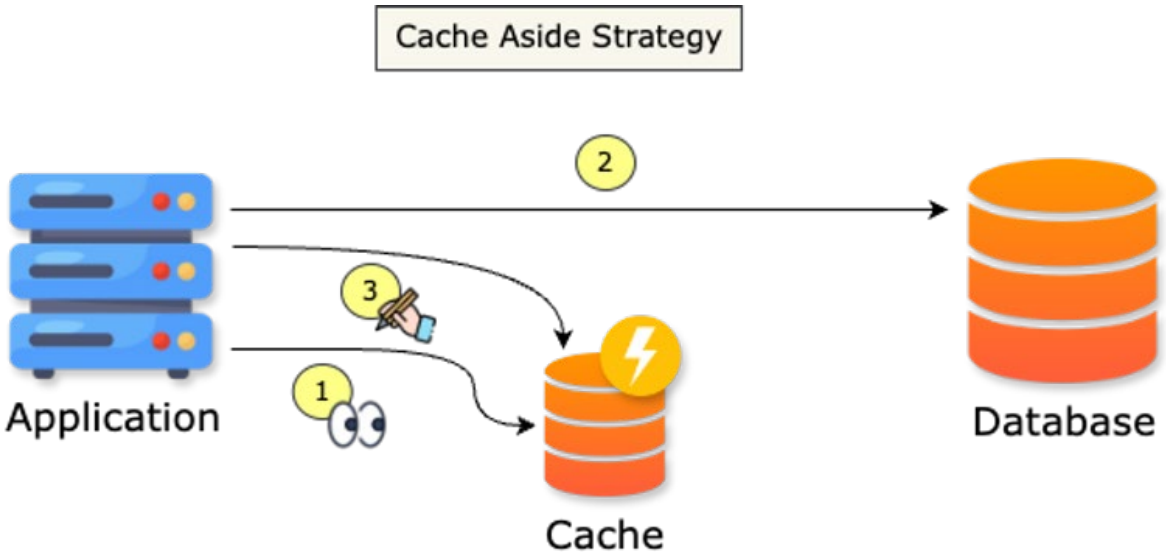
Caching Strategies (Write)

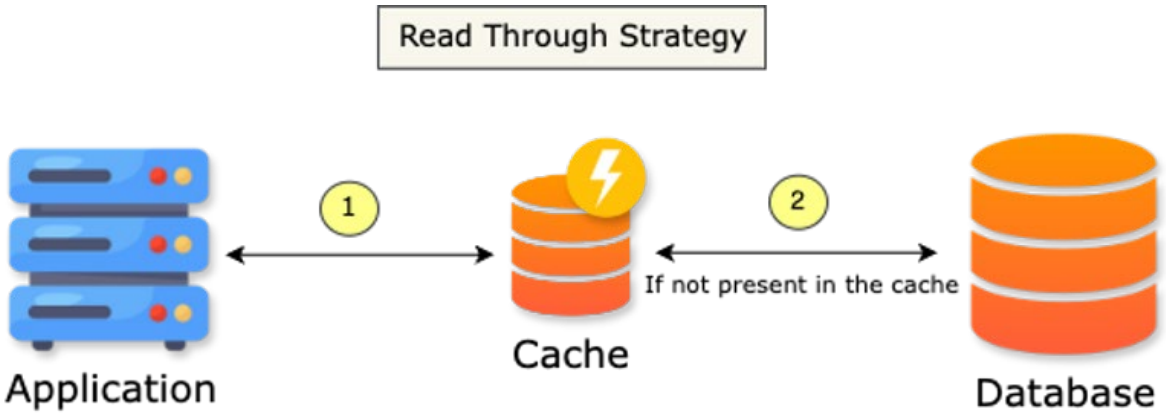




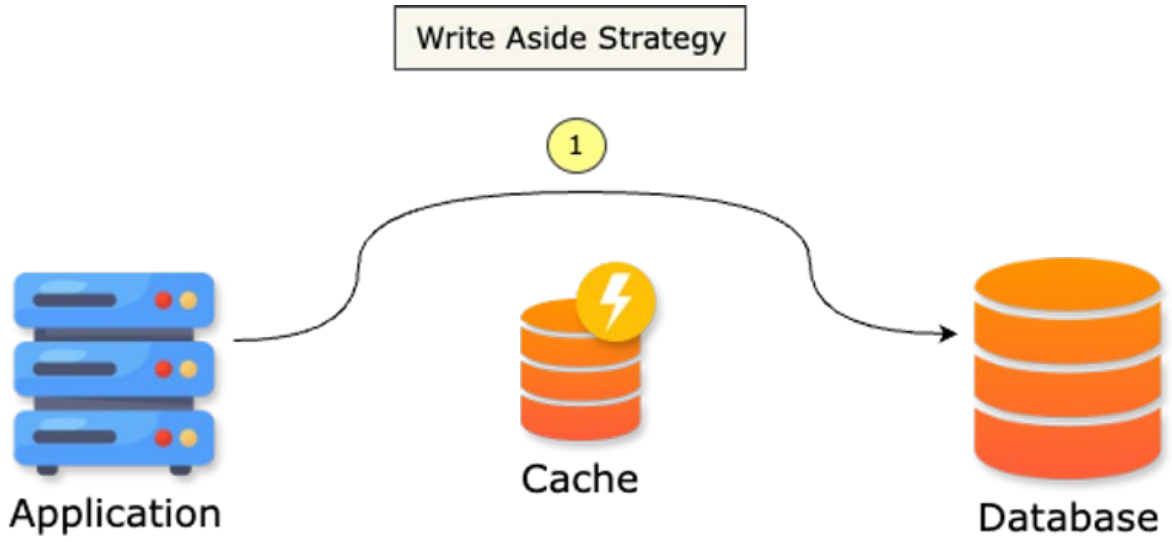


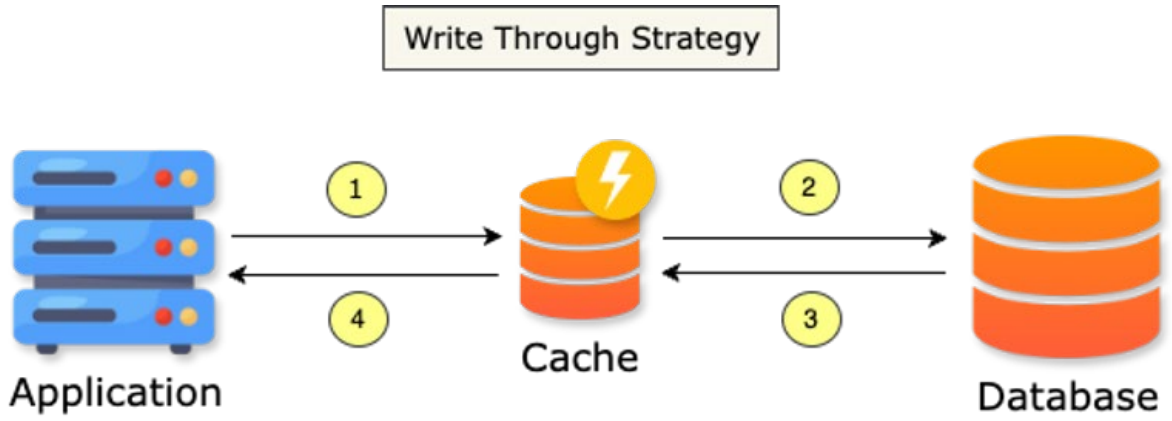
Read Caching Strategies (Pros & Cons)

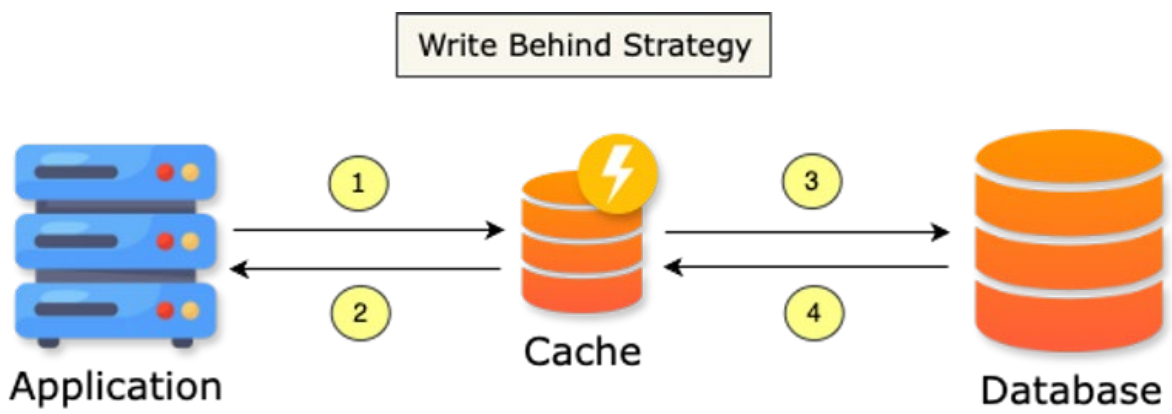




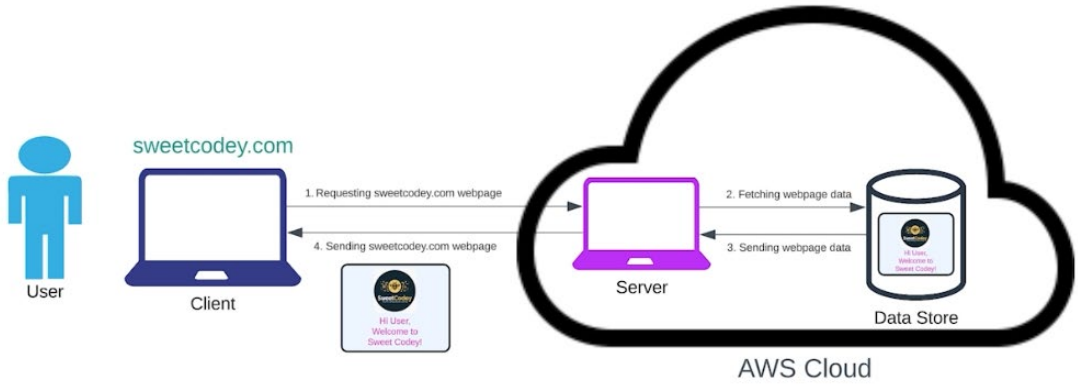
Read Caching Strategies (Pros & Cons)



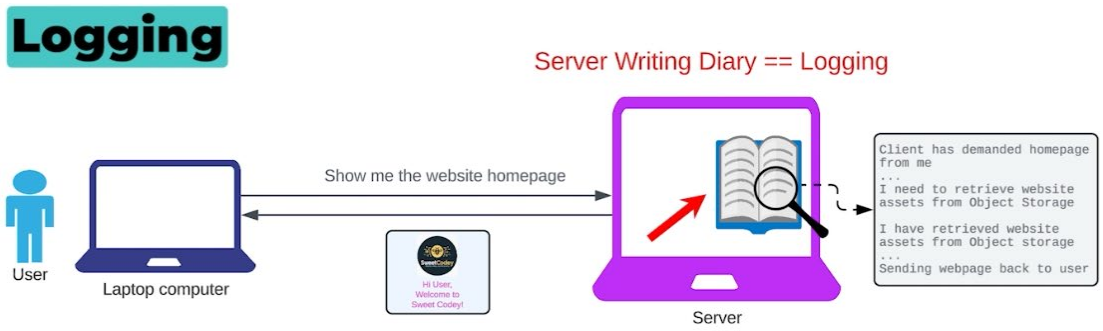


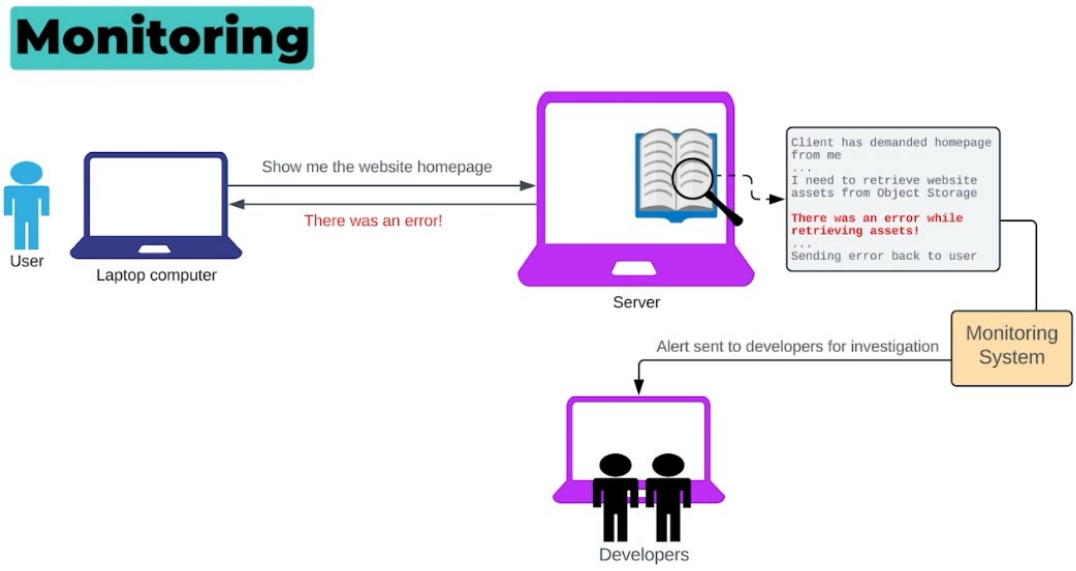


Cloud Computing

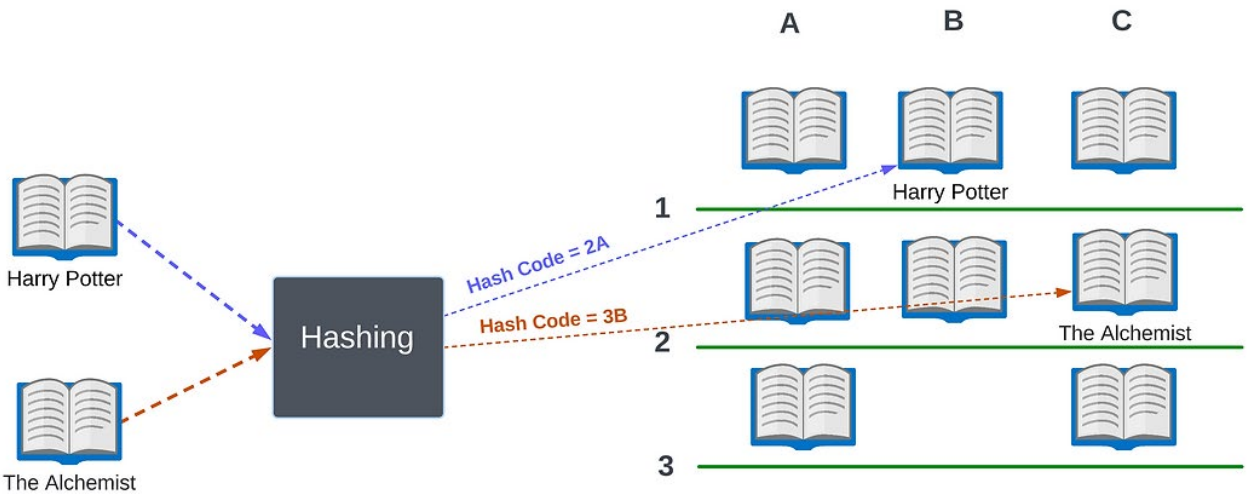


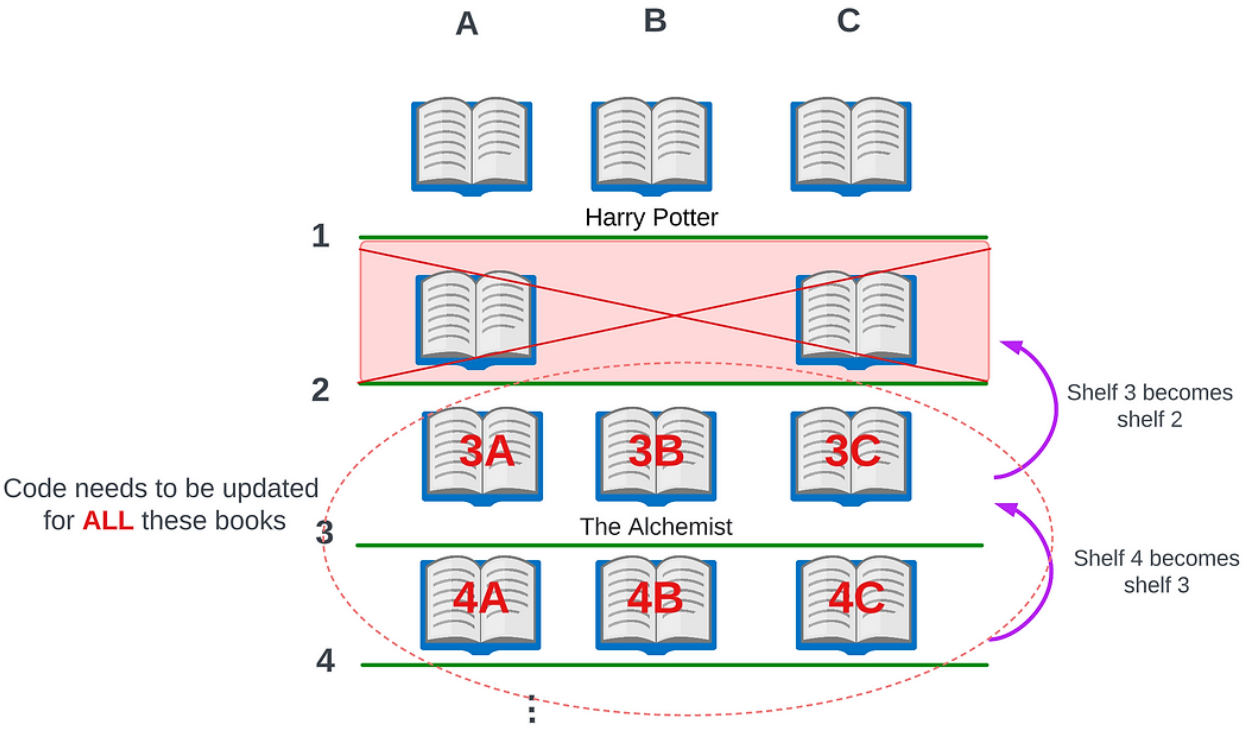
Logging & Monitoring



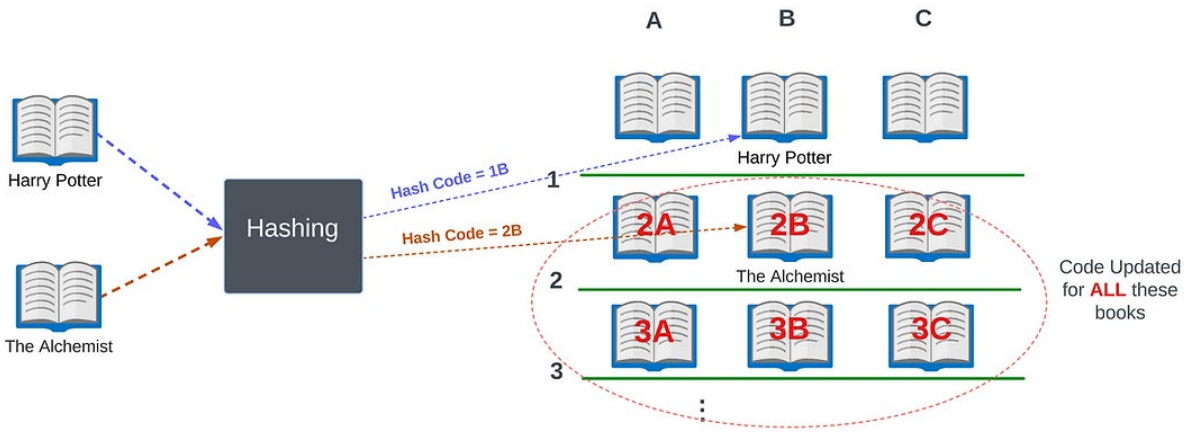


Hashing

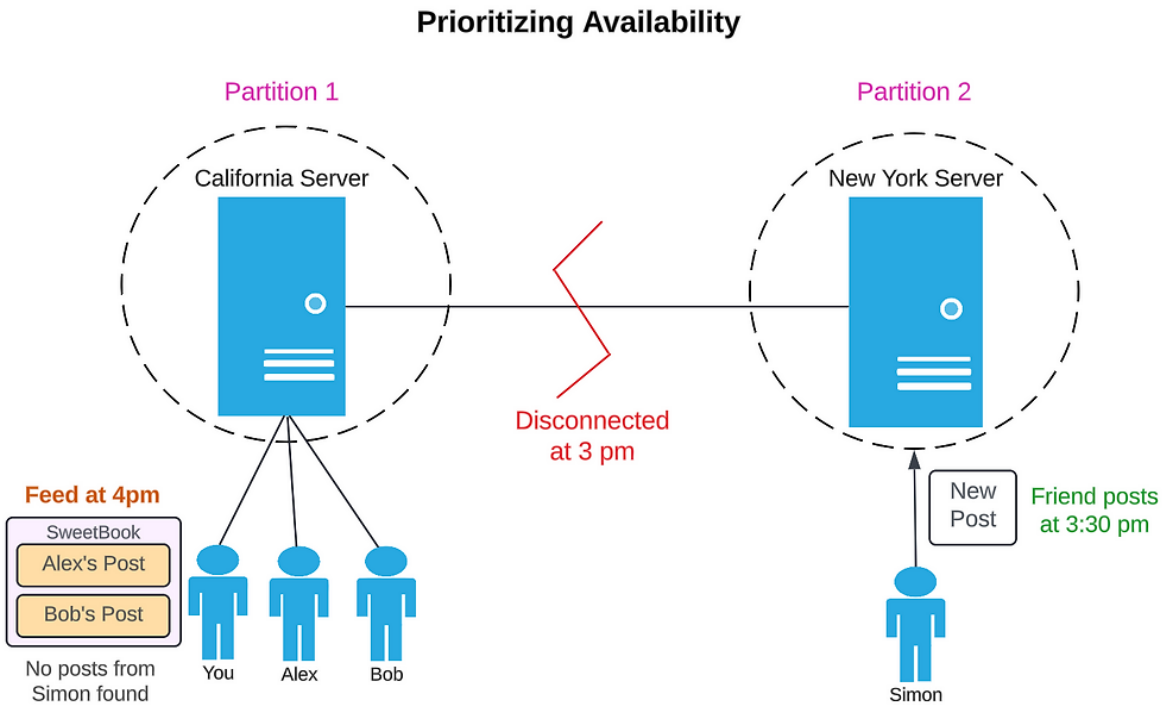

A
B
C
Harry Potter
Hashing
The Alchemist
Hash Code = 2A
Hash Code = 3B
1
2
3
Harry Potter
The Alchemist


A
B
C
Harry Potter
1
2
3
4
3A
3B
3C
4A
4B
4C
The Alchemist
Code needs to be updated for ALL these books
Shelf 3 becomes shelf 2
Shelf 4 becomes shelf 3

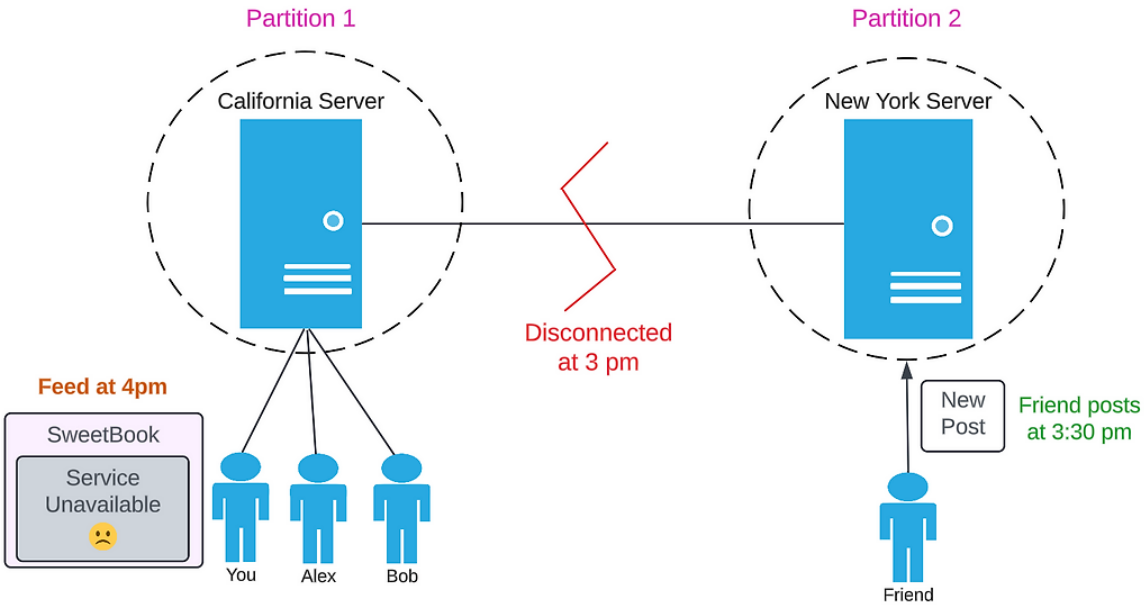
Consistent Hashing



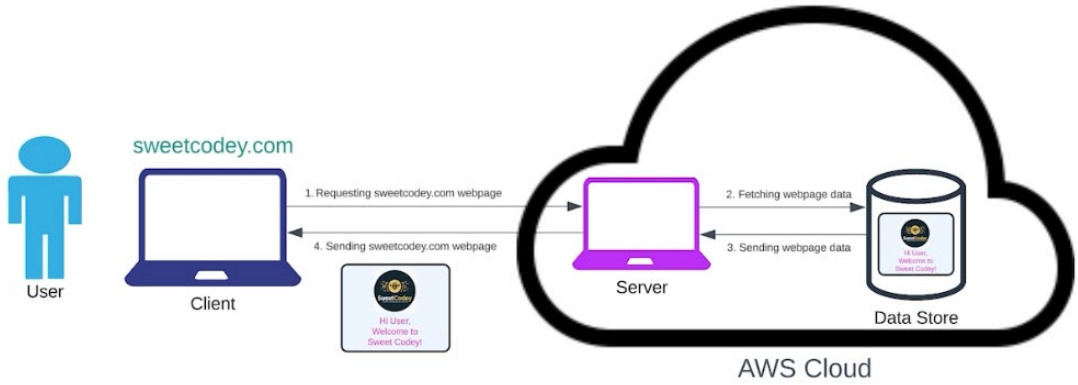
CAP THEORM



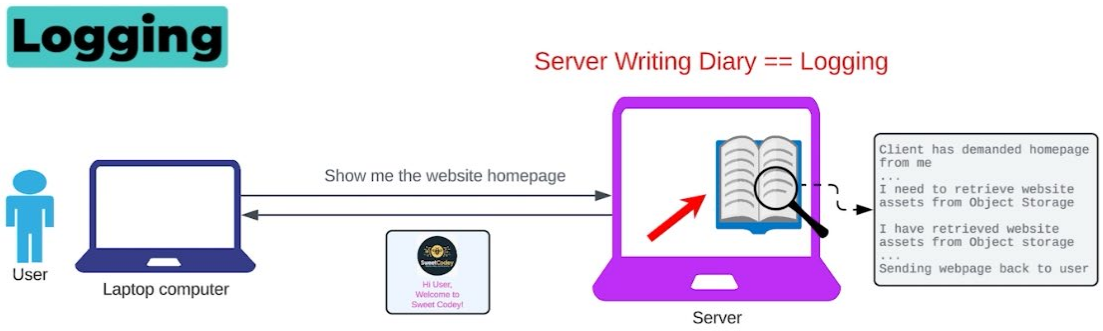
Prioritizing Consistency

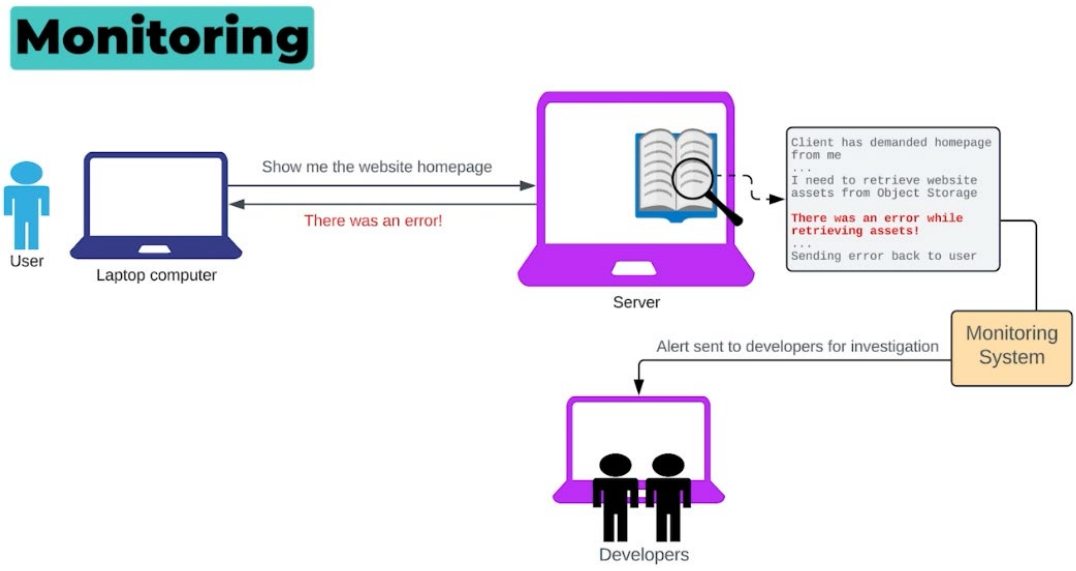


Cloud Computing

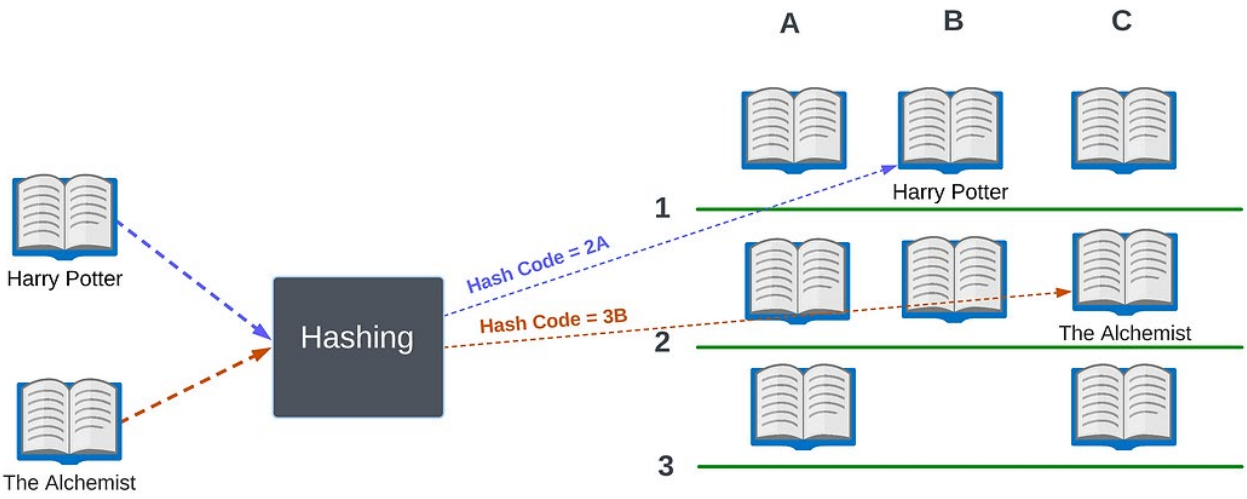


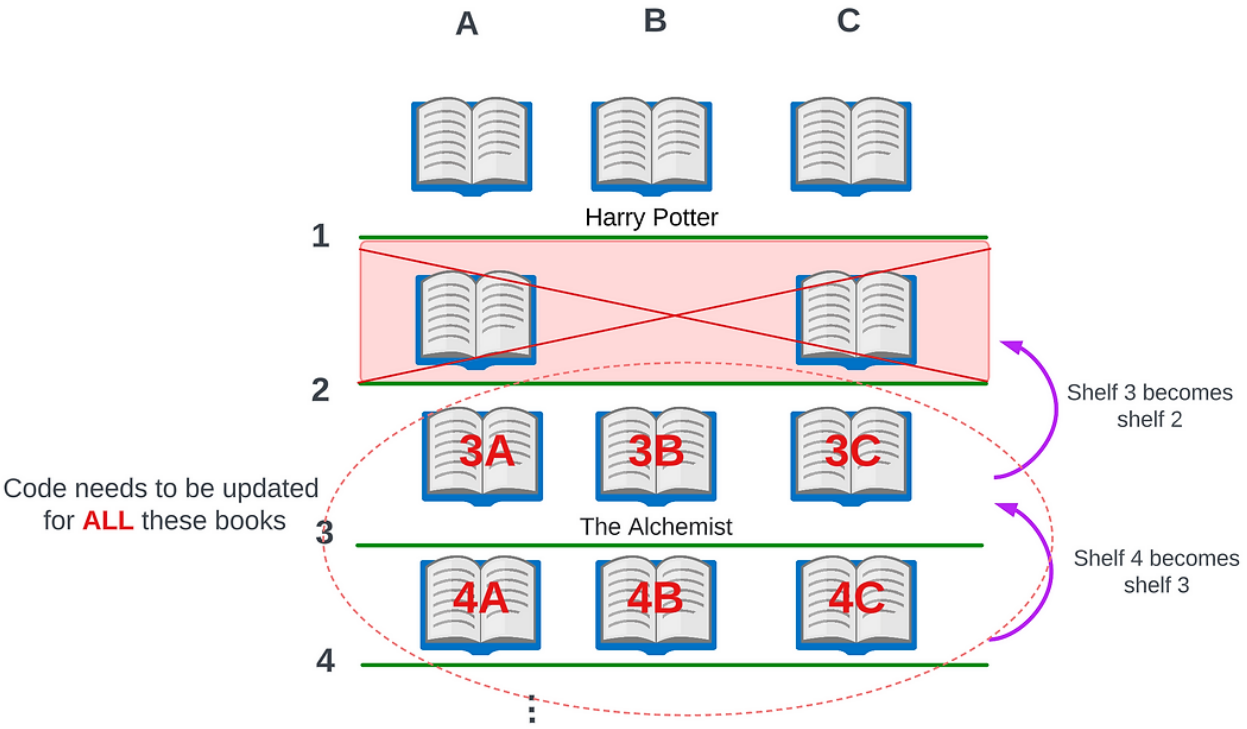
Logging & Monitoring



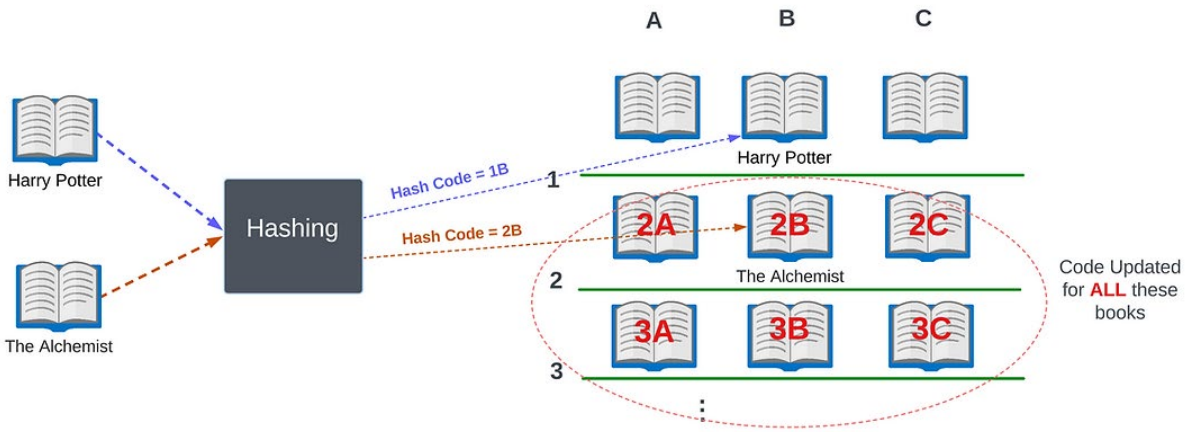


Hashing

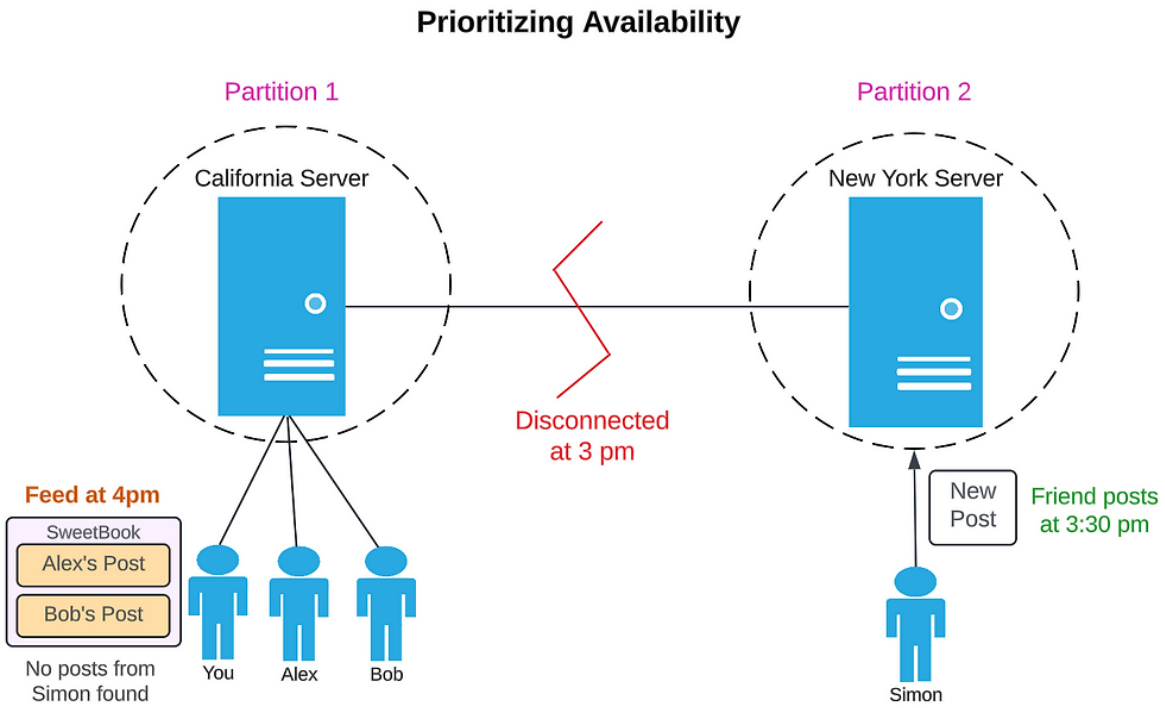

A
B
C
Harry Potter
Hashing
The Alchemist
Hash Code = 2A
Hash Code = 3B
1
2
3
Harry Potter
The Alchemist


A
B
C
Harry Potter
1
2
3
4
3A
3B
3C
4A
4B
4C
The Alchemist
Code needs to be updated for ALL these books
Shelf 3 becomes shelf 2
Shelf 4 becomes shelf 3

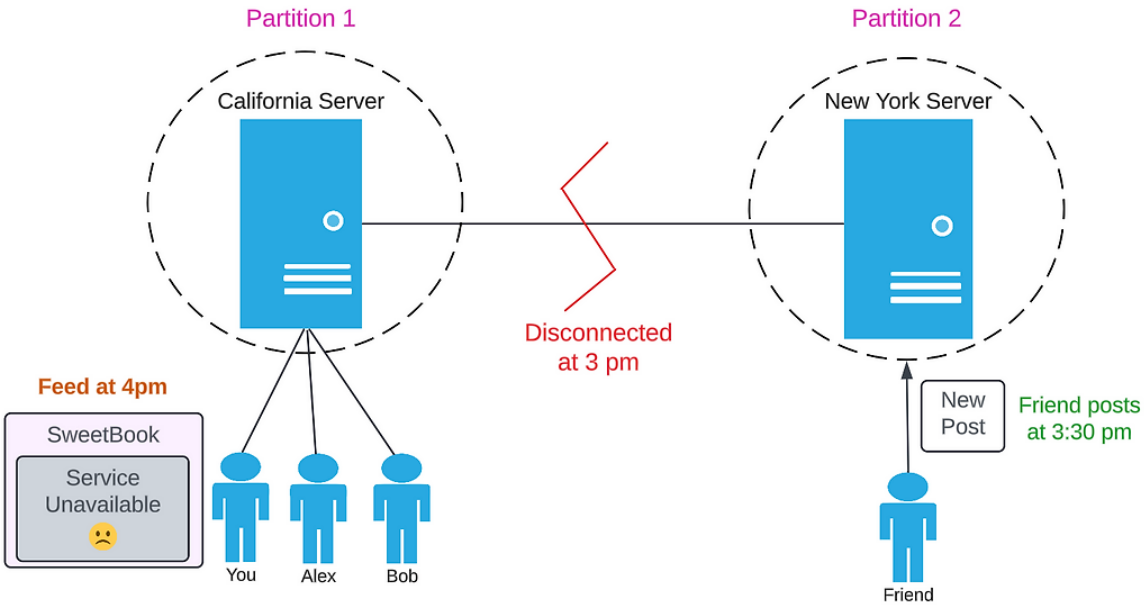
Consistent Hashing



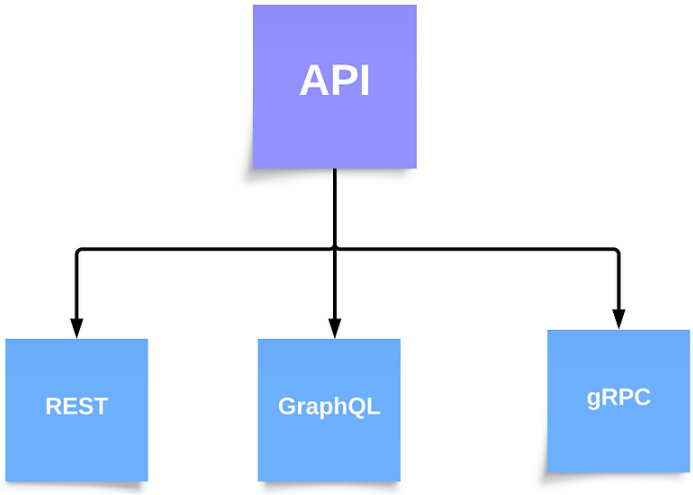
CAP THEORM



Prioritizing Consistency

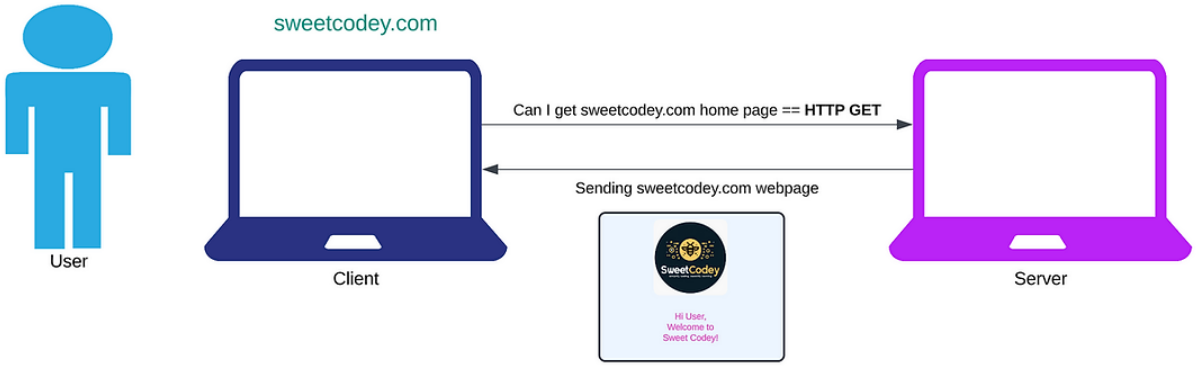


API

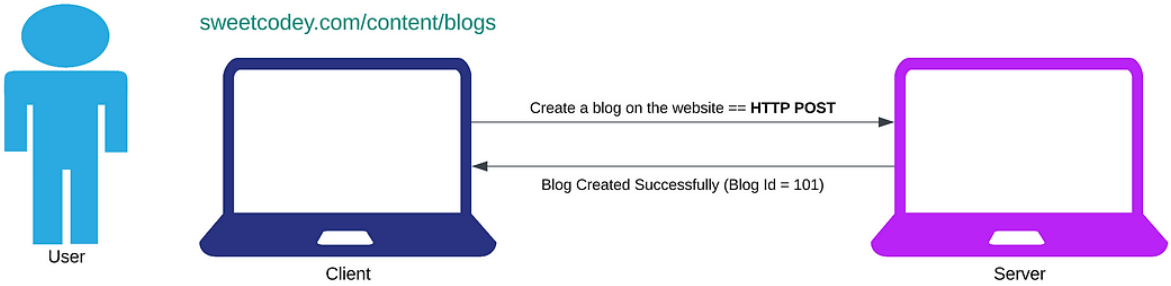


REST API

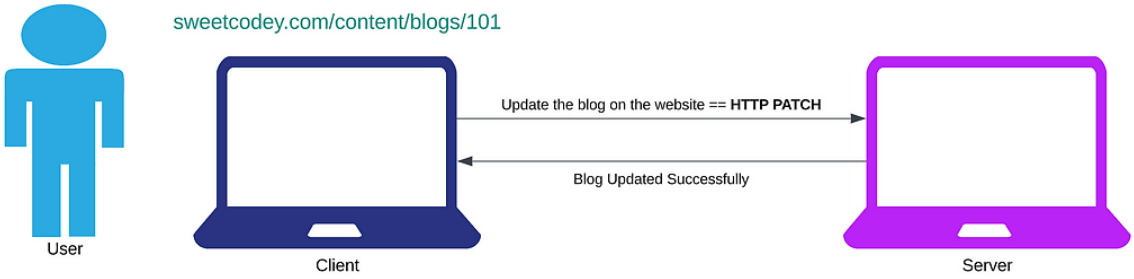
HTTP GET



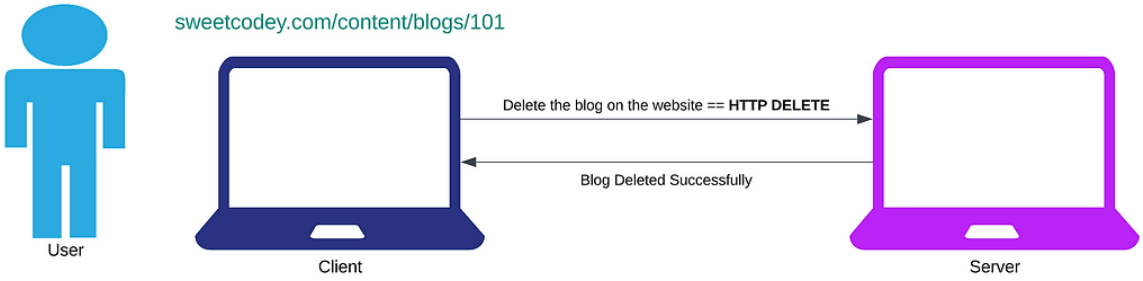
HTTP POST



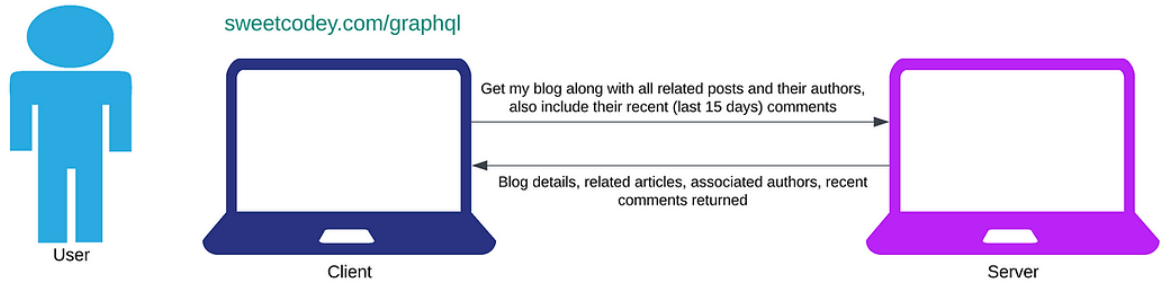
HTTP PATCH



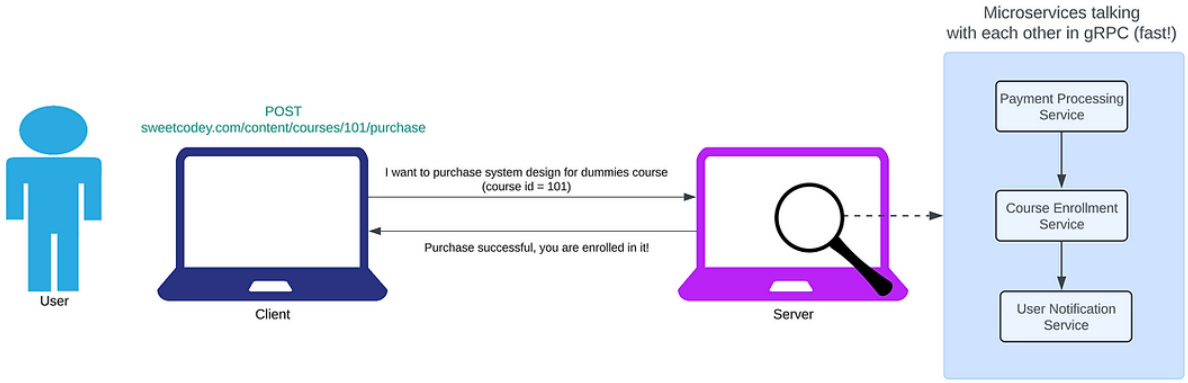
HTTP DELETE



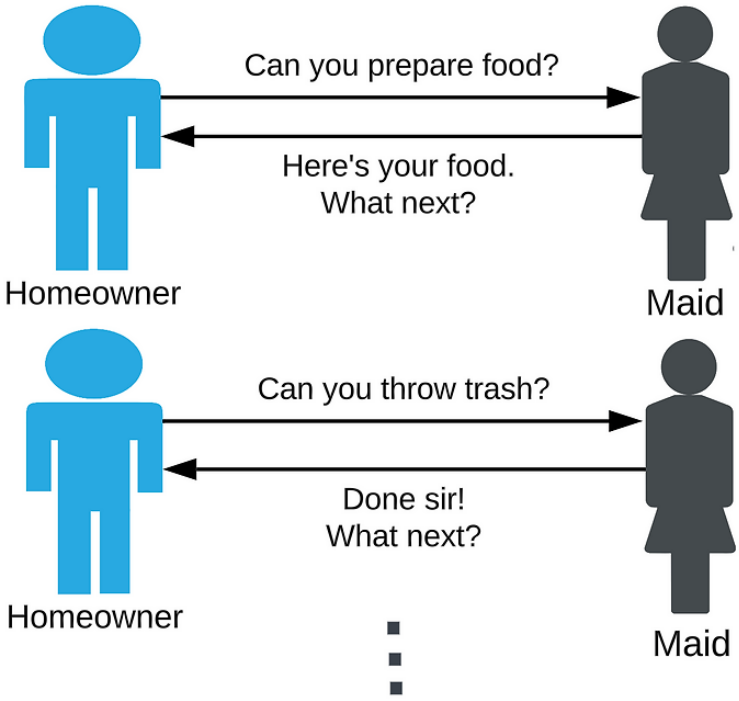
GRAPHQL

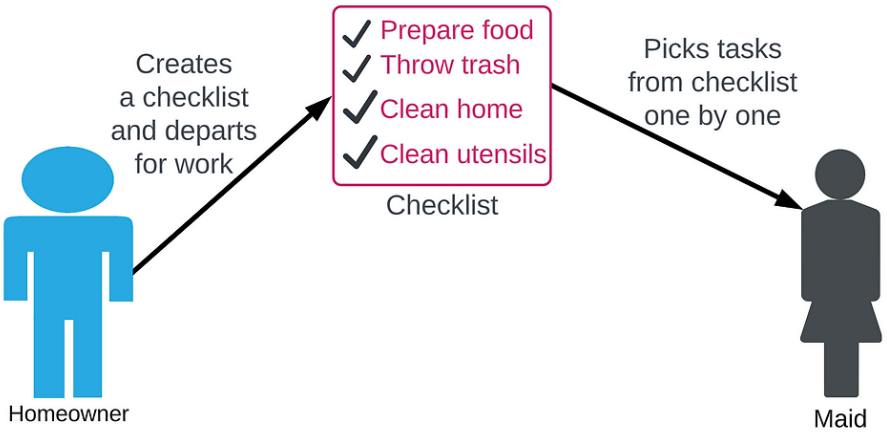


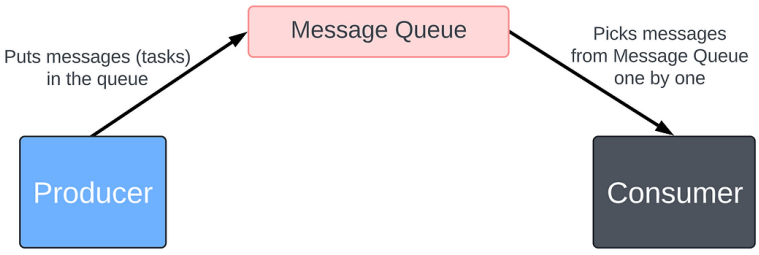
GRPC



Message Queue







Relational Database (SQL)

User Table

1	Bob	Male	28	Single
2	Mark	Male	34	Married
3	Alice	Female	50	Married

User data has a defined structure

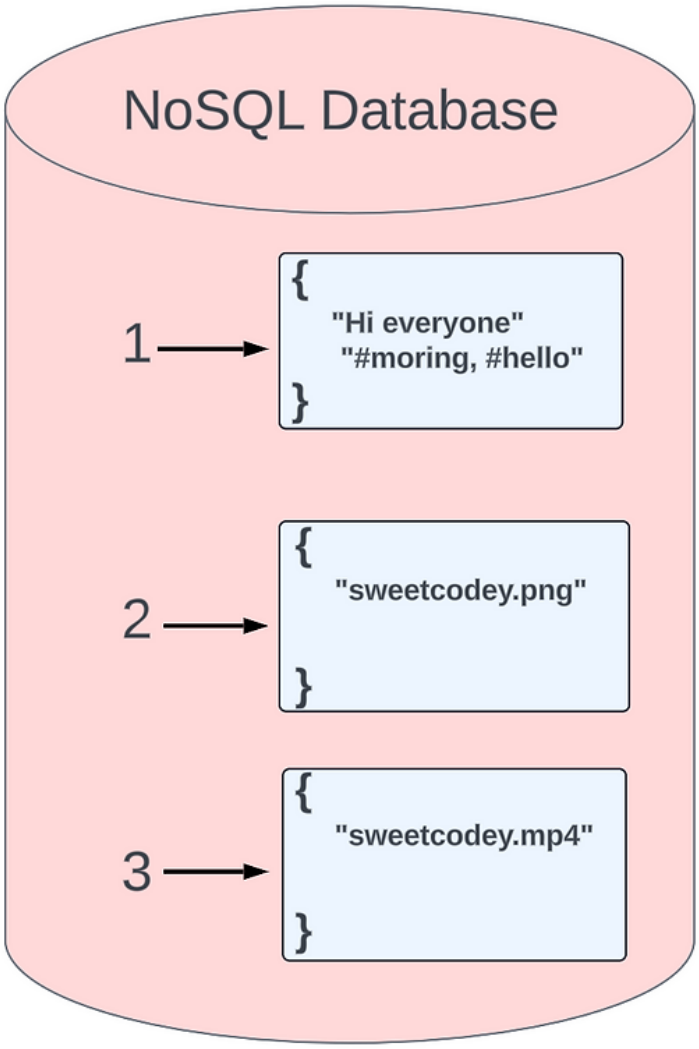
Non-Relational Database (NOSQL)

Social Media Post Table

1	"Hi everyone"			#morning, #hello
2		 sweetcodey.png		
3			 sweetcodey.mp4	

Lot of Empty Entries == Table Space Wasted!

