Designing Instagram | System Design

Last Updated : 17 Jan, 2024

Designing Instagram is an important topic for system design interview questions. Instagram follows strong Architecture. In this article, we will discuss the architecture of posting photos and videos, following and unfollowing users, liking and disliking posts, searching photos and videos, and generating news feeds.

Important Topics for Instagram System Design

1. What is Instagram?

Instagram is an American photo and video-sharing social networking service owned by Meta Platforms. It allows users to upload media that can be edited with filters, be organized by hashtags, and be associated with a location via geographical tagging. Posts can be shared publicly or with preapproved followers.

2. Requirements for Instagram System Design

2.1 Functional Requirements for Instagram System Design

In functional Requirements, we will not discuss the login or signup page of Instagram. Login and Signup architecture is the same for everyone. We will look for further like posting photos, etc.

- **Post photos and videos**: The users can post photos and videos on Instagram.
- **Follow and unfollow users**: The users can follow and unfollow other users on Instagram.
- **Like or dislike posts**: The users can like or dislike posts of the accounts they follow.
- **Search photos and videos**: The users can search photos and videos based on captions and location.
- **Generate news feed**: The users can view the news feed consisting of the photos and videos (in chronological order) from all the users they follow.

2.2 Non-Functional requirements for Instagram System Design

- **Scalability**: The system should be scalable to handle millions of users in terms of computational resources and storage.
- **Latency**: The latency to generate a news feed should be low.
- **Availability**: The system should be highly available.
- **Durability:** Any uploaded content (photos and videos) should never get lost.
- **Consistency**: We can compromise a little on consistency. It is acceptable if the content (photos or videos) takes time to show in followers' feeds located in a distant region.
- **Reliability**: The system must be able to tolerate hardware and software failures.

3. Capacity Estimation for Instagram System Design

We have 1 billion users, with 500 million as daily active users. Assume 60 million photos and 35 million videos are shared on Instagram per day. We can consider 3 MB as the maximum size of each photo and 150 MB as the maximum size of each video uploaded on Instagram.On average, each user sends 20 requests (of any type) per day to our service.

3.1 Storage Per Day

Photos: *60 million photos/day * 3 MB = 180 TeraBytes / day*

Videos: *35 million videos/day * 150 MB = 5250 TB / day*

Total content size = *180 + 5250 = 5430 TB*

The Total Space required for a Year:

*5430 TB/day * 365 (days a year) = 1981950 TB =* ***1981.95 PetaBytes***

3.2 Bandwidth Estimation