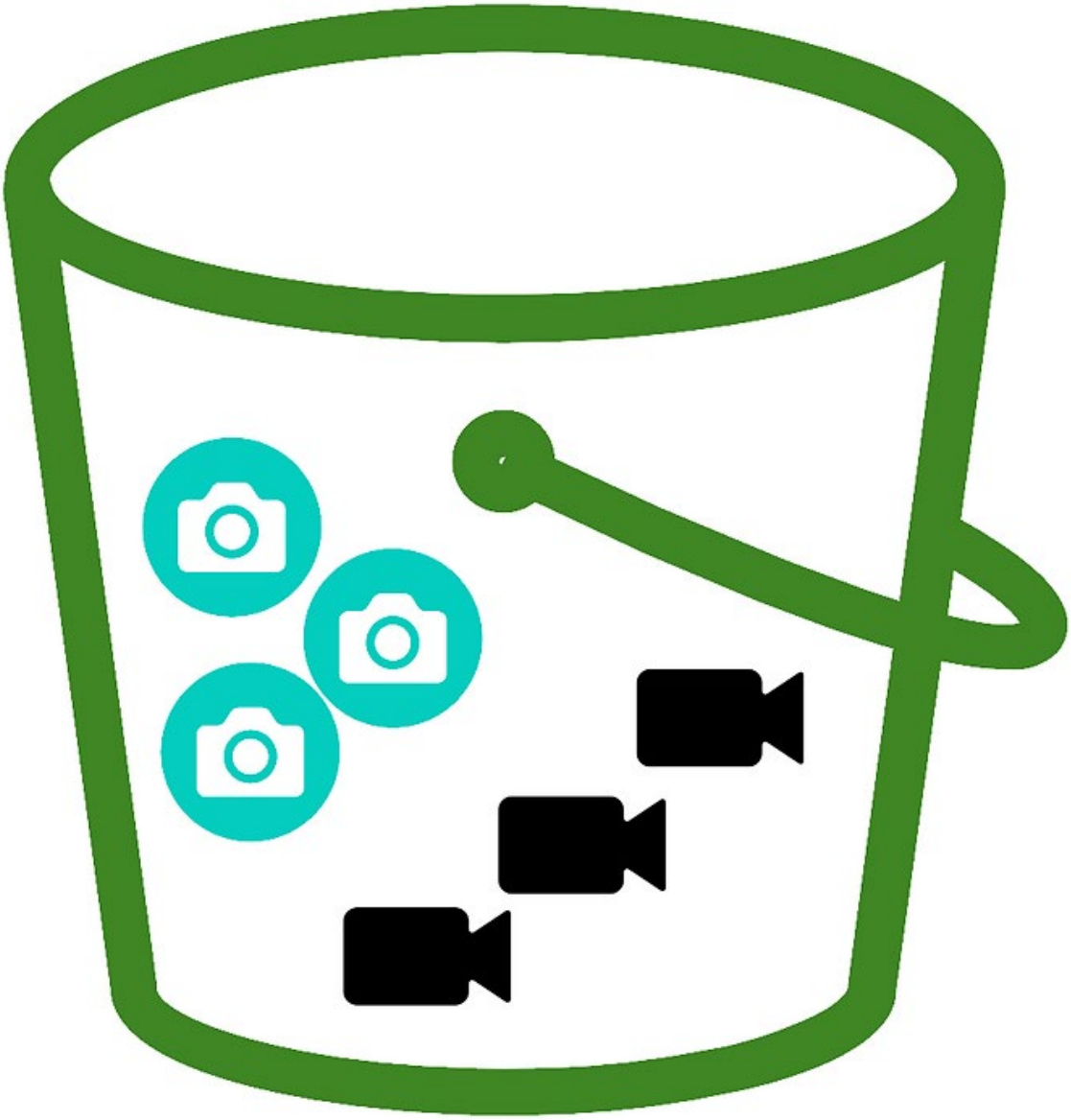
Social Media Post data doesn't have a defined structure



Each post (being unstructured) is stored in a JSON document.

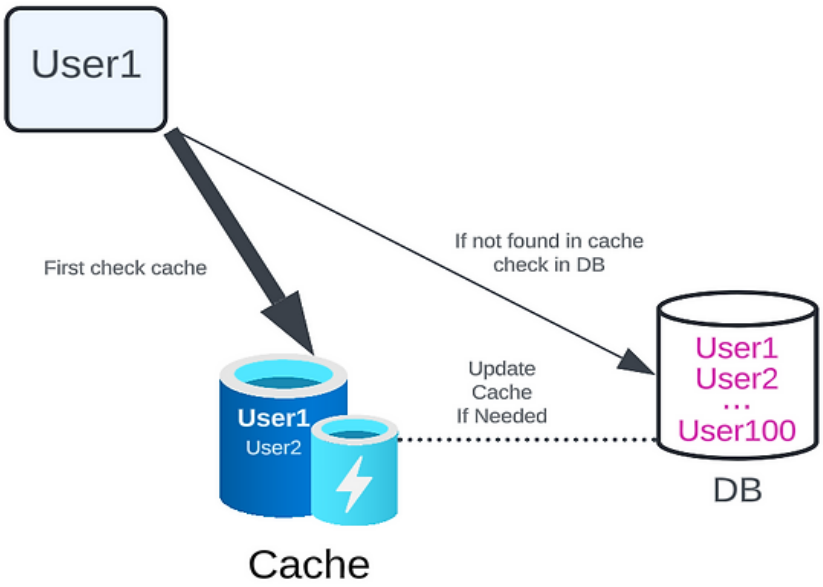
Can be accessed by Key (PostId)

Object Storage



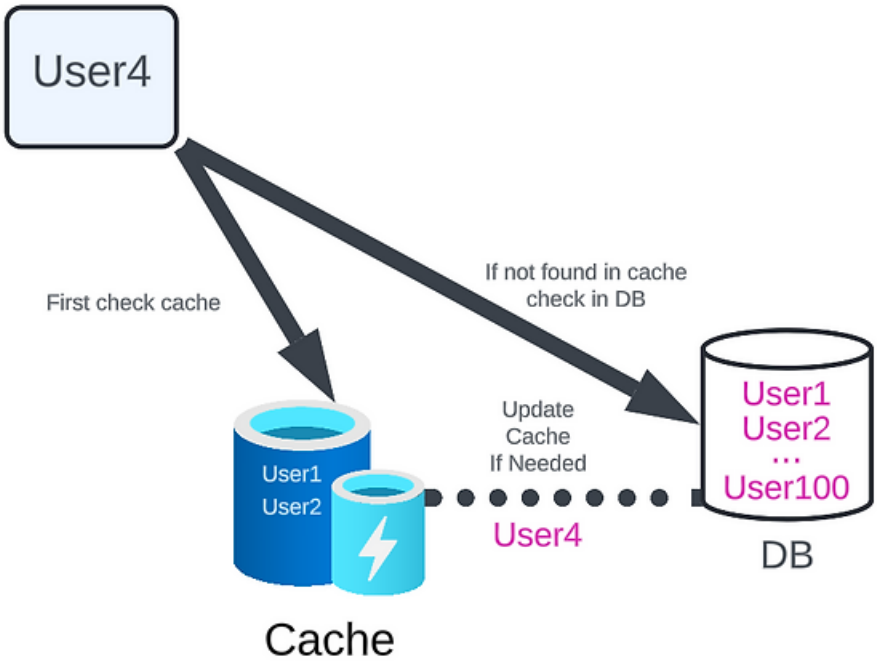
Cache

Cache Hit



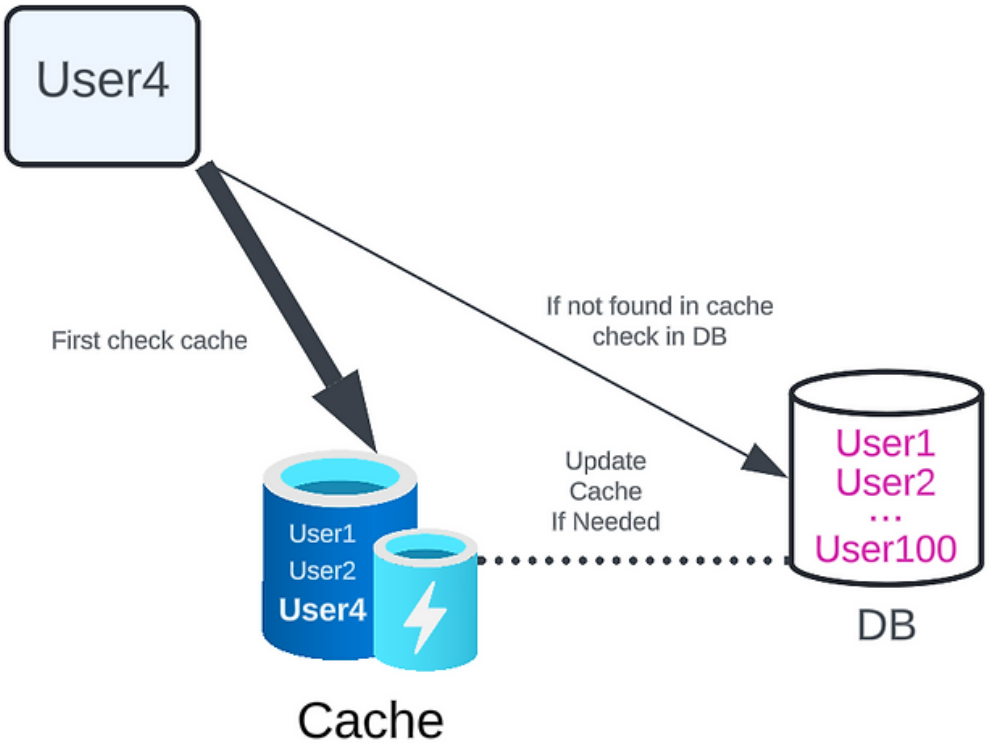
User1 already in Cache

Cache Miss



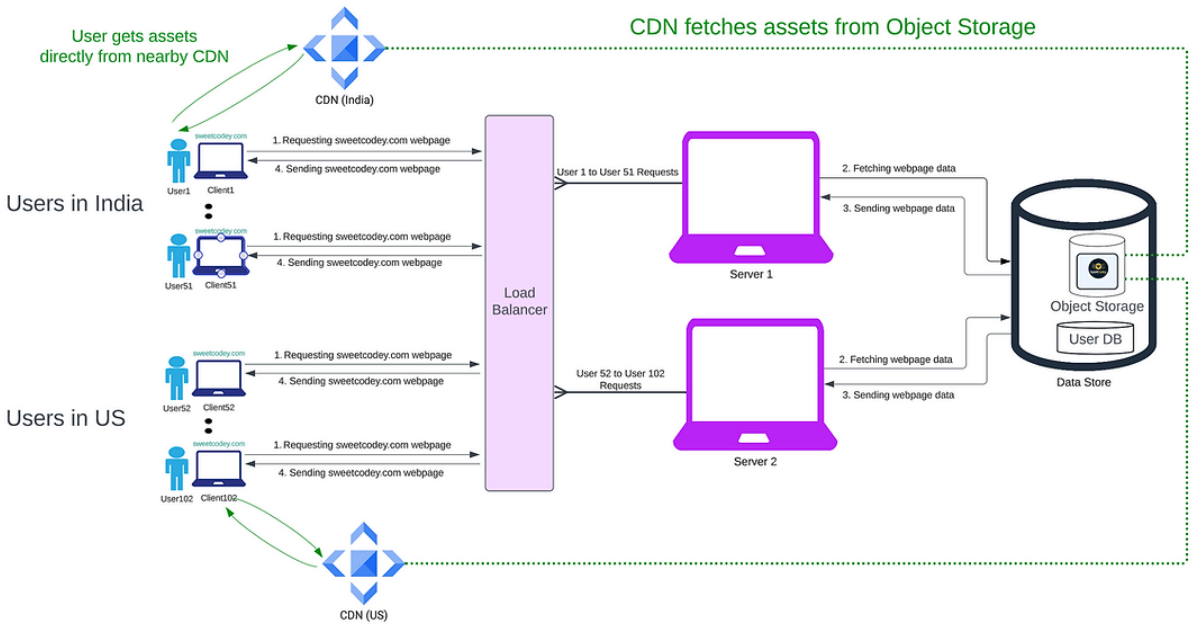
User4 NOT in Cache
Cache Updated with User4

Cache Hit

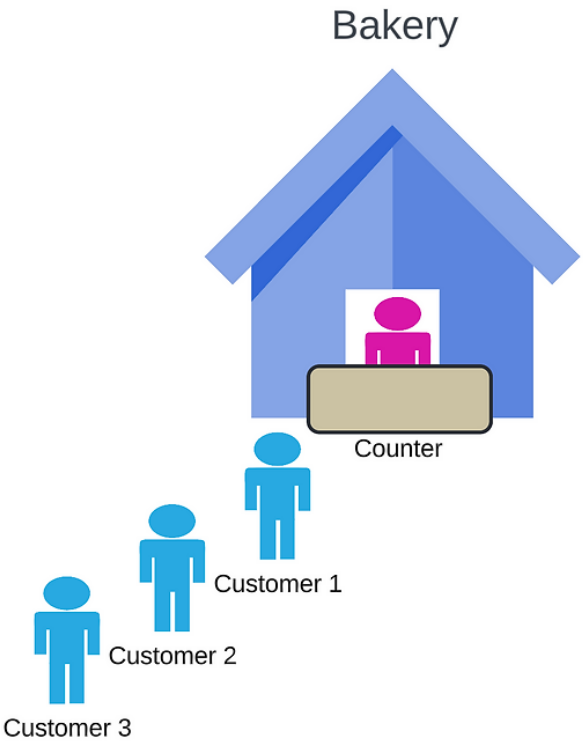


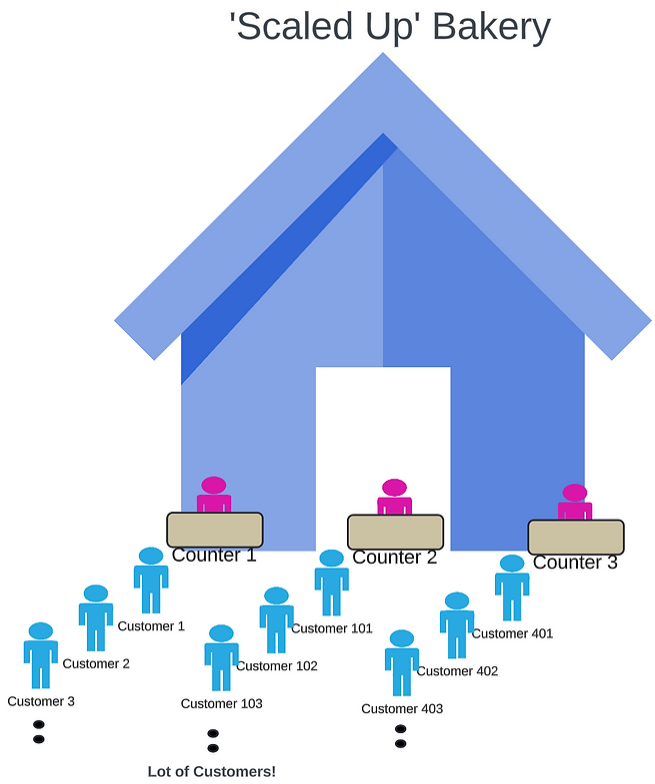
User4 is in Cache

CDN

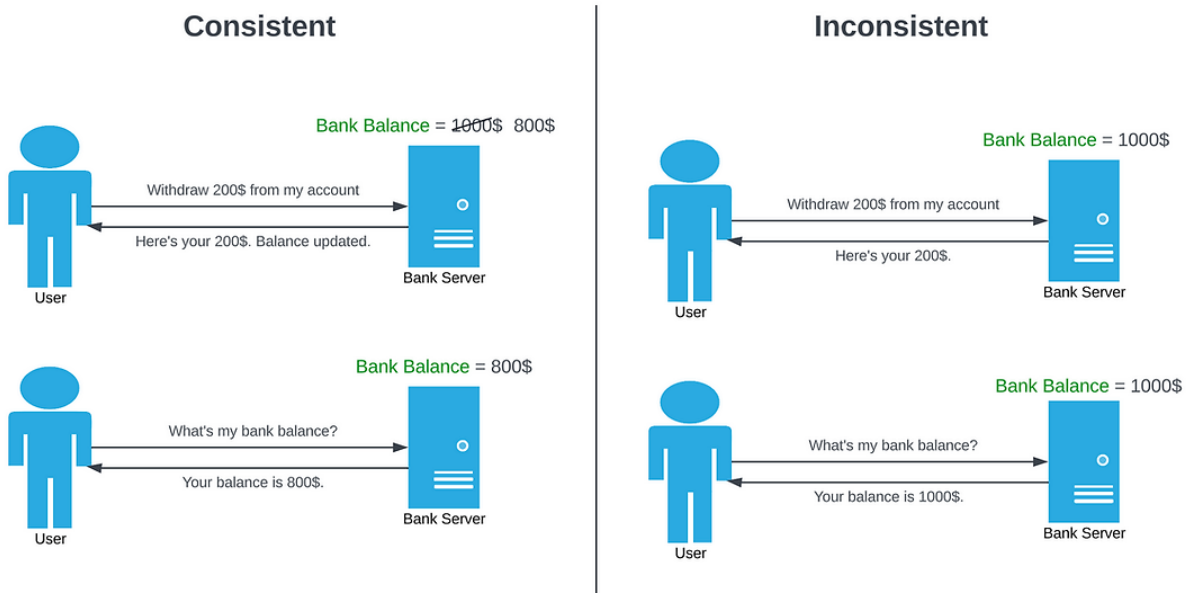


Scalability

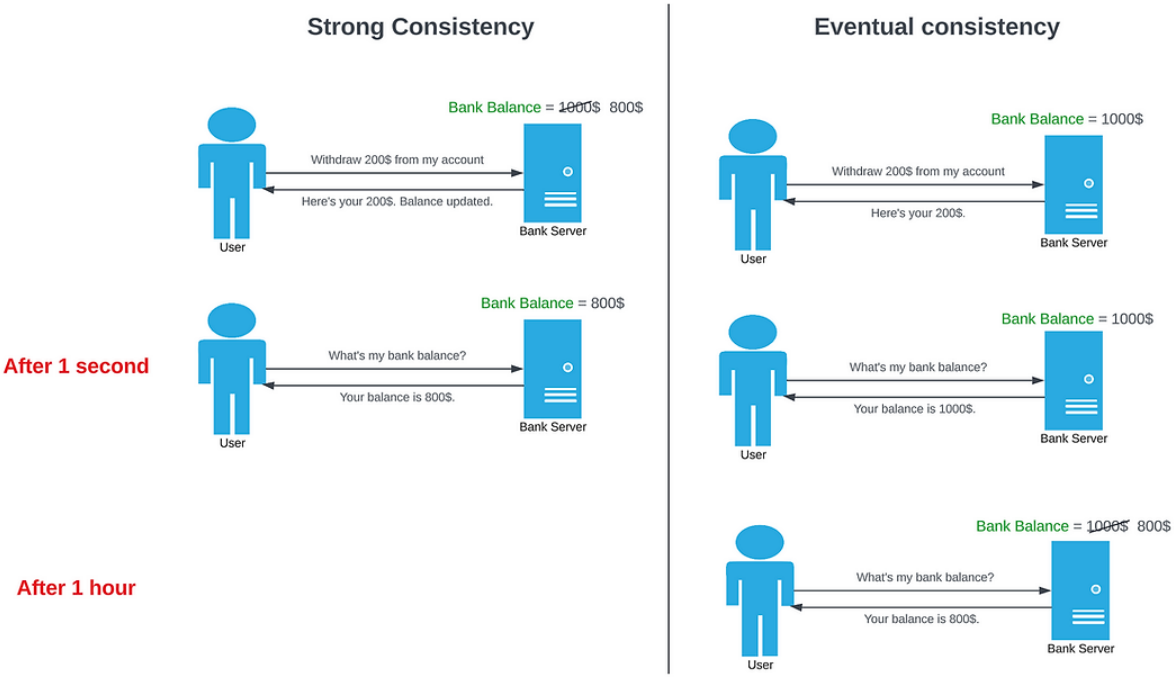




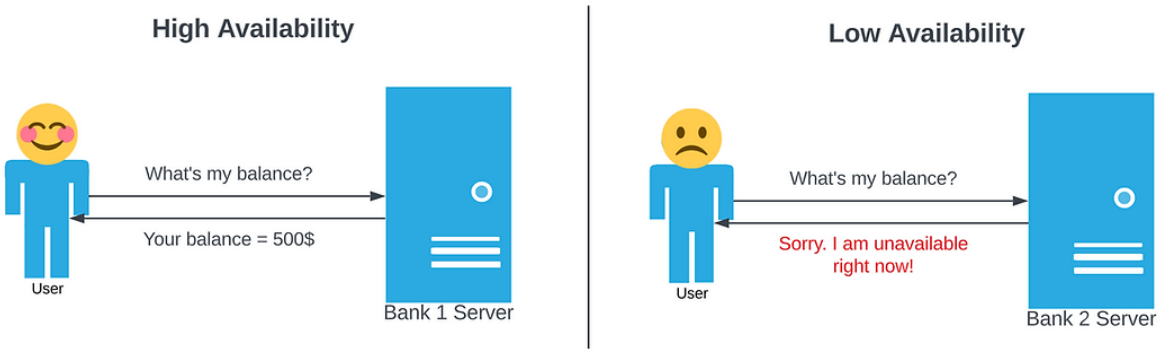
Consistency



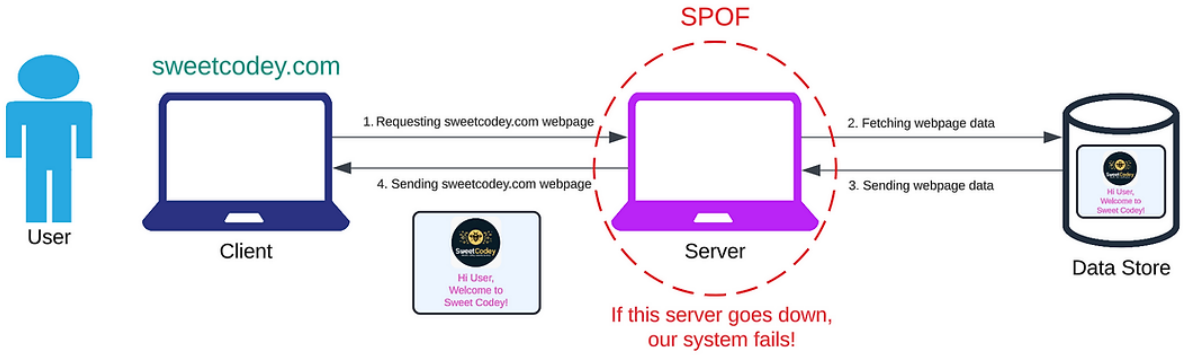
Strong/Eventual Consistency



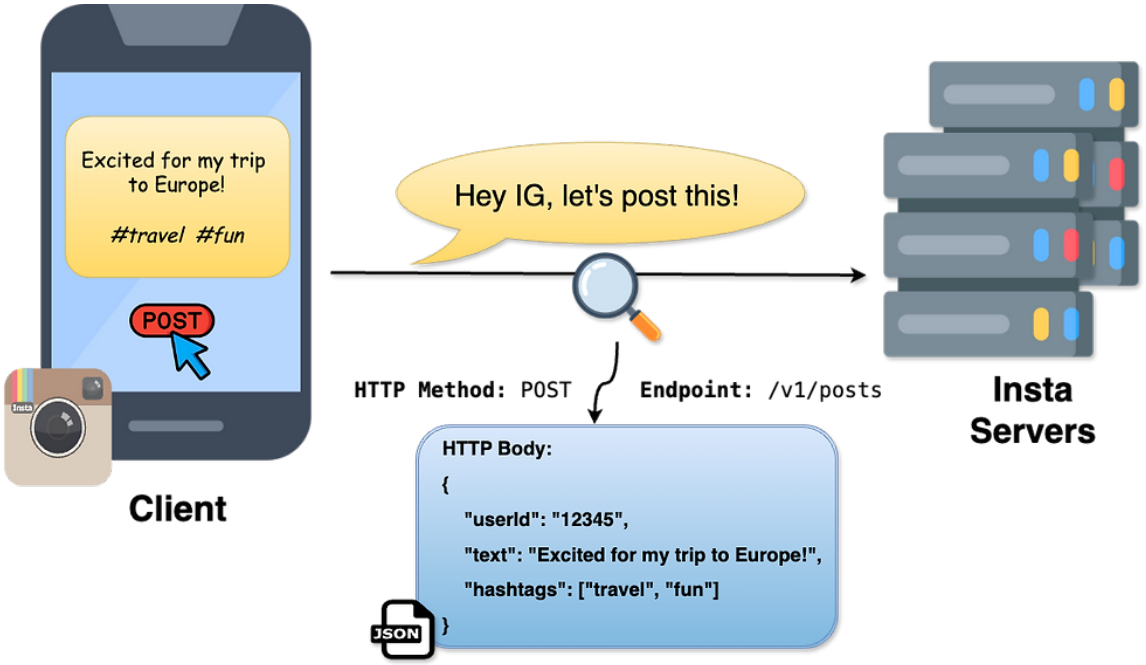
Availability



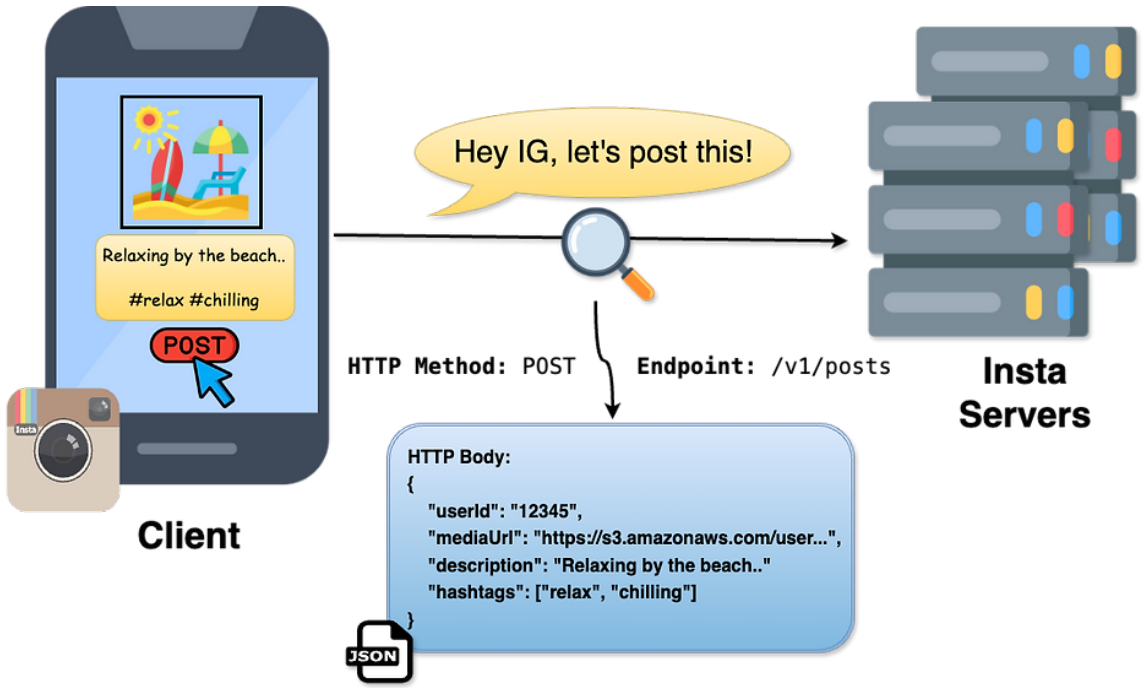
Single Point of Failure



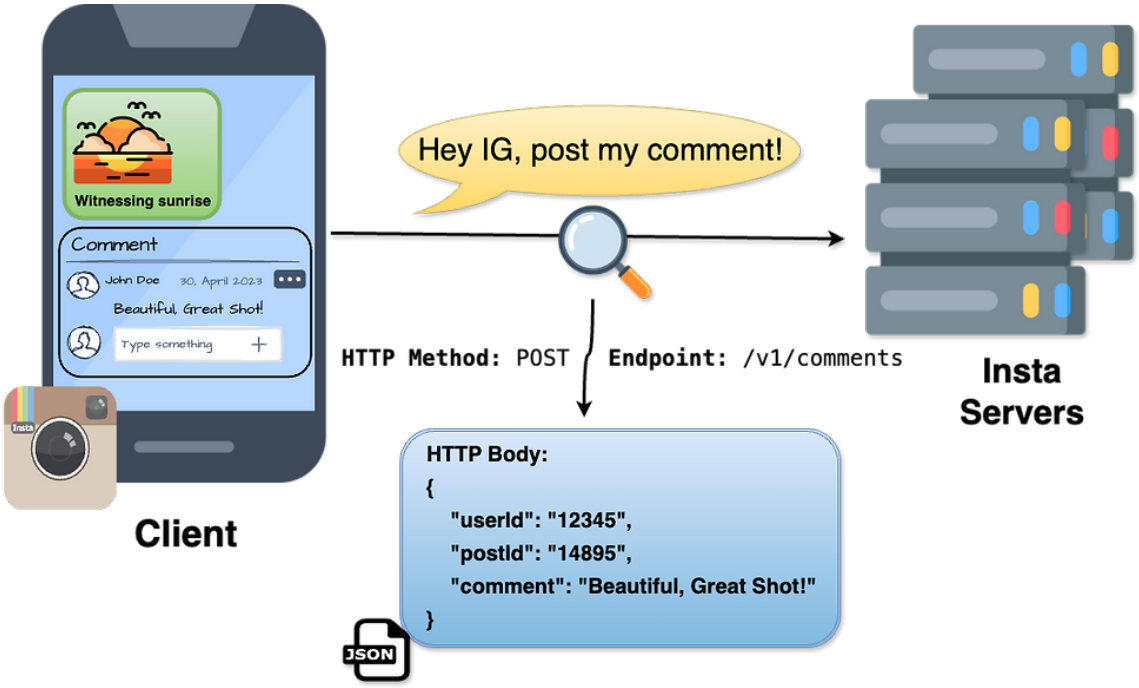
API Design-Create a Text Post



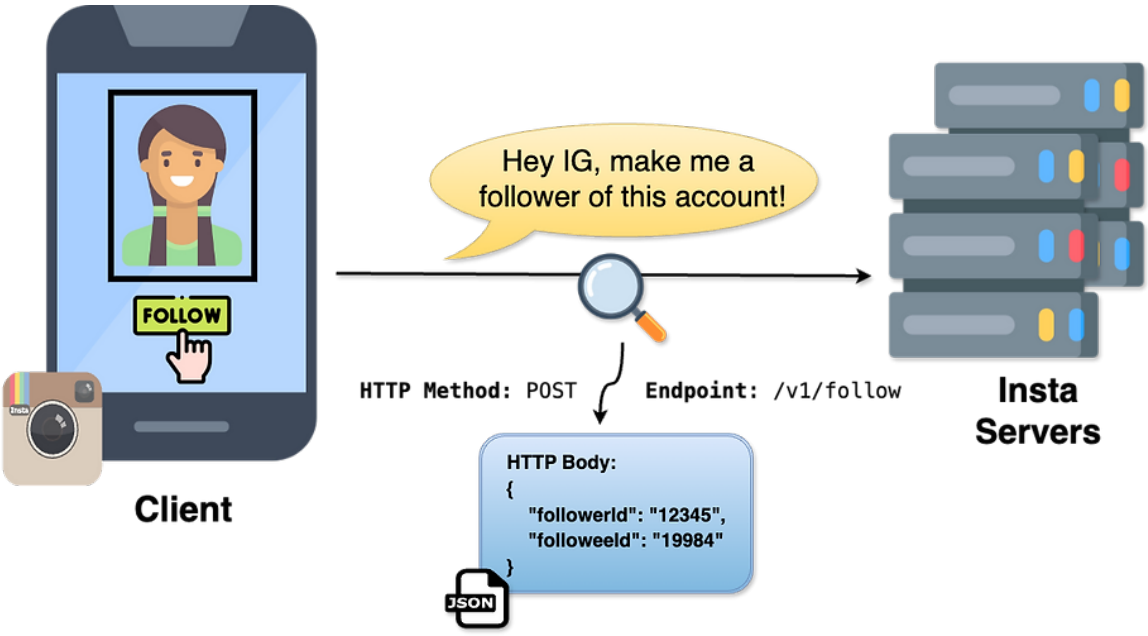
API Design-Create Image/ Video Post



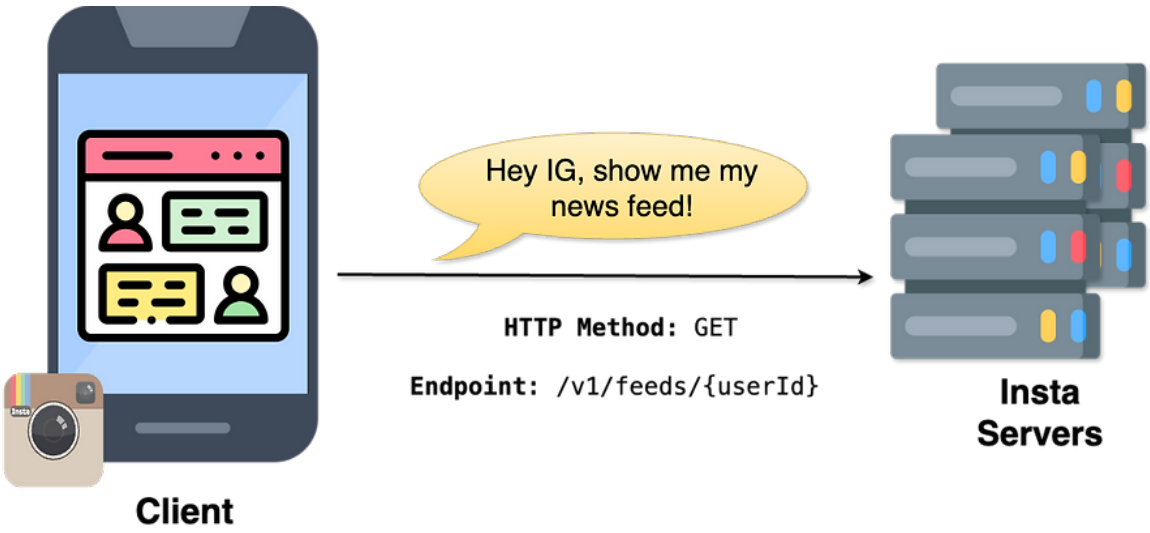
API Design-Like /Comment a Post



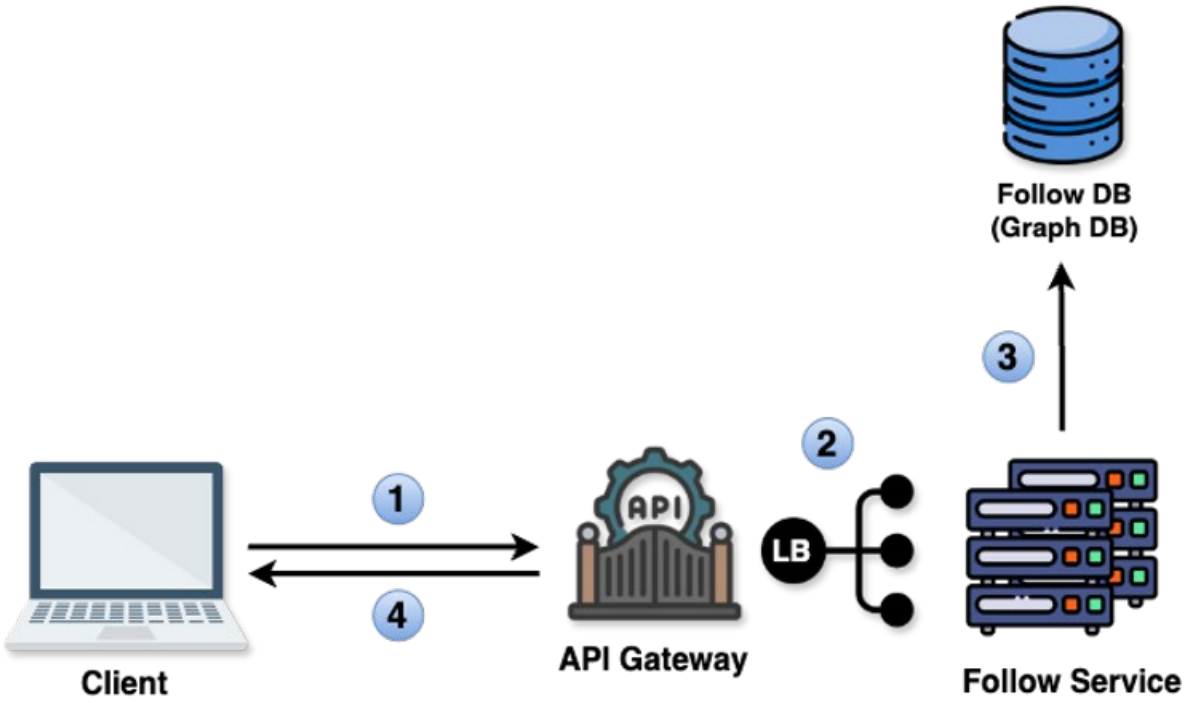
API Design-Follow / Unfollow another User



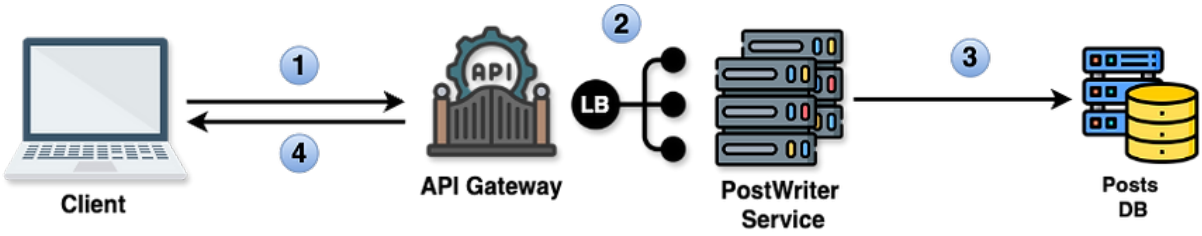
API Design-Read the Newsfeed (aka Timeline)

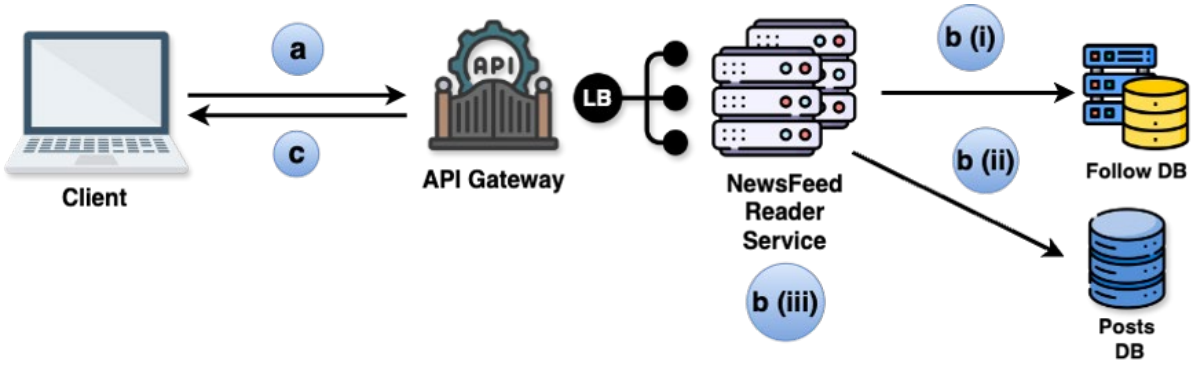


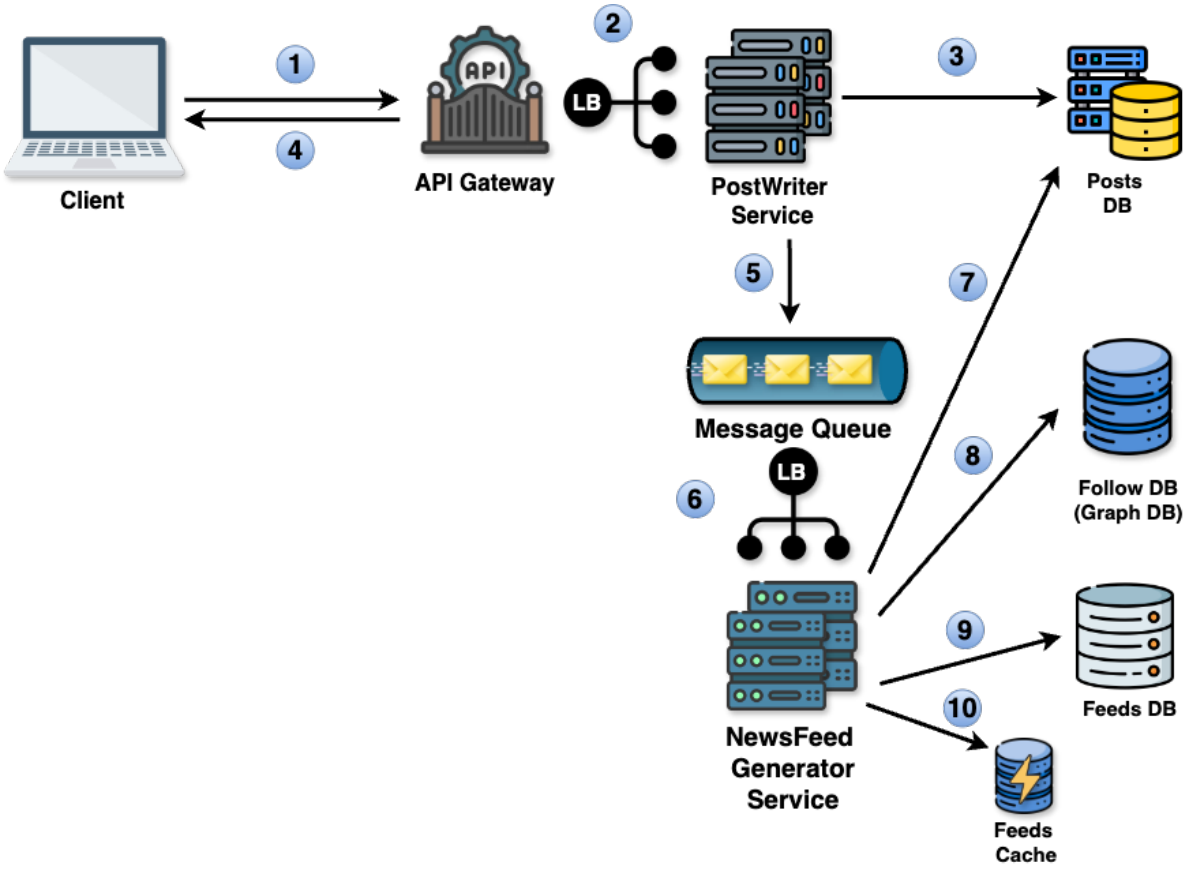
High level Design- Follow / Unfollow another User



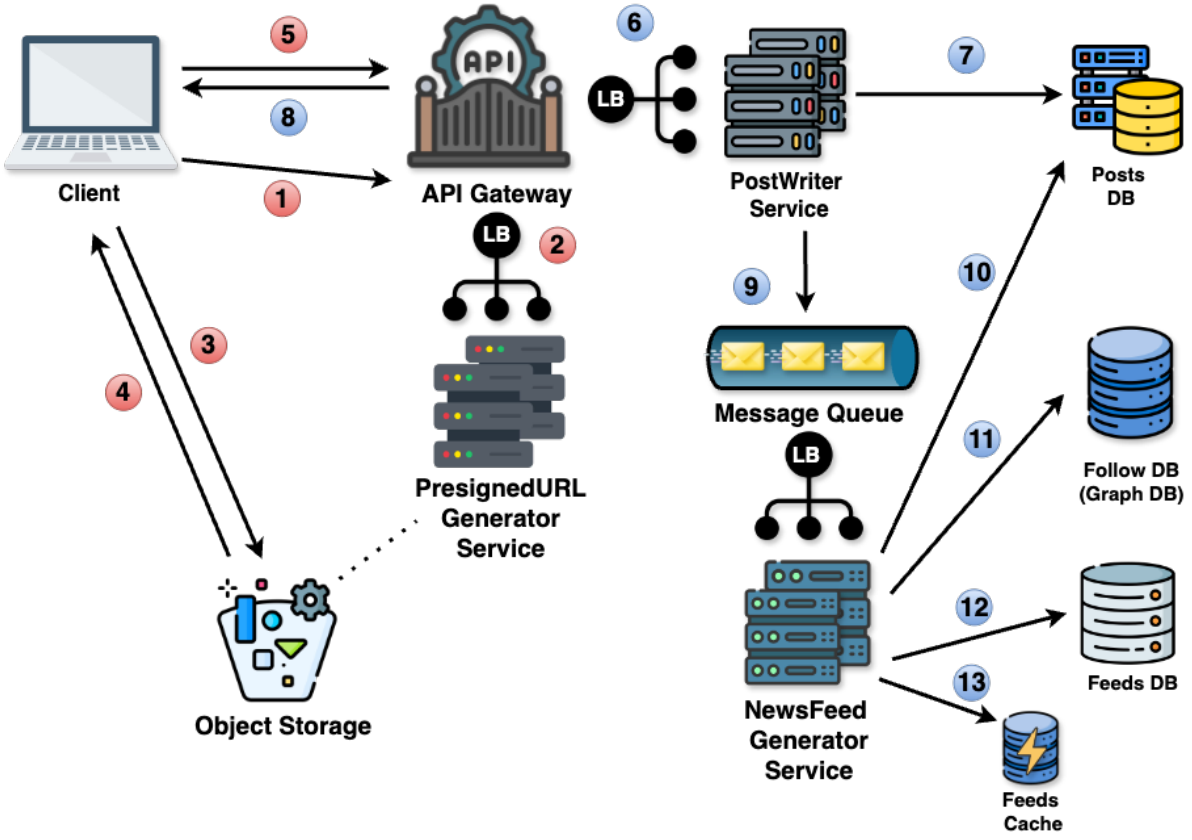
High level Design- Create Text Post



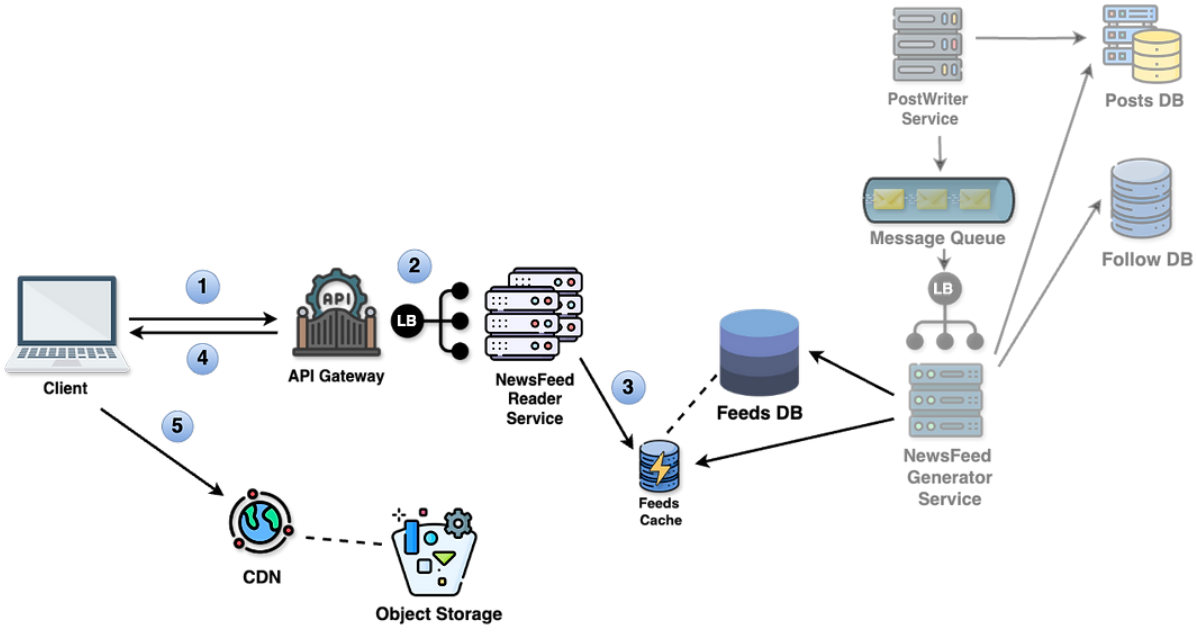




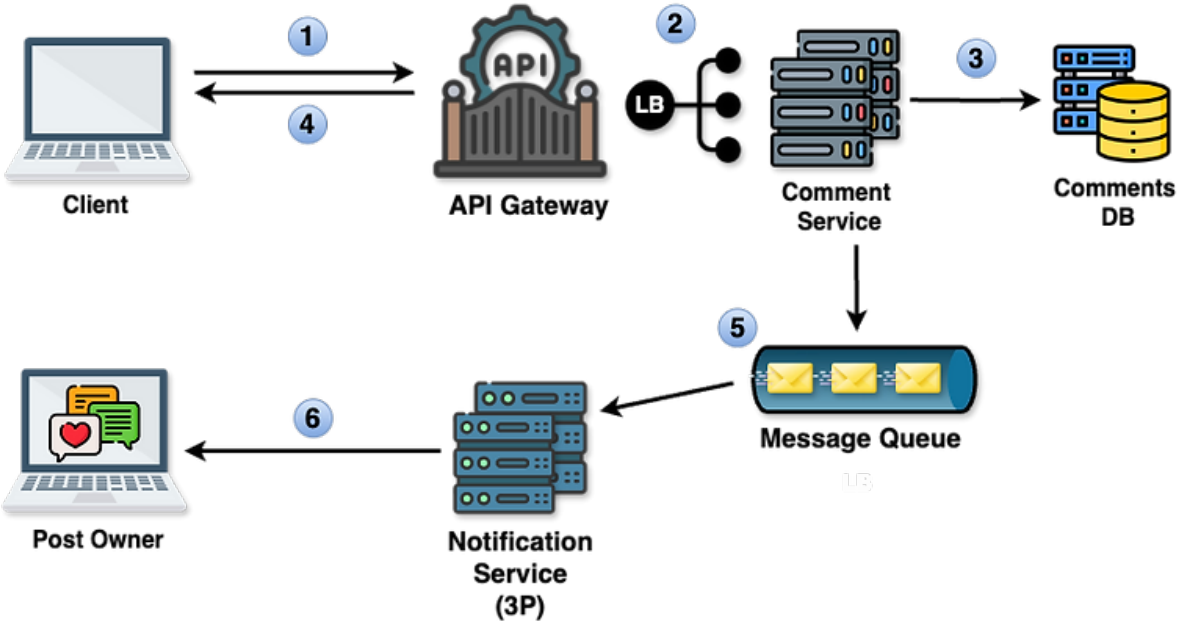
High level Design- Create Image/Video Post



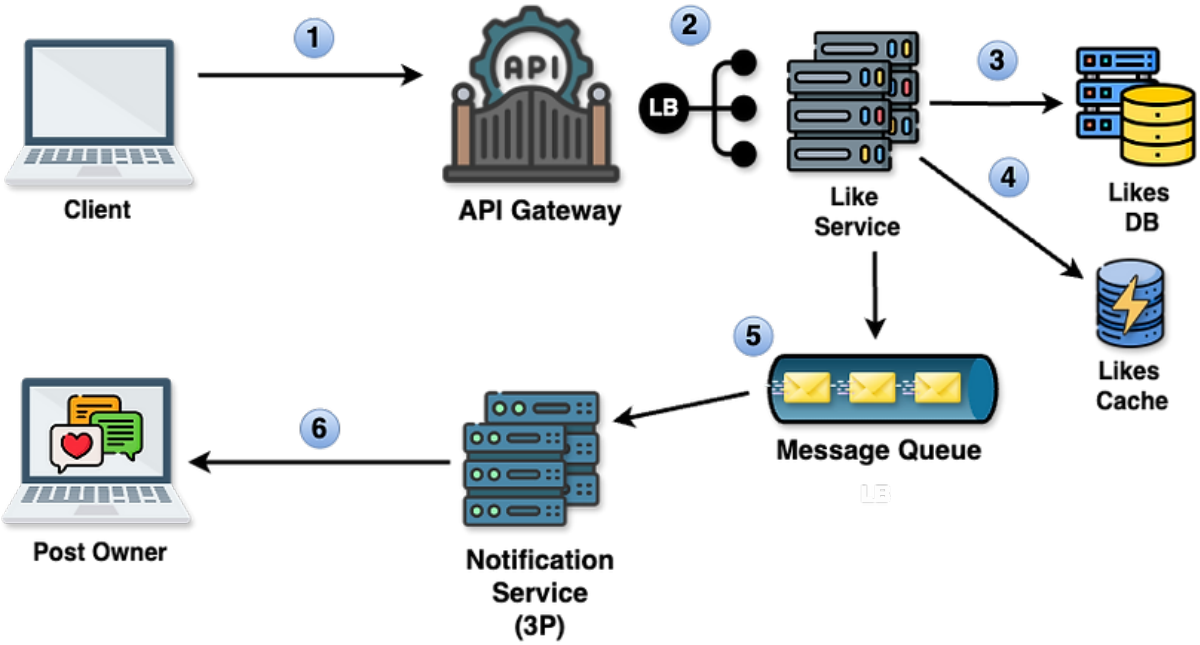
High level Design- Read the Newsfeed (aka timeline)

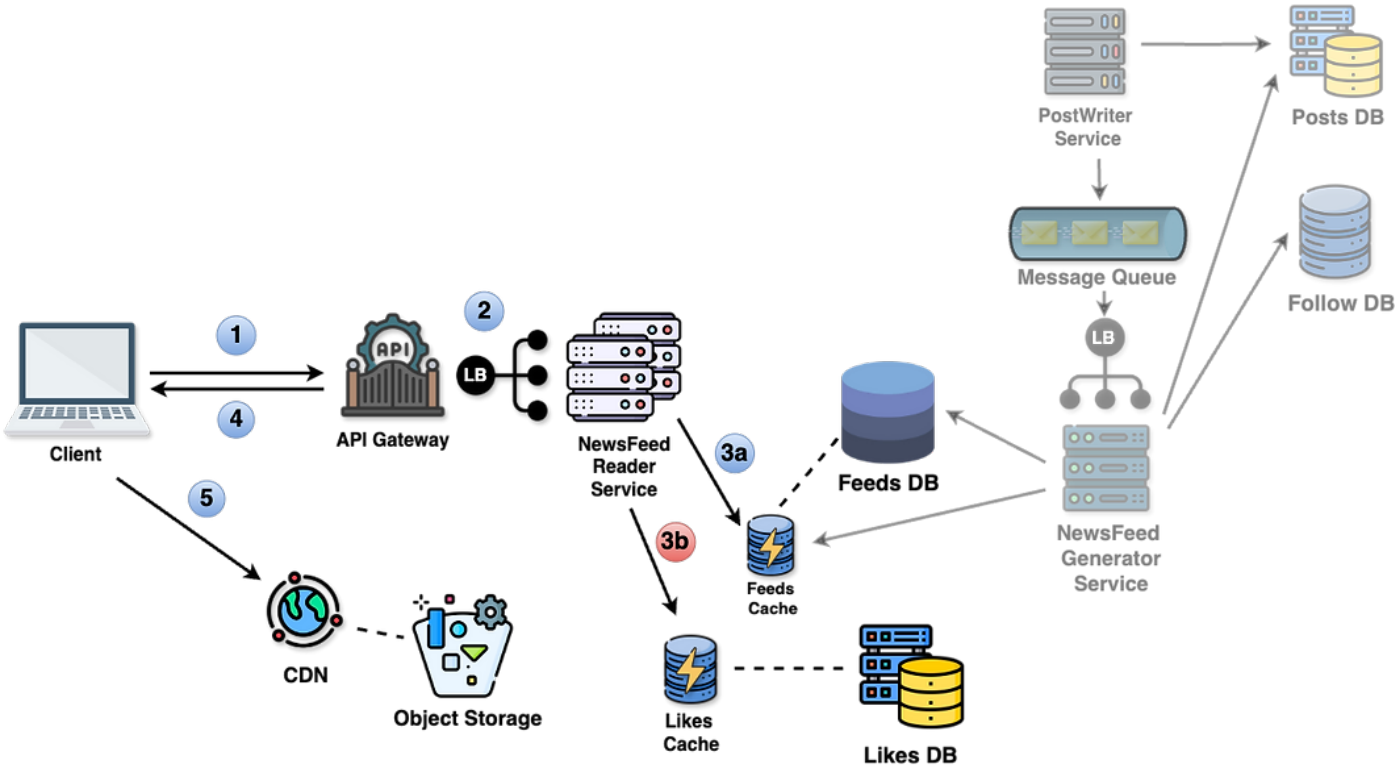


High level Design-Comment on a Post

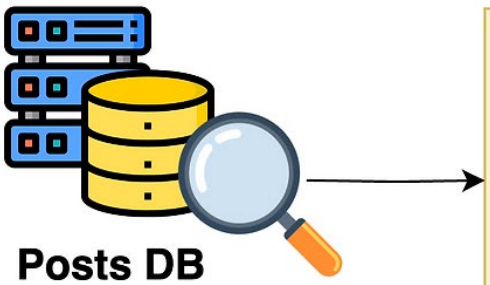


High level Design-Like on a Post



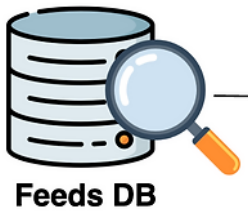


Posts Schema



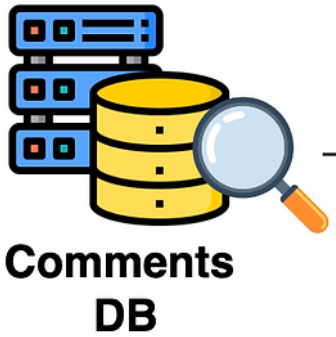

```
{
  "postId": "unique_post_id",
  "userId": "user_id",
  "text": "post text",
  "mediaUrls": ["url1", "url2"],
  "timestamp": "1720748150"
}
```

Feeds Schema



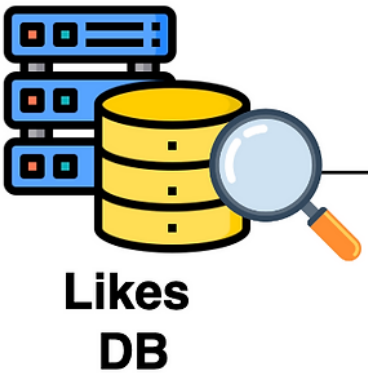

```
{
    "userId": "unique_user_id",
    "feedItems": [
        {
            "postId": "13452",
            "userId": "84231",
            "text": "sample post text 1",
            "timestamp": "1720748111"
        },
        {
            "postId": "86475",
            "userId": "12394",
            "text": "sample post text 2",
            "mediaUrls": ["url1", "url2", ...]
            "timestamp": "1720713459"
        }
        // More posts...
    ]
}
```

Comments Schema



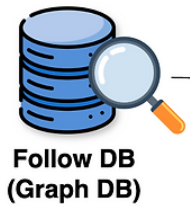

```
{
  "commentId": "unique_comment_id",
  "userId": "user_id",
  "postId": "post_id",
  "comment": "comment text",
  "timestamp": "1720748150"
}
```

Likes Schema

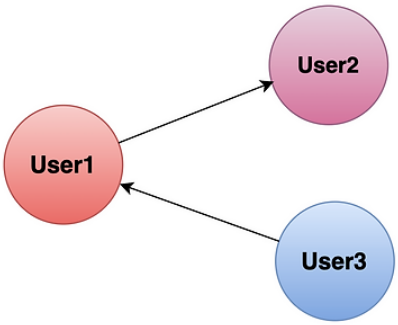



```
{
  "likeId": "unique_like_id",
  "userId": "user_id",
  "postId": "post_id",
  "timestamp": "1720741345"
}
```

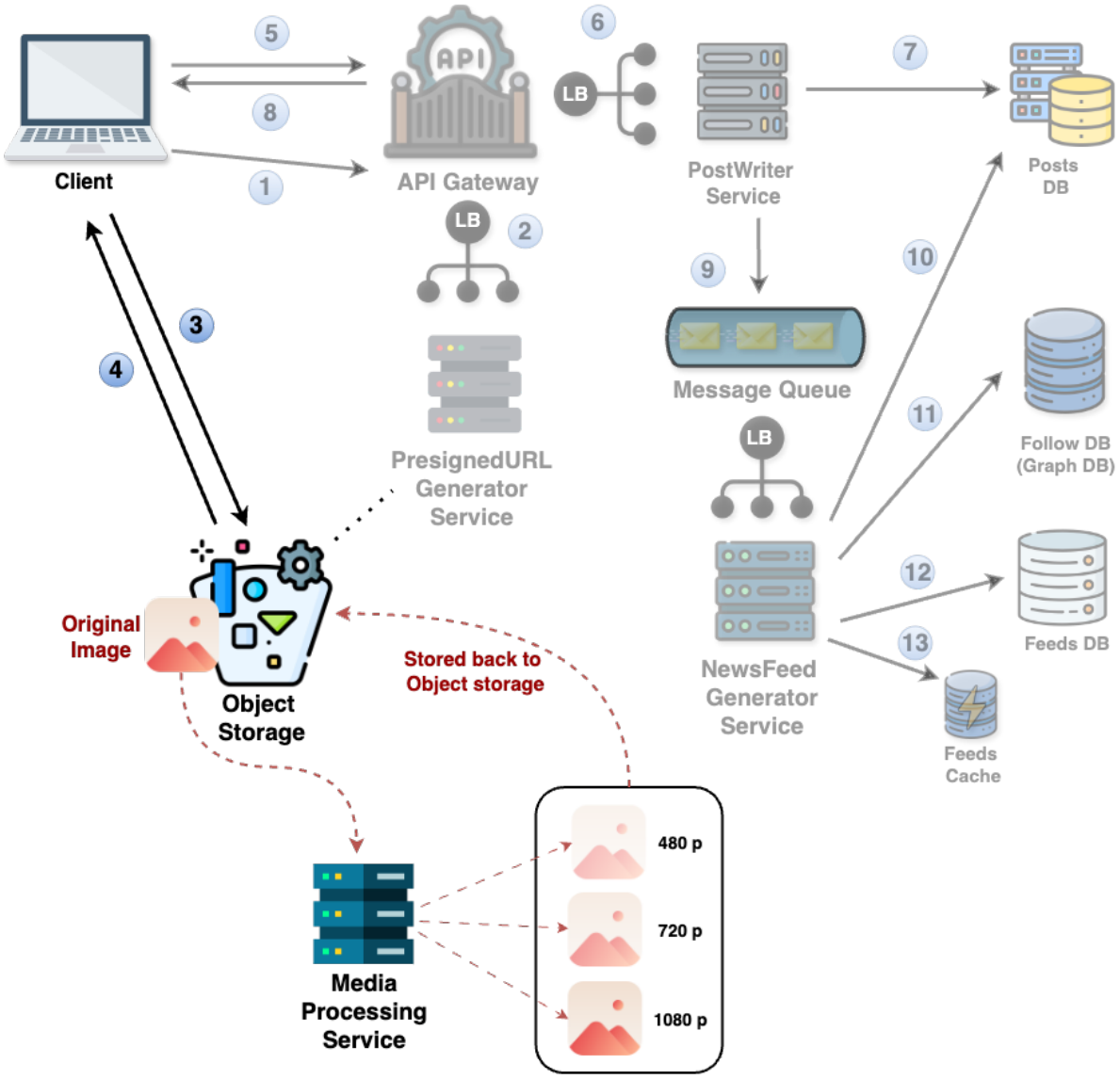
Follow Schema



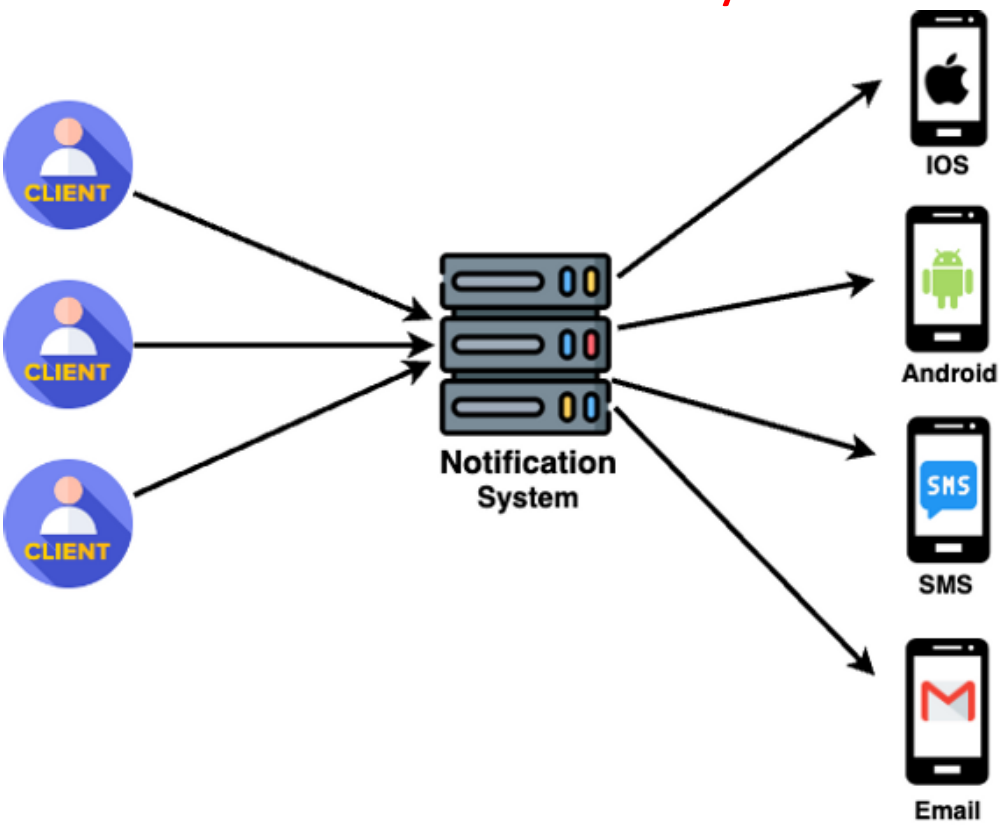

```
{
   "userId": "<user1_id>",
   "followers": [
       {
         "followerId": "<user2_id>",
         "timestamp": "1720748150"
       },
       // More followers...
    ],
    "followees": [
       {
         "followeeId": "<user3_id>",
         "timestamp": "1720748150"
       },
       // More followees...
    ]
}
```



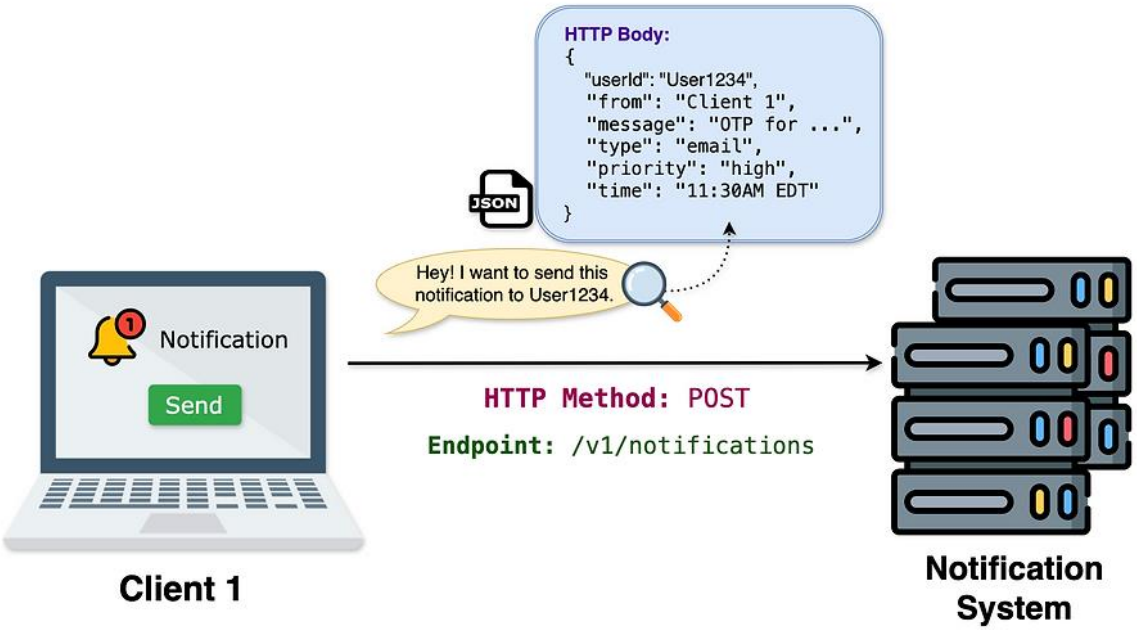
Media Processing



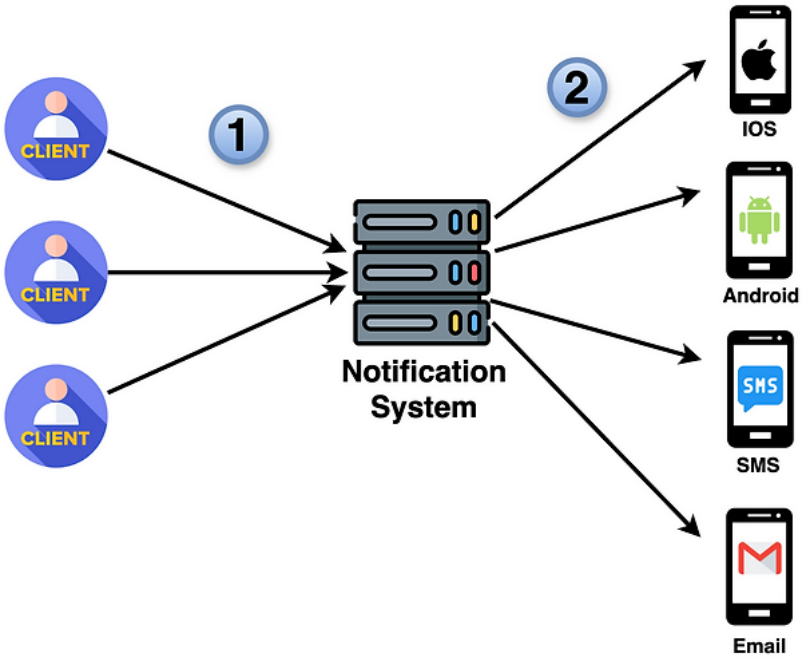
What is Notification System?



API Design- Sending a Notification

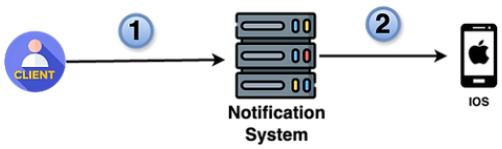


High Level Design- Basic Flow

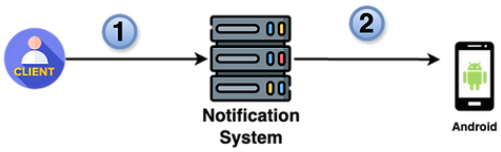


High Level Design- Problems with Basic High Level Design

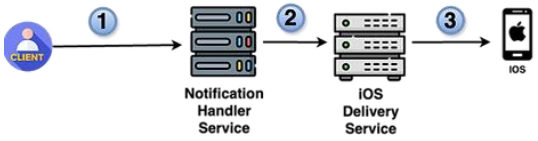
1. Validate the notification request.
2. Process the notification.
3. Create a notification record in the system.
4. Connect to Apple Servers.
5. Send the notification through the Apple APIs.
6. Receive delivery confirmation.



1. Validate the notification request.
2. Process the notification.
3. Create a notification record in the system.
4. Connect to Android Servers.
5. Send the notification through Android APIs.
6. Receive delivery confirmation.

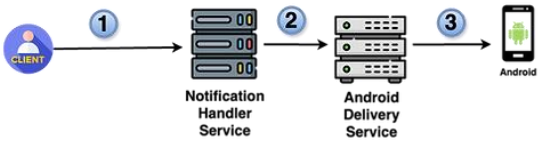


1. Validate the notification request.
2. Process the notification.
3. Create a notification record in the system.

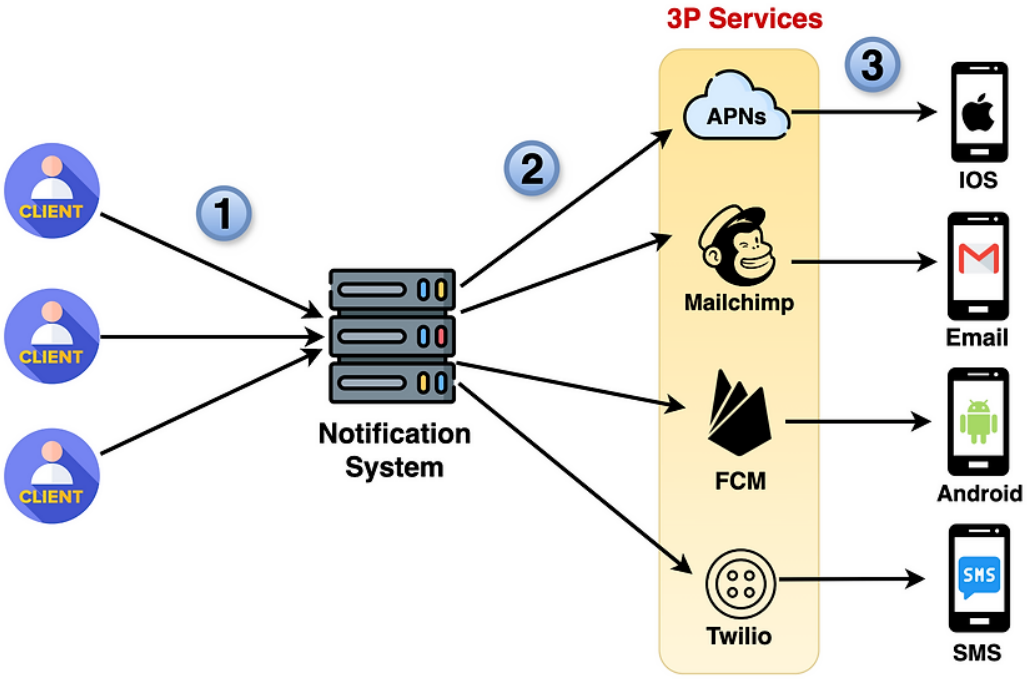


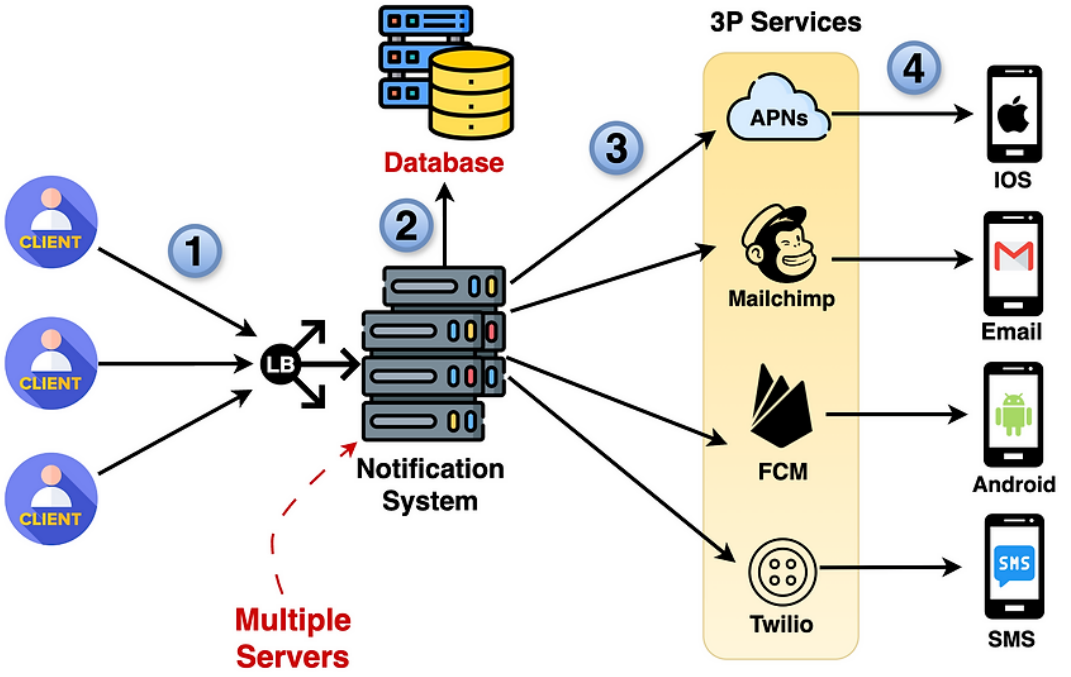
3. Connect to Apple Servers.
4. Send the notification through the Apple APIs.
5. Receive delivery confirmation.

1. Validate the notification request.
2. Process the notification.
3. Create a notification record in the system.

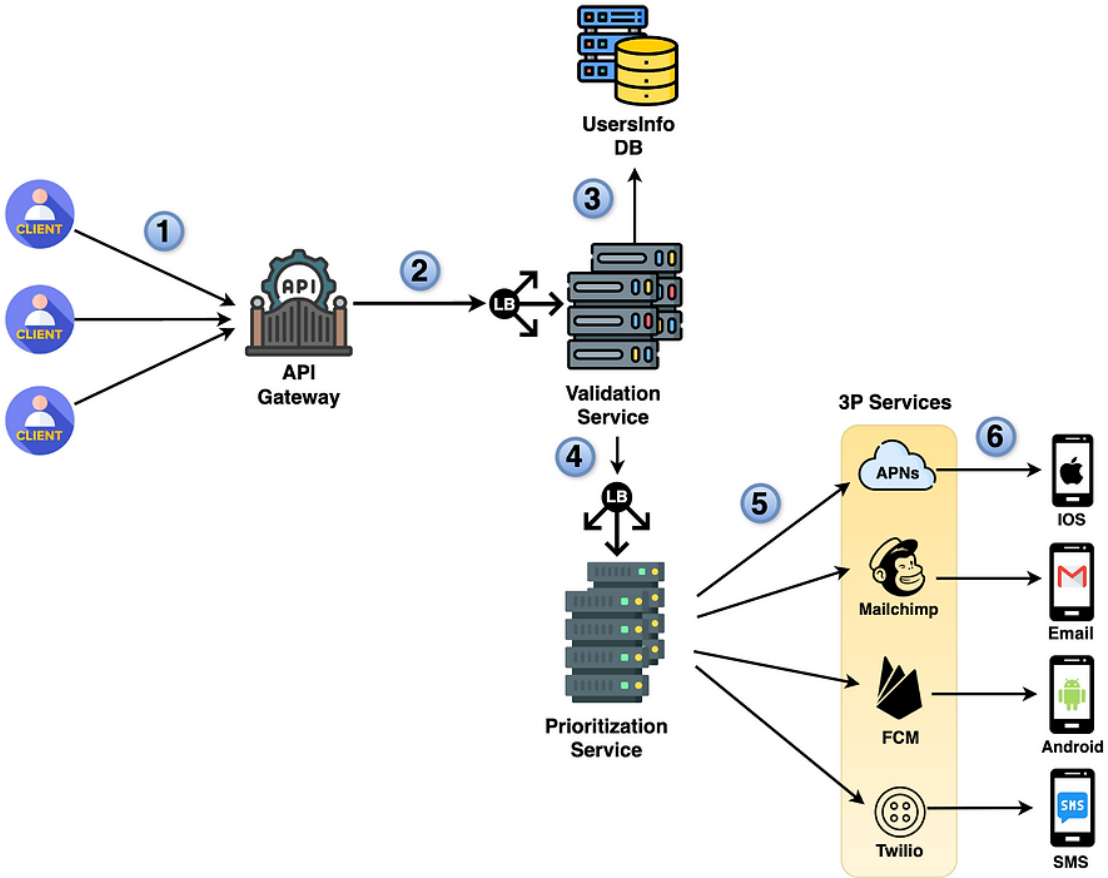


3. Connect to Android Servers.
4. Send the notification through Android API.
5. Receive delivery confirmation.

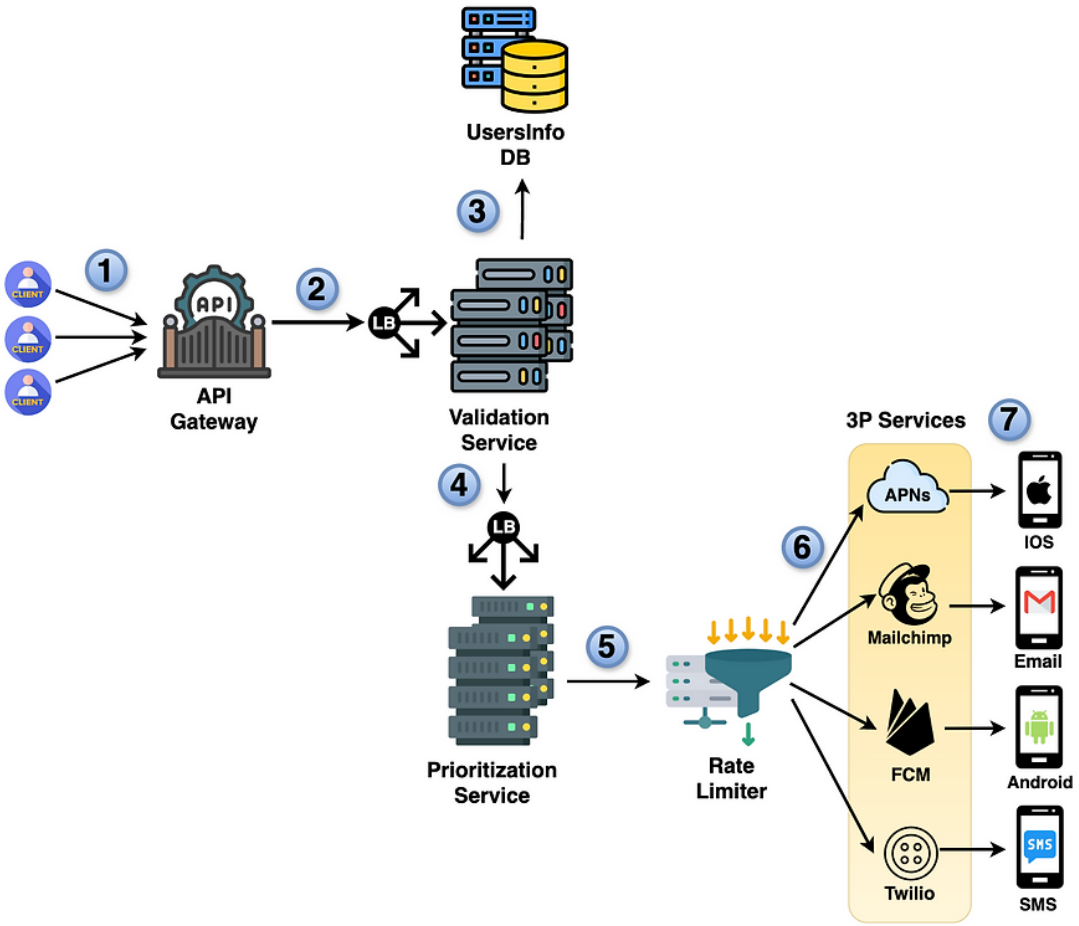




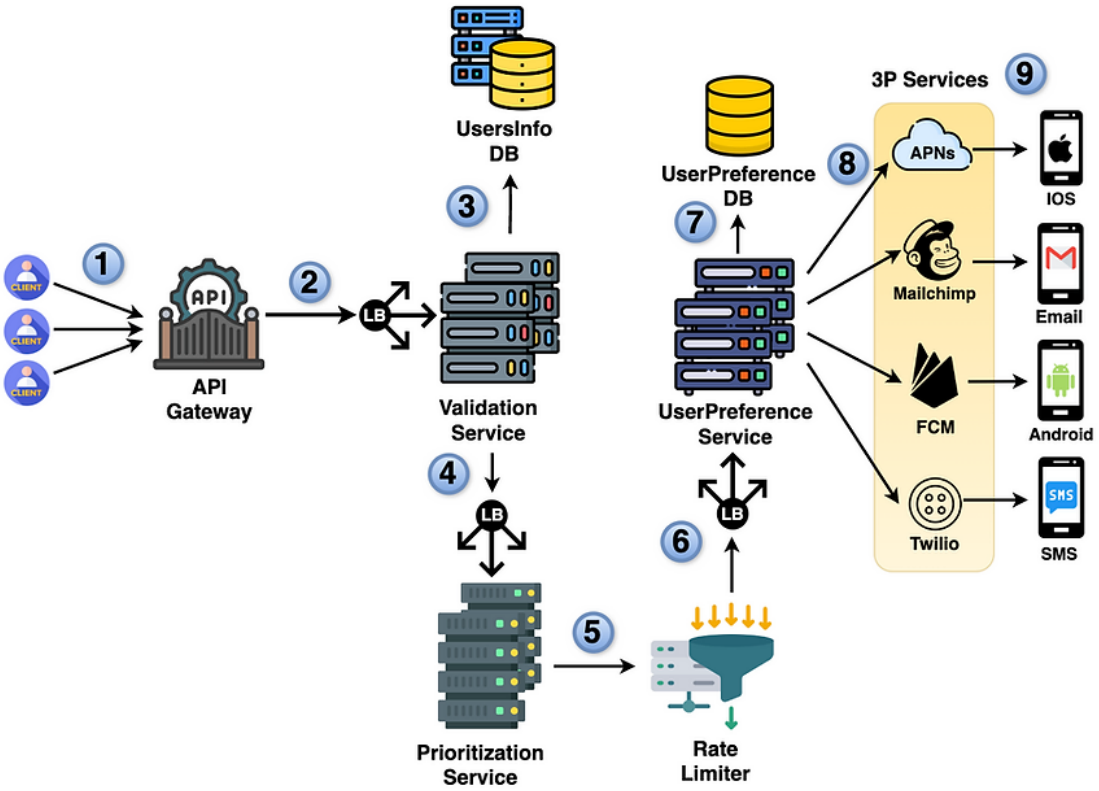
High Level Design-Validation and Prioritization



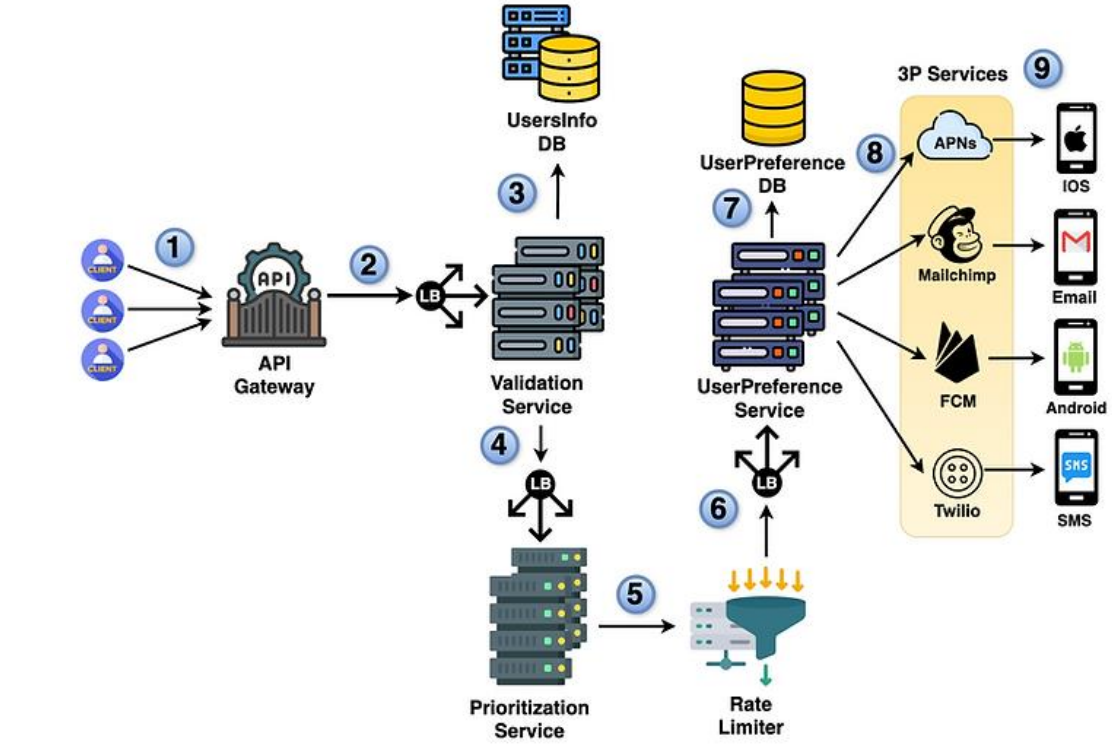
High Level Design-Rate Limiting Notifications

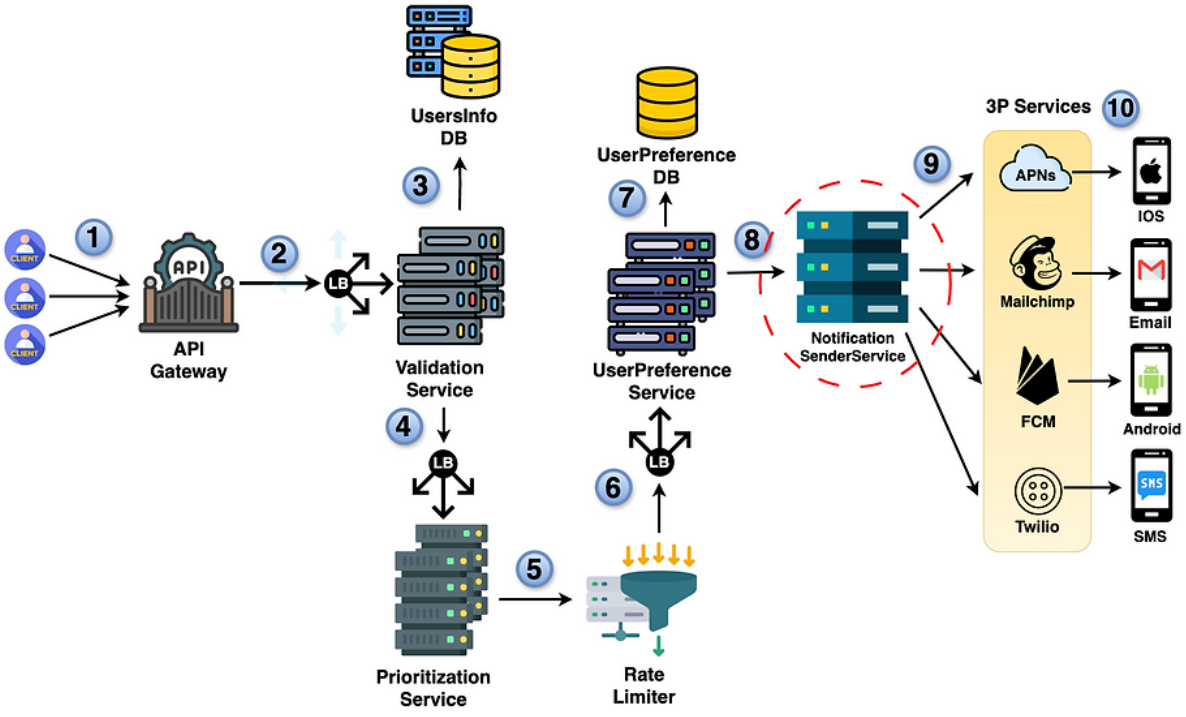


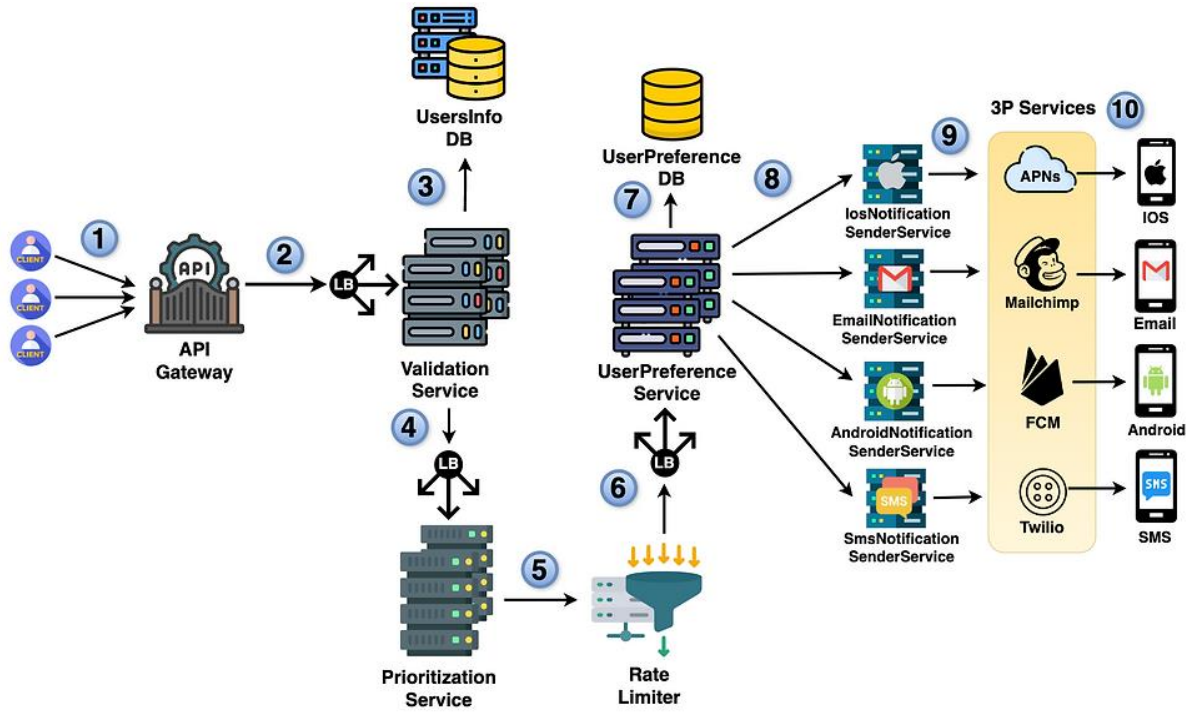
High Level Design-Adjusting to User Preferences



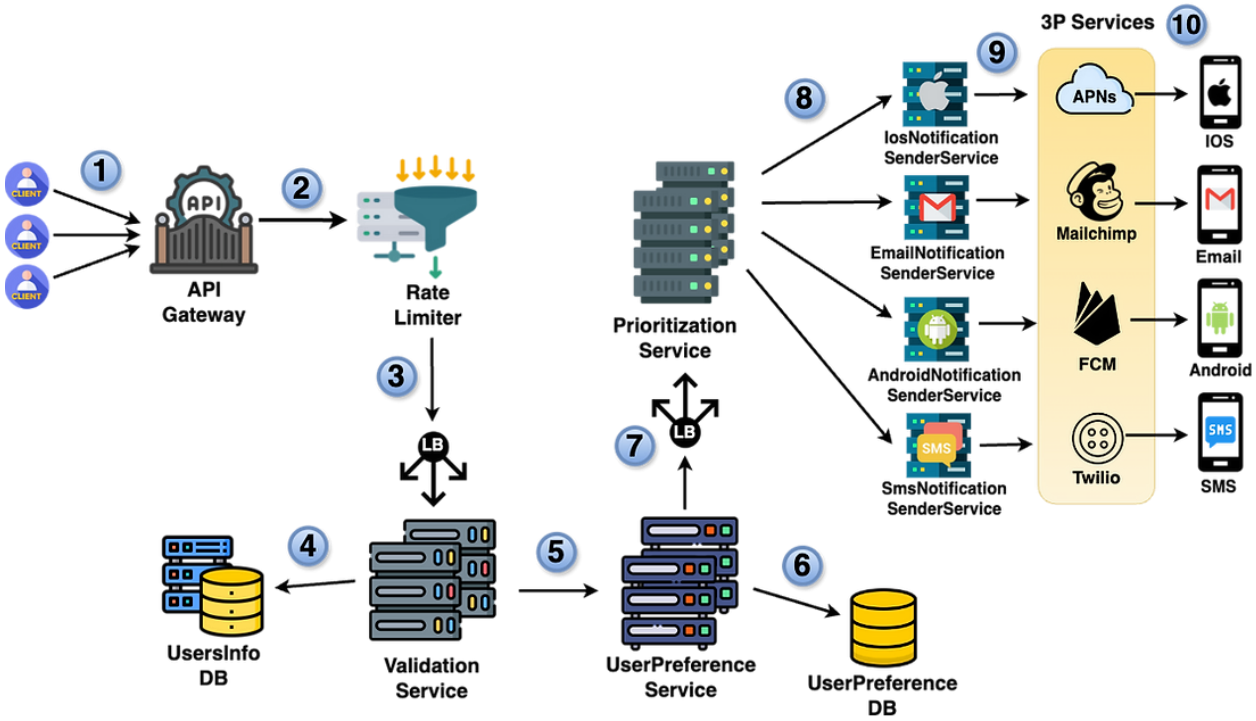
High Level Design-Handling different types of Notifications



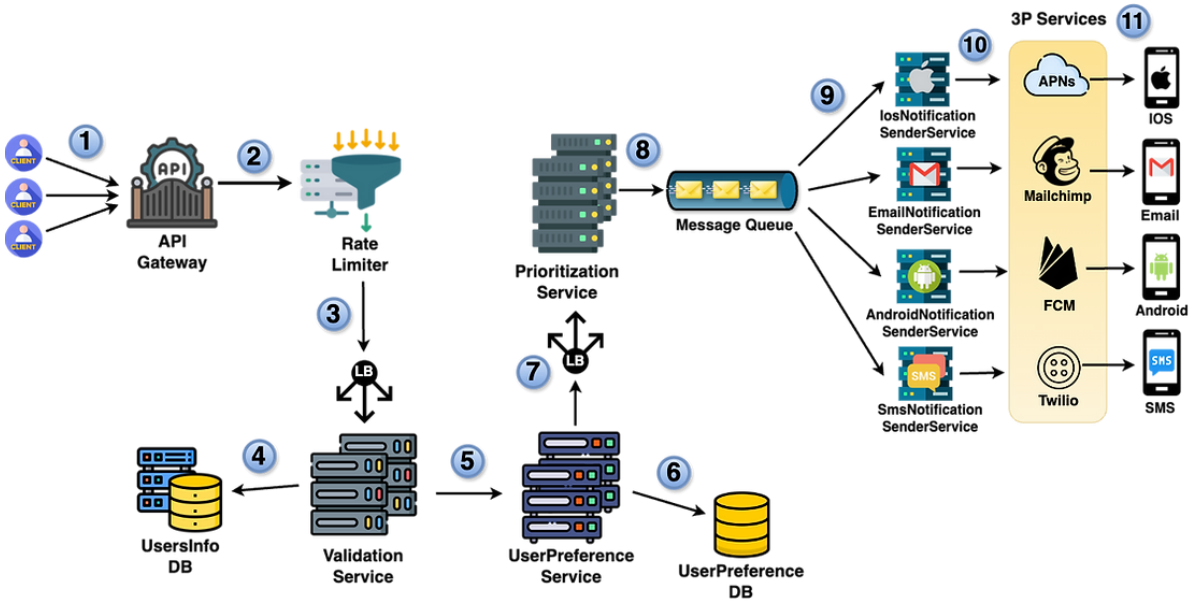




High Level Design-Reorganizing Services



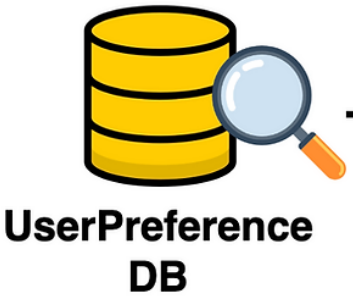
High Level Design-Decoupling & Optimizing Design



Deep Dive Insights- Database Modeling (User Info DB Schema)

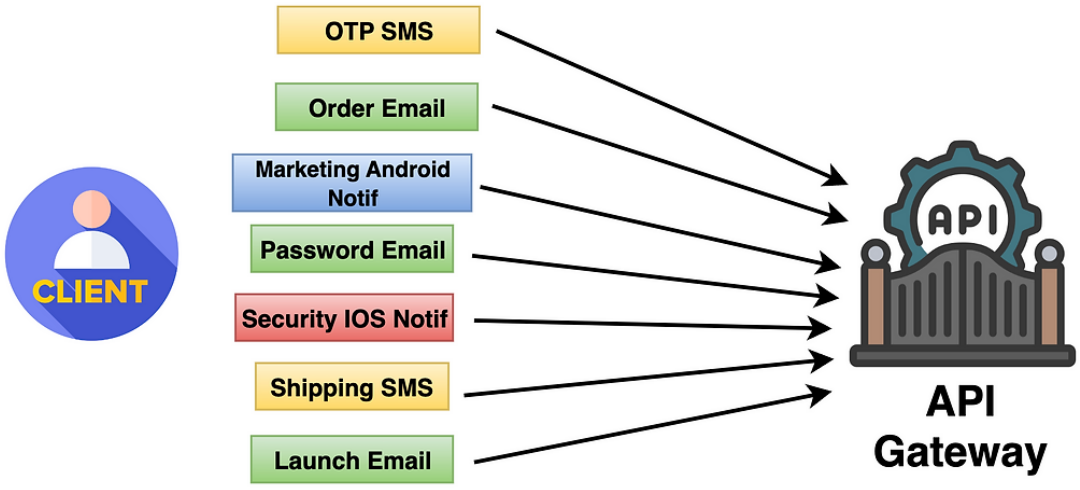
Column Name	Data Type	Description
userId	VARCHAR	The unique identifier for each user (PK).
name	VARCHAR	The user's full name.
email	VARCHAR	The user's email address (unique).
phoneNumber	VARCHAR	The user's phone number (unique).
isActive	BOOLEAN	Indicates if the user's account is active.
createdAt	TIMESTAMP	When the user was created.
updatedAt	TIMESTAMP	When the user's information was last updated.

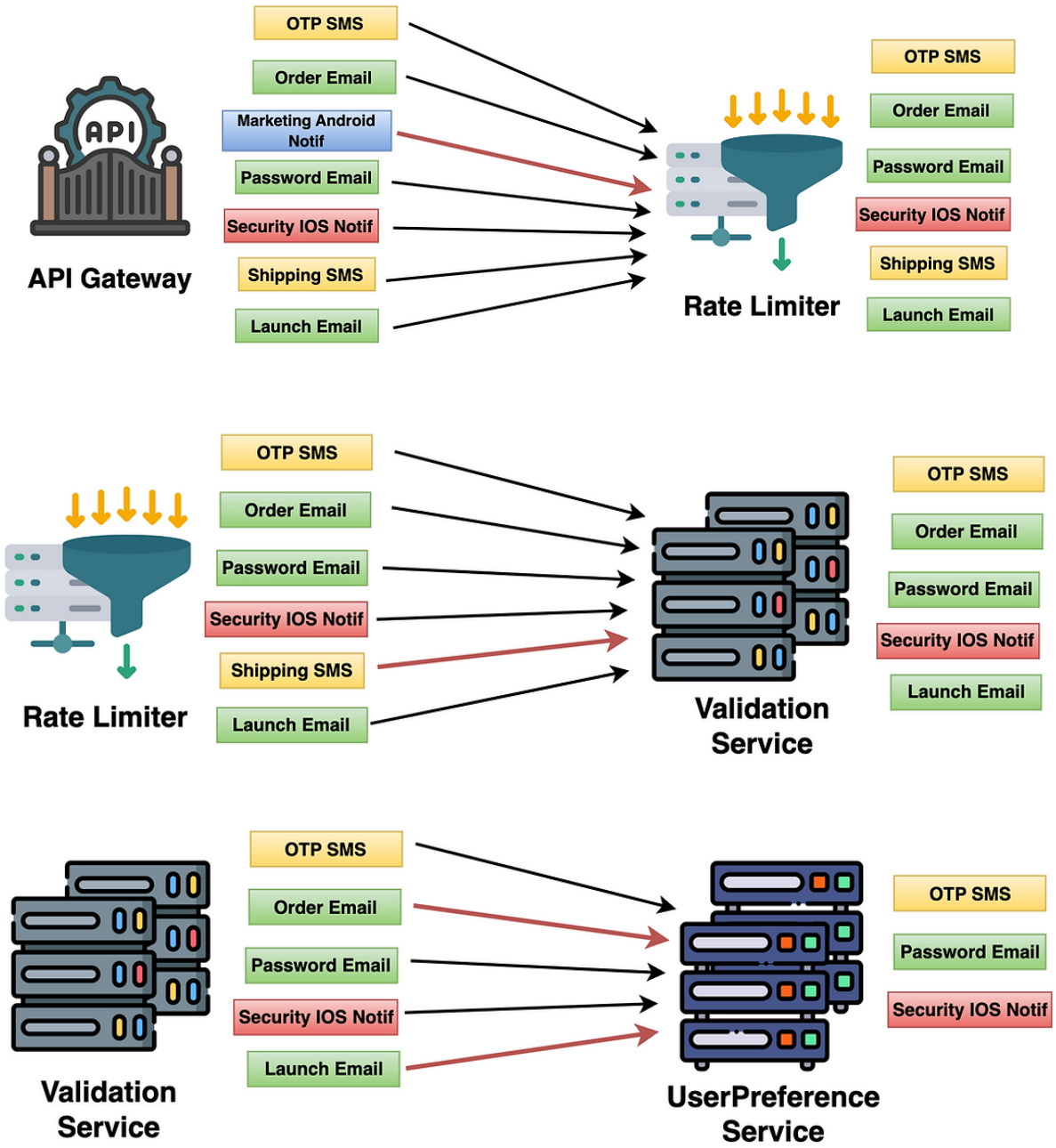
Deep Dive Insights- Database Modeling (User Preference DB Schema)

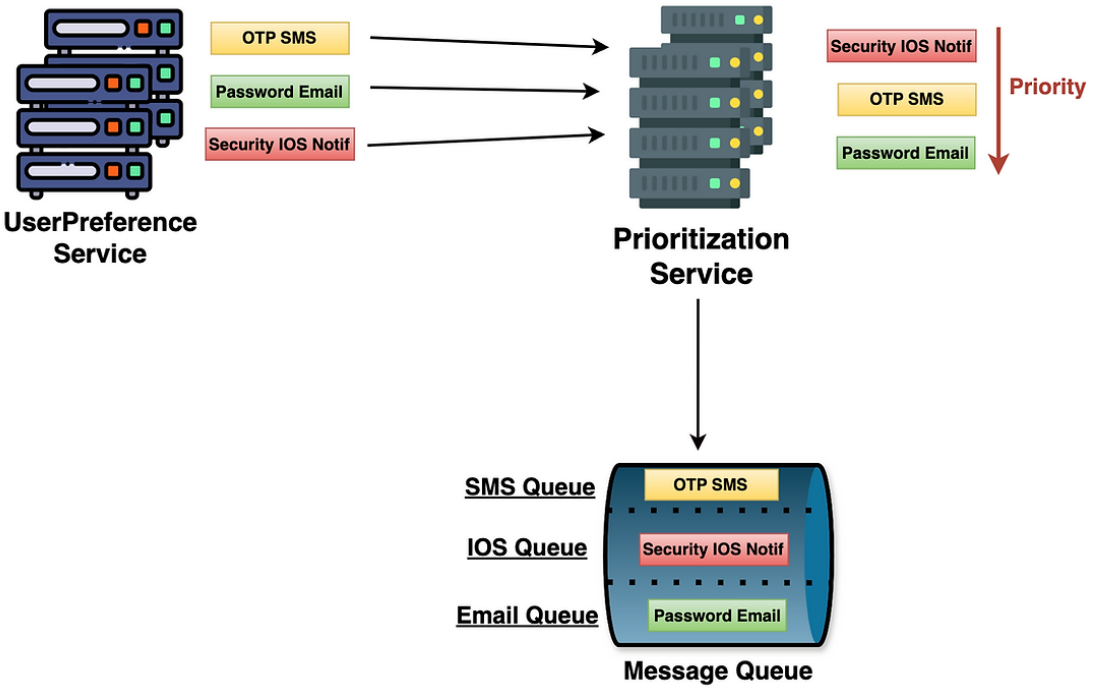


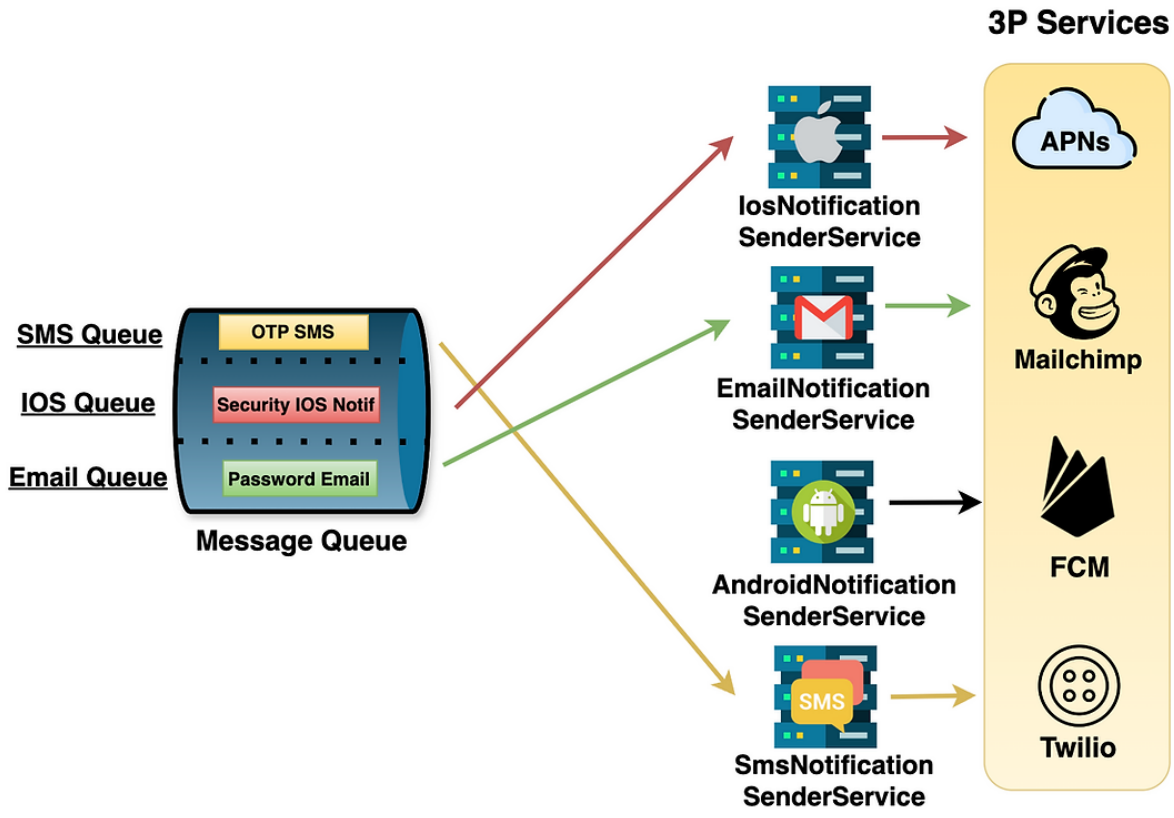

```
{
  "userId": "unique_user_id",
  "preferences": {
    "email": {
      "optIn": true,
      "notificationTypes": ["orderConfirmation", ..]
    },
    "sms": {
      "optIn": true,
      "notificationTypes": ["OTP", ..]
    },
    ...
  },
  "createdAt": "1720789150",
}
```

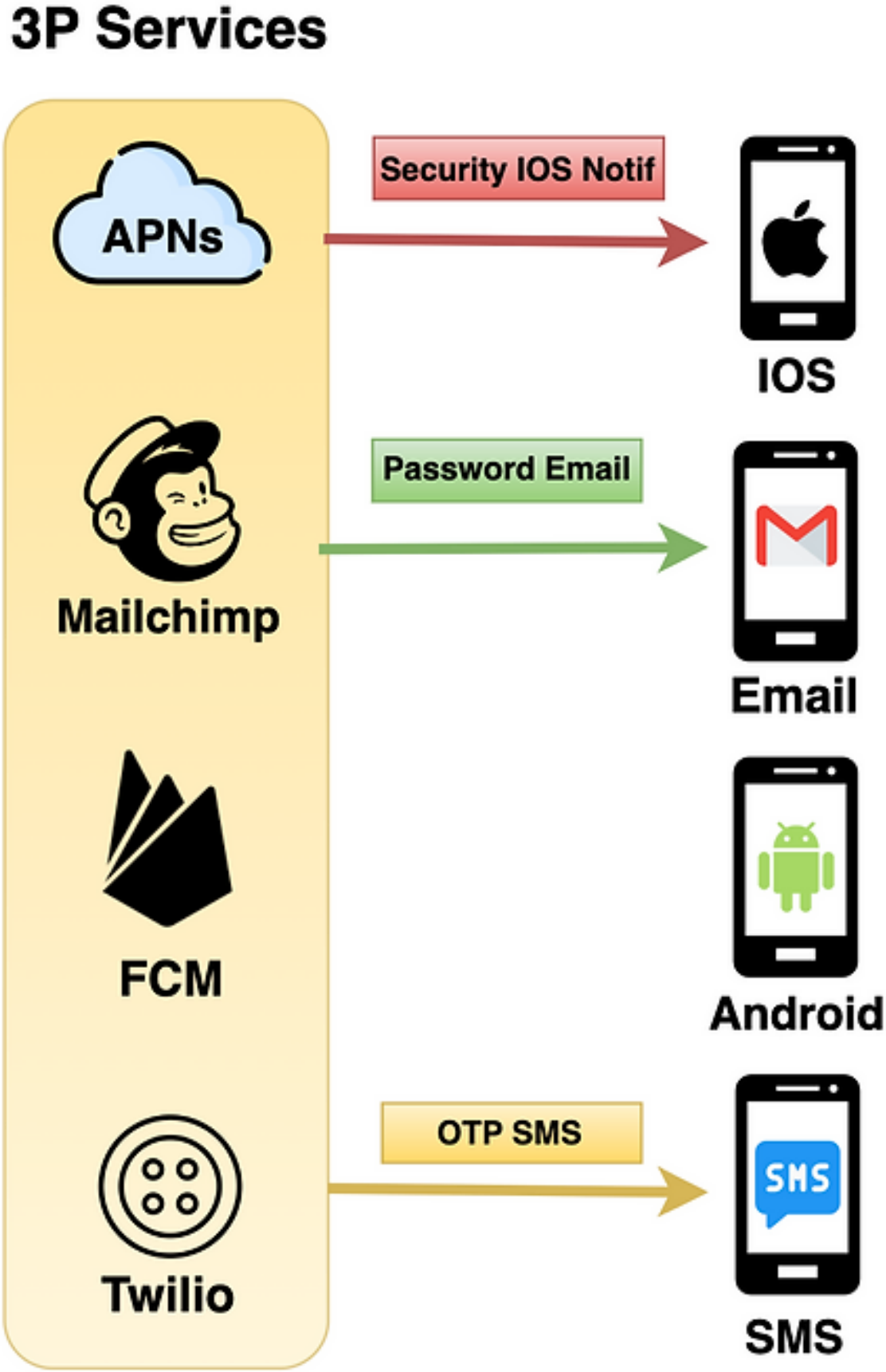
Deep Dive Insights- Notifications Overall Flow



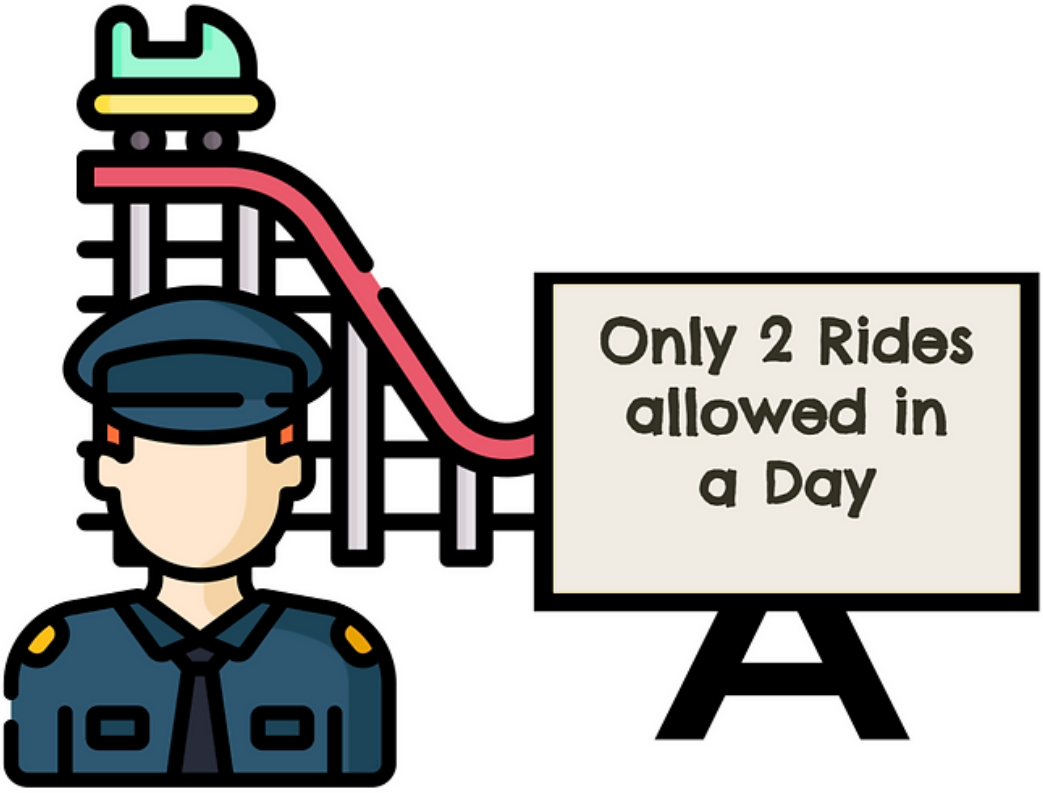

API
API Gateway
OTP SMS
Order Email
Marketing Android Notif
Password Email
Security IOS Notif
Shipping SMS
Launch Email
Rate Limiter
OTP SMS
Order Email
Password Email
Security IOS Notif
Shipping SMS
Launch Email
Rate Limiter
OTP SMS
Order Email
Password Email
Security IOS Notif
Shipping SMS
Launch Email
Validation Service
OTP SMS
Order Email
Password Email
Security IOS Notif
Launch Email
Validation Service
OTP SMS
Order Email
Password Email
Security IOS Notif
Launch Email
UserPreference Service
OTP SMS
Password Email
Security IOS Notif


OTP SMS
Password Email
Security IOS Notif
UserPreference Service
Prioritization Service
Security IOS Notif
OTP SMS
Password Email
Priority
SMS Queue
IOS Queue
Email Queue
OTP SMS
Security IOS Notif
Password Email
Message Queue


3P Services
SMS Queue
IOS Queue
Email Queue
OTP SMS
Security IOS Notif
Password Email
Message Queue
IosNotification SenderService
EmailNotification SenderService
AndroidNotification SenderService
SmsNotification SenderService
SMS
APNs
Mailchimp
FCM
Twilio

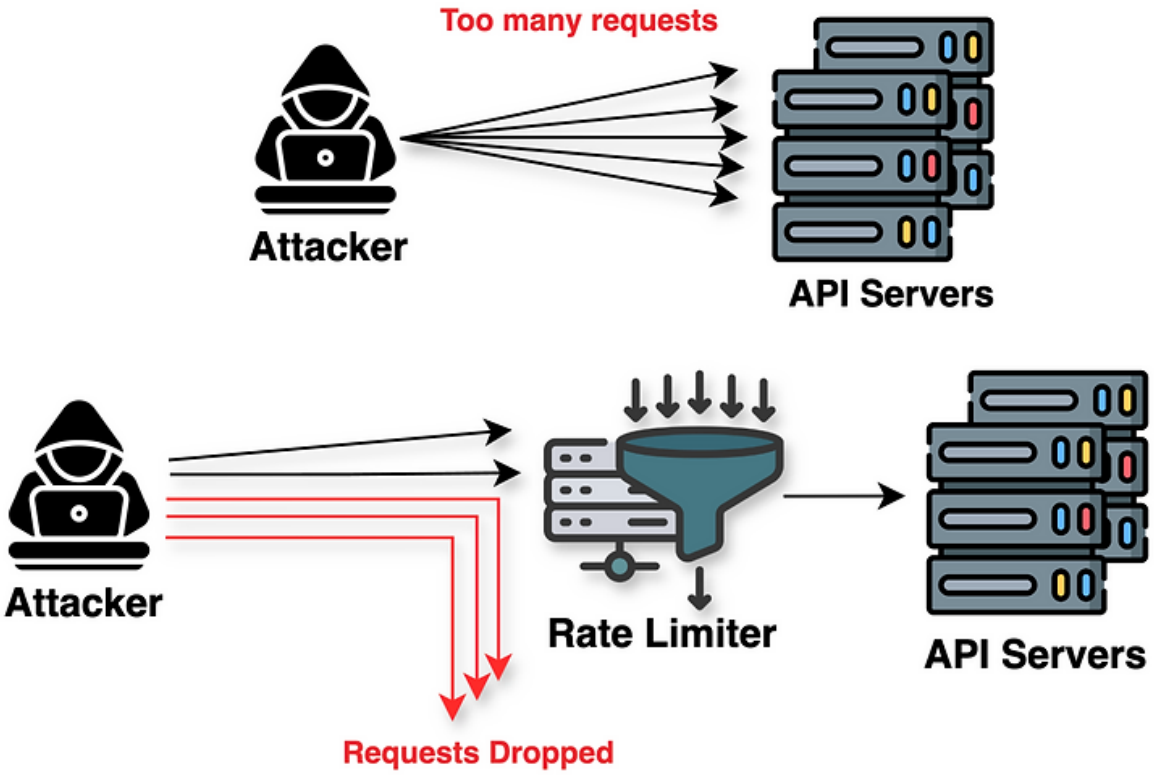

3P Services
APNs
Security IOS Notif
IOS
Mailchimp
Password Email
Email
FCM
Android
Twilio
OTP SMS
SMS
SMS

Analogy

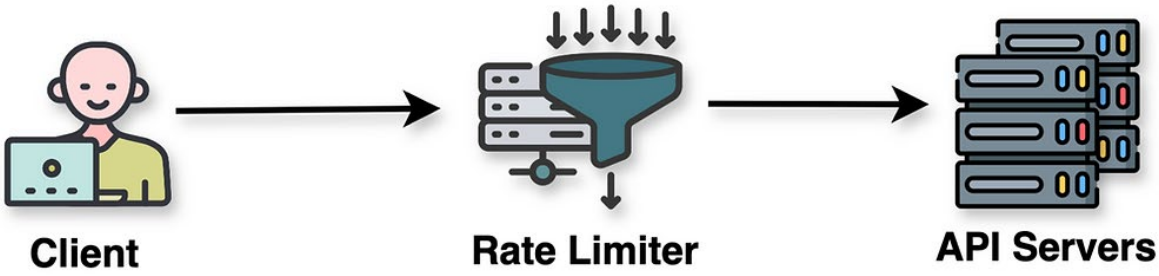


What is a Rate Limiter?

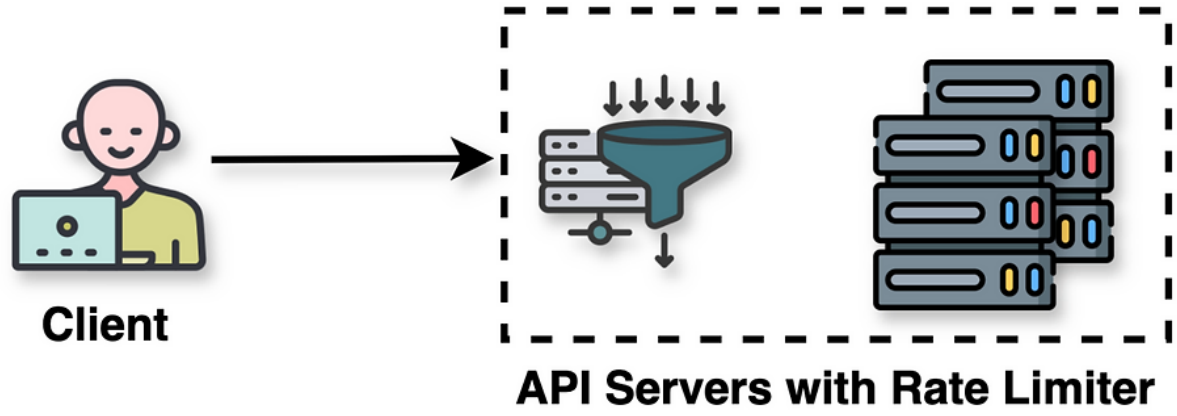
Why do we need a Rate Limiter?



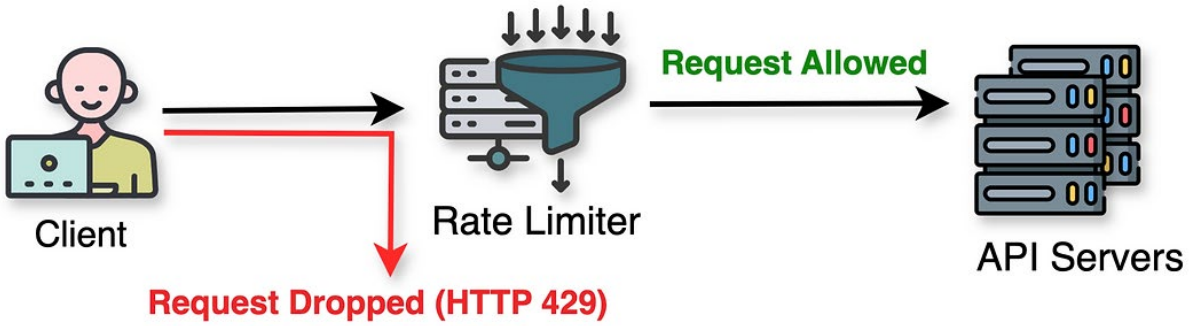
Rate Limiter comes before API Servers



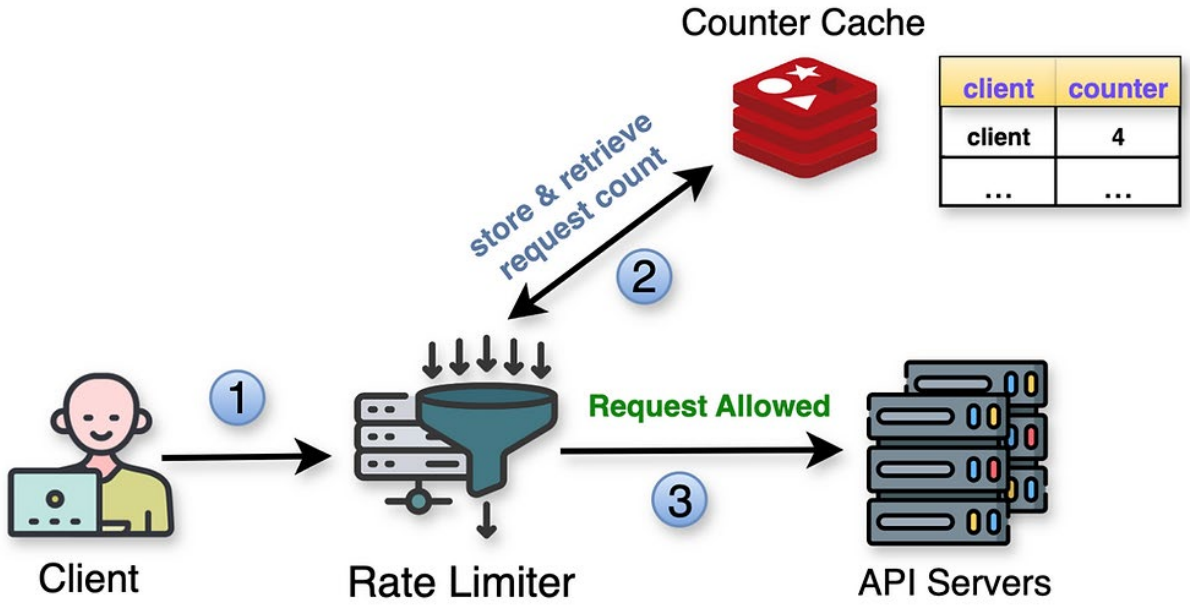
Rate Limiter comes with API Servers



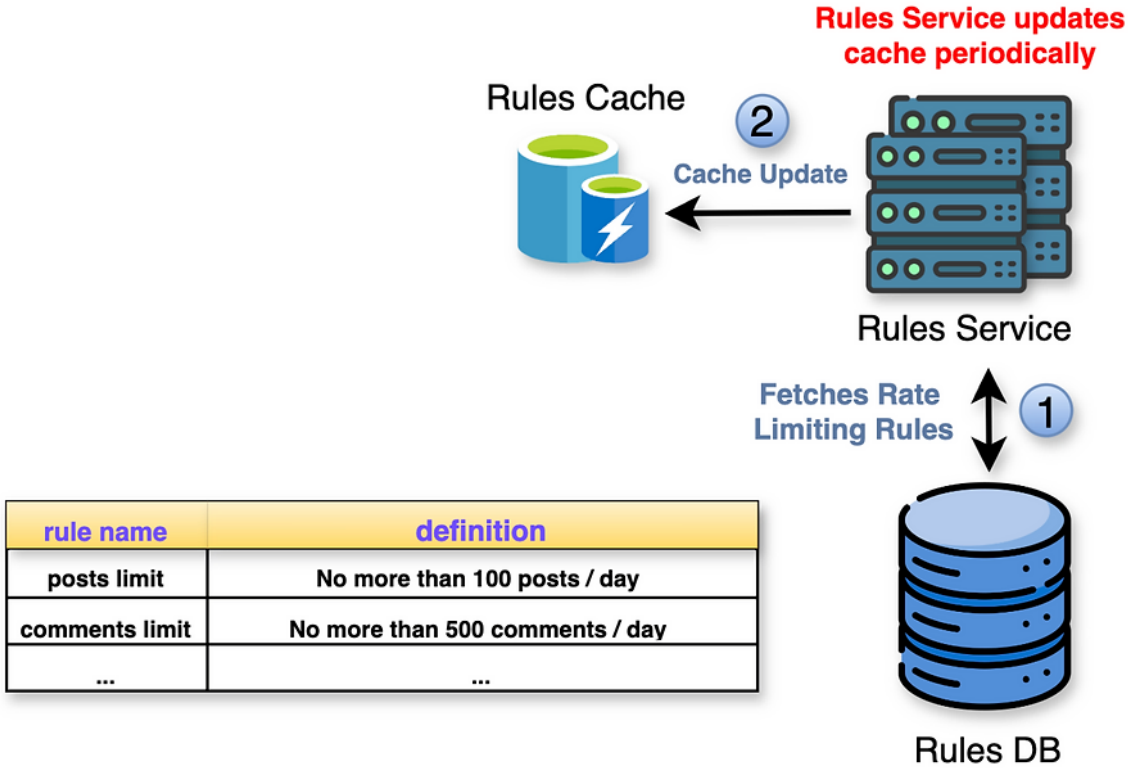
Basic High-Level Flow



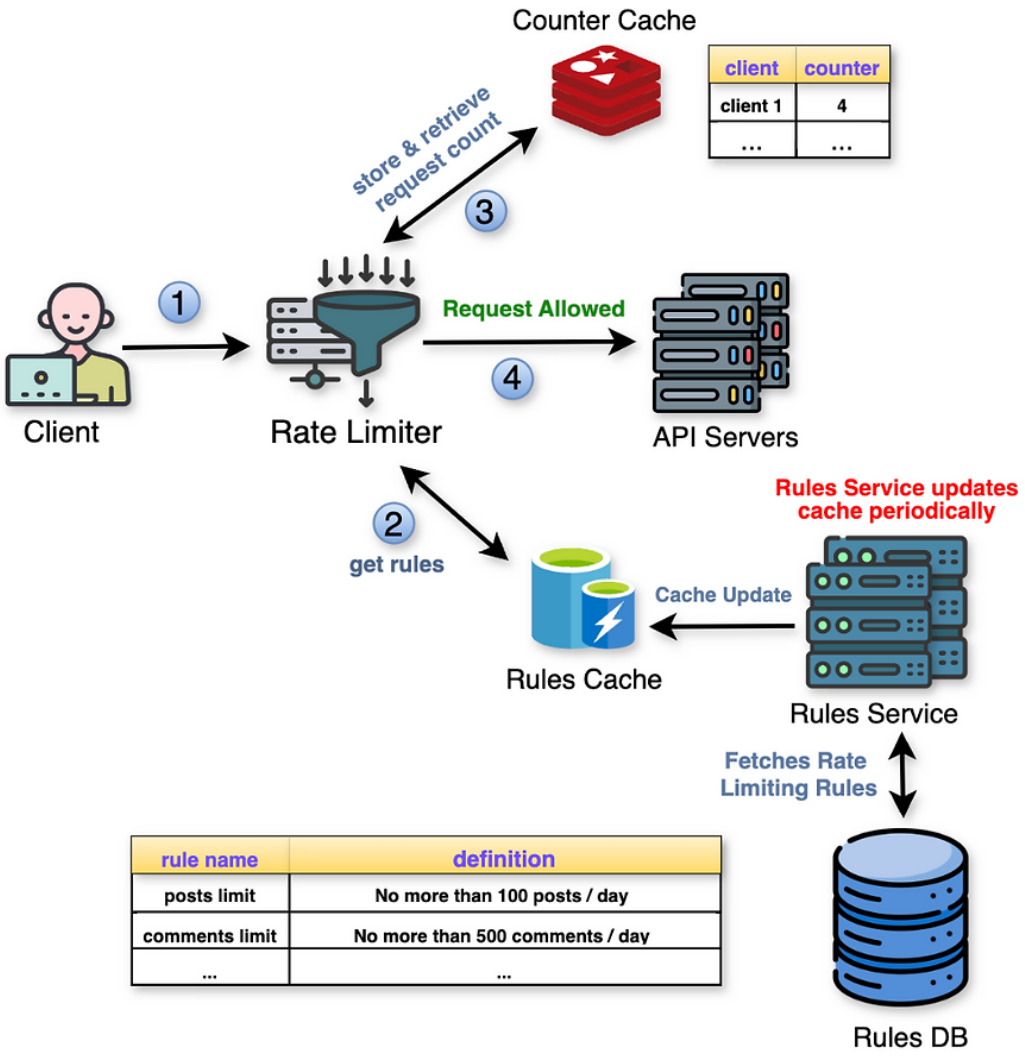
Storing Request Counters



Storing Rate Limiter Rules

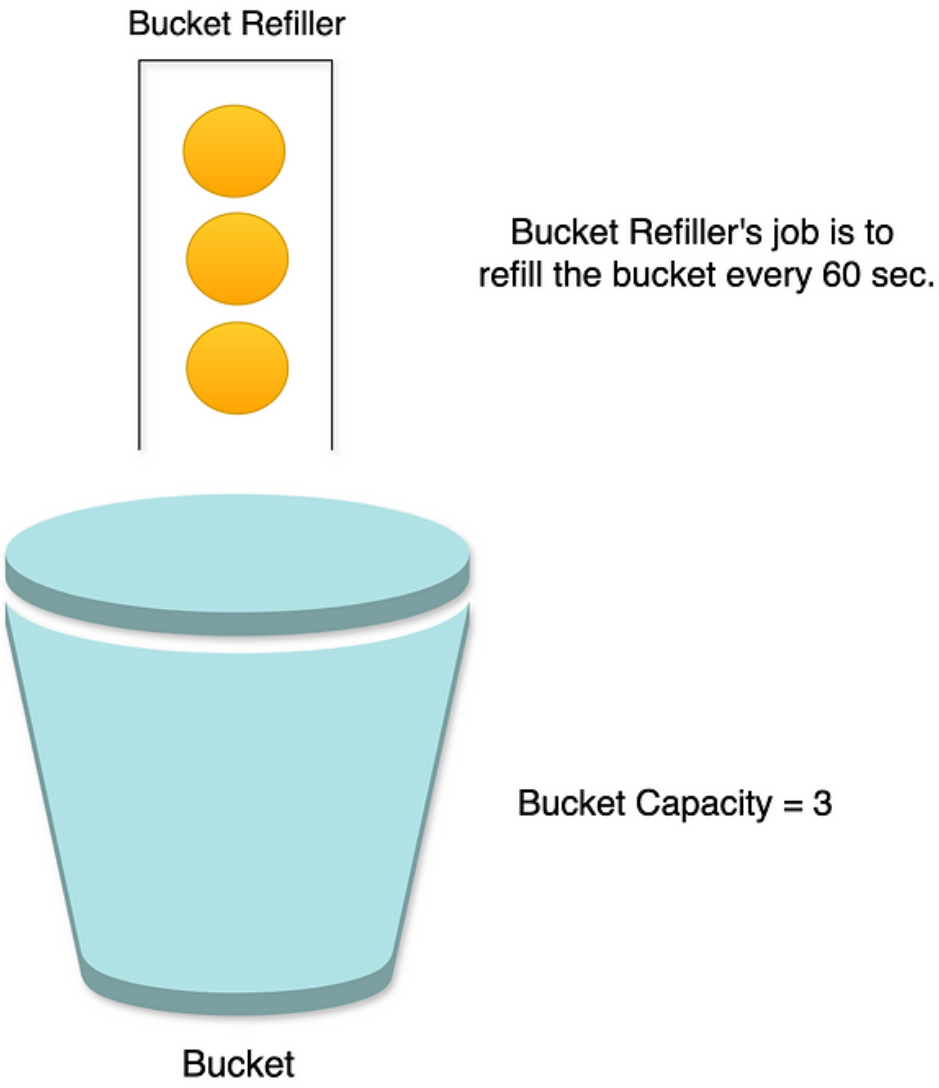


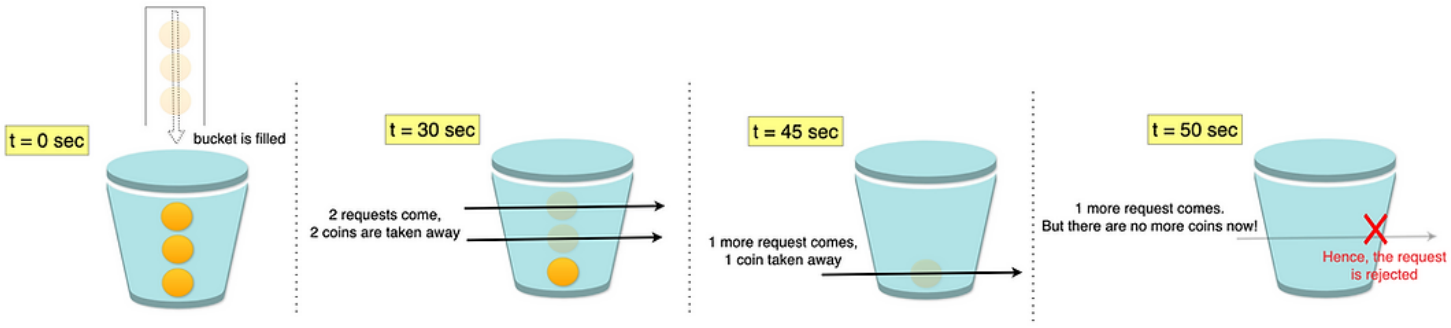
rule name	definition
posts limit	No more than 100 posts / day
comments limit	No more than 500 comments / day
...	...

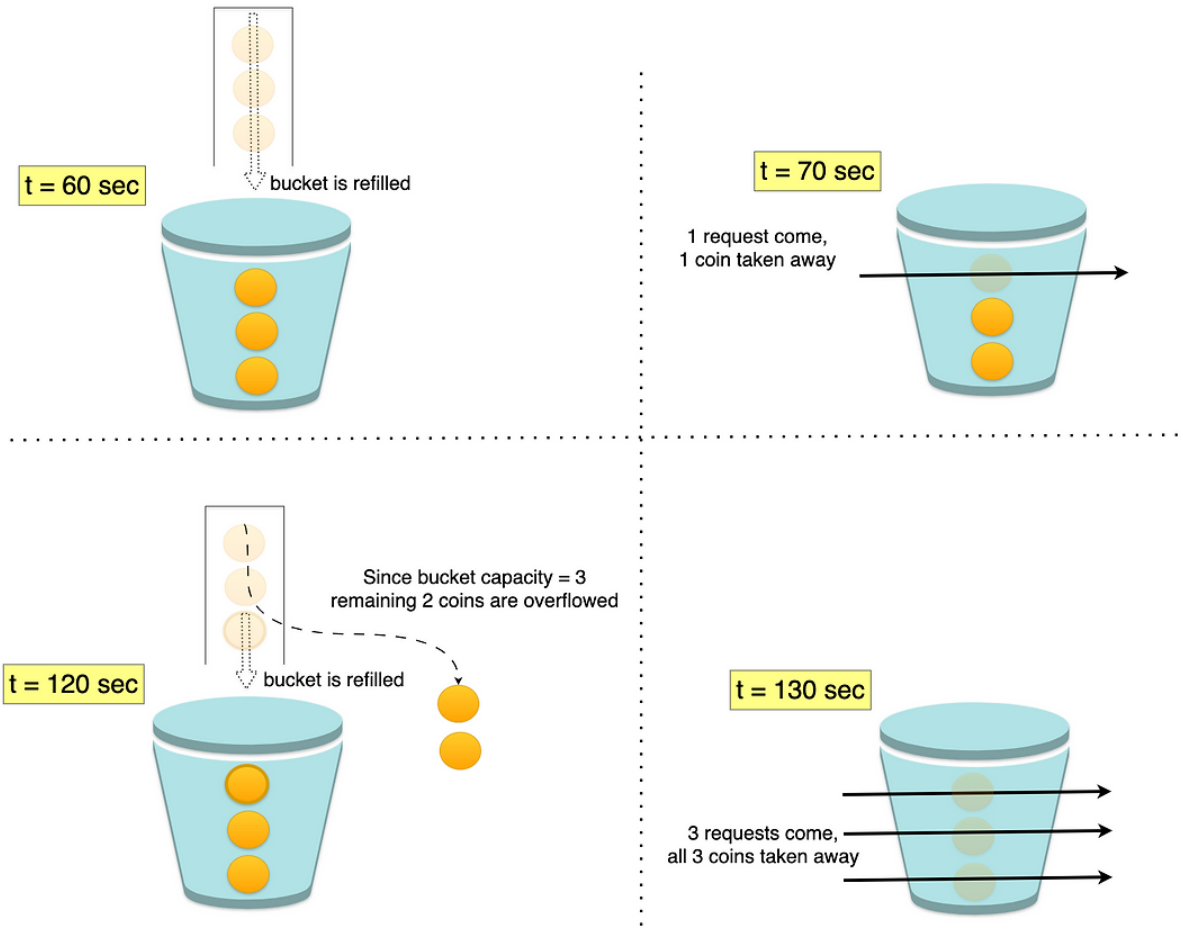


rule name	definition
posts limit	No more than 100 posts / day
comments limit	No more than 500 comments / day
...	...

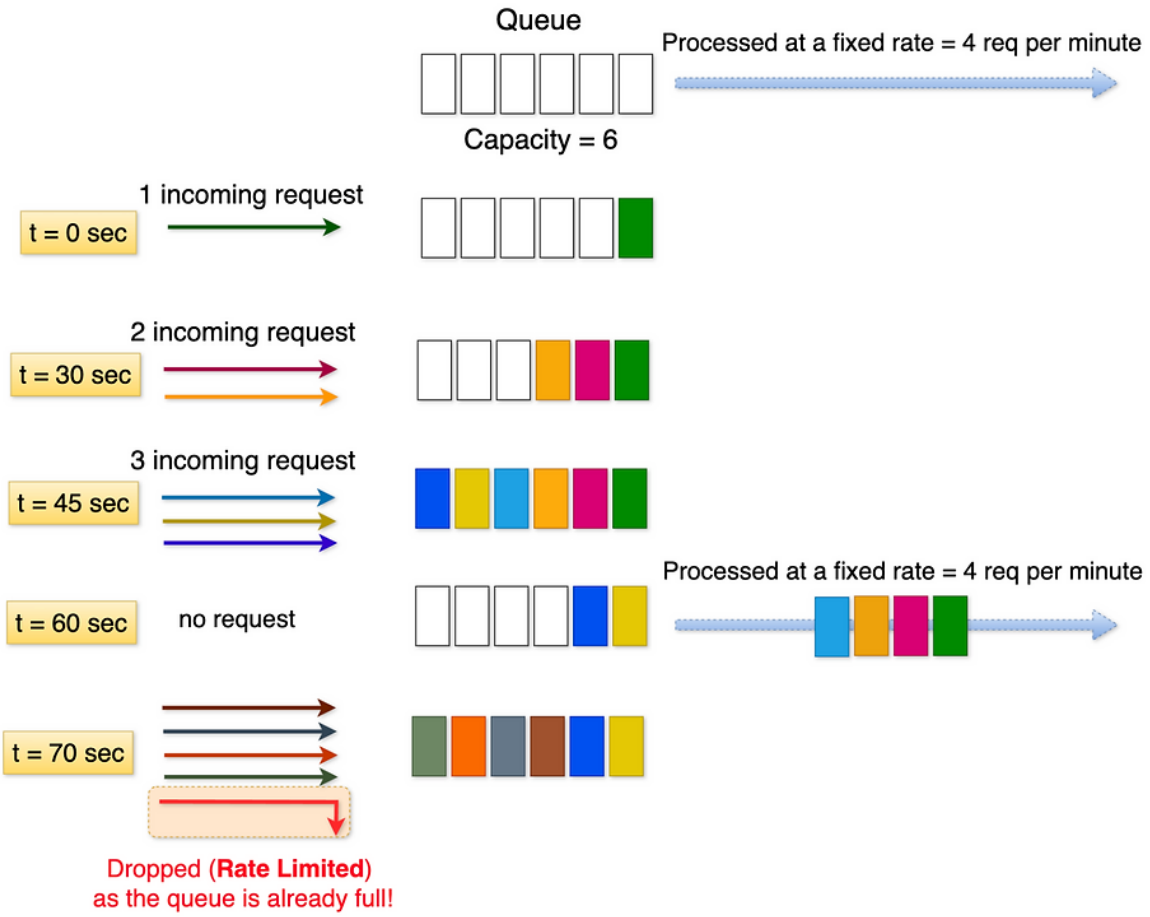
Token Bucket Algorithm





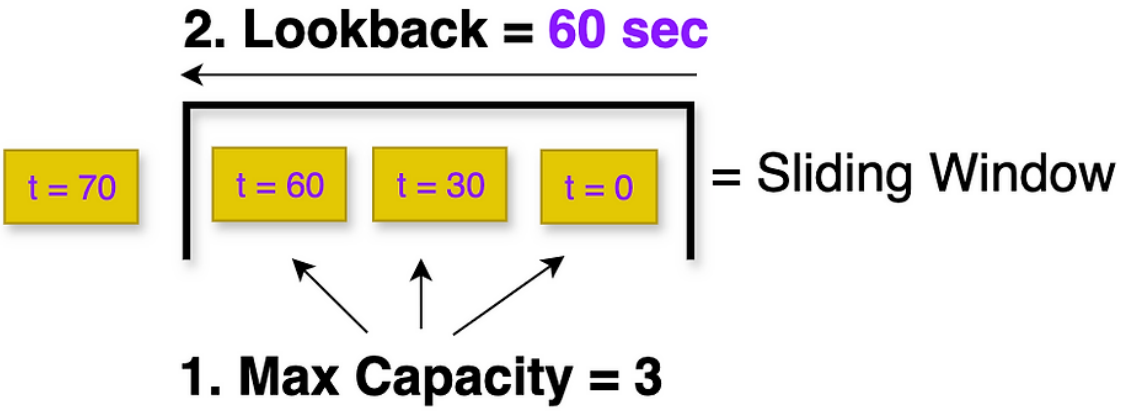


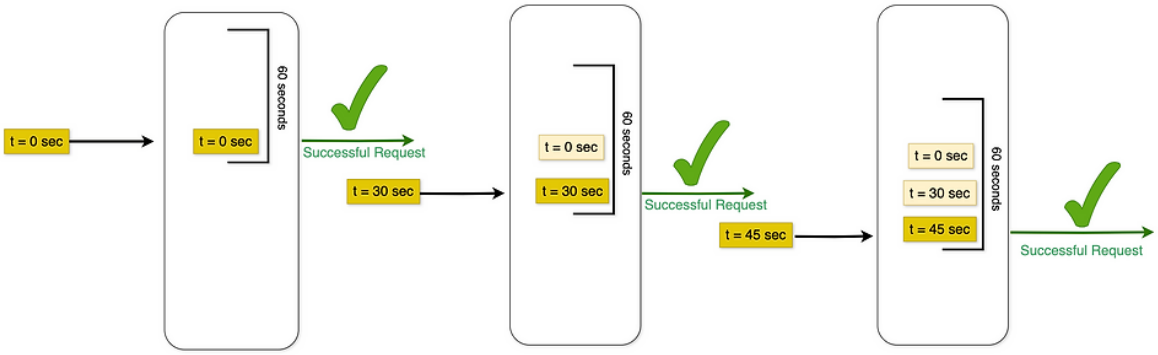
Leaky Bucket Algorithm

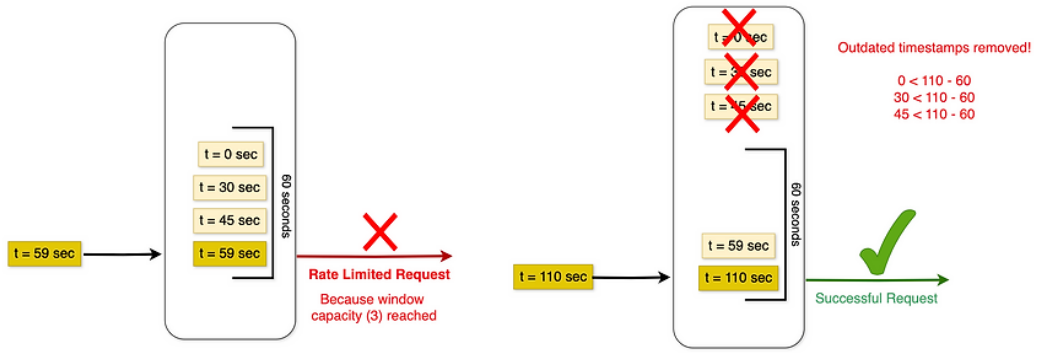


Fixed Window Counter Algorithm

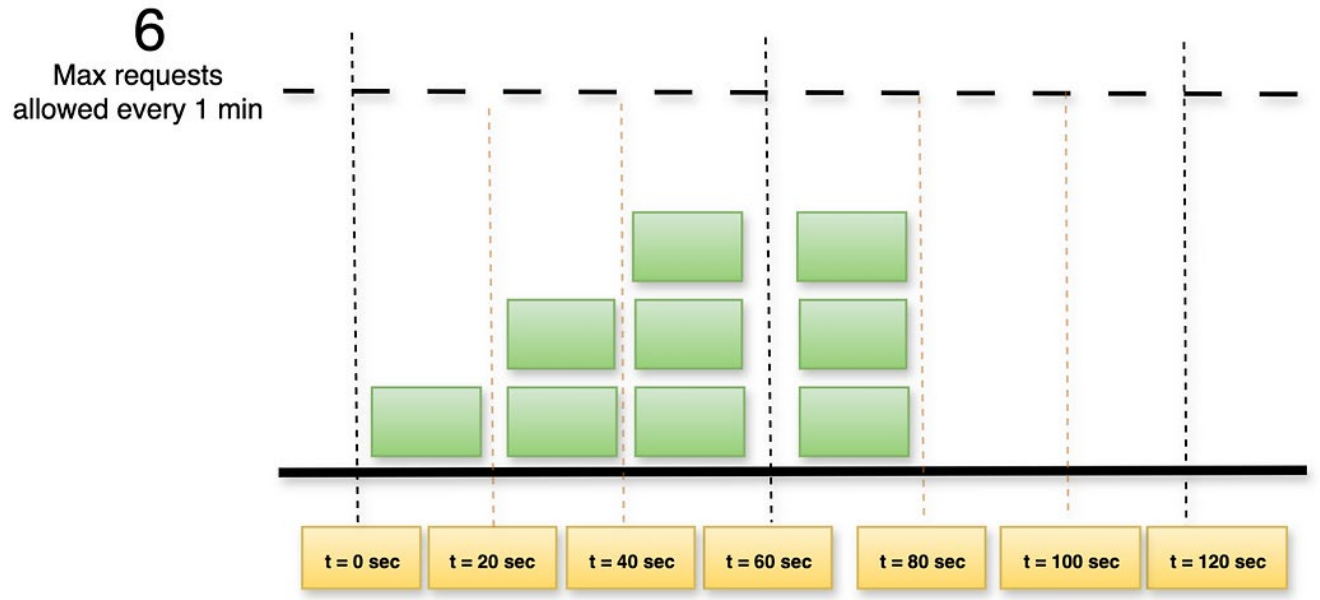
Sliding Window Log Algorithm

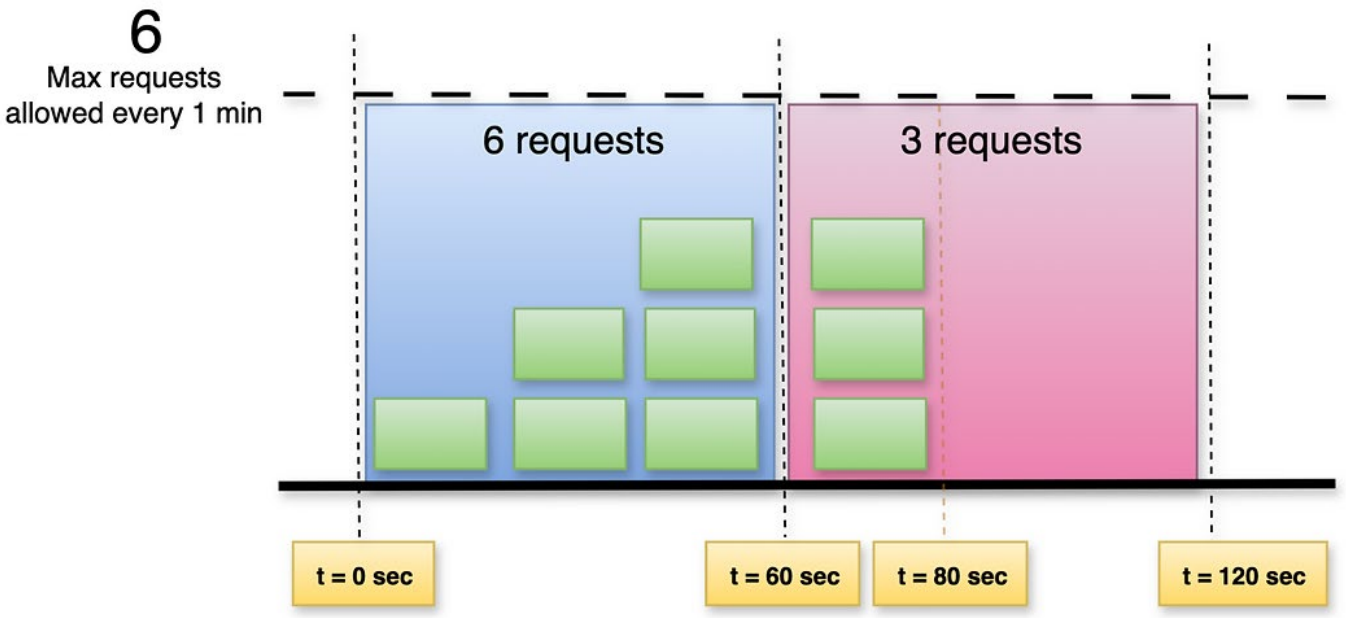


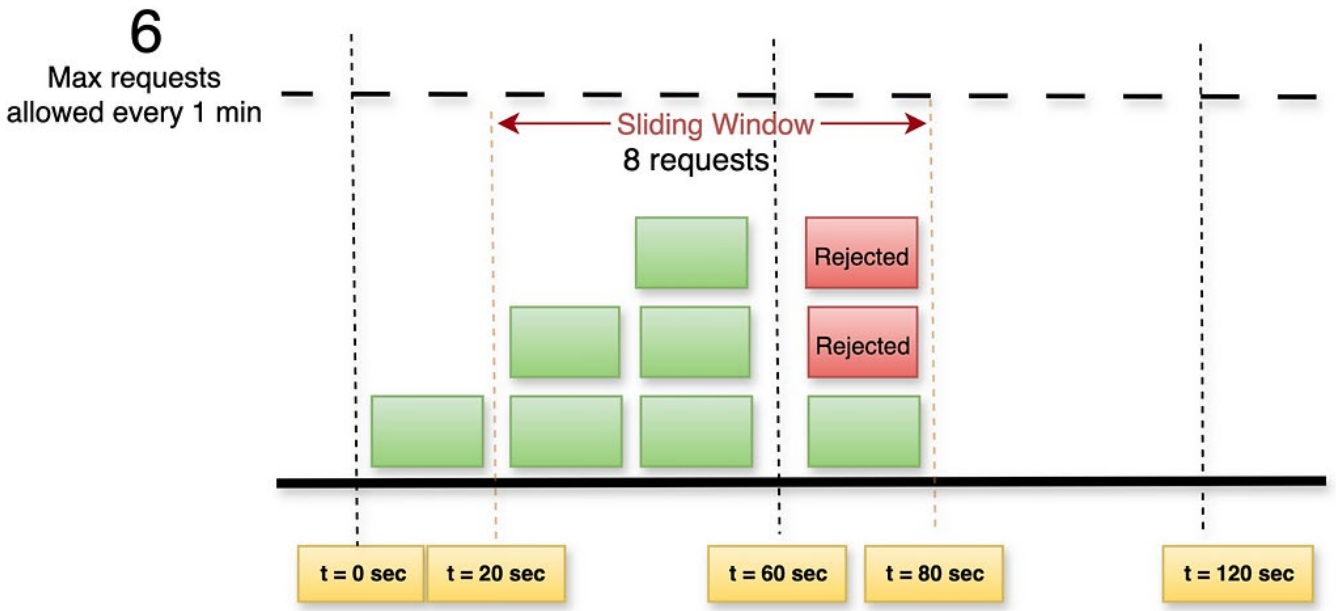


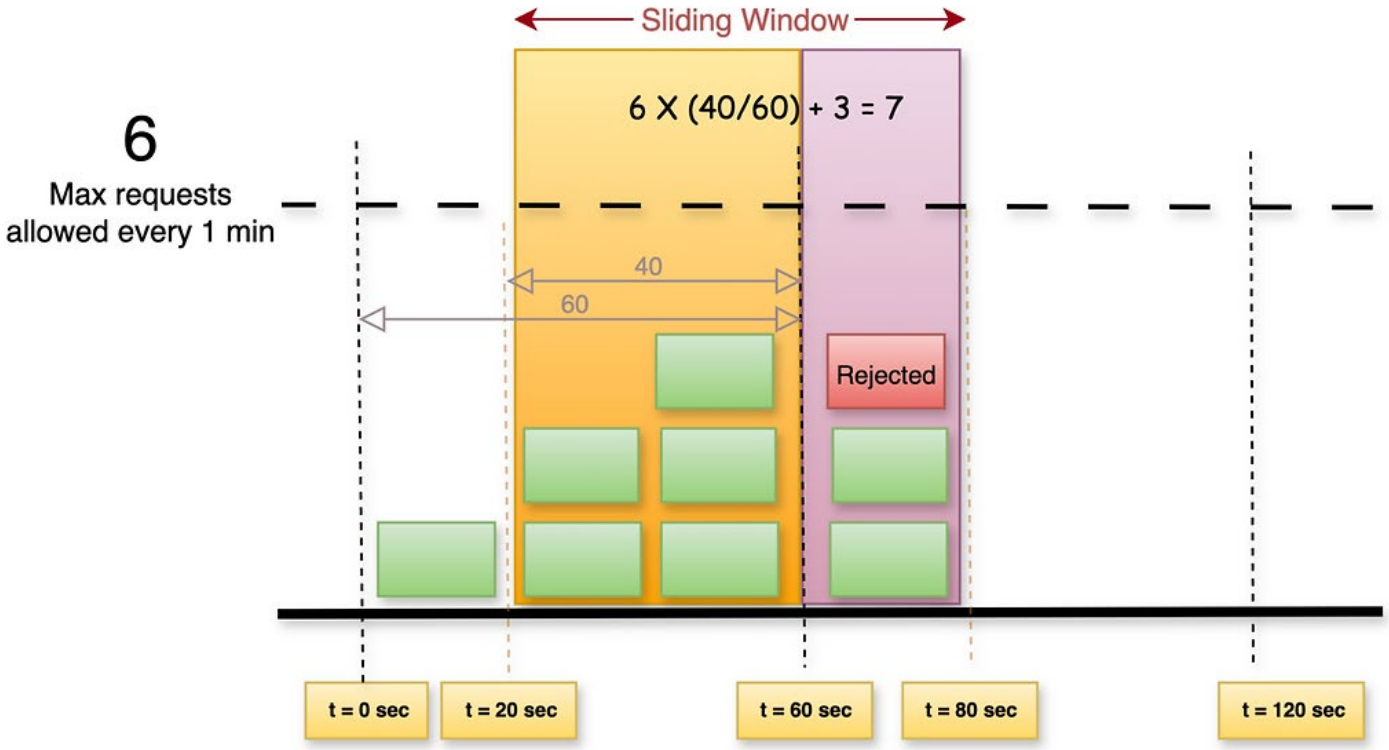


Sliding Window Counter Algorithm

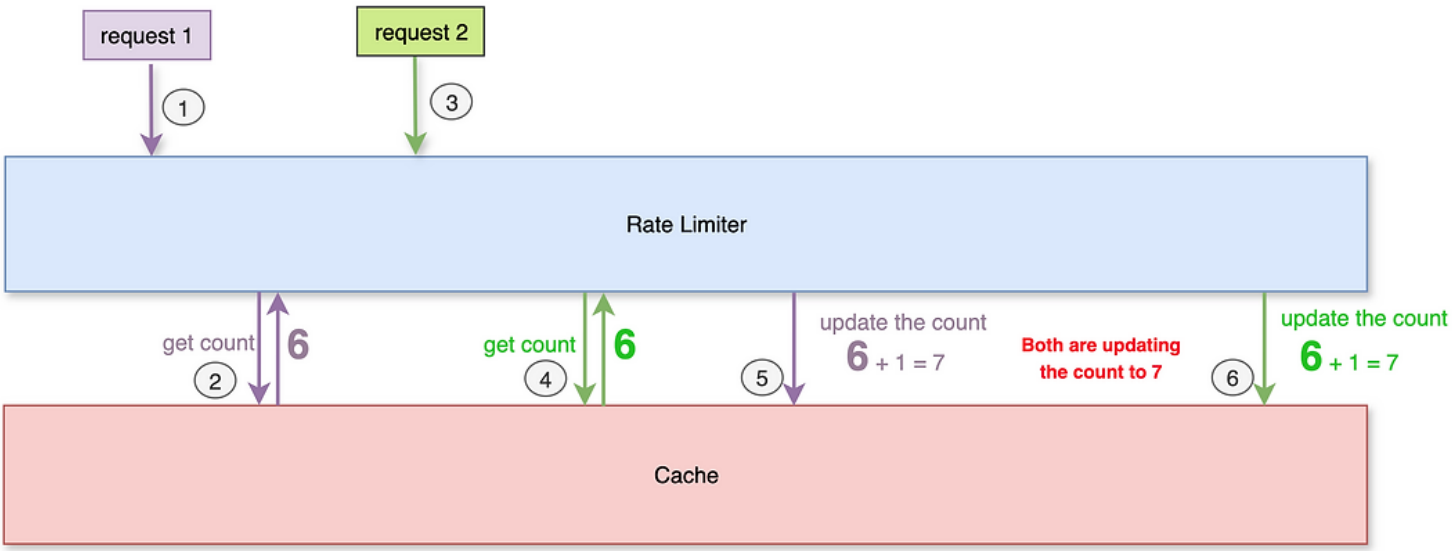


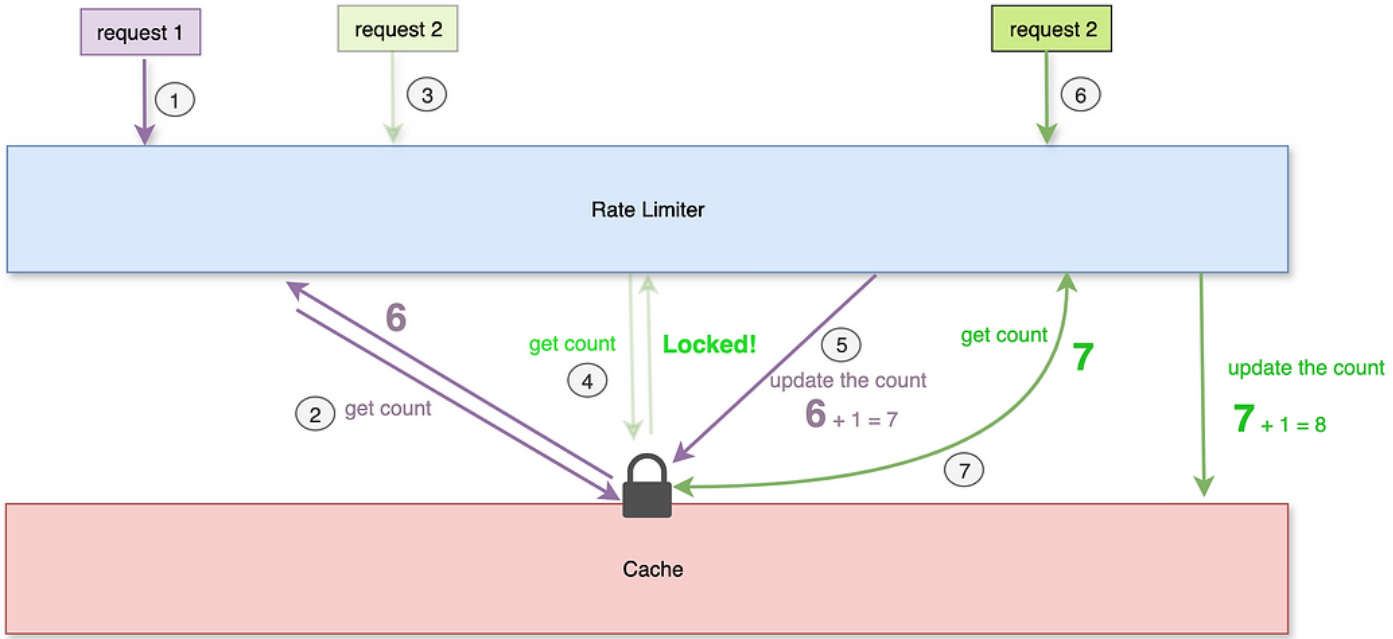



6
Max requests
allowed every 1 min
Sliding Window
8 requests
Rejected
Rejected
t = 0 sec
t = 20 sec
t = 60 sec
t = 80 sec
t = 120 sec

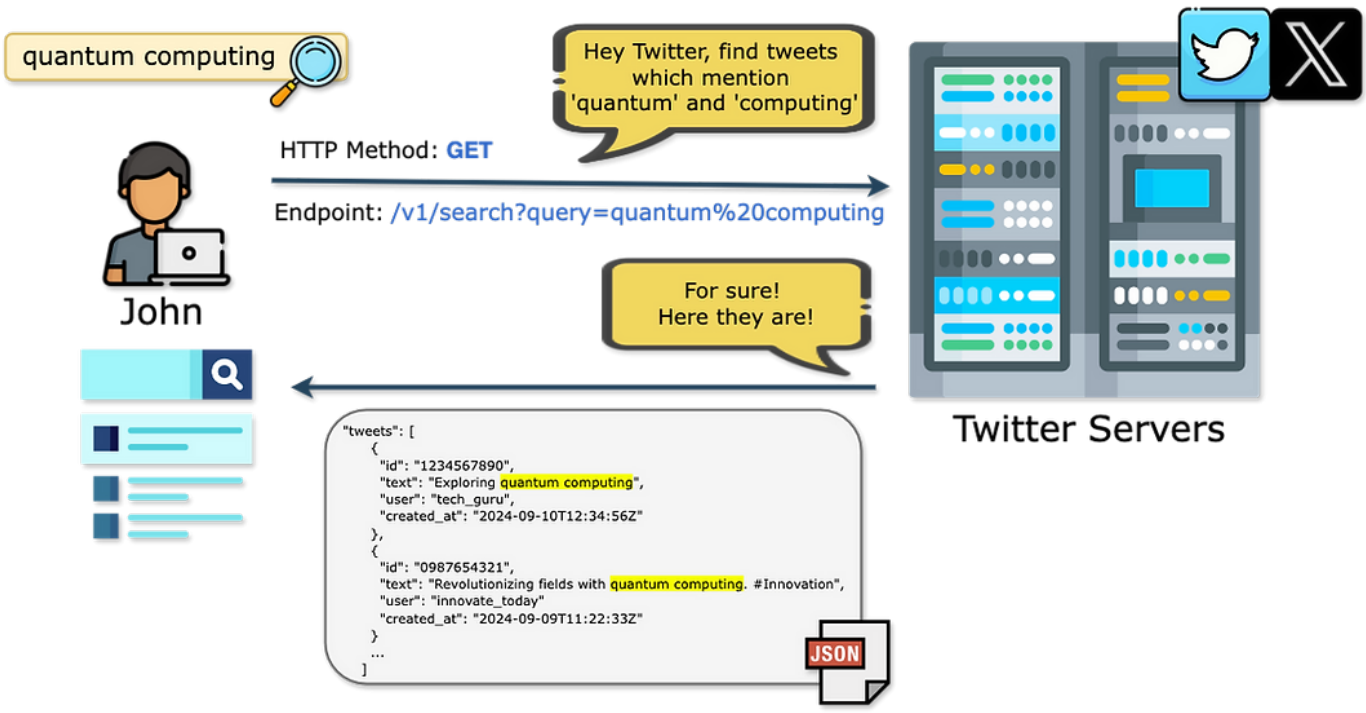

Sliding Window
6 X (40/60) + 3 = 7
6
Max requests
allowed every 1 min
40
60
Rejected
t = 0 sec
t = 20 sec
t = 60 sec
t = 80 sec
t = 120 sec

Handling Race Condition



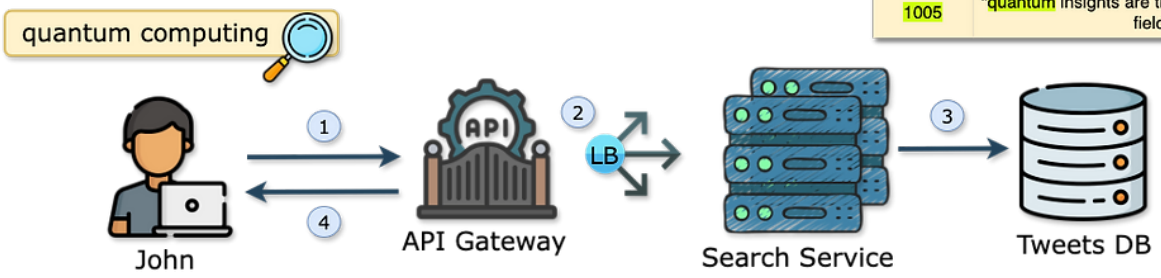


API Design- Twitter Search

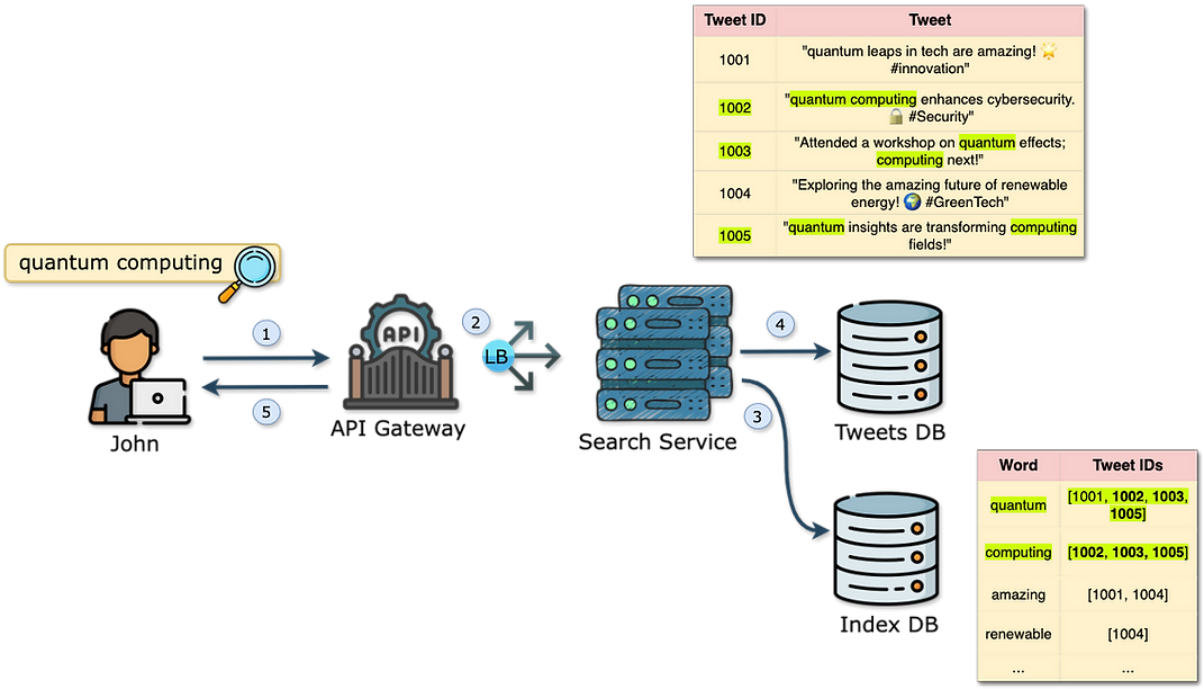


High Level Design- Twitter Search

Tweet ID	Tweet
1001	"quantum leaps in tech are amazing! 💥 #innovation"
1002	"quantum computing enhances cybersecurity. 🔒 #Security"
1003	"Attended a workshop on quantum effects; computing next!"
1004	"Exploring the amazing future of renewable energy! 🌍 #GreenTech"
1005	"quantum insights are transforming computing fields!"



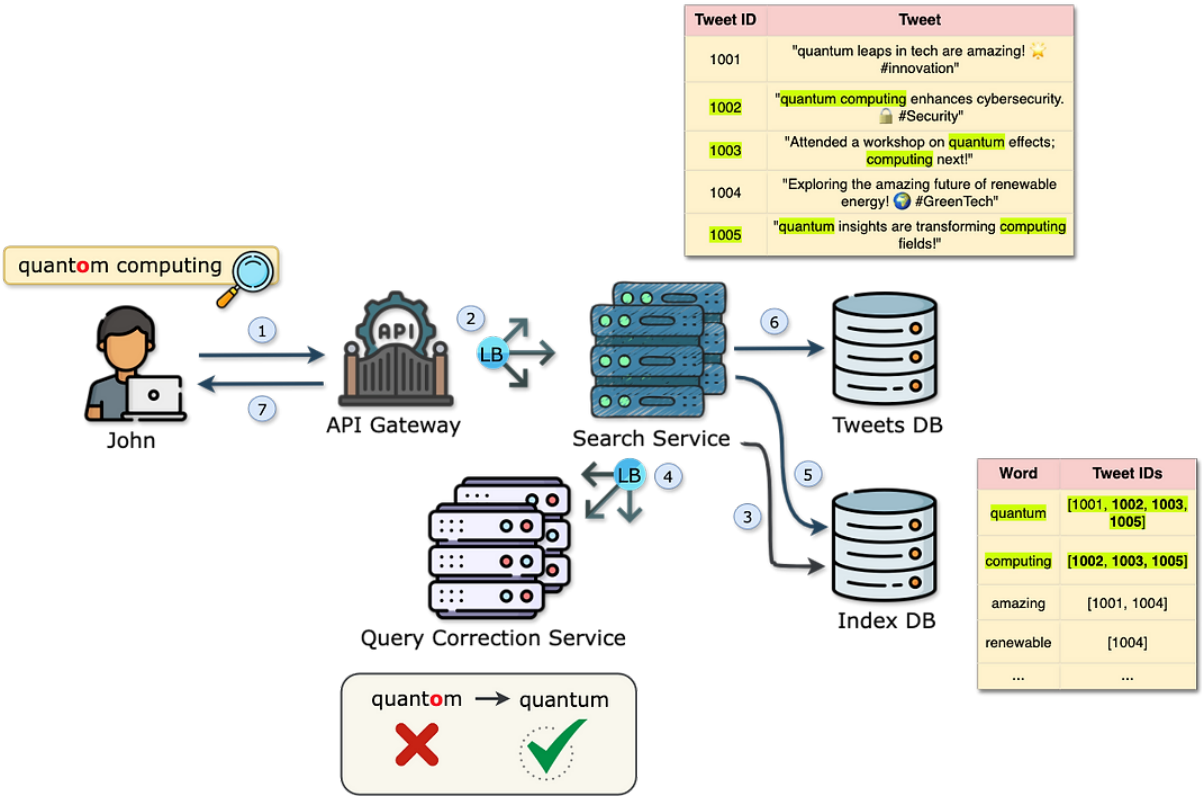
High Level Design- Twitter Search (Indexing)



Tweet ID	Tweet
1001	"quantum leaps in tech are amazing! 💥 #innovation"
1002	"quantum computing enhances cybersecurity. 🔒 #Security"
1003	"Attended a workshop on quantum effects; computing next!"
1004	"Exploring the amazing future of renewable energy! 🌍 #GreenTech"
1005	"quantum insights are transforming computing fields!"

Word	Tweet IDs
quantum	[1001, 1002, 1003, 1005]
computing	[1002, 1003, 1005]
amazing	[1001, 1004]
renewable	[1004]
...	...

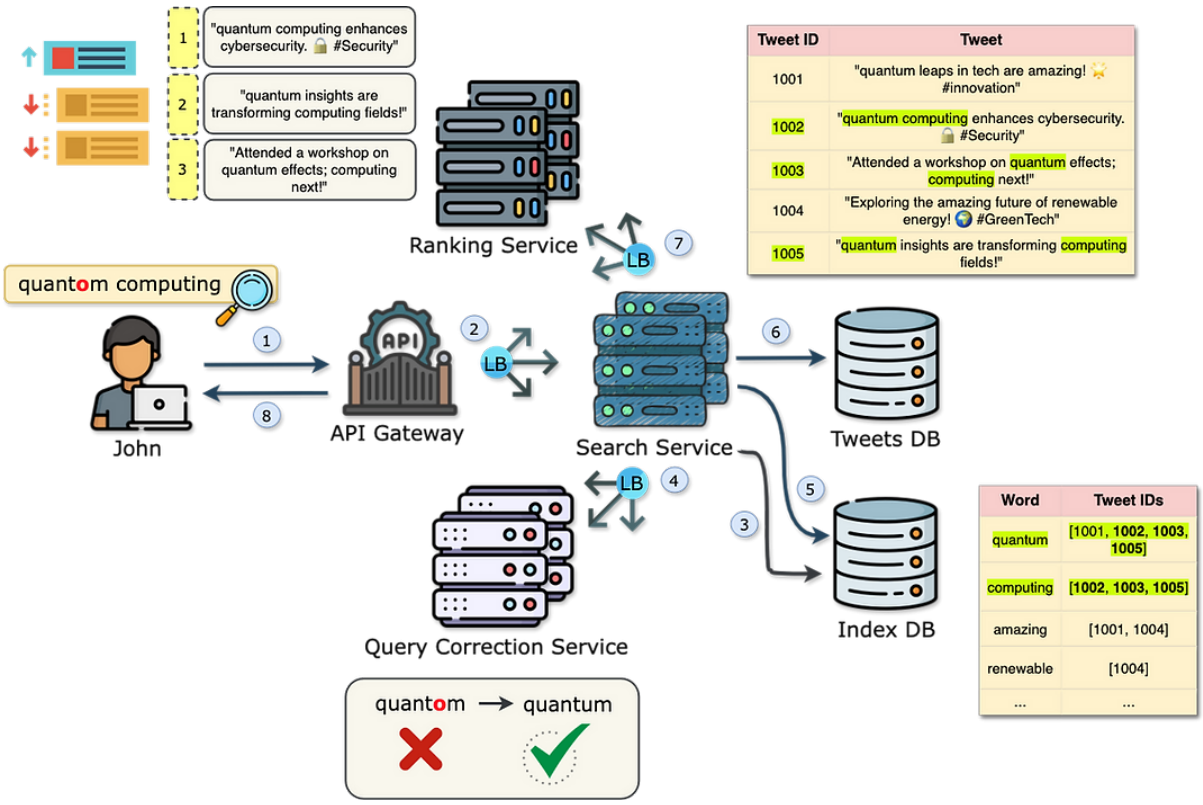
High Level Design- Twitter Search (Fuzzy Search)



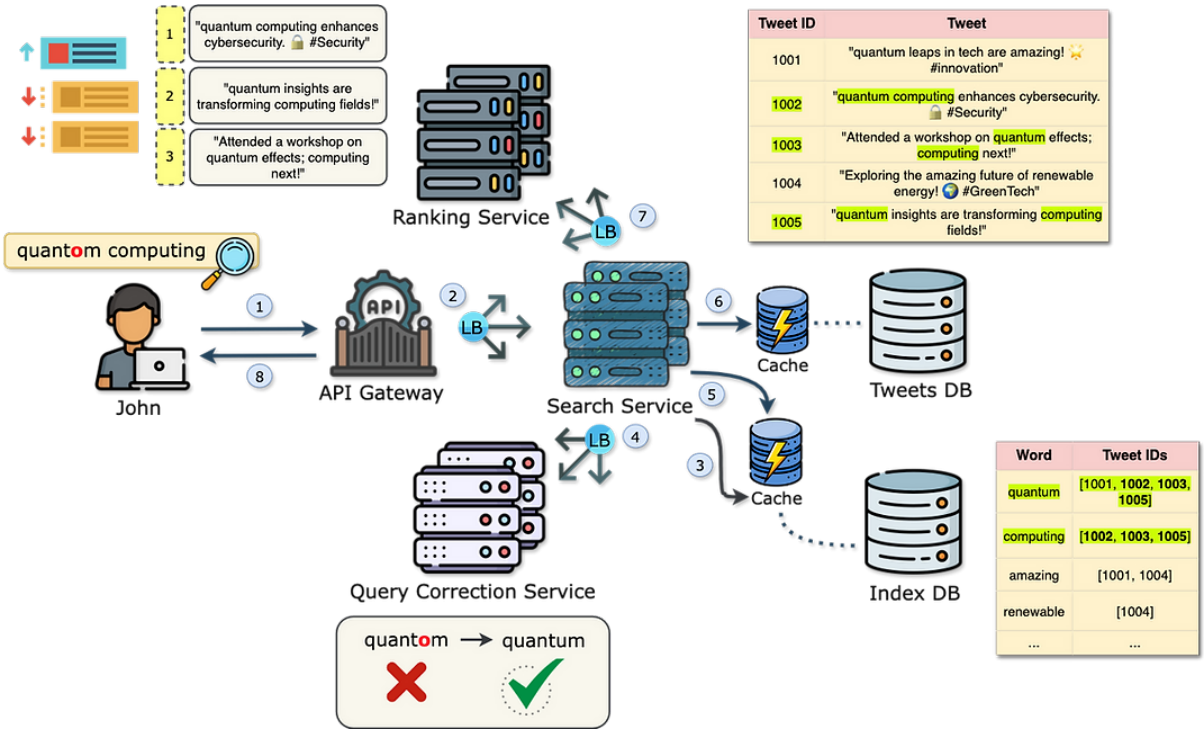
Tweet ID	Tweet
1001	"quantum leaps in tech are amazing! 💥 #innovation"
1002	"quantum computing enhances cybersecurity. 🔒 #Security"
1003	"Attended a workshop on quantum effects; computing next!"
1004	"Exploring the amazing future of renewable energy! 🌍 #GreenTech"
1005	"quantum insights are transforming computing fields!"

Word	Tweet IDs
quantum	[1001, 1002, 1003, 1005]
computing	[1002, 1003, 1005]
amazing	[1001, 1004]
renewable	[1004]
...	...

High Level Design- Twitter Search (Ranking)

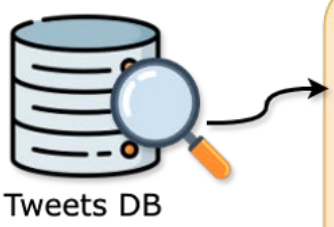


High Level Design- Twitter Search (Final Design)



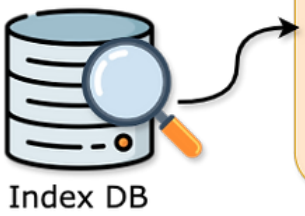
DATABSE SCHEMAS

Tweets DB Schema




```
{
  "TweetId": "Unique identifier for each tweet",
  "Text": "Content of the tweet",
  "UserId": "Identifier for the user who posted the tweet",
  "CreatedAt": "Timestamp when the tweet was created"
}
```

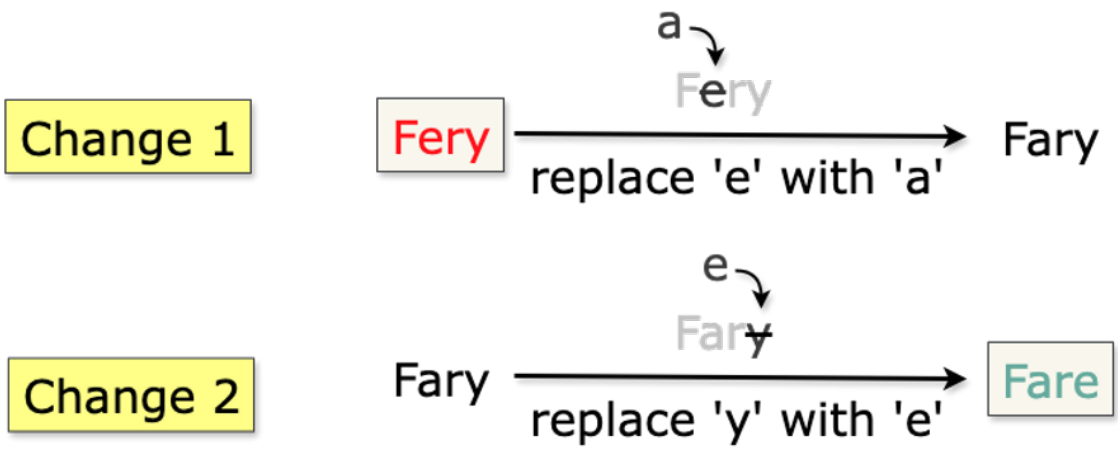
Index DB Schema



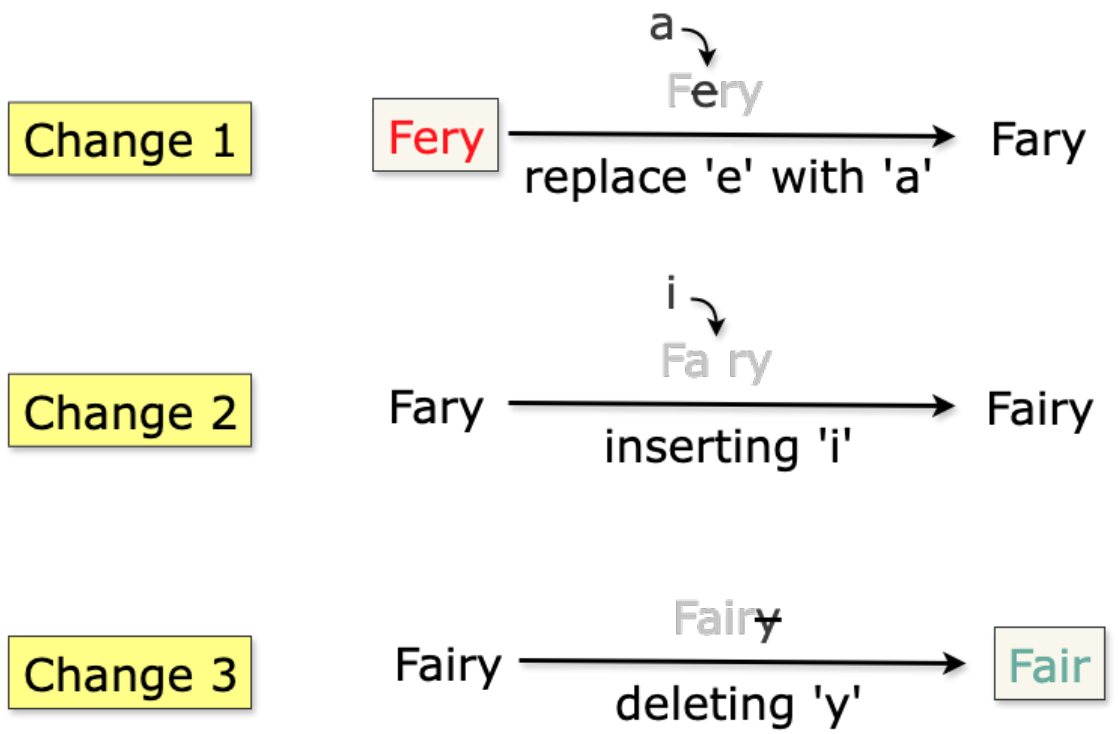

```
{
  "Keyword": "A specific word or hashtag",
  "TweetIds": ["List of tweet Ids having this keyword in the tweet"]
}
```

Query Correction Service (Fery to Ferry)

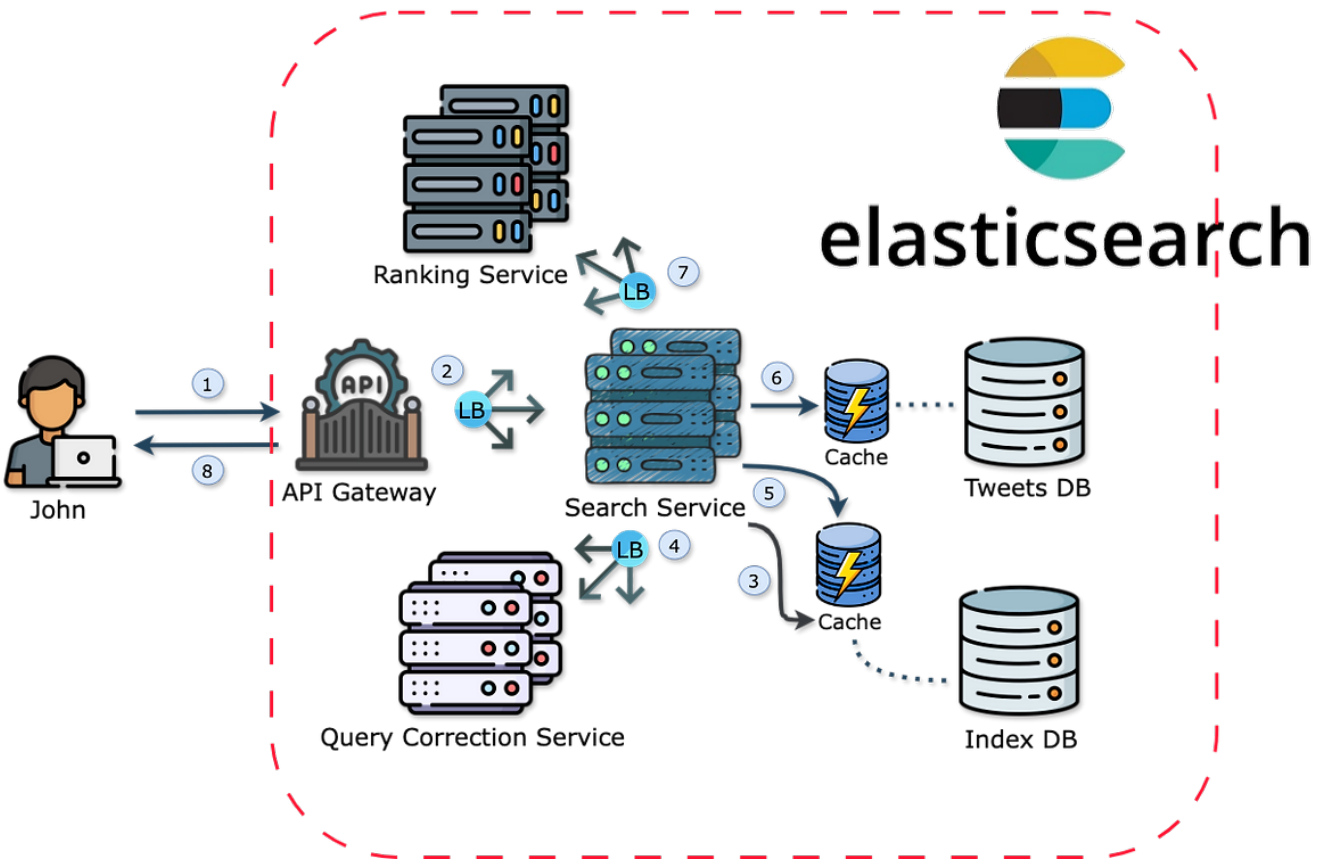
Query Correction Service (Fery to Fare)



Query Correction Service (Fery to Fair)

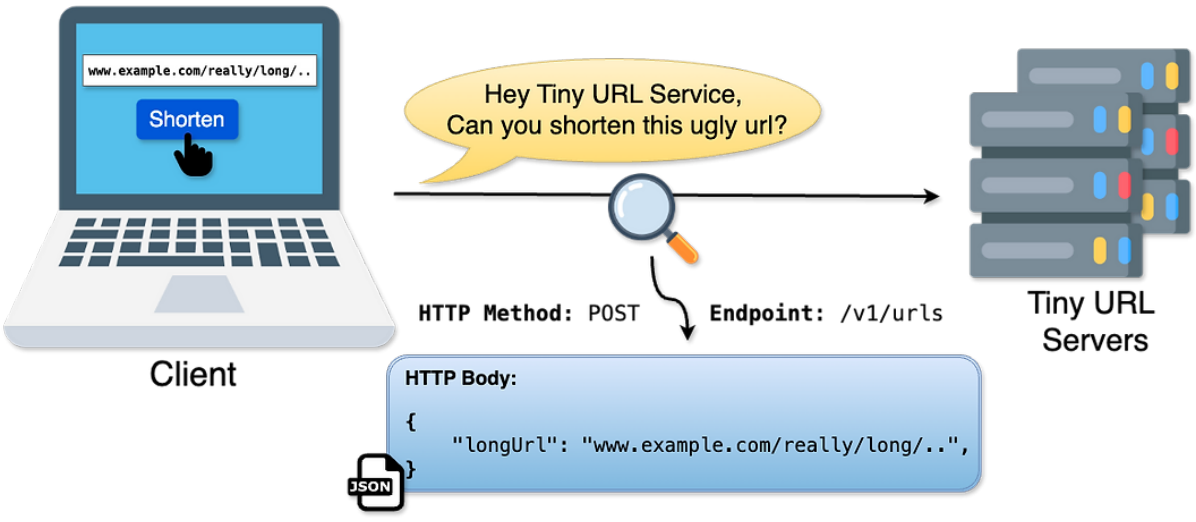


Elastic Search

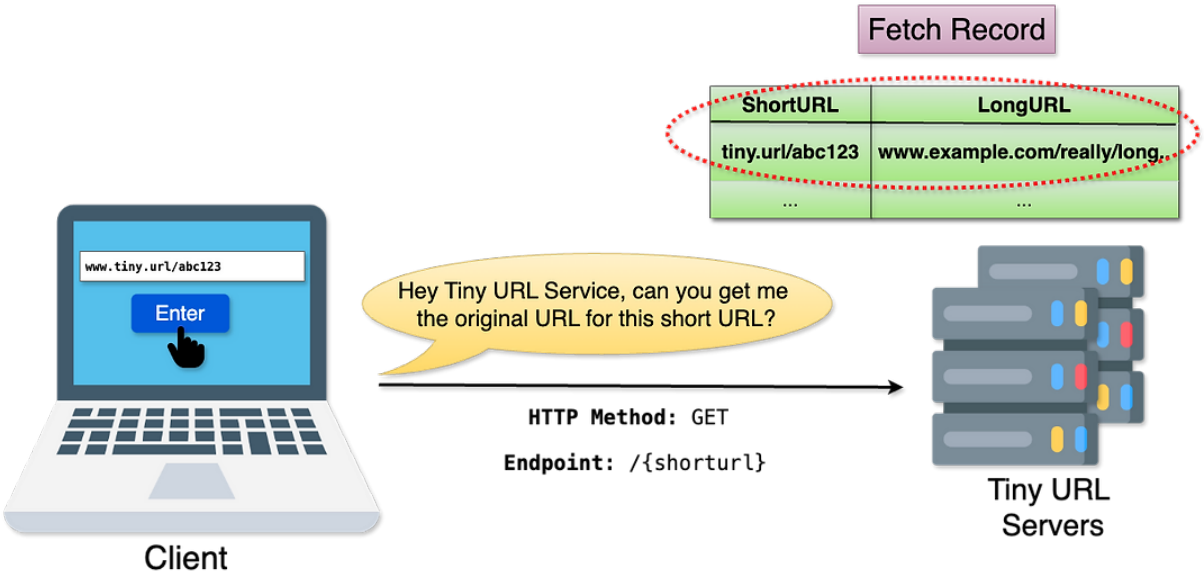


What is a Tiny URL?

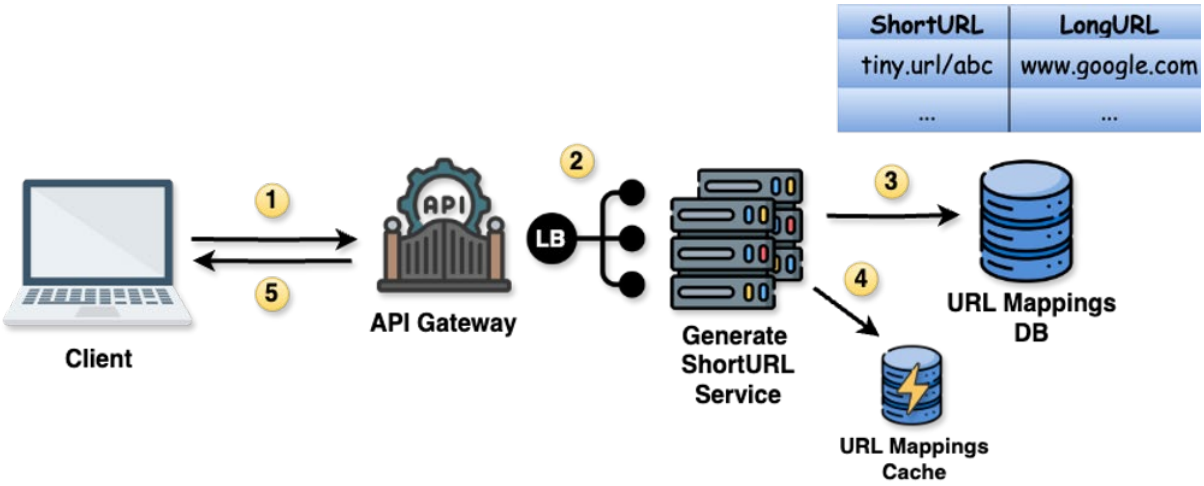
API Design- Generate Short URL



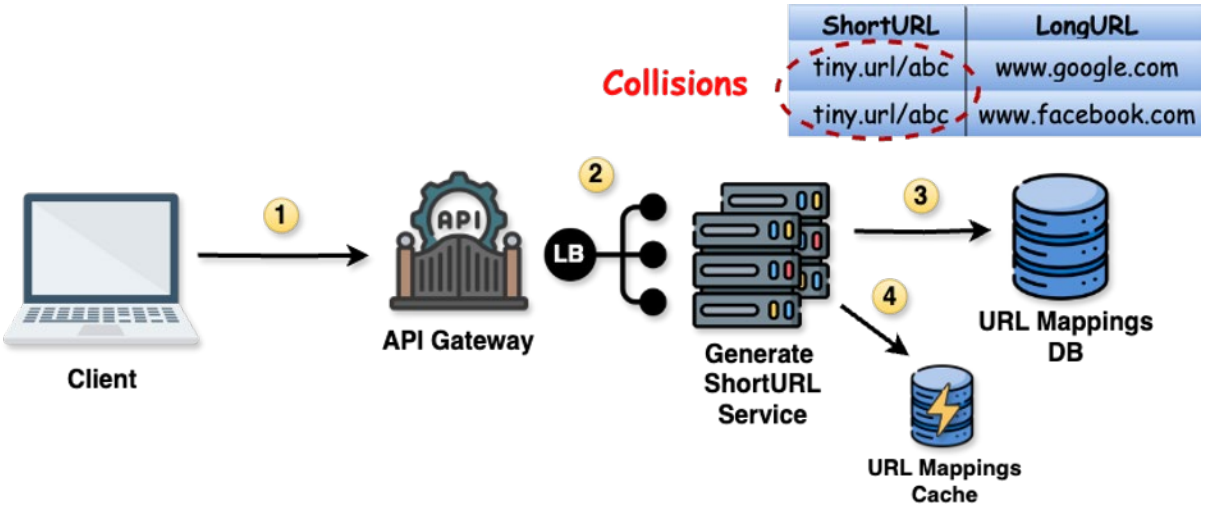
API Design- Get Long URL



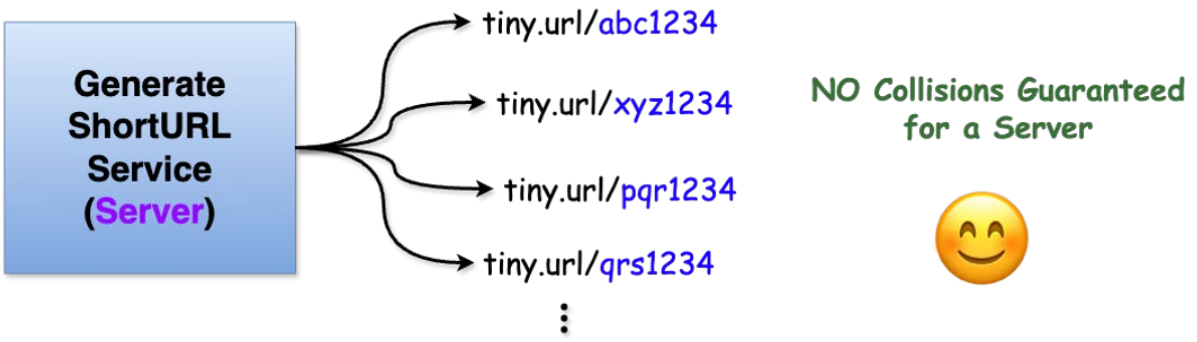
High Level Design-Generate Short URL

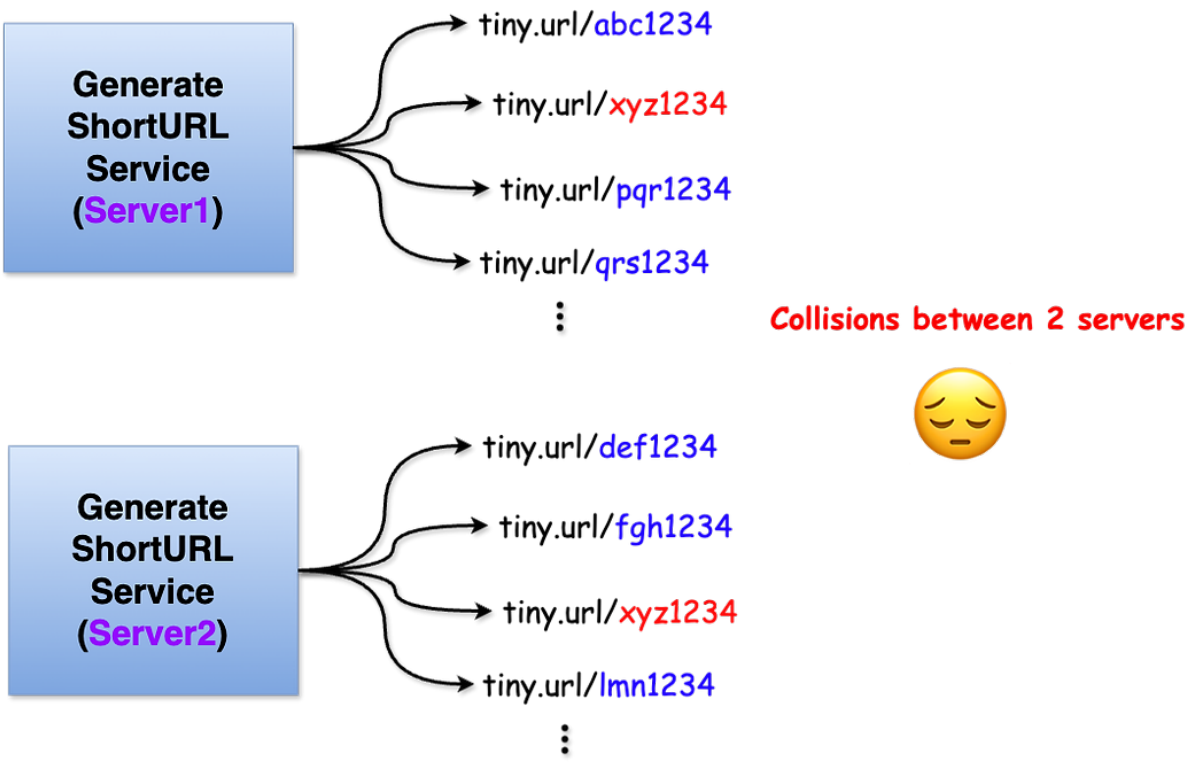


High Level Design-Collison's

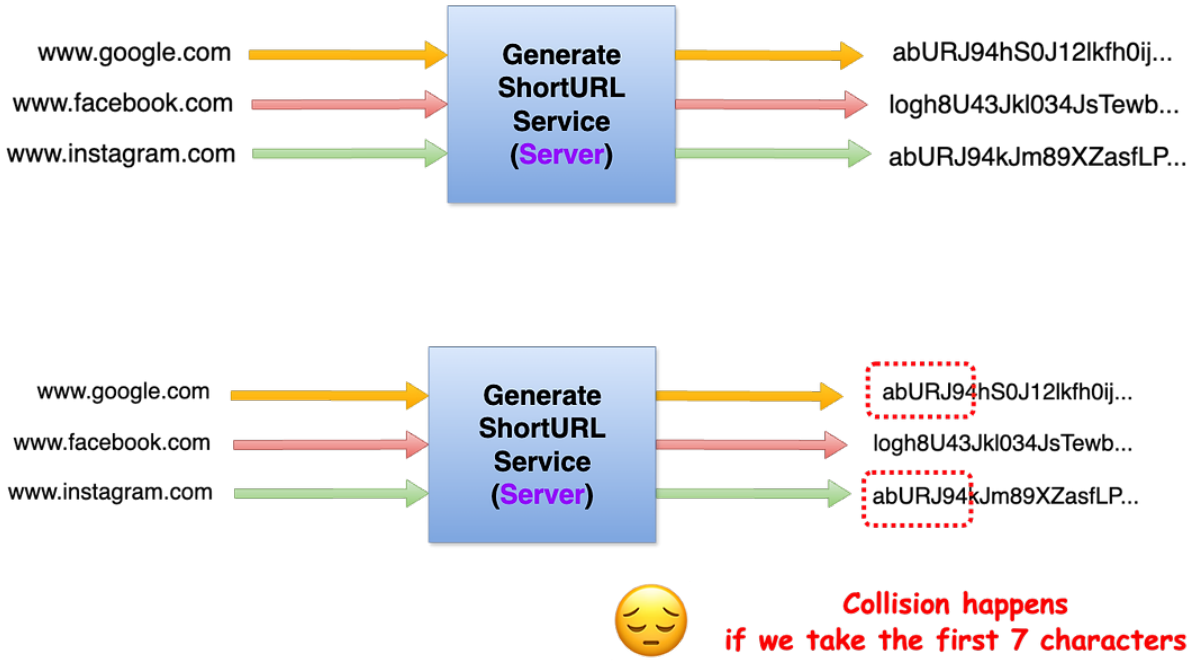


High Level Design-Approach1(Random String Generation)

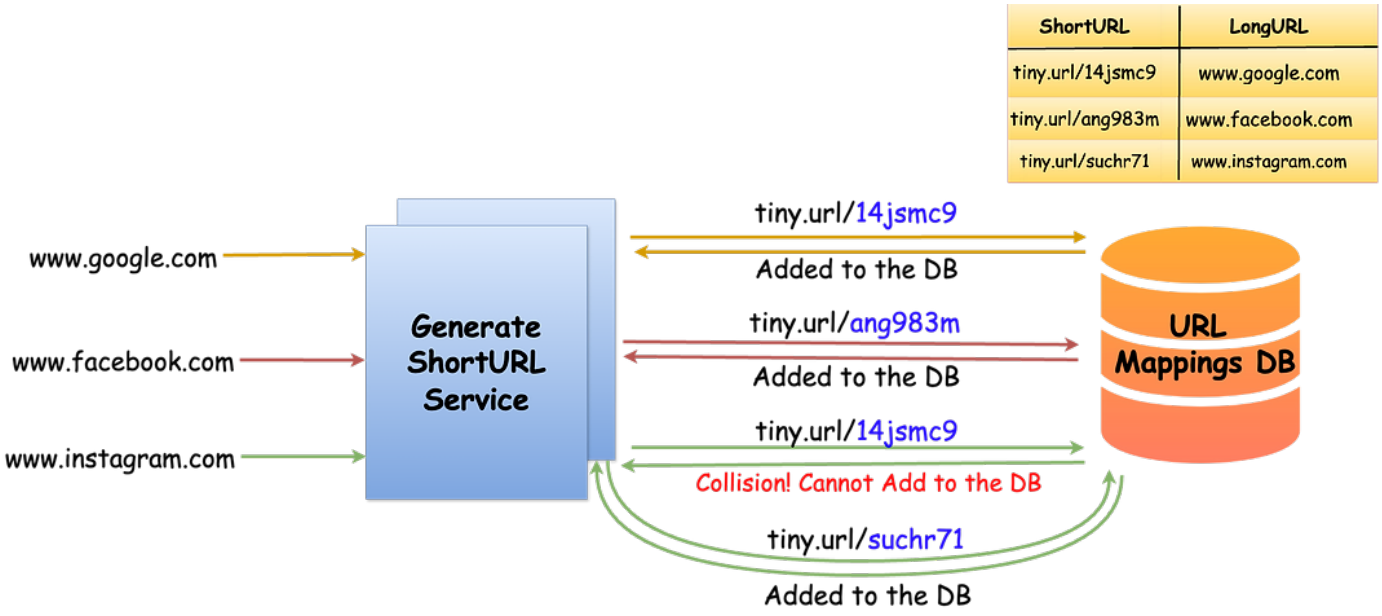




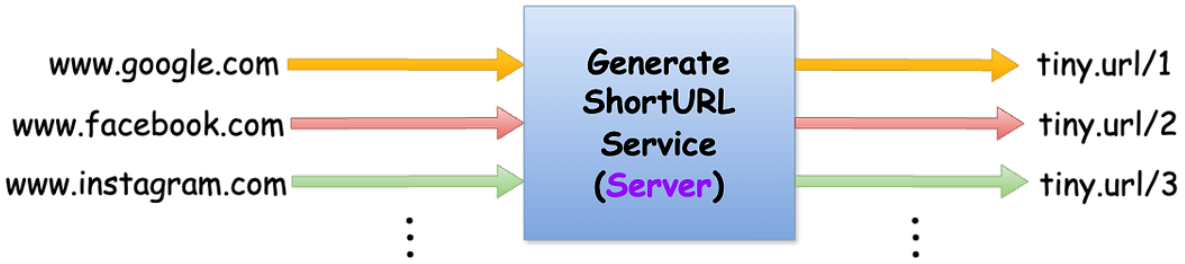
High Level Design- Approach 2: Random String Generation using Long URL

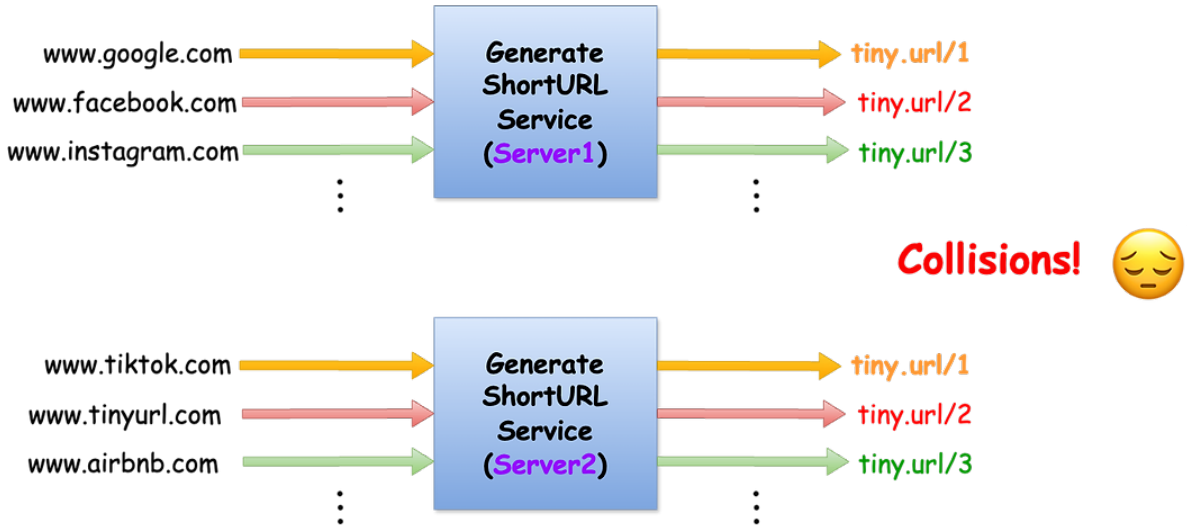


High Level Design- Approach 3: Check DB for Collisions

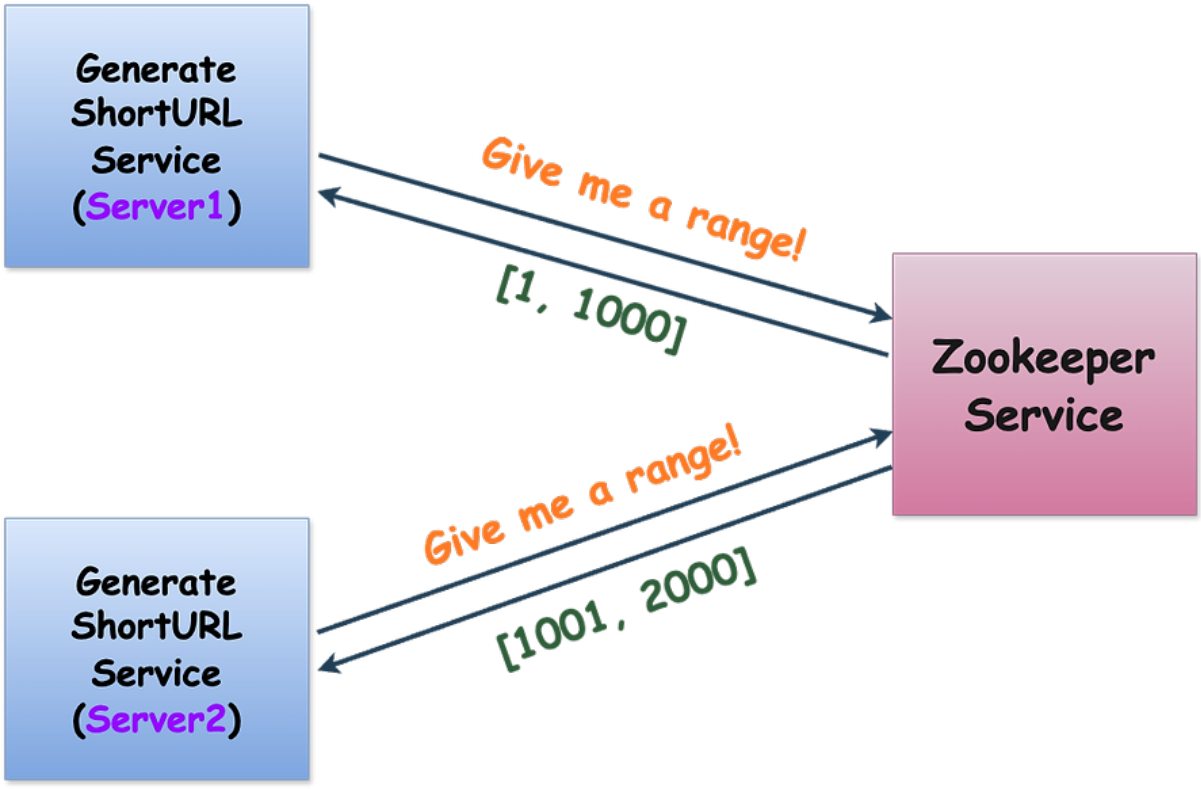


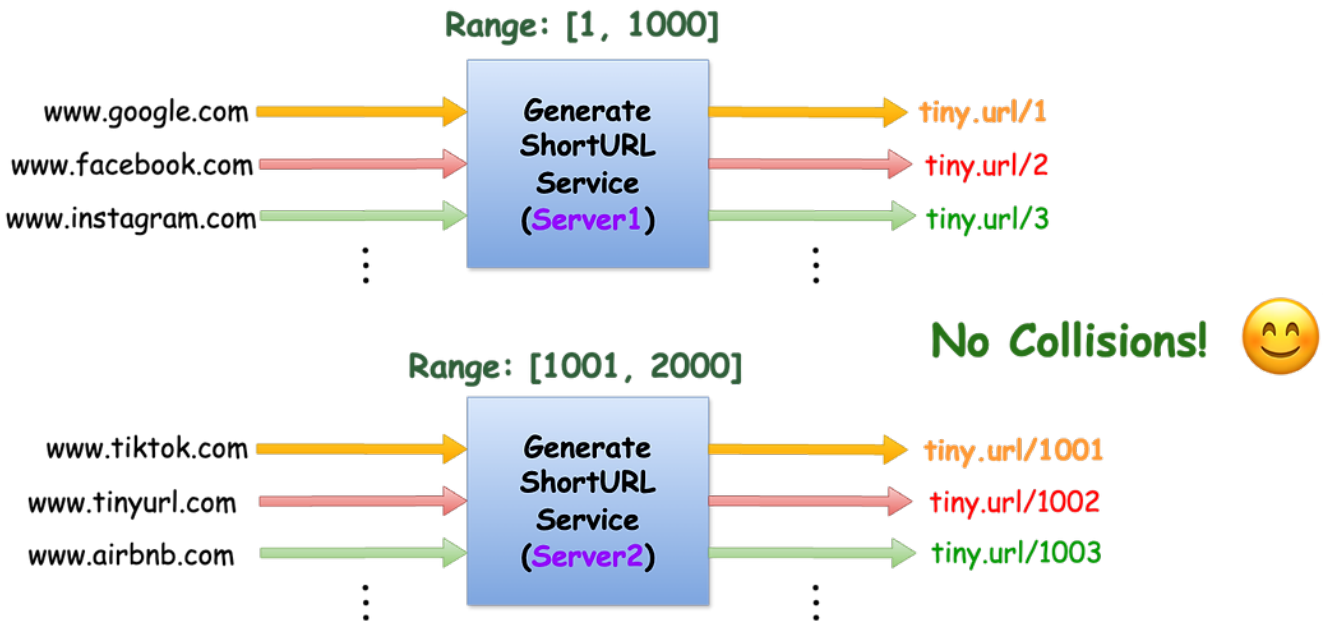
High Level Design- Approach 4: Let's Keep Counters





High Level Design- Approach 4: Zookeeper

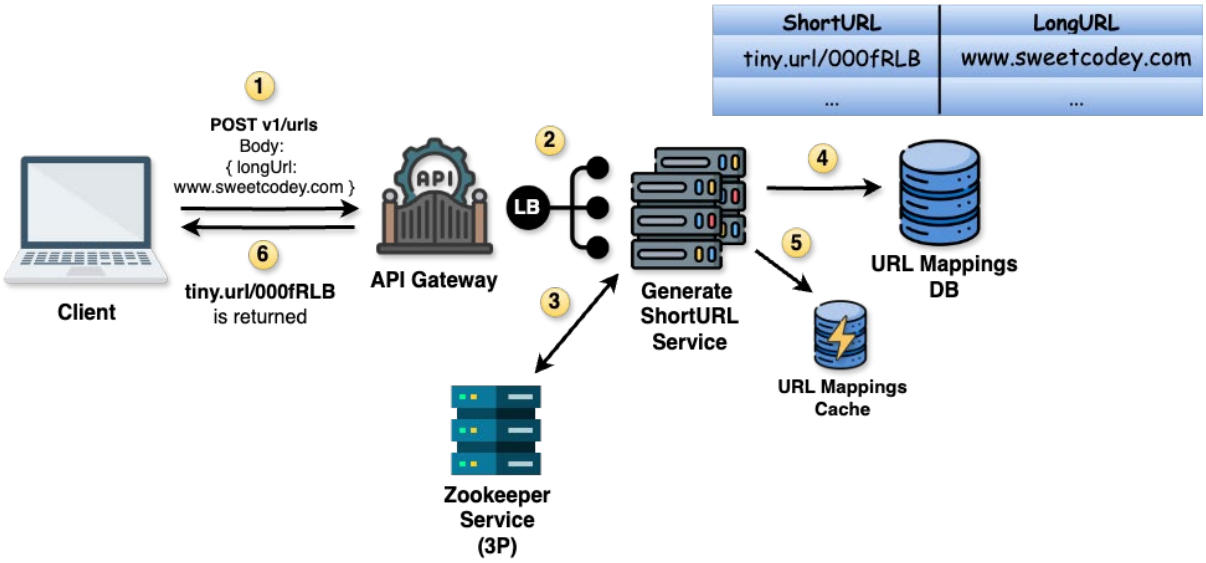




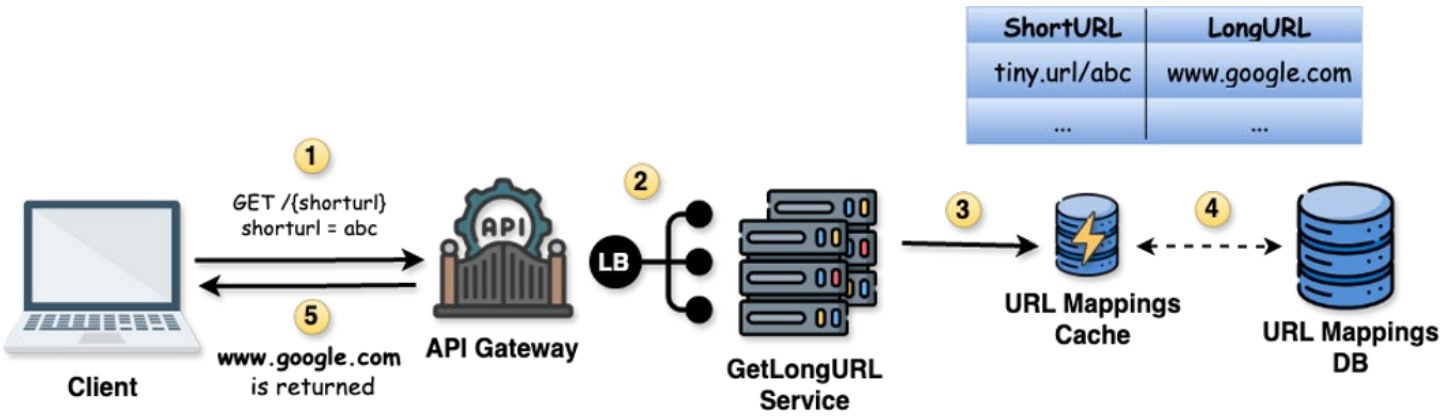
High Level Design- Base-62 Table

Decimal	Binary	Base62	Decimal	Binary	Base62	Decimal	Binary	Base62	Decimal	Binary	Base62
0	000000	0	16	010000	G	32	100000	W	48	110000	m
1	000001	1	17	010001	H	33	100001	X	49	110001	n
2	000010	2	18	010010	I	34	100010	Y	50	110010	o
3	000011	3	19	010011	J	35	100011	Z	51	110011	p
4	000100	4	20	010100	K	36	100100	a	52	110100	q
5	000101	5	21	010101	L	37	100101	b	53	110101	r
6	000110	6	22	010110	M	38	100110	c	54	110110	s
7	000111	7	23	010111	N	39	100111	d	55	110111	t
8	001000	8	24	011000	O	40	101000	e	56	111000	u
9	001001	9	25	011001	P	41	101001	f	57	111001	v
10	001010	A	26	011010	Q	42	101010	g	58	111010	w
11	001011	B	27	011011	R	43	101011	h	59	111011	x
12	001100	C	28	011100	S	44	101100	i	60	111100	y
13	001101	D	29	011101	T	45	101101	j	61	111101	z
14	001110	E	30	011110	U	46	101110	k			
15	001111	F	31	011111	V	47	101111	l			

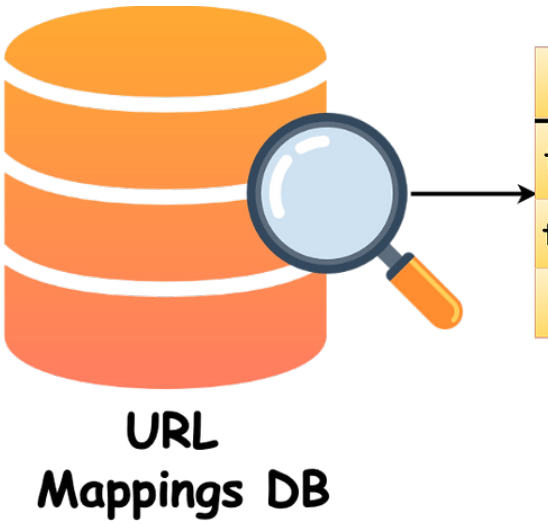
High Level Design-Approach:4-Final Design Diagram



High Level Design- Get Long URL

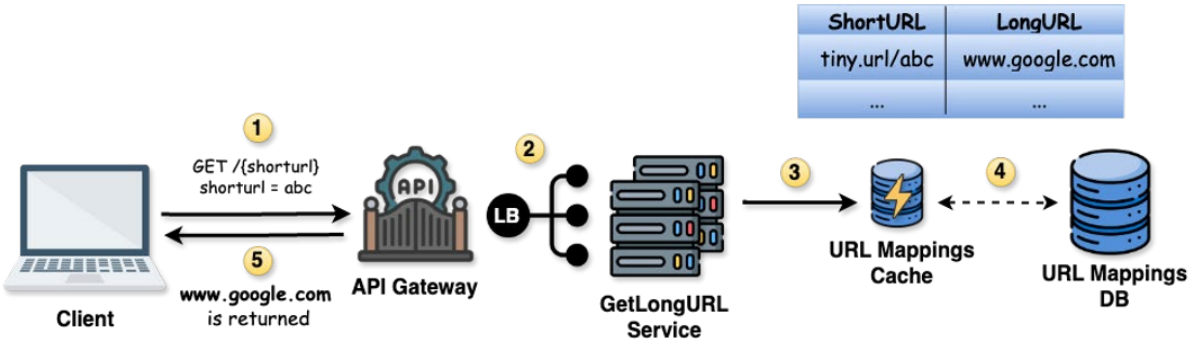


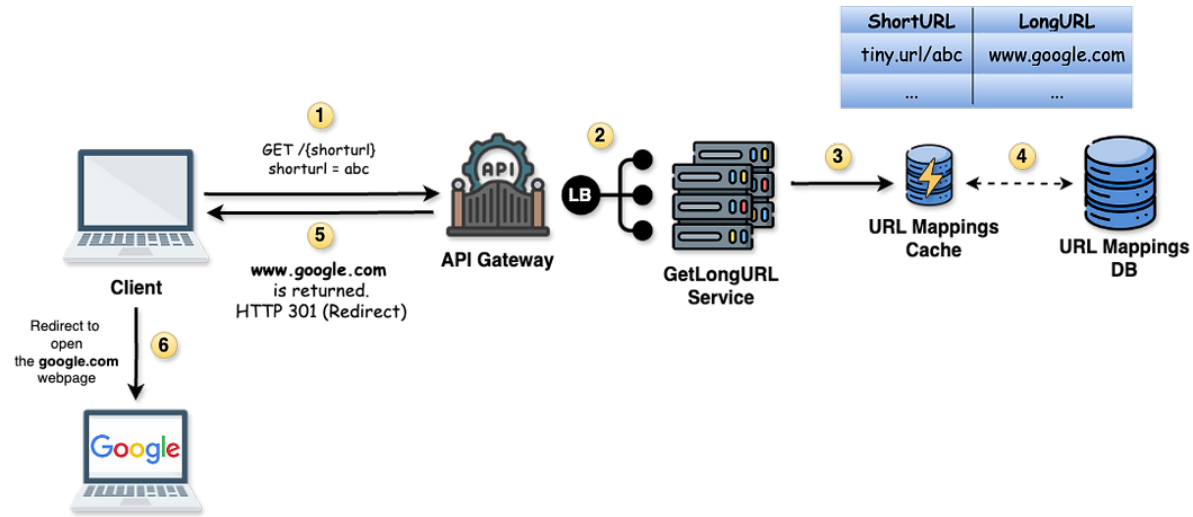
URL Mappings Schema



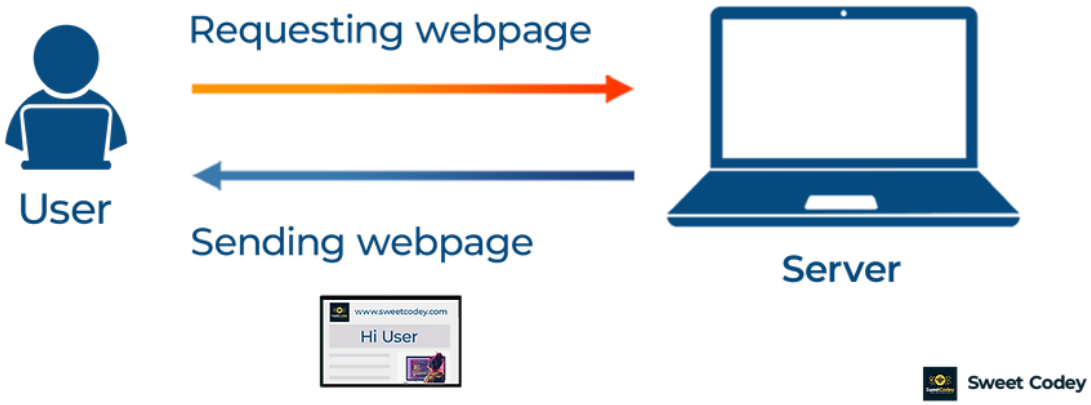
ShortURL	LongURL
tiny.url/00000G9	www.tiktok.com
tiny.url/00000GA	www.twitter.com
tiny.url/000fRLB	www.sweetcodey.com

Redirection from Short URL

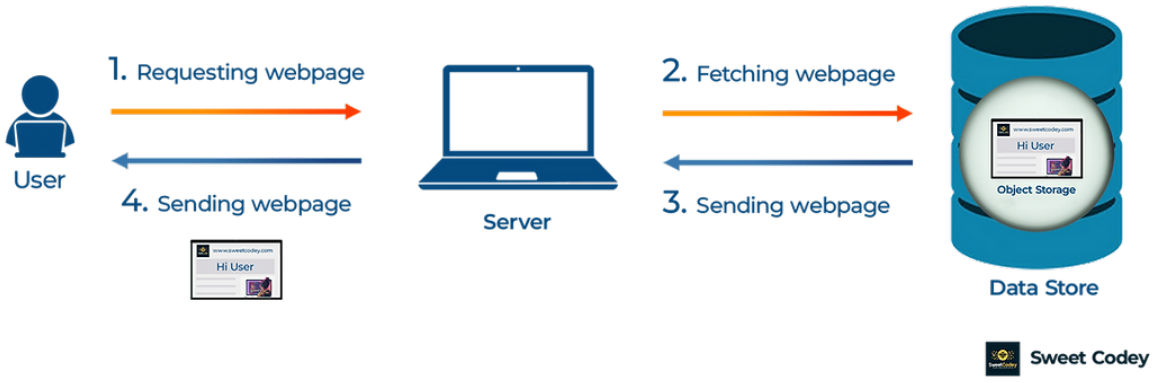




Step 1: Client & Server

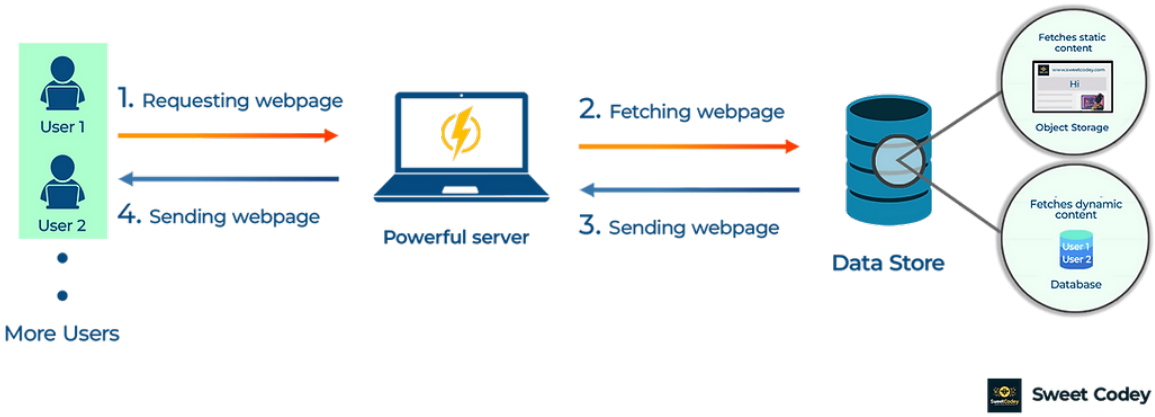


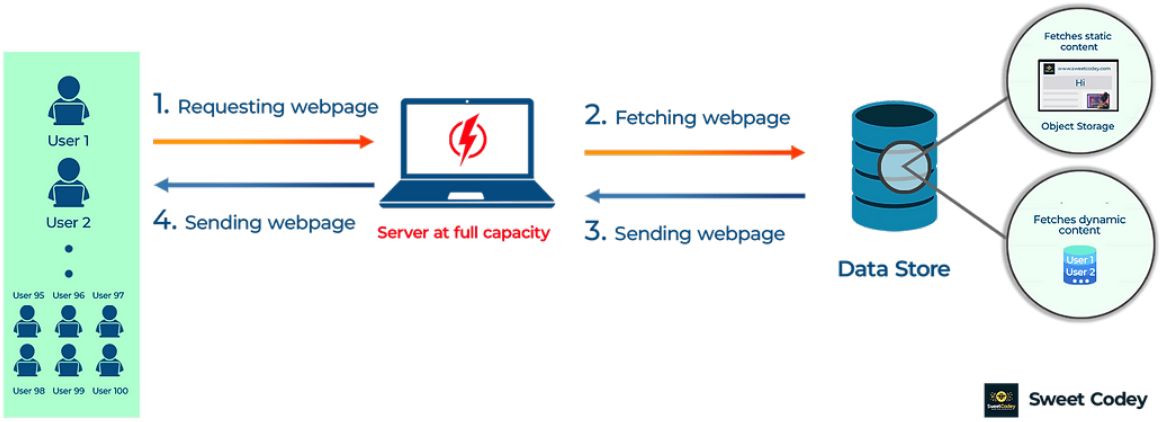
Step 2: Database



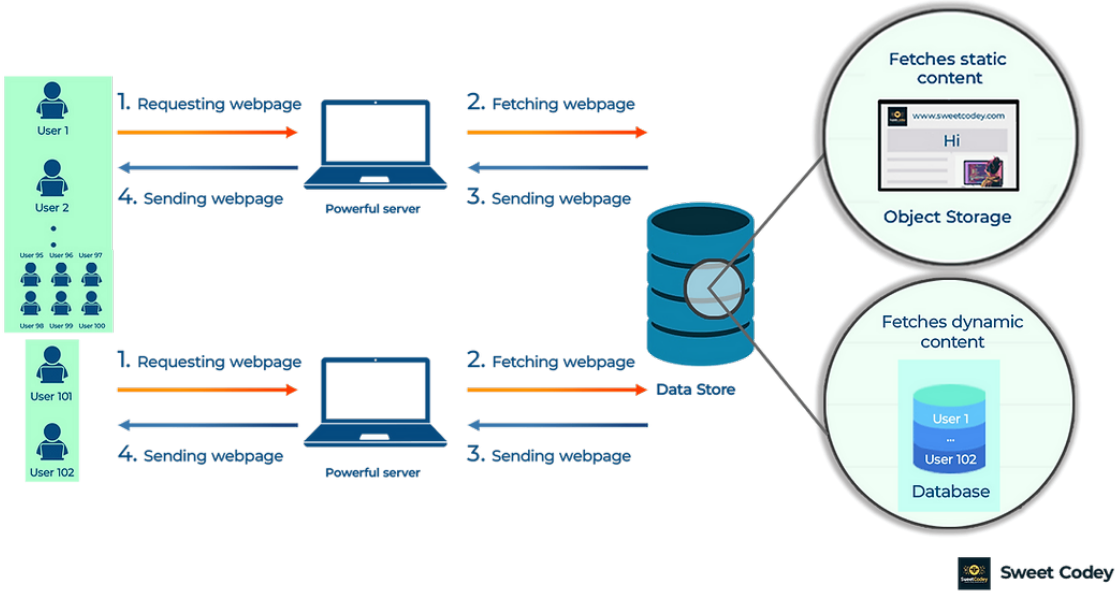
Step 3: Vertical vs Horizontal Scaling

Vertical Scaling

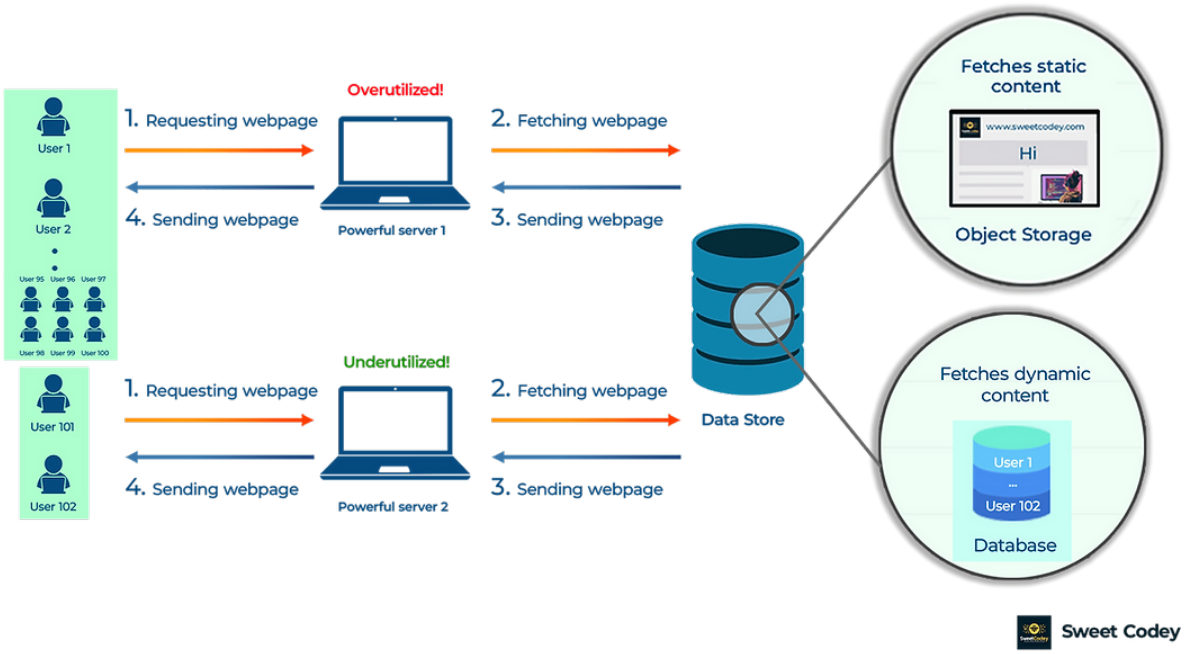


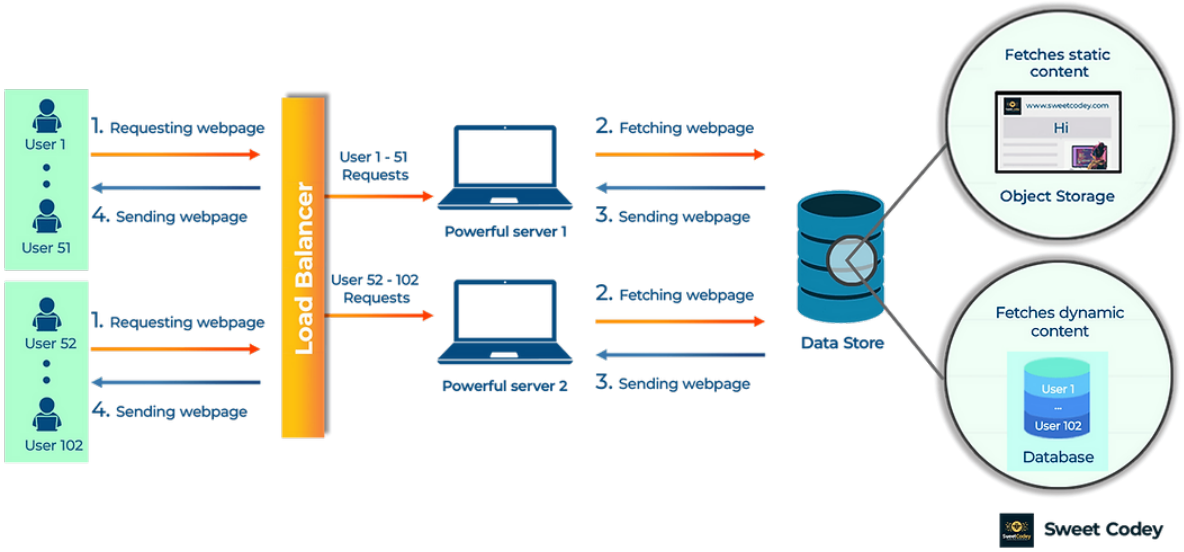


Horizontal Scaling

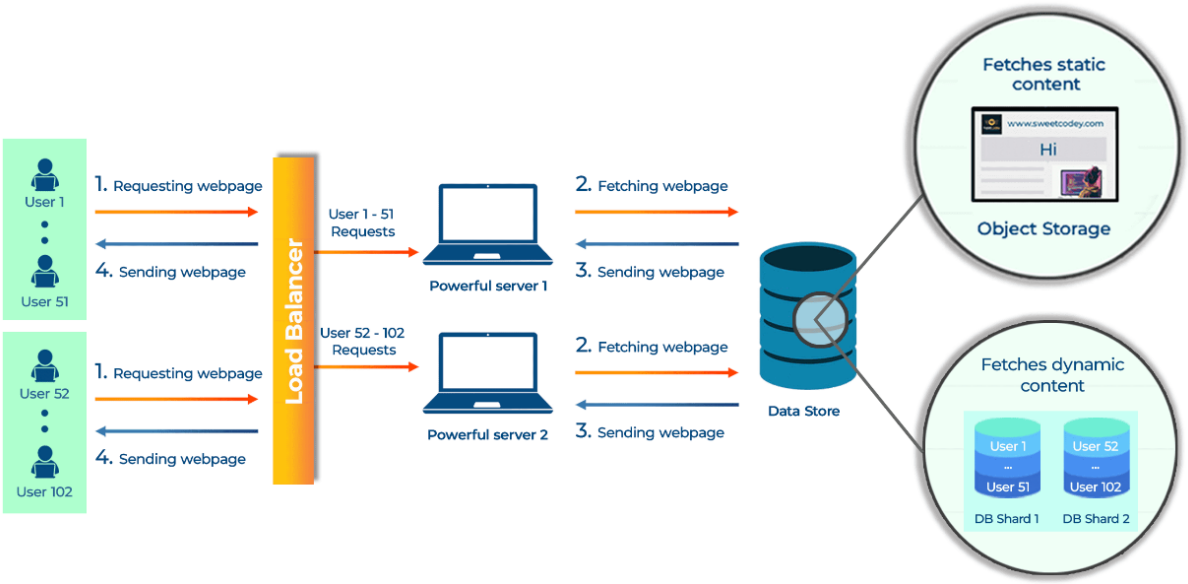


Step 4: Load Balancer

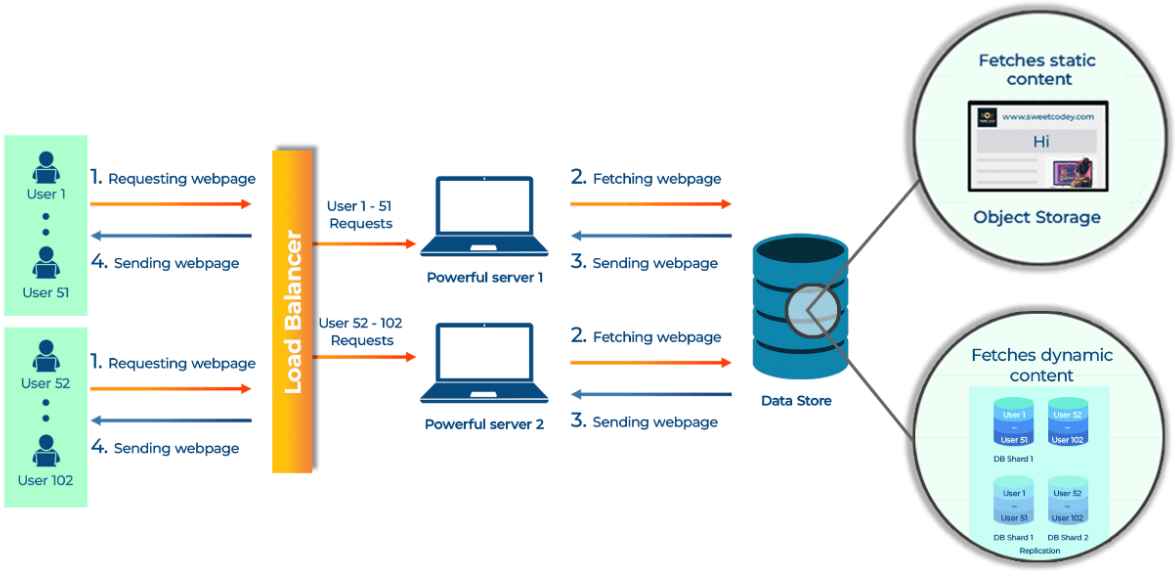




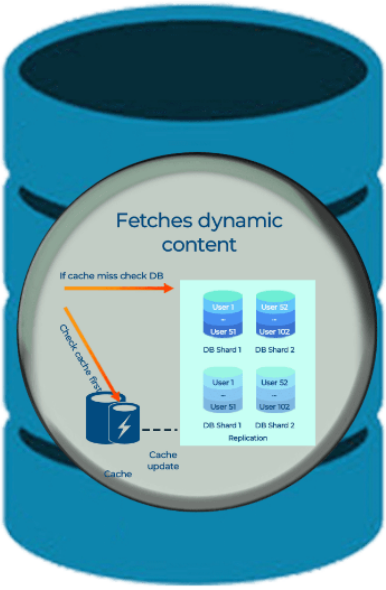
Step 5a: Database Sharding



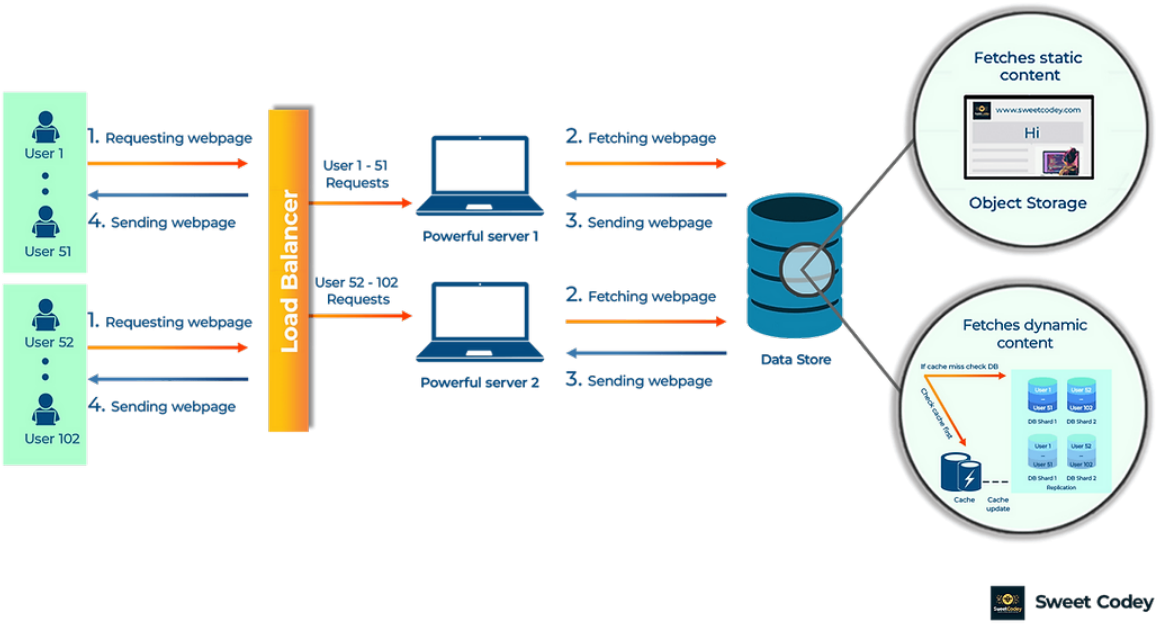
Step 5b: Database Replication



Step 6: Cache

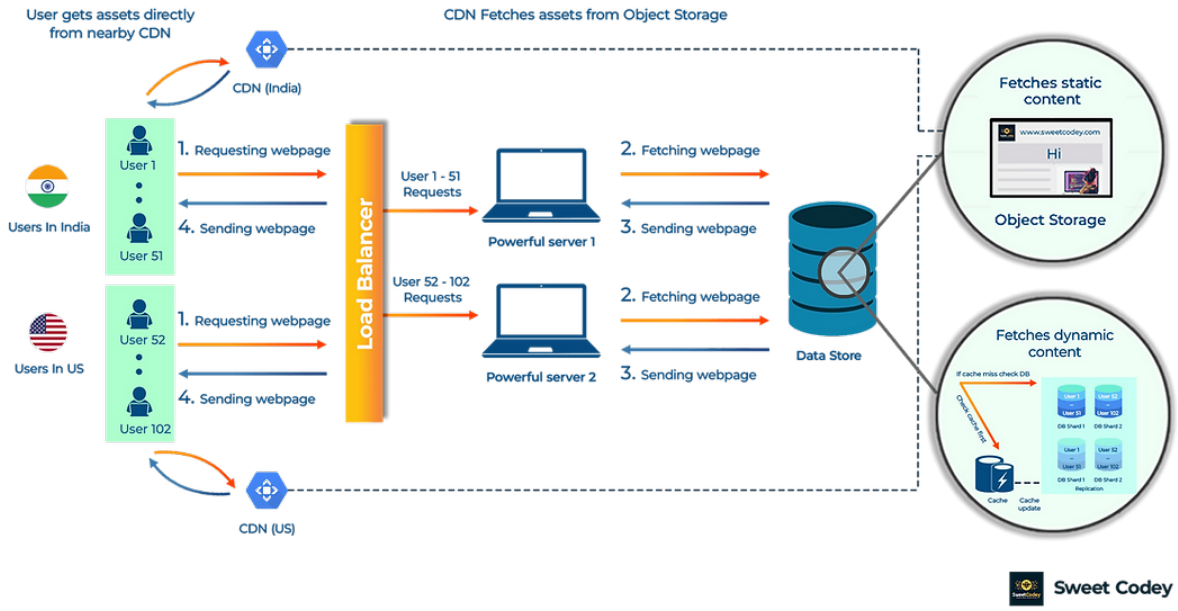


Data Store



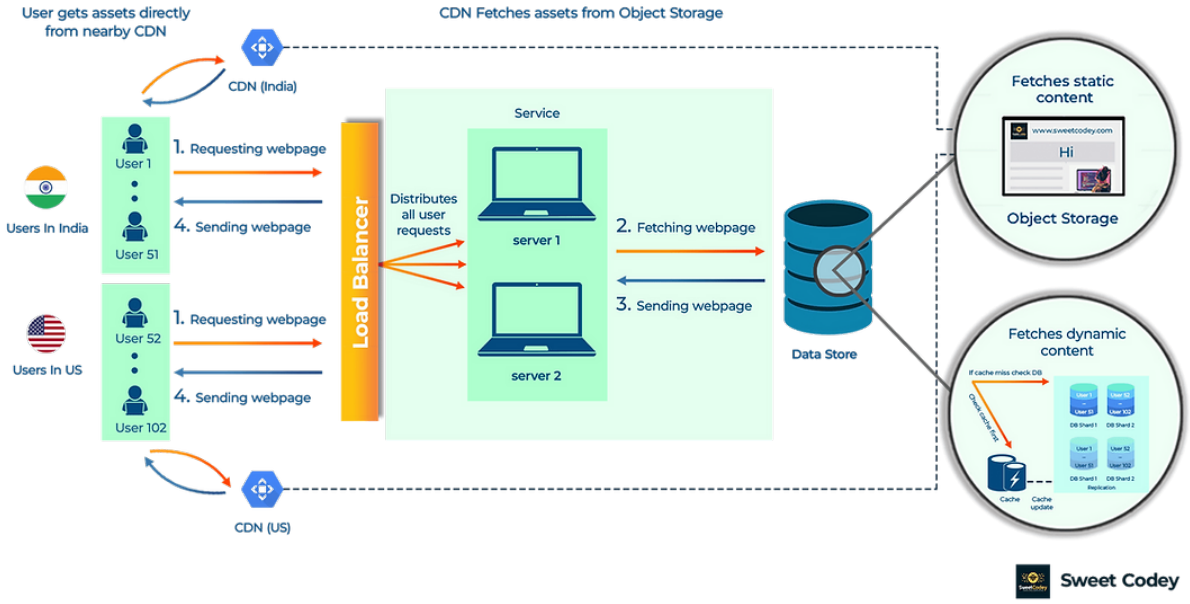
Sweet Codey

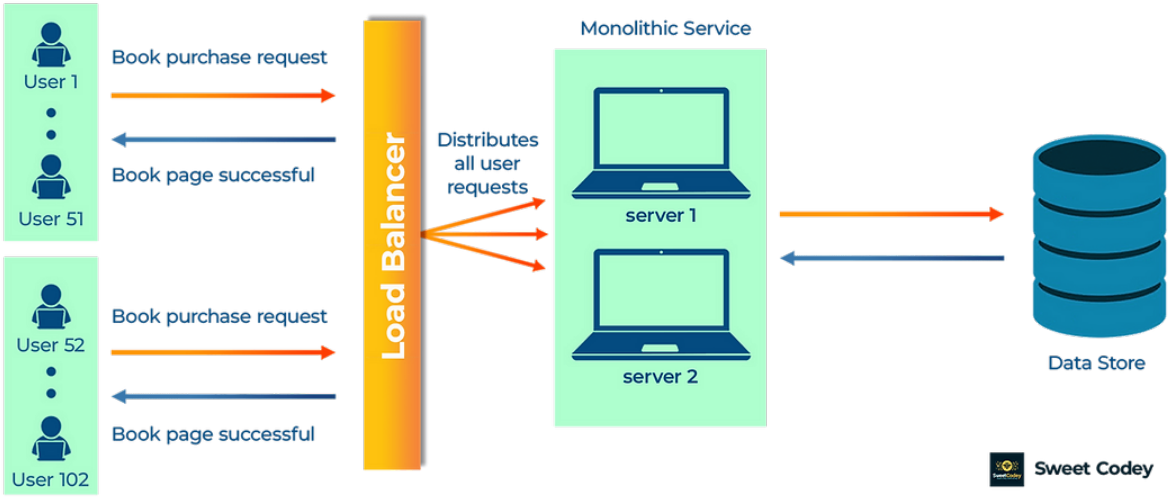
Step 7: Content Delivery Network (CDN)

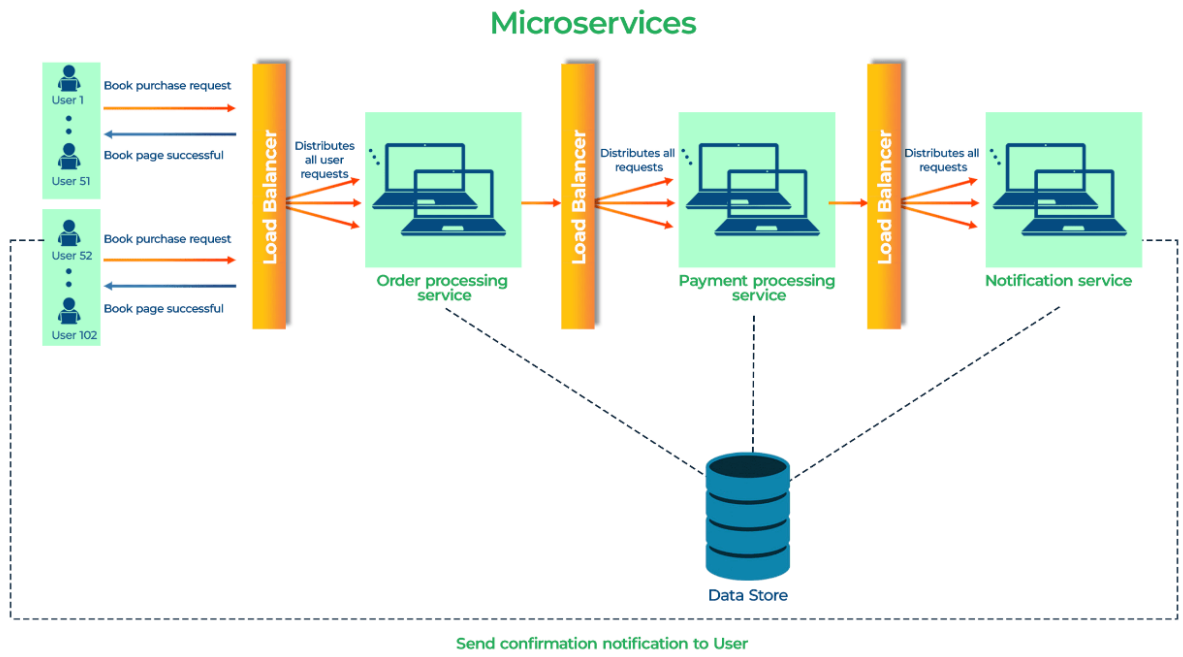


Step 8: Monolith & Microservice

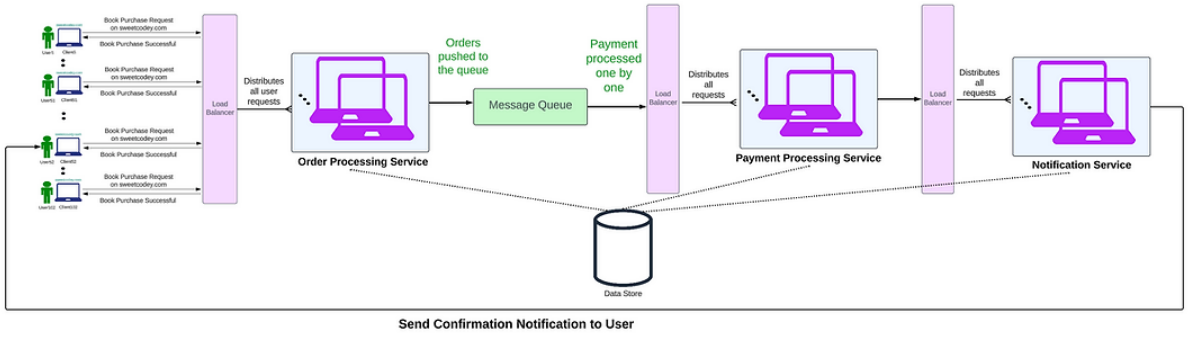
Monolith



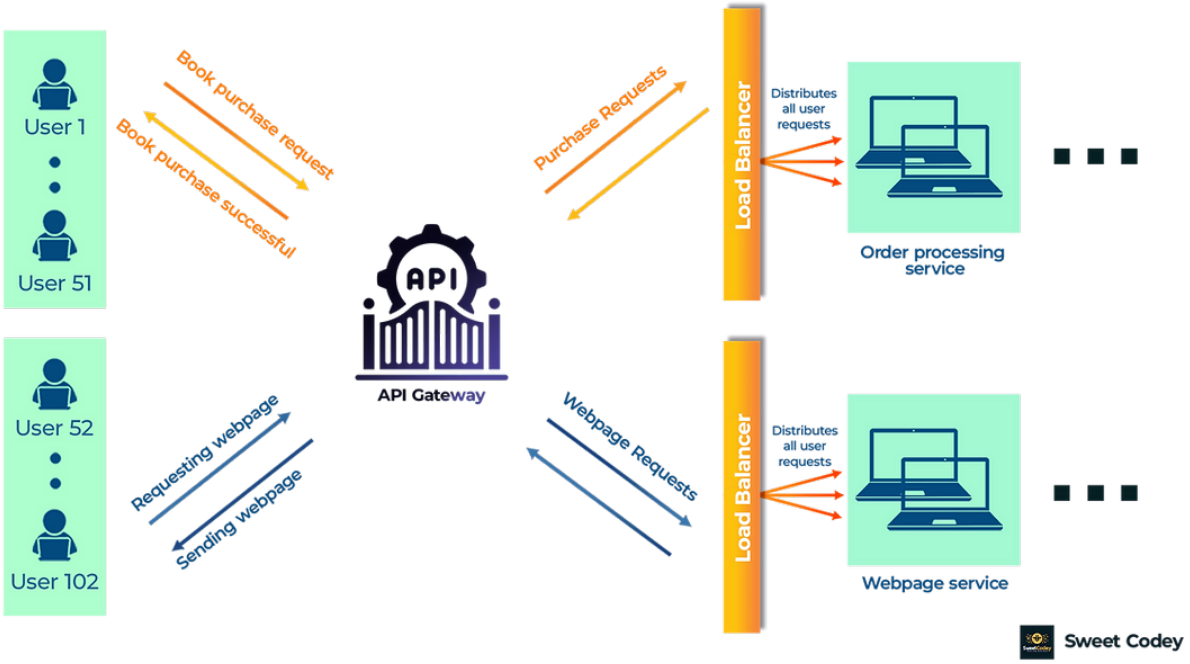




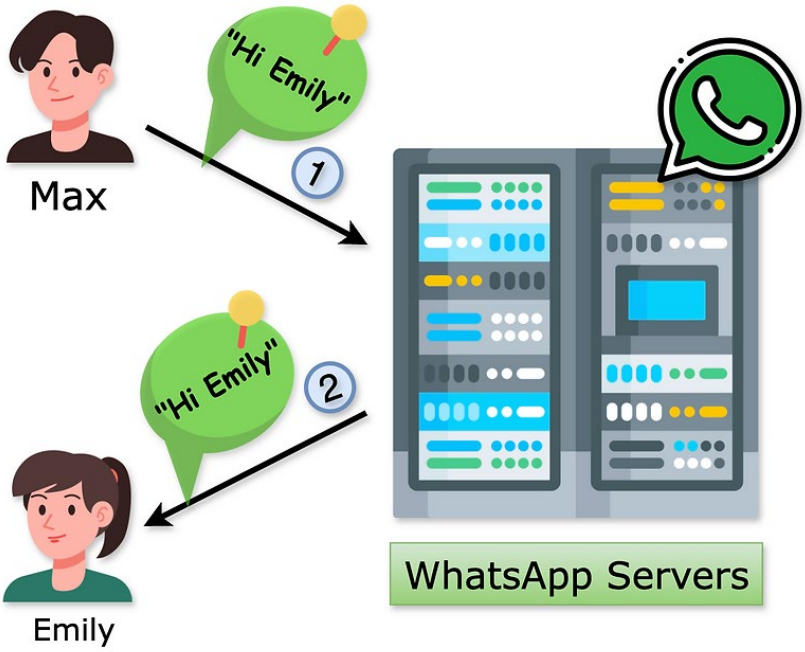
Step 9: Message Queue

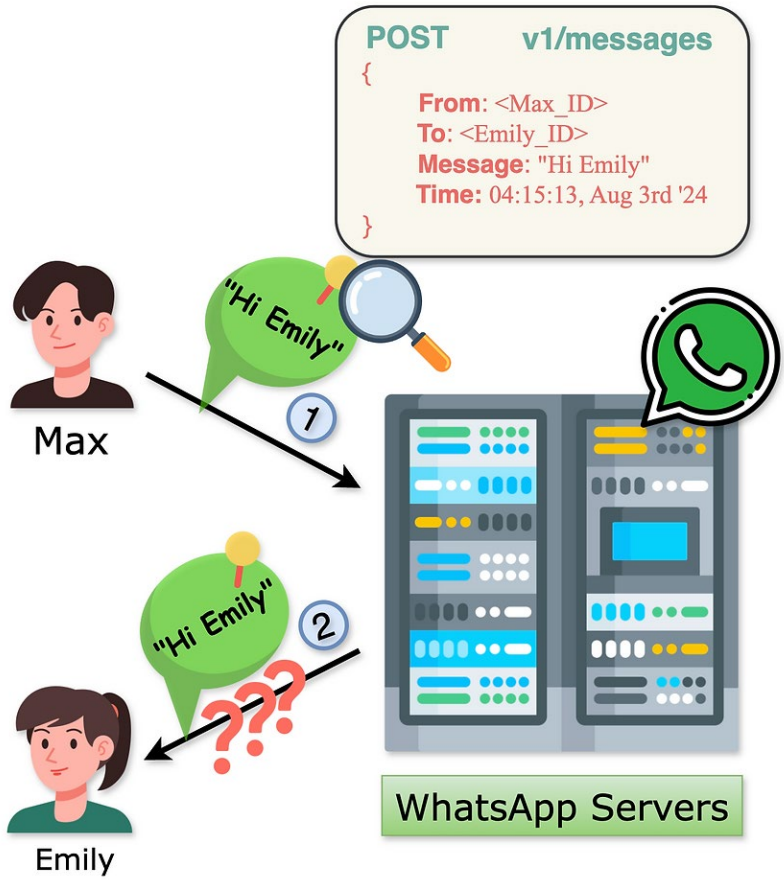


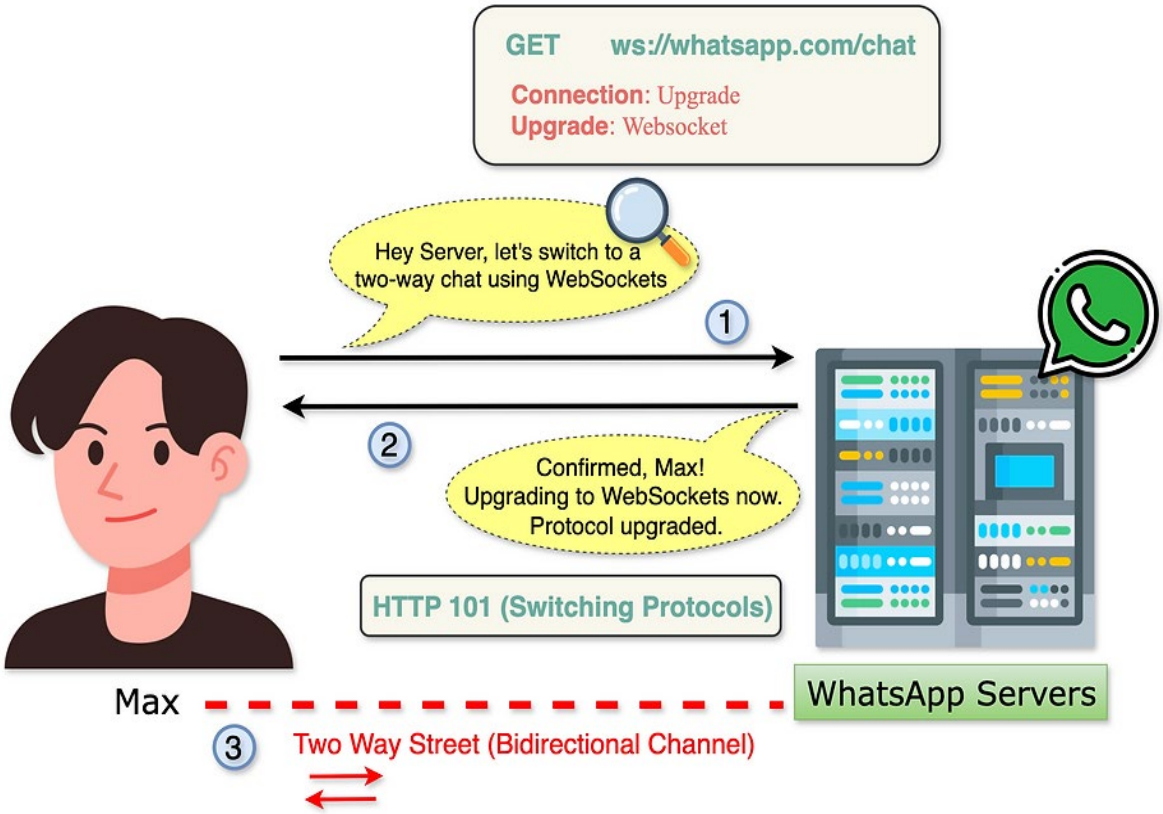
Step 10: API Gateway



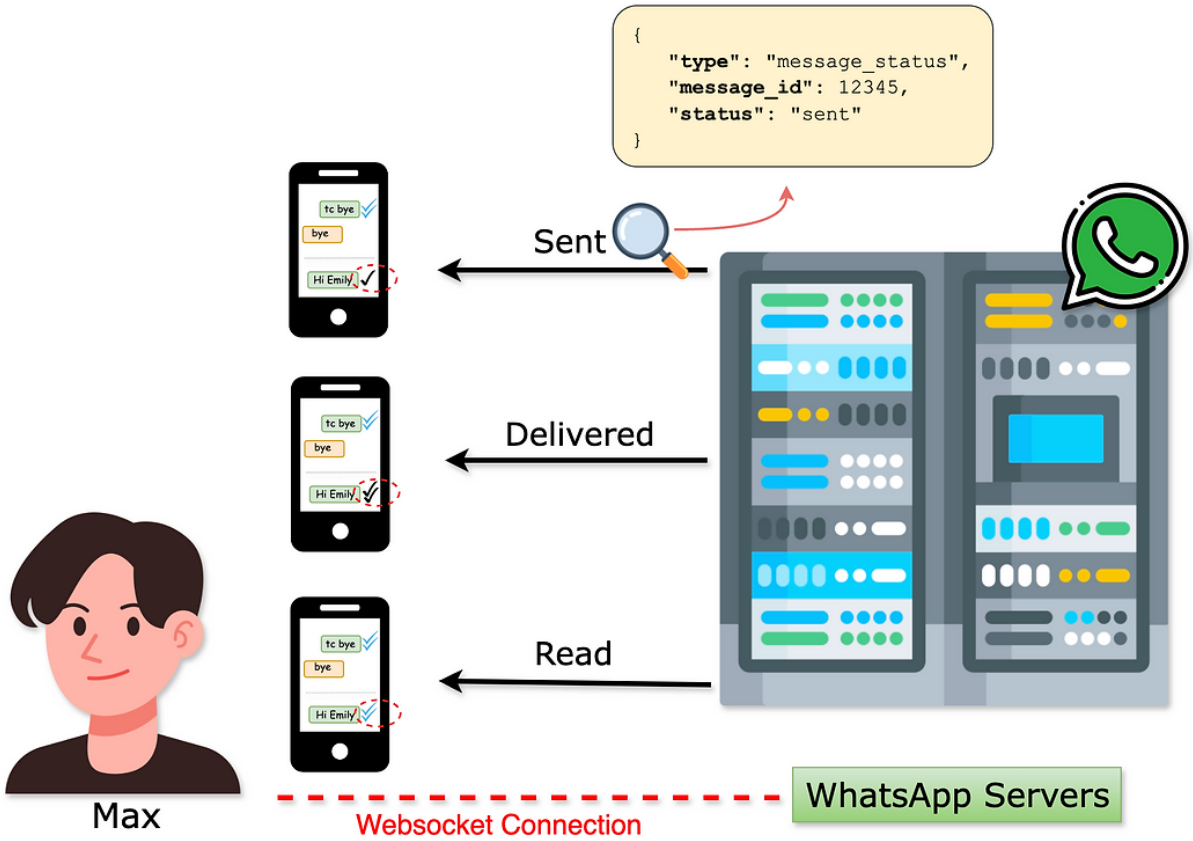
API Design: One-One Messaging



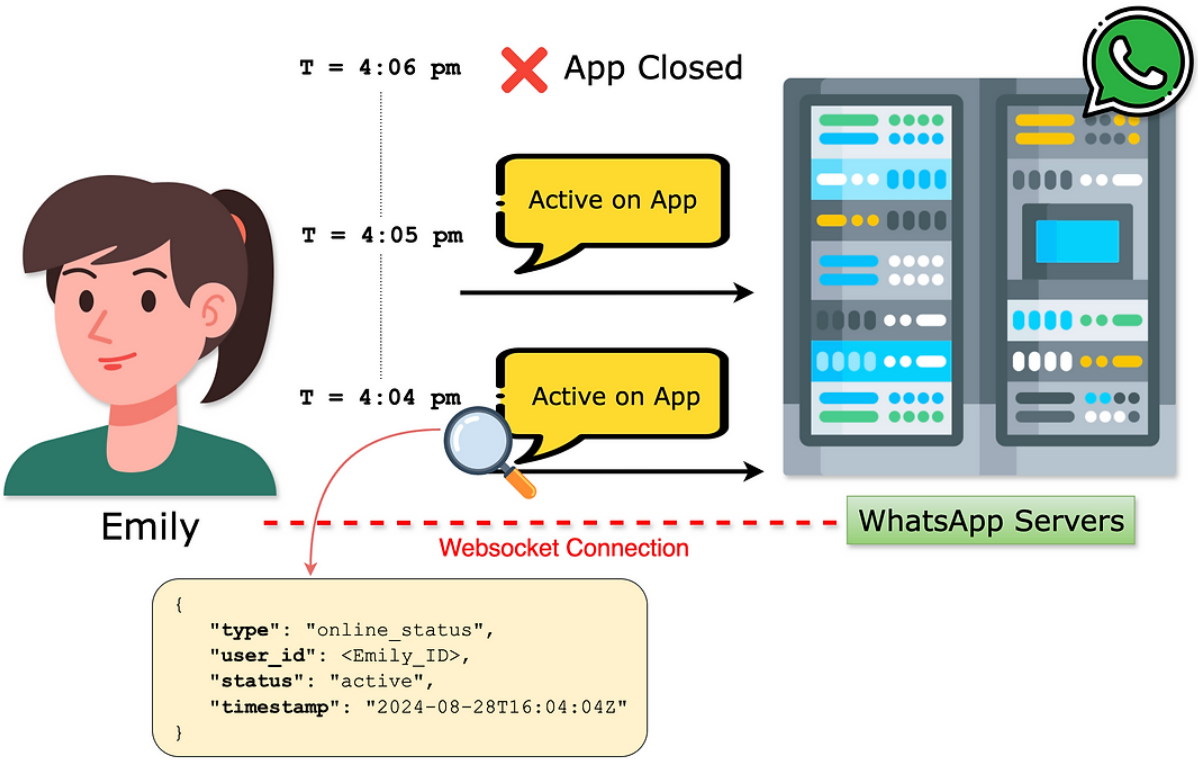


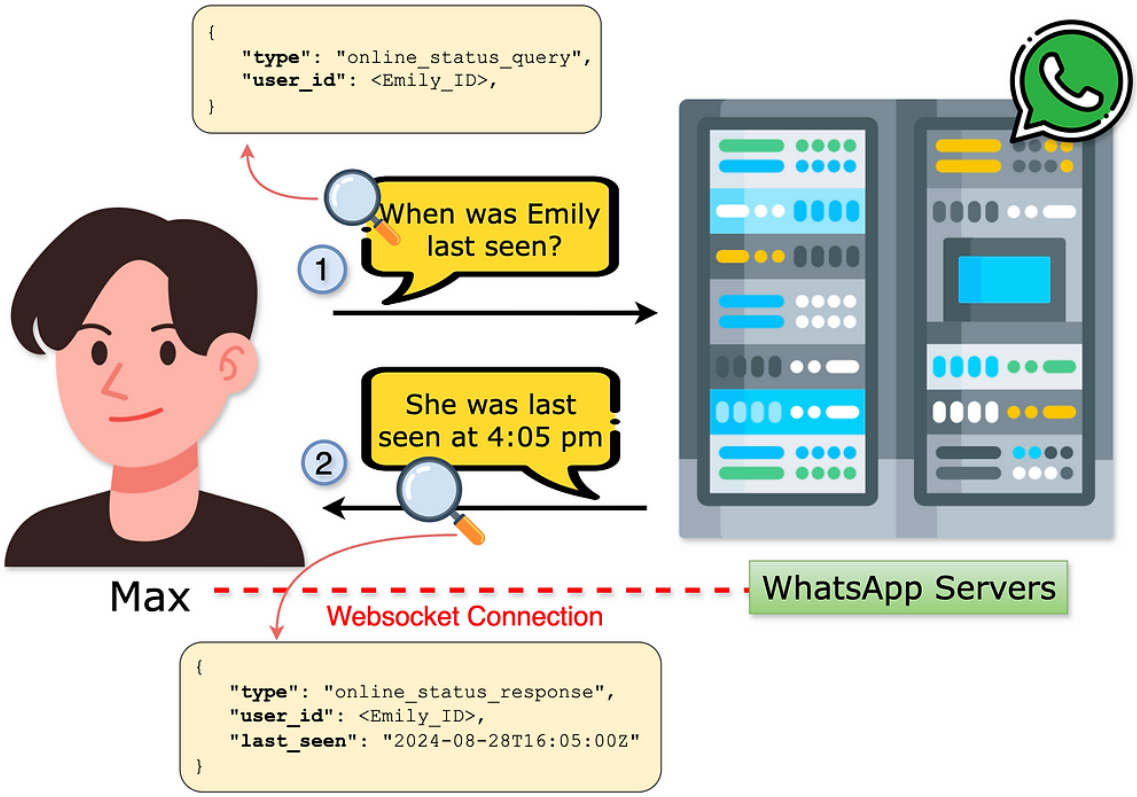


API Design: Message Status

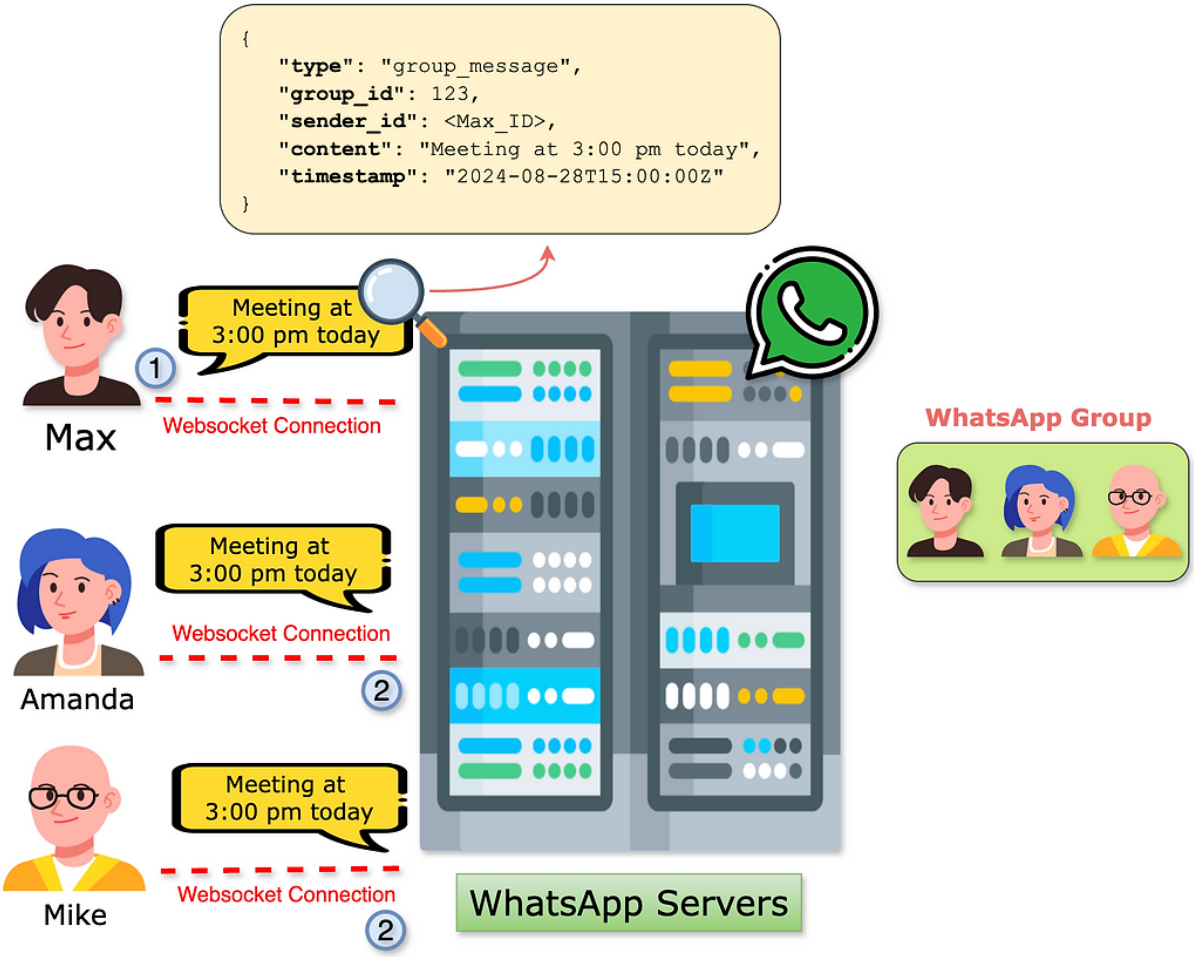


API Design: Online Status

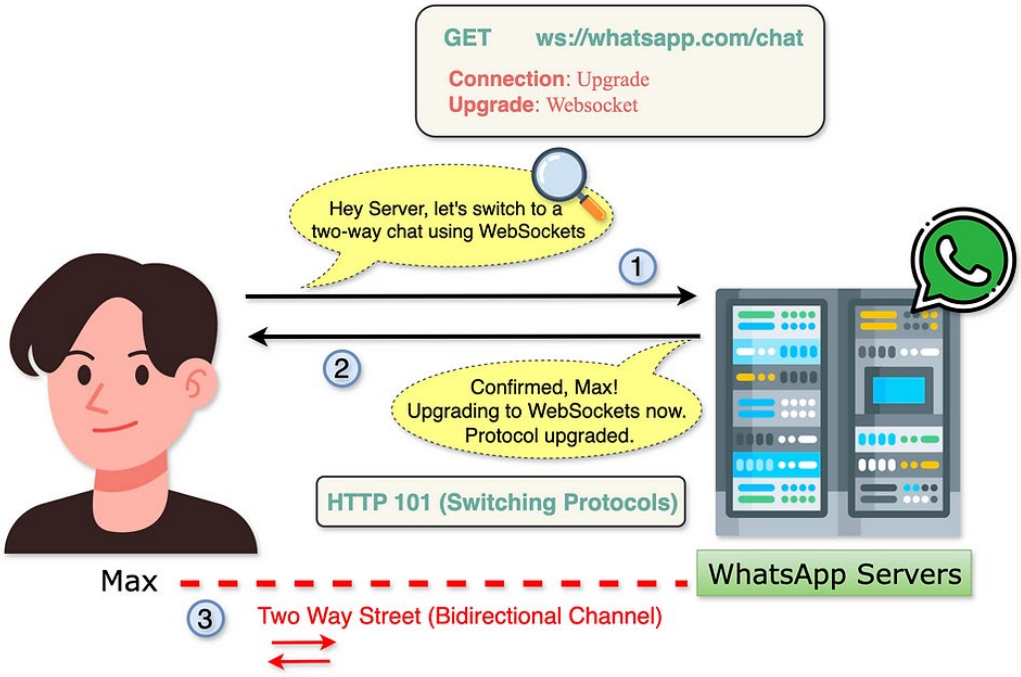


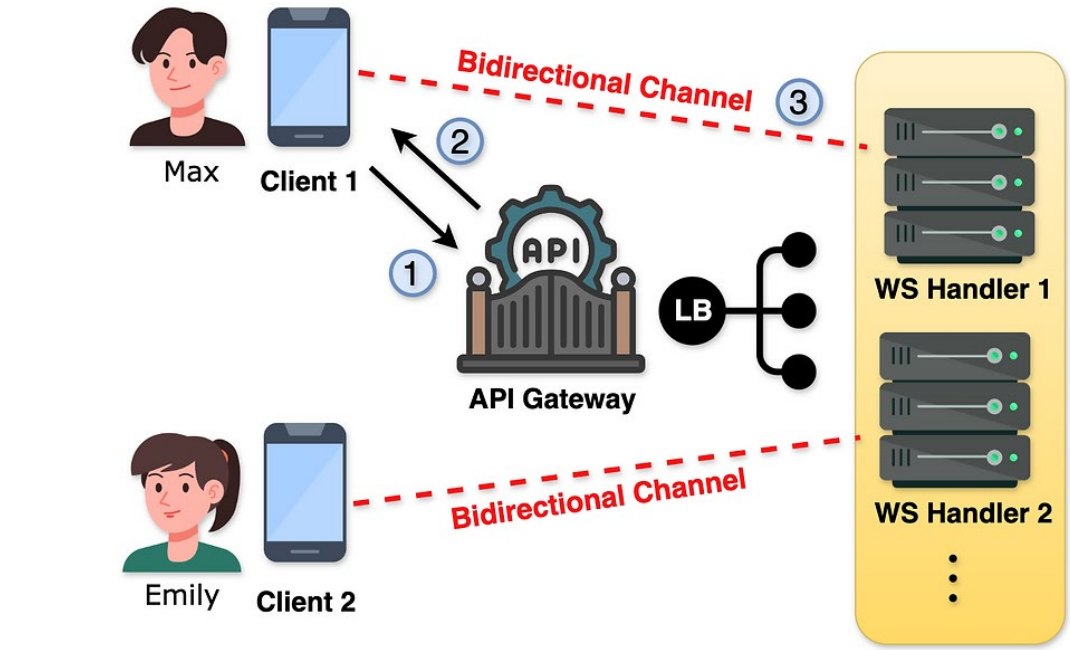


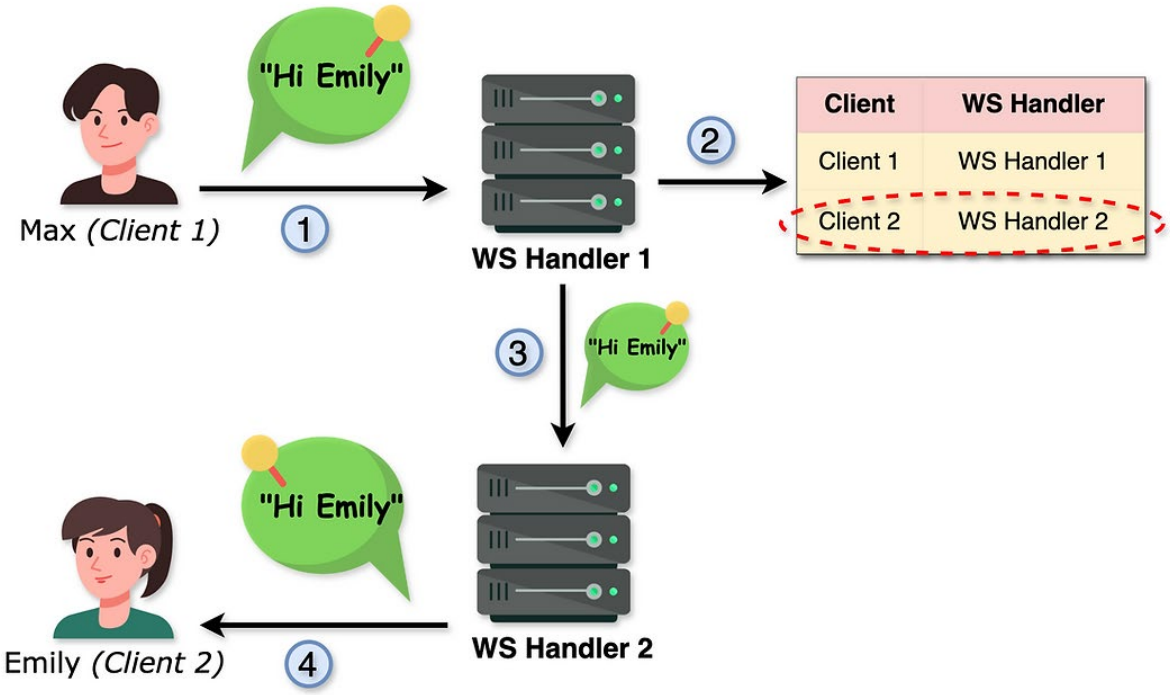
API Design: Group Messaging

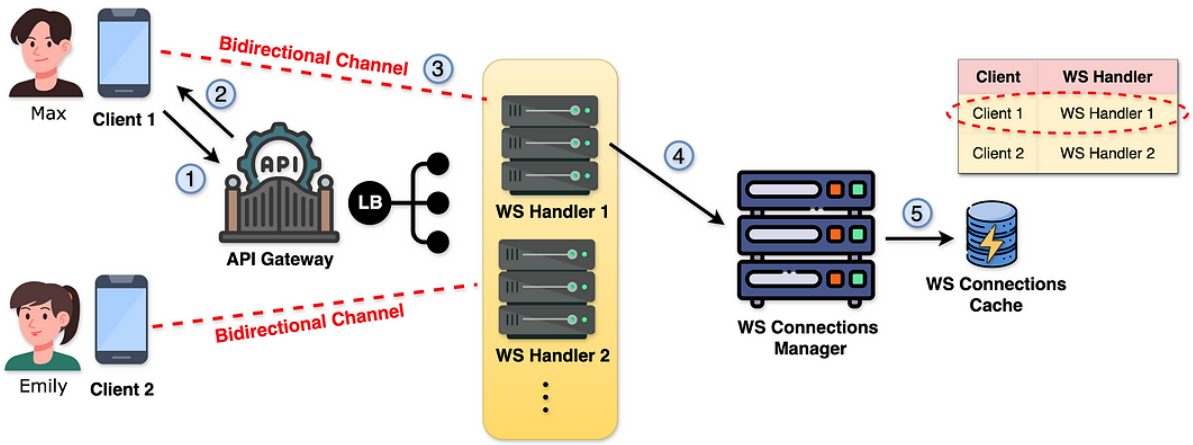


Establishing WebSocket connection

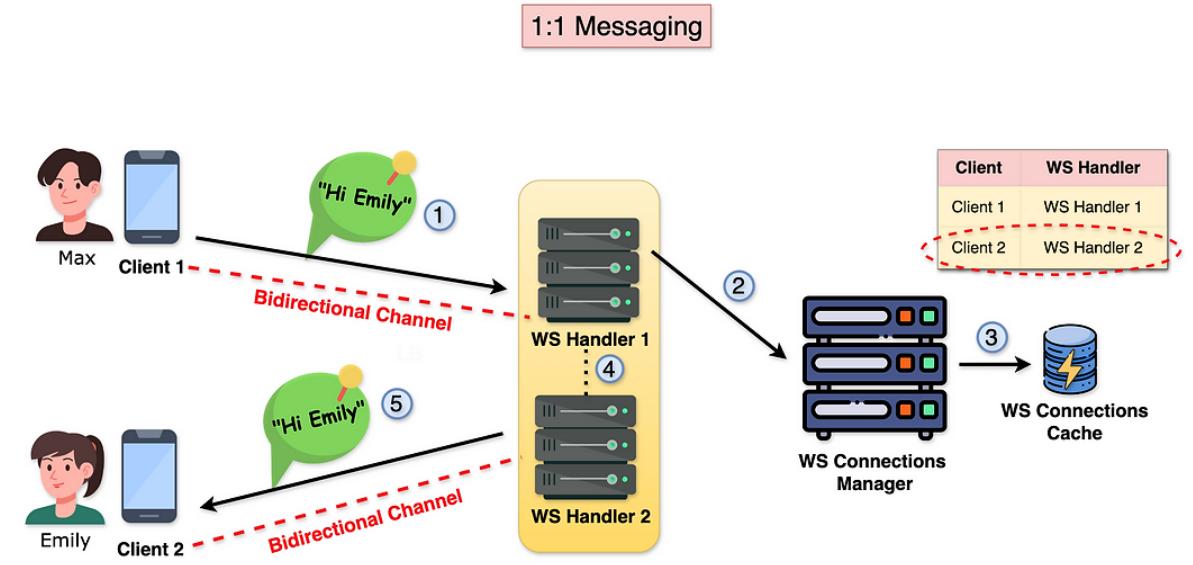


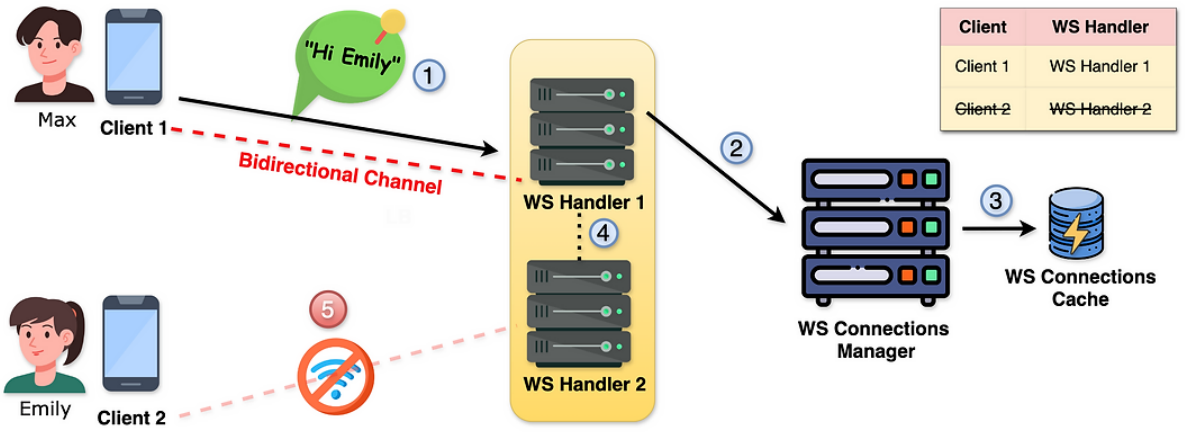

Max
Client 1
Bidirectional Channel
3
2
1
API
LB
API Gateway
WS Handler 1
WS Handler 2
Emily
Client 2
Bidirectional Channel


"Hi Emily"
Max (Client 1)
1
WS Handler 1
2
Client
WS Handler
Client 1
WS Handler 1
Client 2
WS Handler 2
3
"Hi Emily"
"Hi Emily"
Emily (Client 2)
4
WS Handler 2

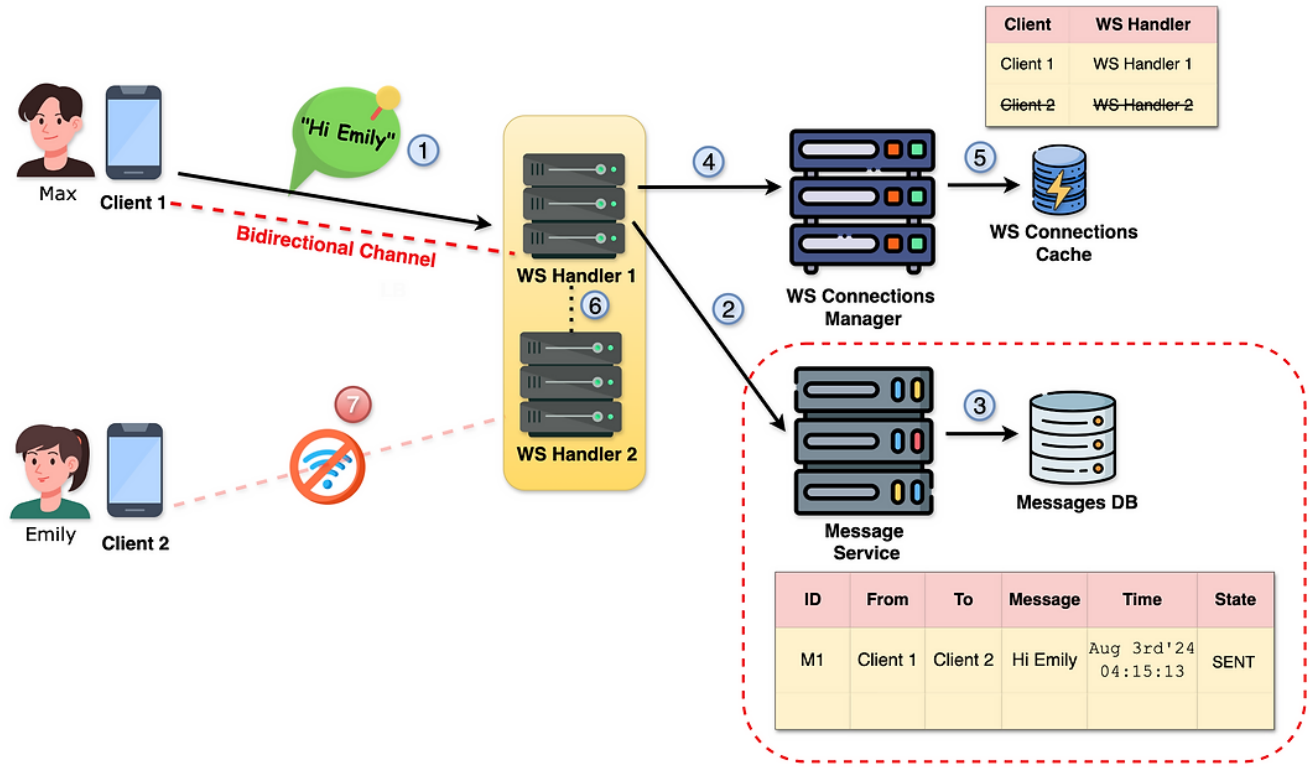


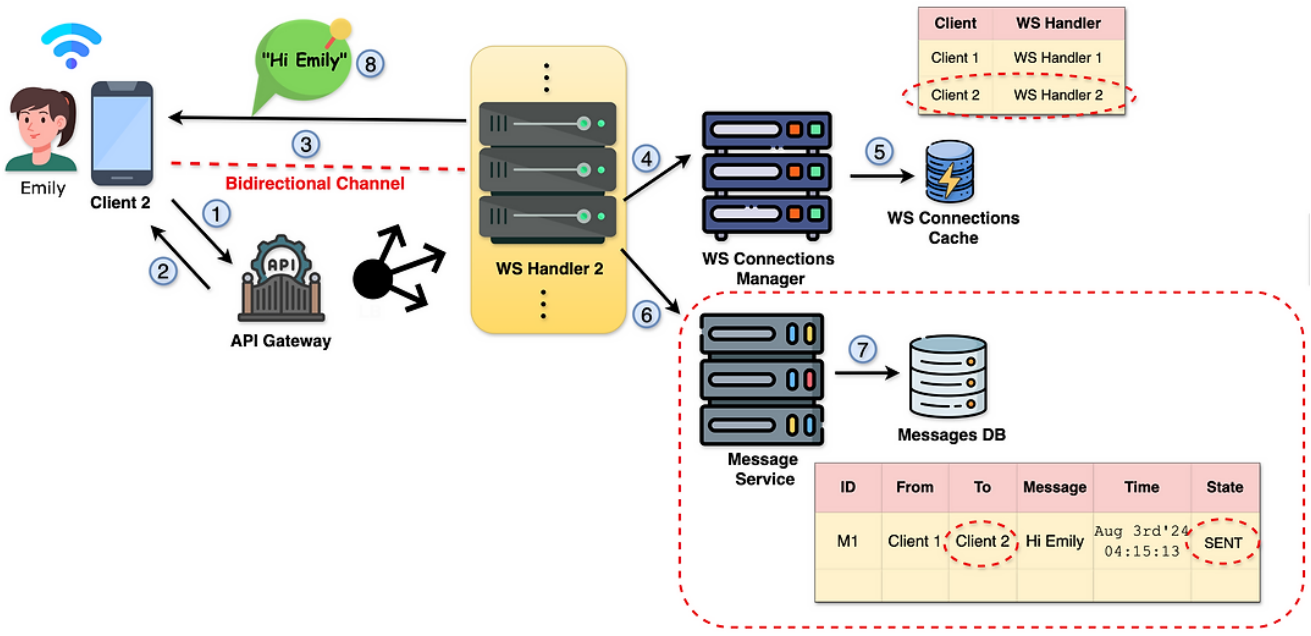
High Level Design: One-One Messaging



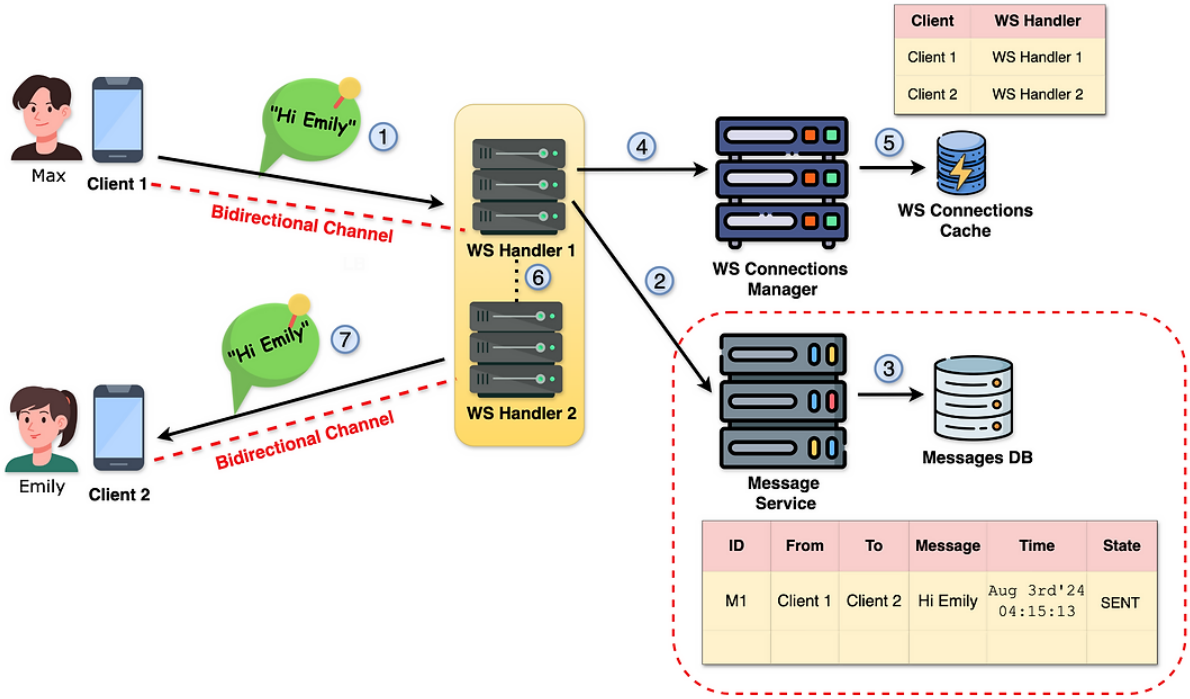


High Level Design: One-One Messaging (Offline Client)

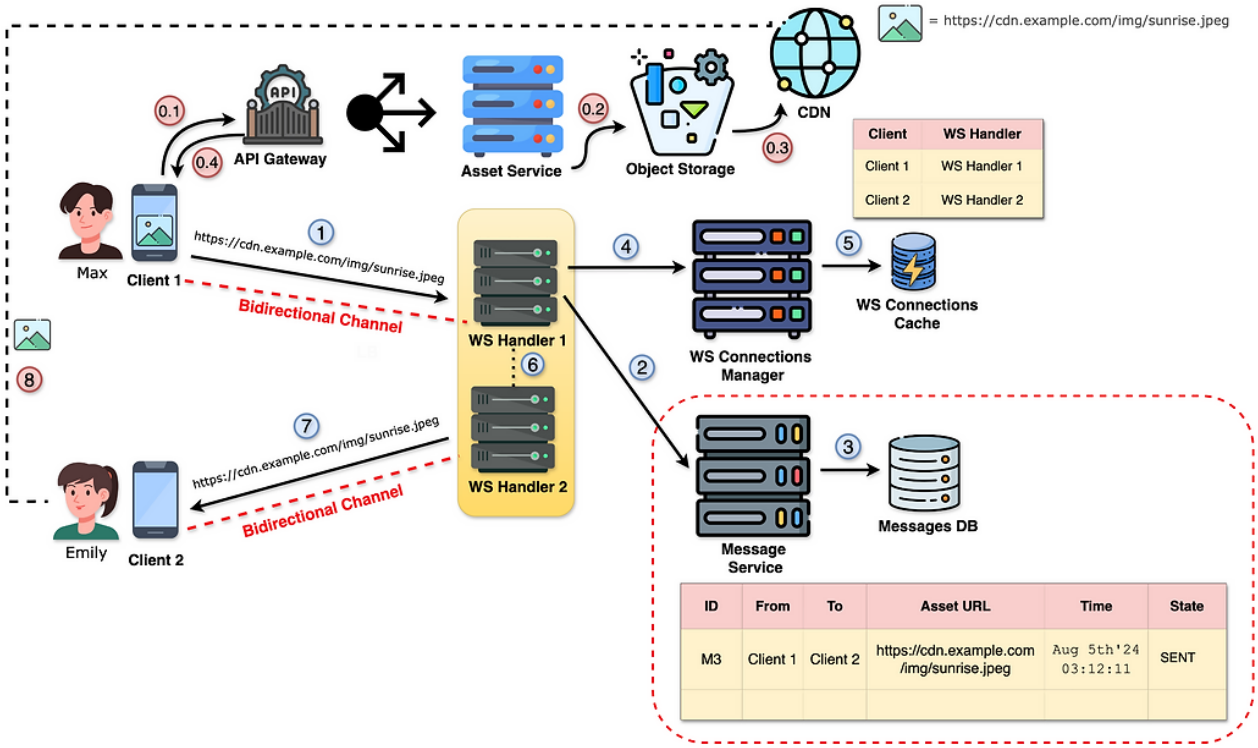




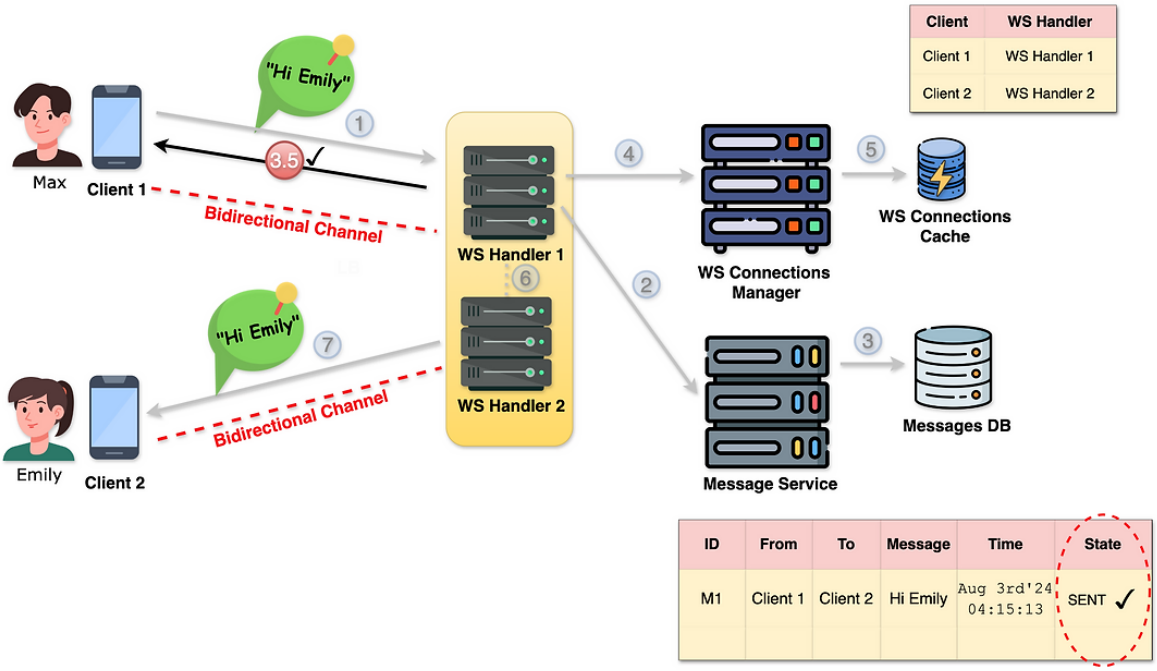
High Level Design: One-One Messaging Final Design



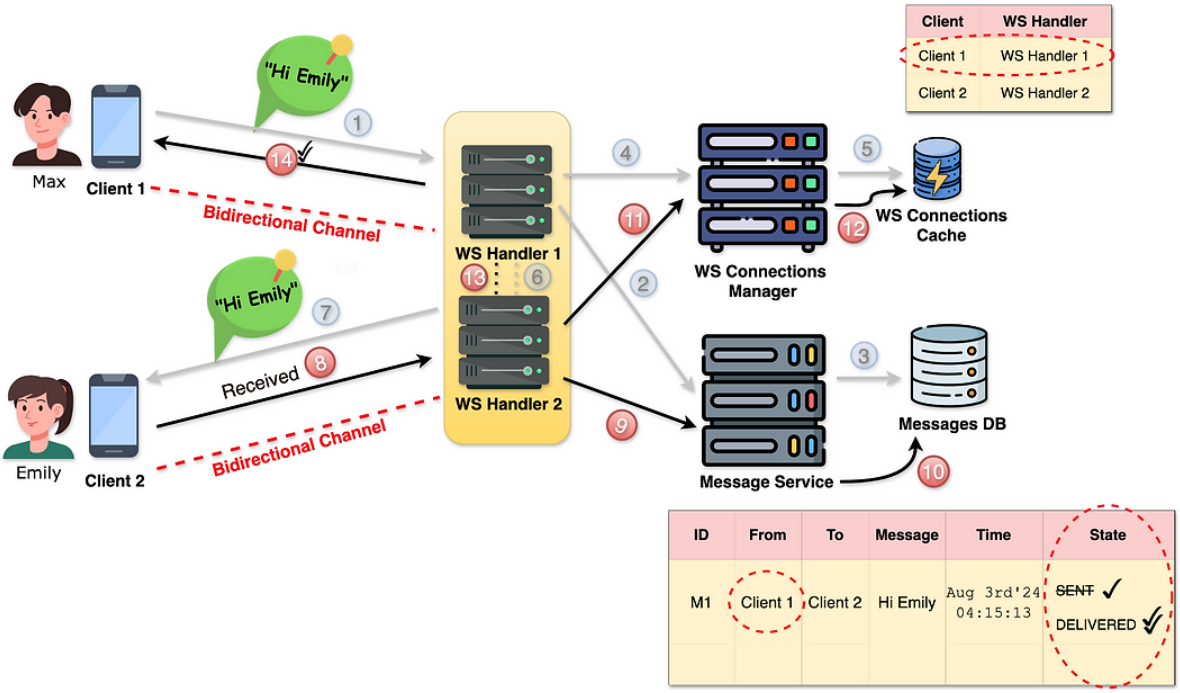
High Level Design: One-One Messaging (Image, Video, Document)



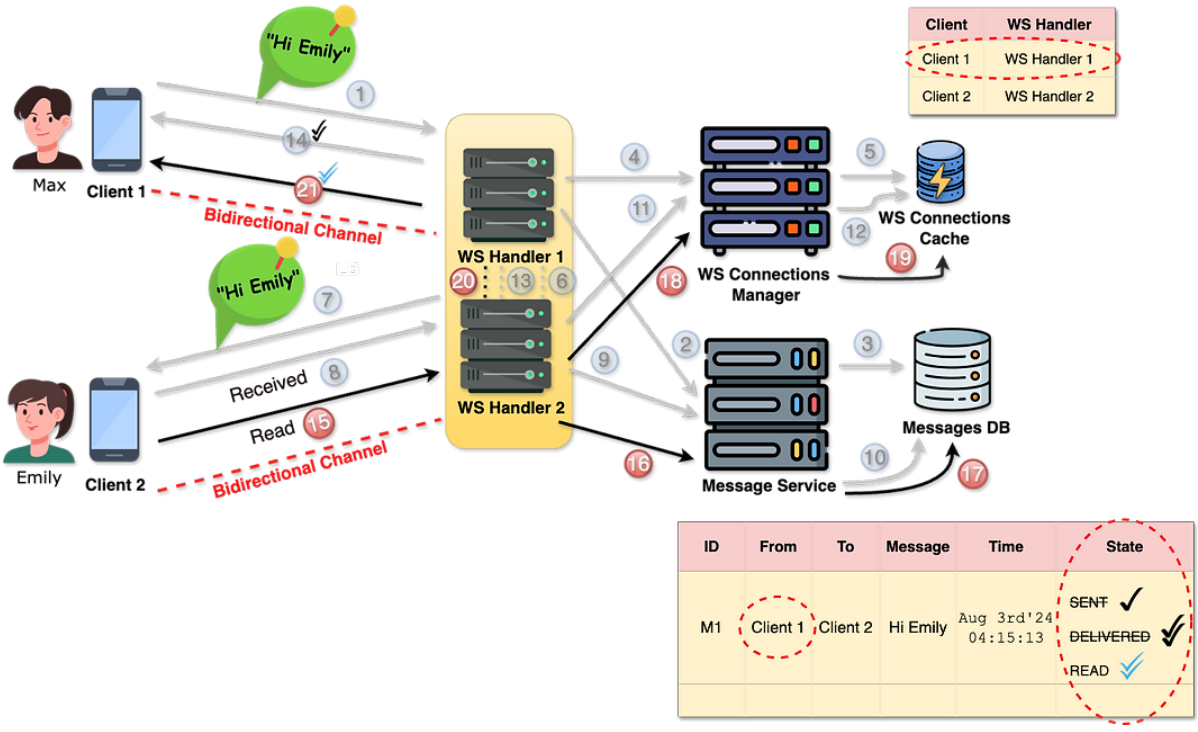
High Level Design: Message Status (Sent)



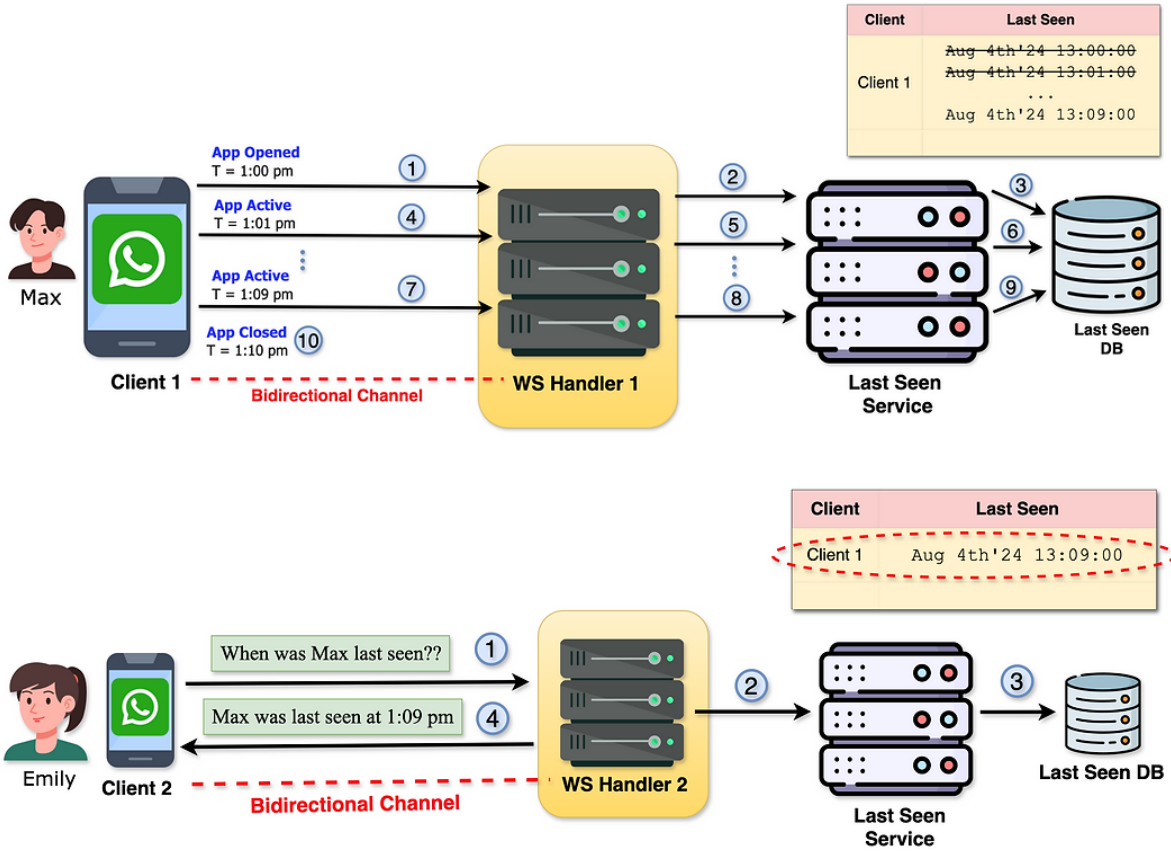
High Level Design: Message Status (Delivered)



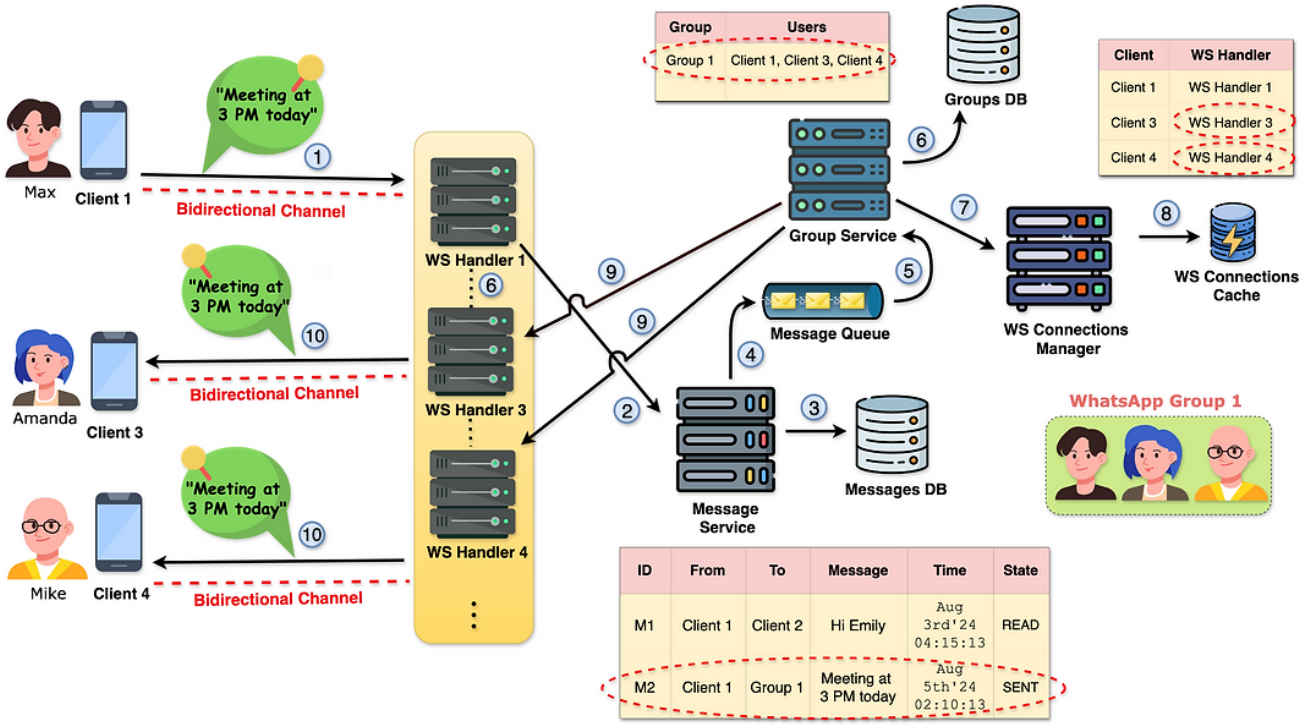
High Level Design: Message Status (Read)



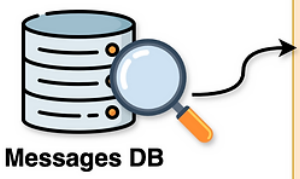
High Level Design: Online Status



High Level Design: Group Messaging



Messages DB schema

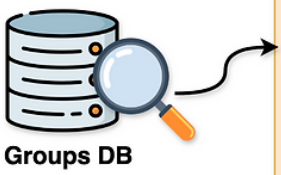


{

MessageId: A unique ID for each message.
SenderId ("From"): ID of the user sending the message.
ReceiverId ("To"): ID of the user or group getting the message.
MessageType: Type of the message, such as text, image, or video.
Message: The actual text of the message.
AssetURL: Link to any images, videos, or files attached.
State: Current status of the message, like 'Sent', 'Delivered', or 'Read'.
Time: When the message was sent.

}

Groups DB schema

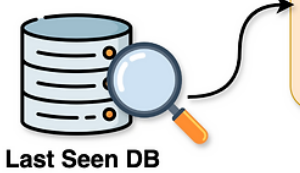


{

GroupId: A unique identifier for each group.
GroupName: The name of the group.
GroupDescription: A short description or purpose of the group.
CreatorId: The user ID of the person who created the group.
Members: A list of user IDs that are members of the group.
CreatedTime: Timestamp when the group was created.
LastUpdatedTime: Timestamp for the last update to the group such as adding or removing members.

}

Last Seen DB schema

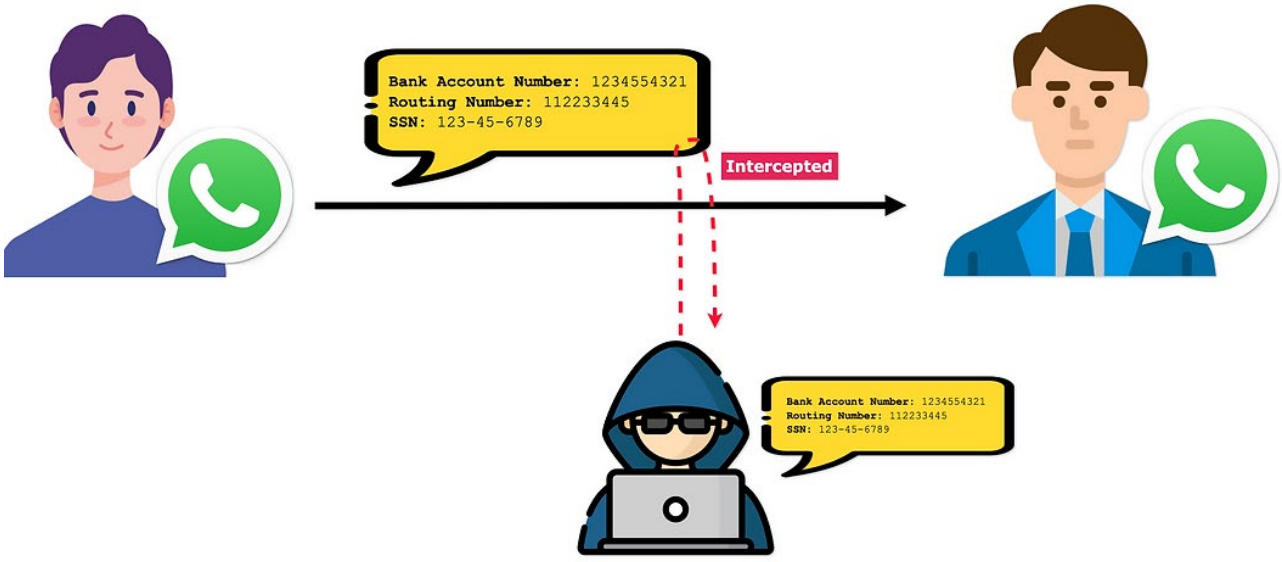


{

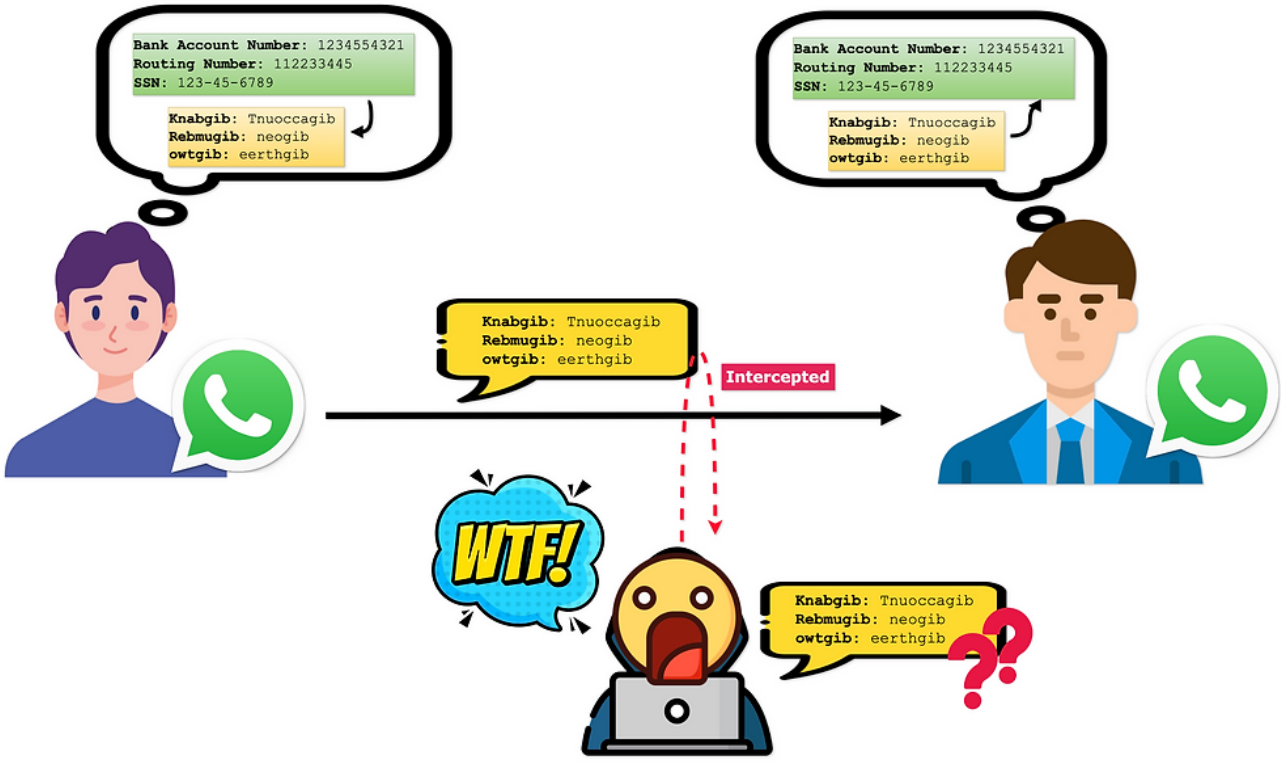
ClientId: A unique identifier for each user/client.
LastSeenTime: The timestamp when the user was last active on the app.

}

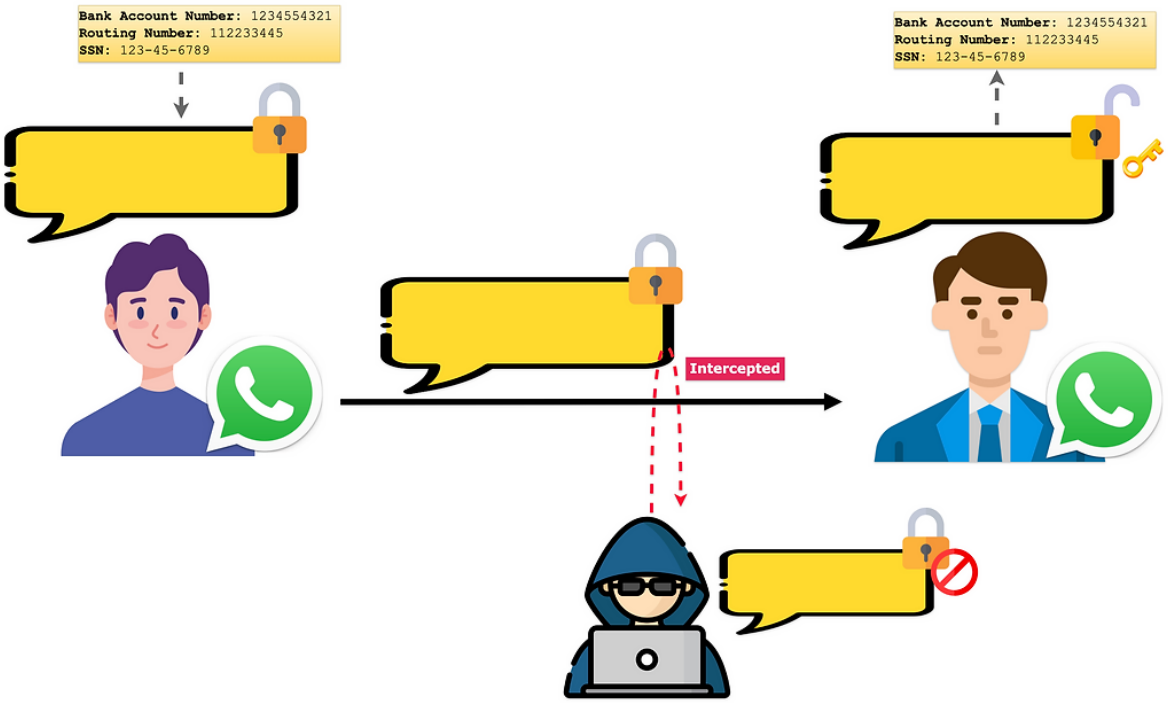
End-to-End Encryption (without Encryption)



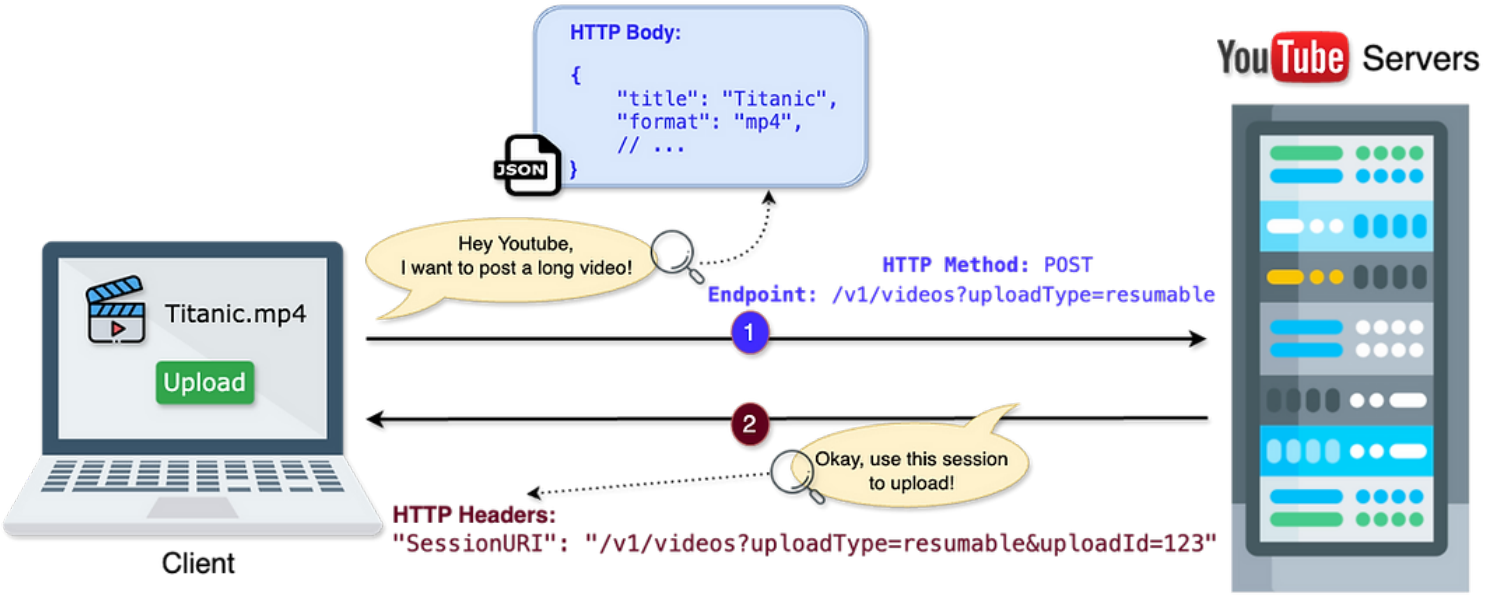
End-to-End Encryption (with Encryption)

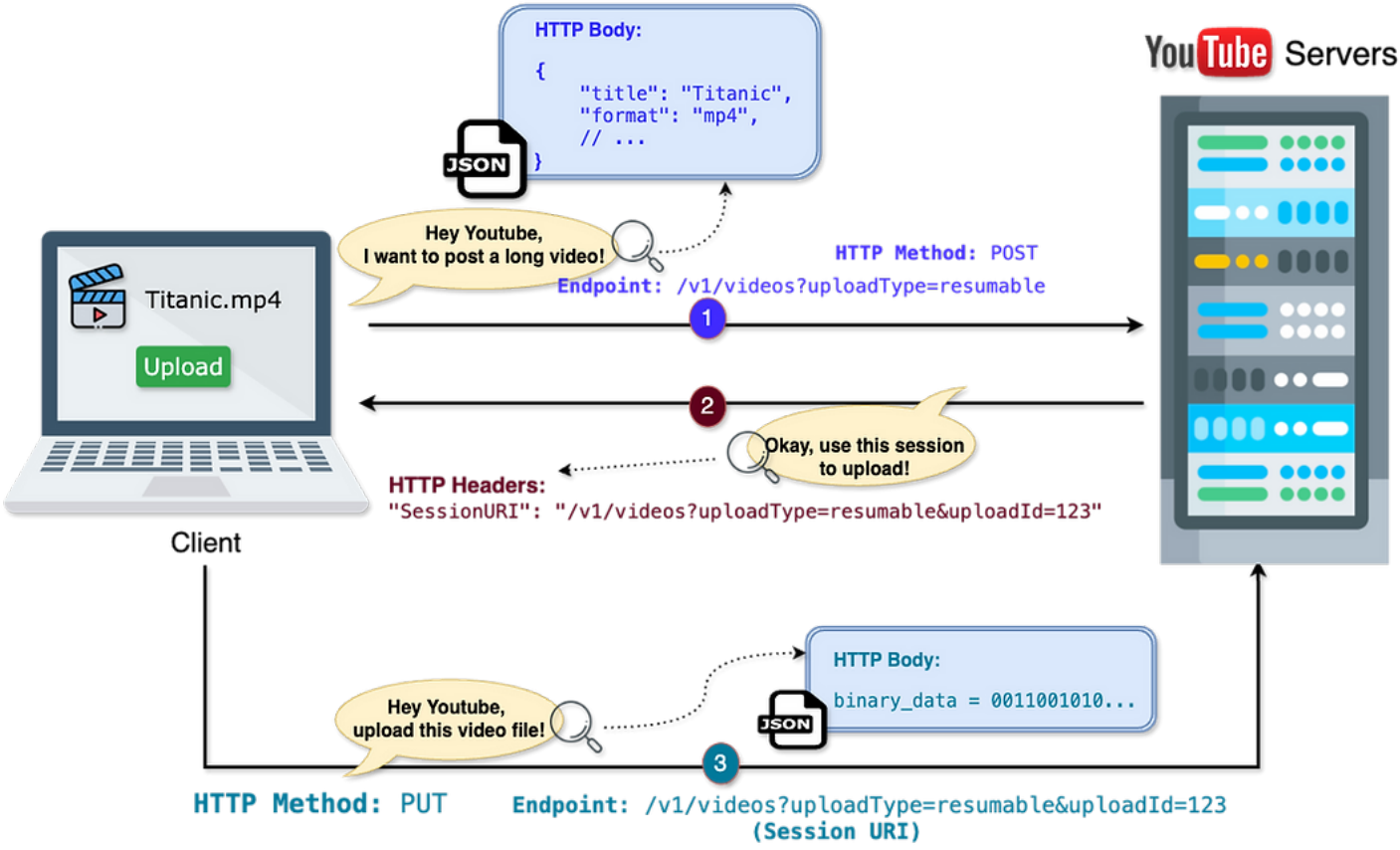


End-to-End Encryption (Encryption with lock & Key)

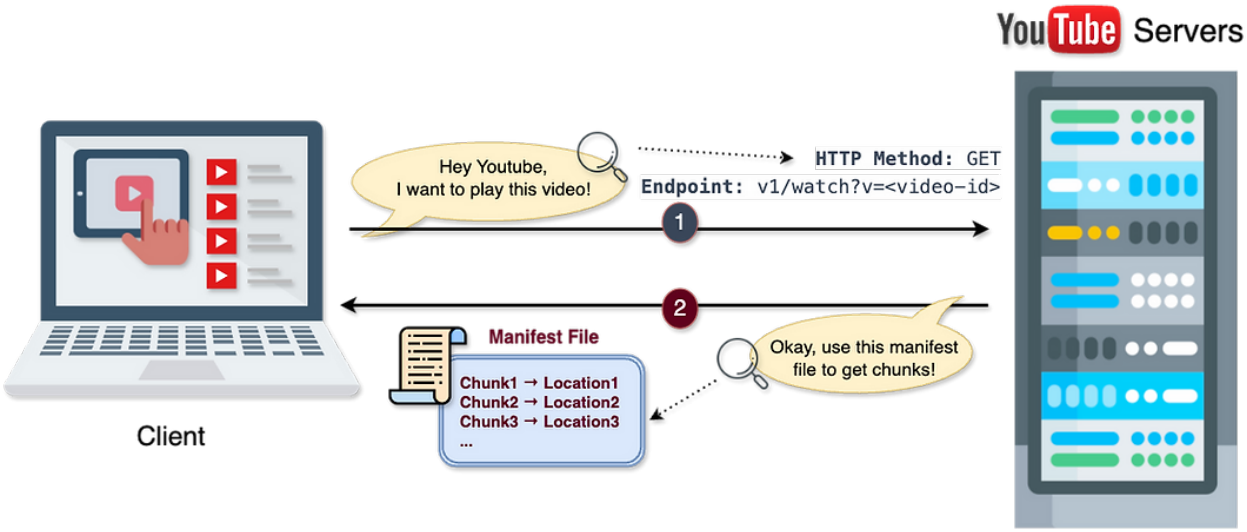


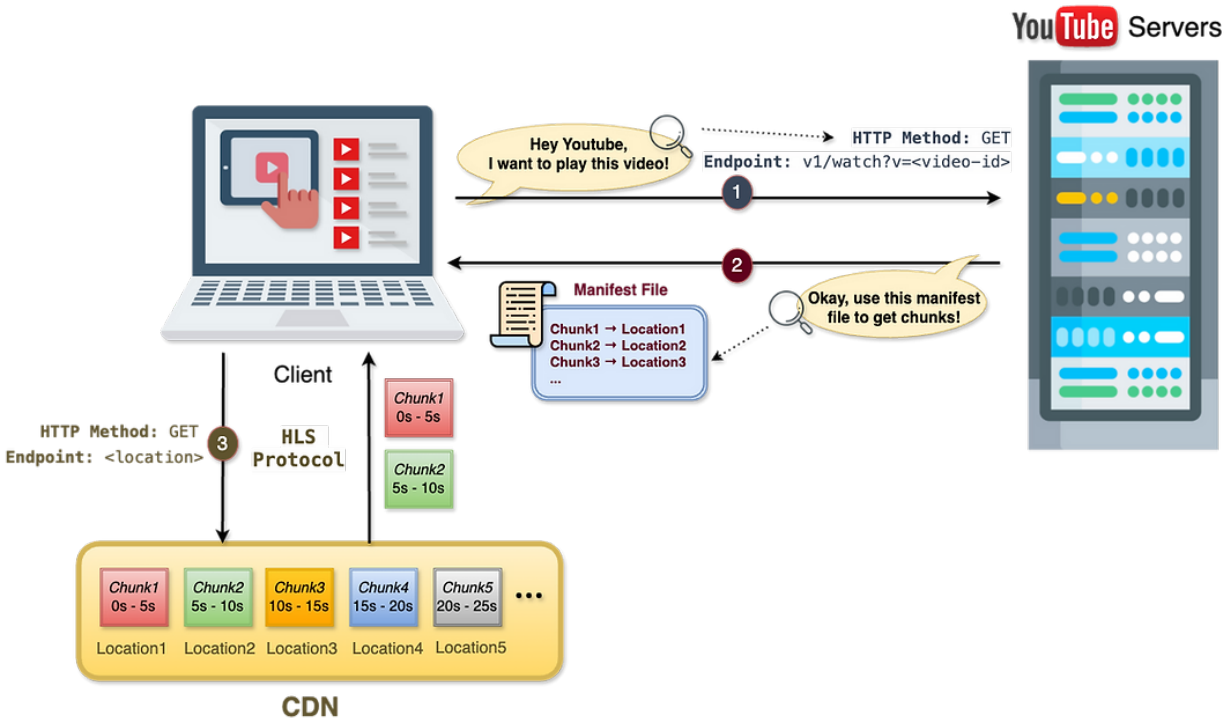
API DESIGN-Upload Content

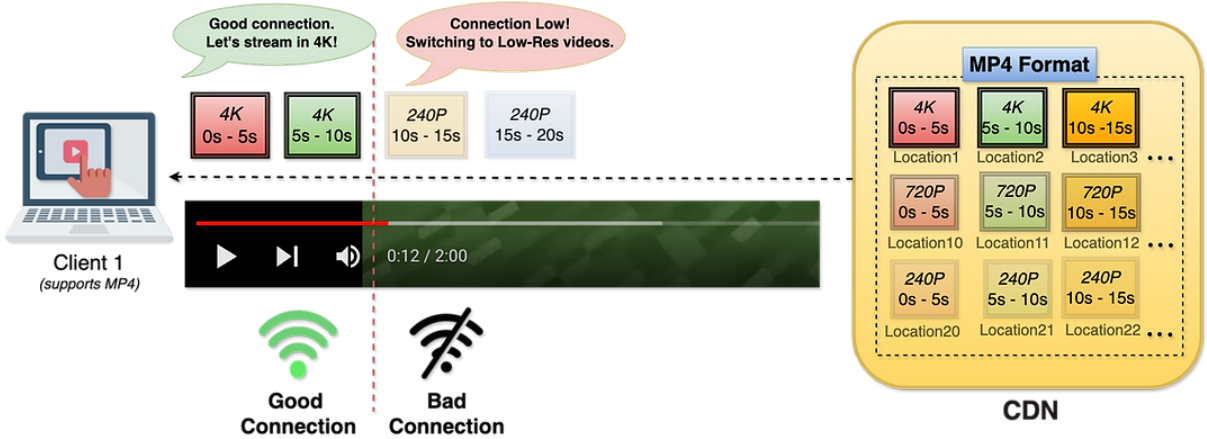




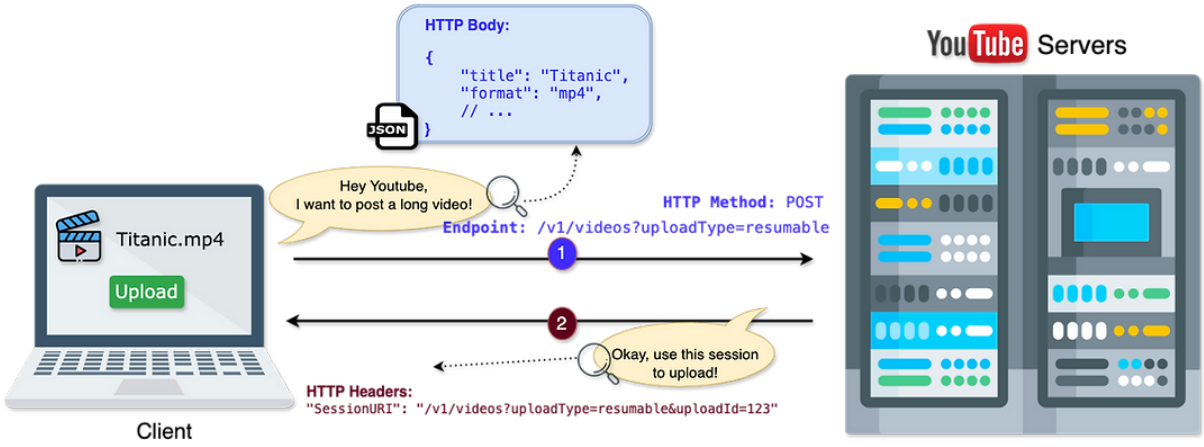
API DESIGN-Stream Content

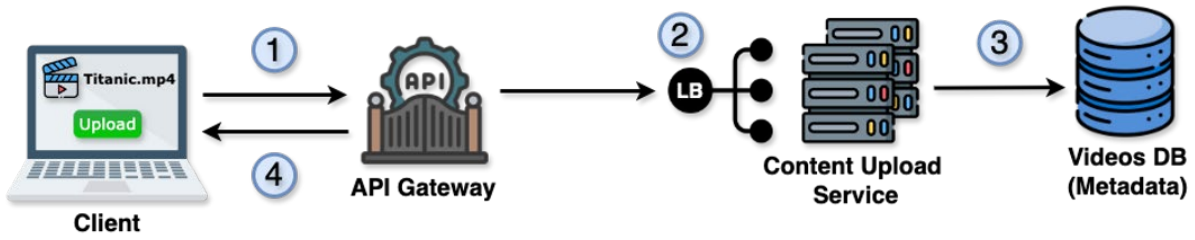


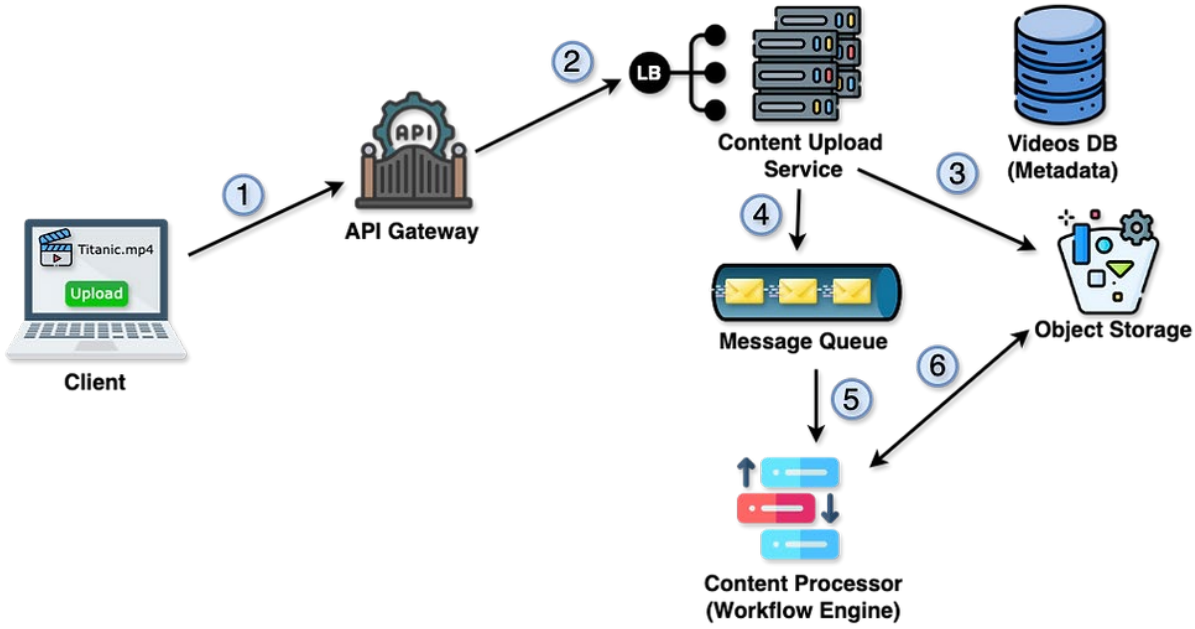


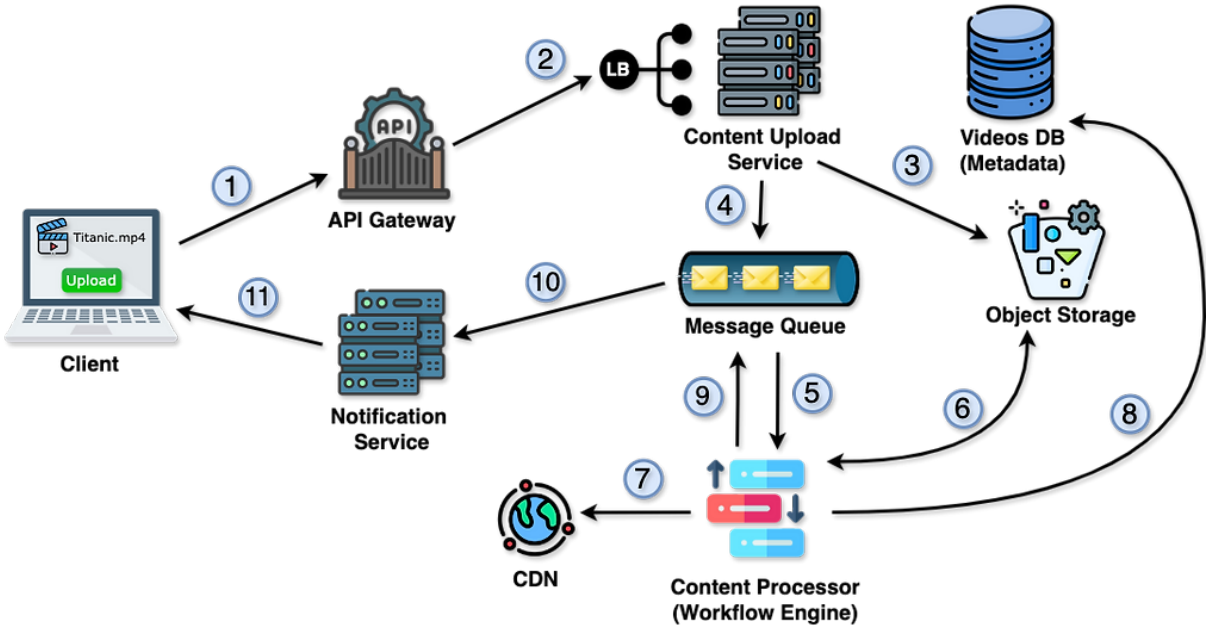


High Level Design-Upload Content

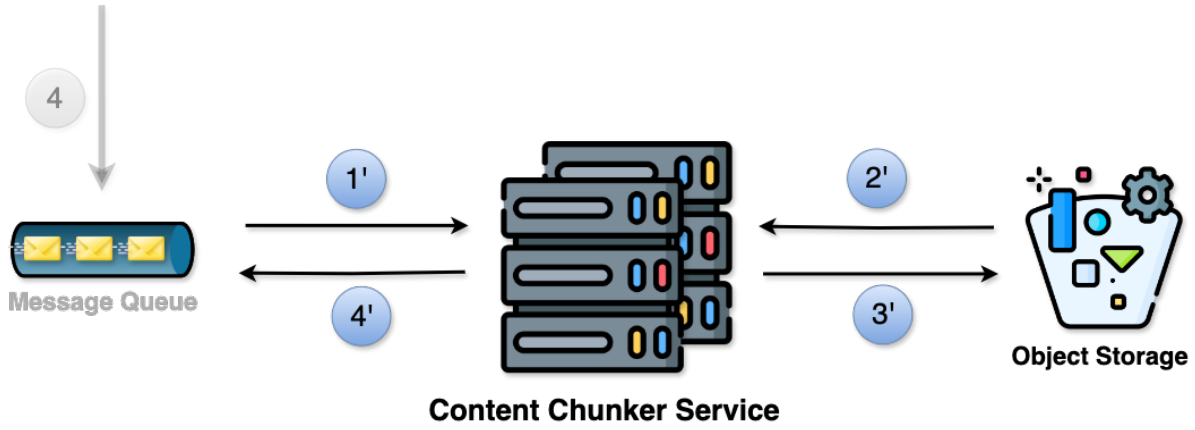




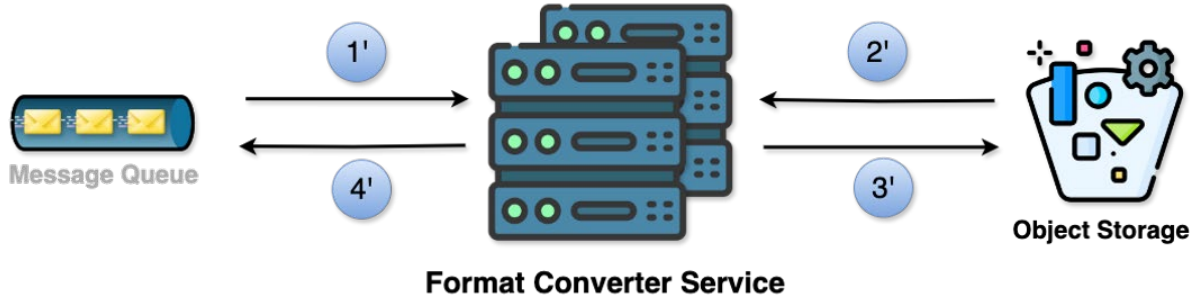




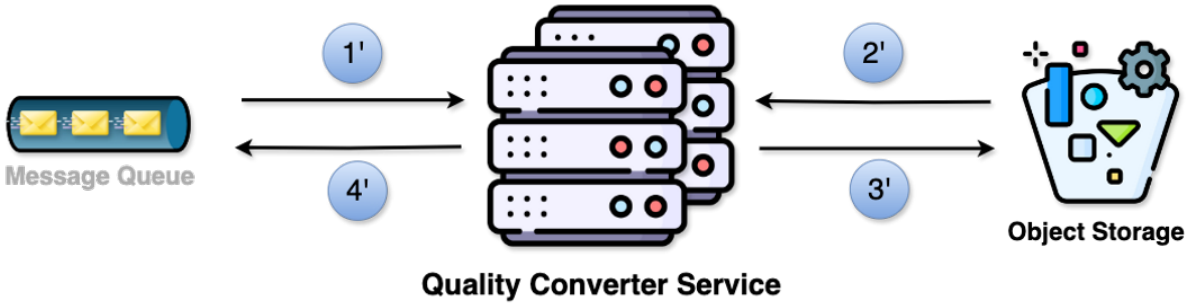
Content Chunker Service



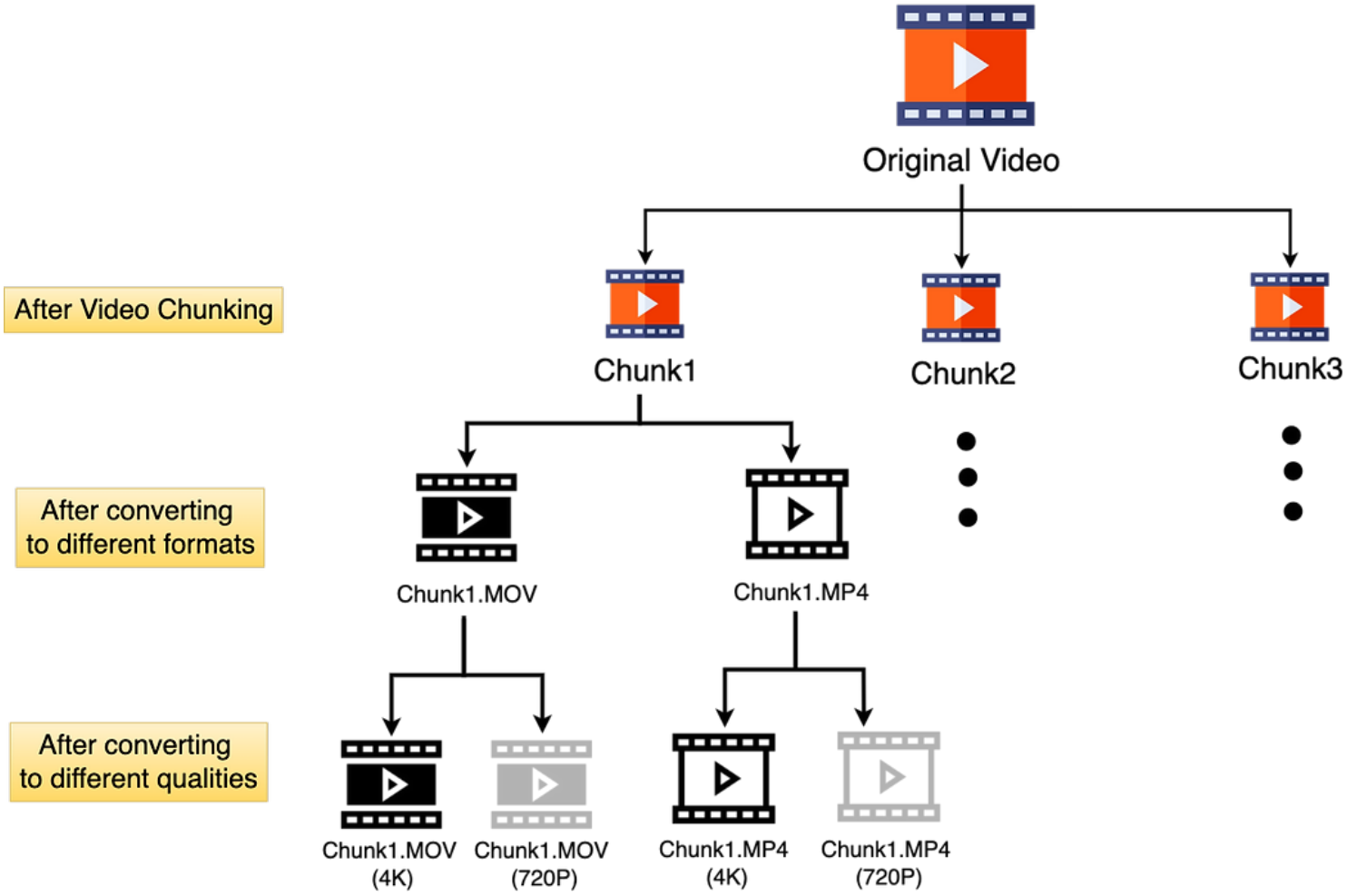
Format Converter Service



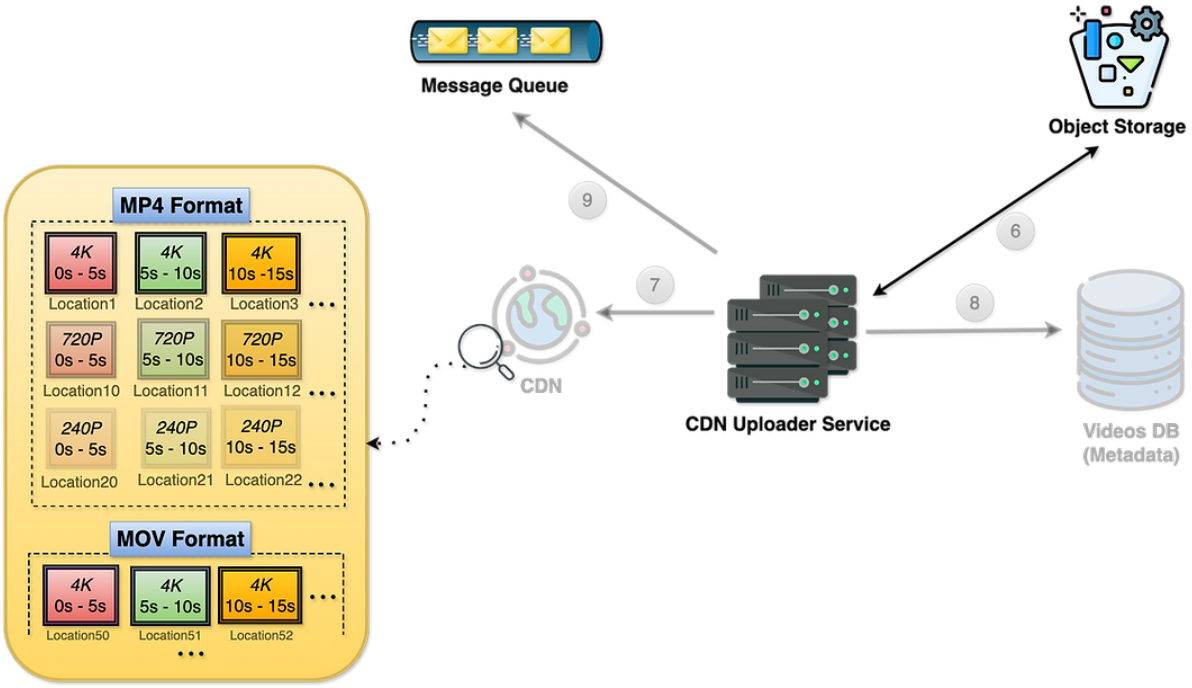
Quality Converter Service



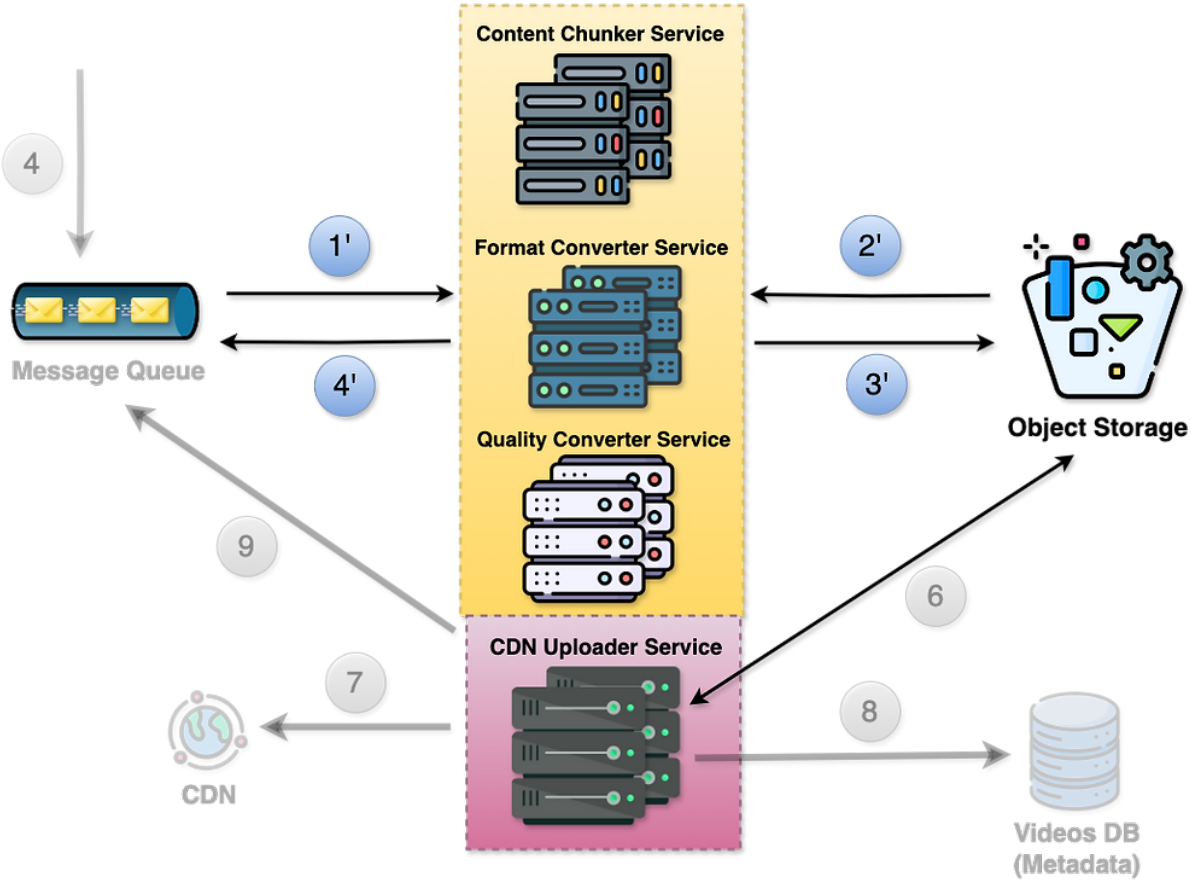
Chunking Videos



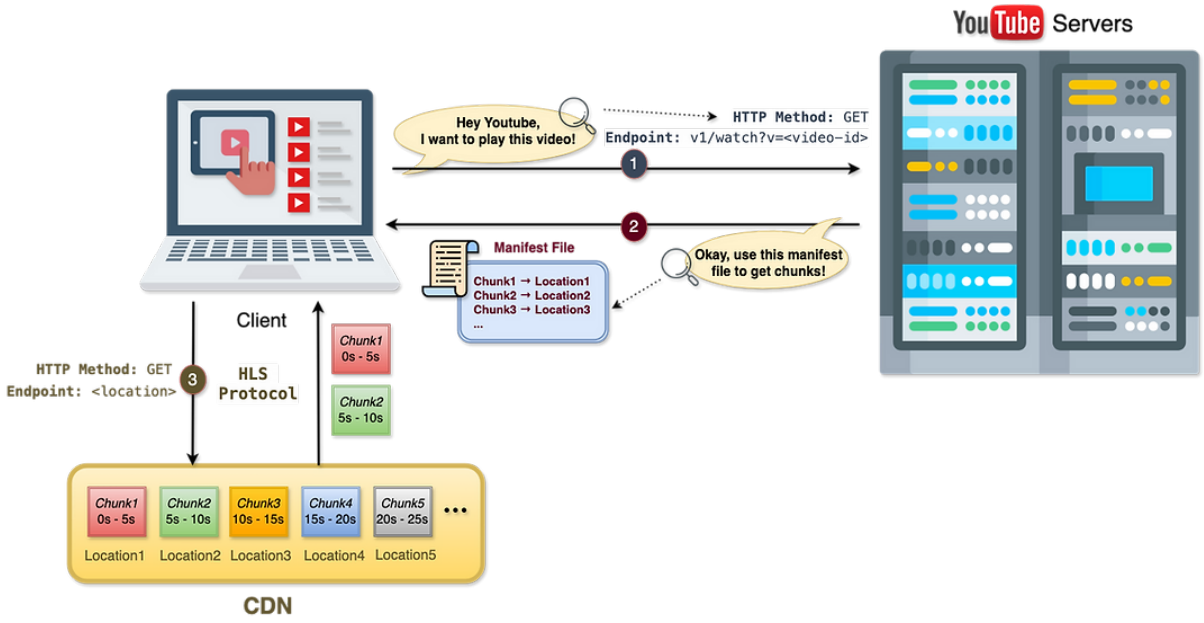
CDN Uploader Service

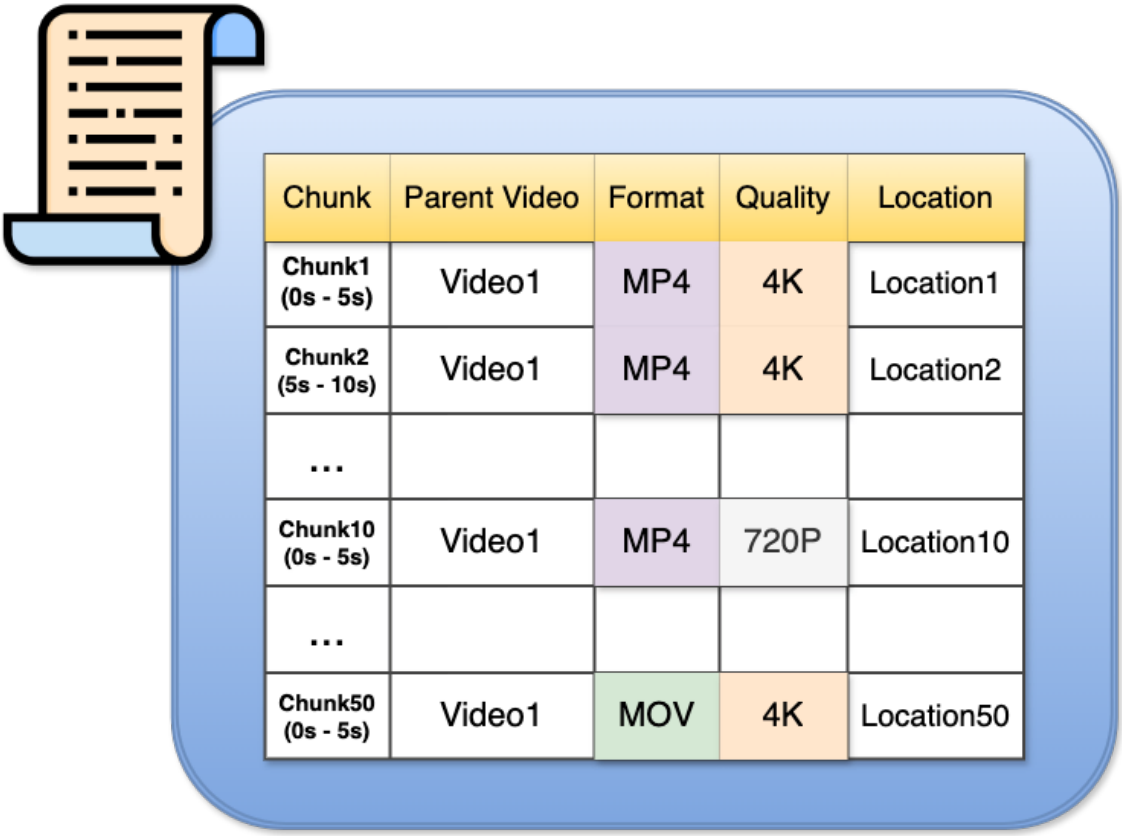


Overall Chunking Flow

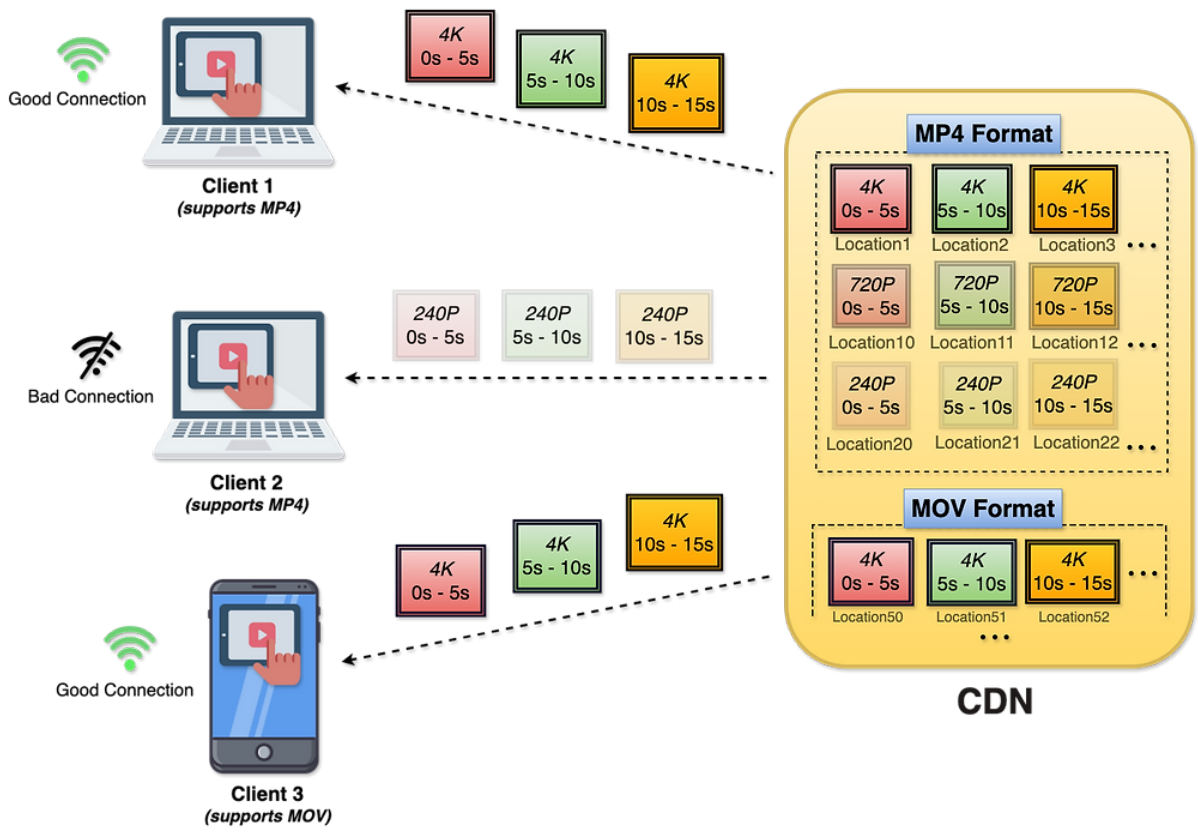


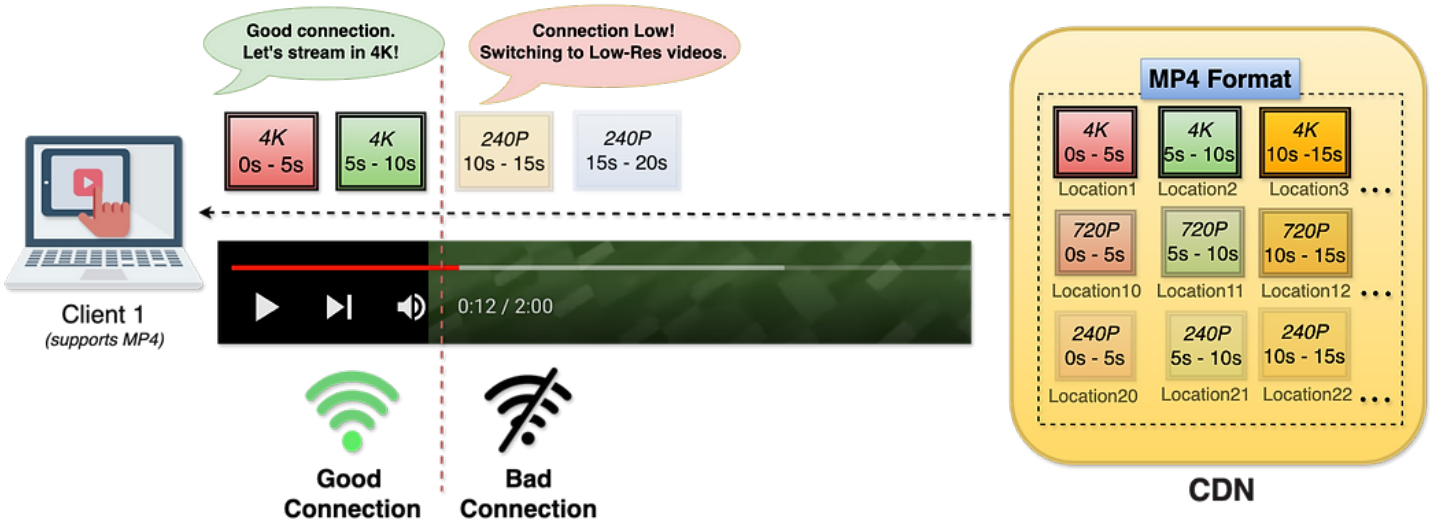
High Level Design-Stream Content





Chunk	Parent Video	Format	Quality	Location
Chunk1 (0s - 5s)	Video1	MP4	4K	Location1
Chunk2 (5s - 10s)	Video1	MP4	4K	Location2
...				
Chunk10 (0s - 5s)	Video1	MP4	720P	Location10
...				
Chunk50 (0s - 5s)	Video1	MOV	4K	Location50


Good Connection
Client 1
(supports MP4)
4K 0s - 5s
4K 5s - 10s
4K 10s - 15s
Bad Connection
Client 2
(supports MP4)
240P 0s - 5s
240P 5s - 10s
240P 10s - 15s
Good Connection
Client 3
(supports MOV)
4K 0s - 5s
4K 5s - 10s
4K 10s - 15s
MP4 Format
4K 0s - 5s
4K 5s - 10s
4K 10s -15s
Location1
Location2
Location3
720P 0s - 5s
720P 5s - 10s
720P 10s - 15s
Location10
Location11
Location12
240P 0s - 5s
240P 5s - 10s
240P 10s - 15s
Location20
Location21
Location22
MOV Format
4K 0s - 5s
4K 5s - 10s
4K 10s - 15s
Location50
Location51
Location52
CDN


Good connection.
Let's stream in 4K!
Connection Low!
Switching to Low-Res videos.
4K 0s - 5s
4K 5s - 10s
240P 10s - 15s
240P 15s - 20s
Client 1
(supports MP4)
0:12 / 2:00
Good Connection
Bad Connection
MP4 Format
4K 0s - 5s
4K 5s - 10s
4K 10s -15s
Location1
Location2
Location3
720P 0s - 5s
720P 5s - 10s
720P 10s - 15s
Location10
Location11
Location12
240P 0s - 5s
240P 5s - 10s
240P 10s - 15s
Location20
Location21
Location22
CDN

Video’s DB Schema

Videos DB (Metadata)

```
{
  "videoId": "unique_video_id",

  // Technical Metadata
  "format": "mp4",
  "duration": "length_of_video_in_s",
  ...

  // General Metadata
  "creatorId": "creator_id",
  "title": "video_title",
  "description": "video_description_text",
  ...


  // Content Delivery Metadata
   "cdnUrls" : {
      "MP4": {
          "4K": ["chunkurl1", "chunkurl2", ...],
          "720P": ["chunkurl1", "chunkurl2", ...],
          ...
      }
      "MOV": {
          "4K": ["chunkurl1", "chunkurl2", ...],
          "720P": ["chunkurl1", "chunkurl2", ...],
          ...
      }
      ...
   }
}
```

Chunking Workflow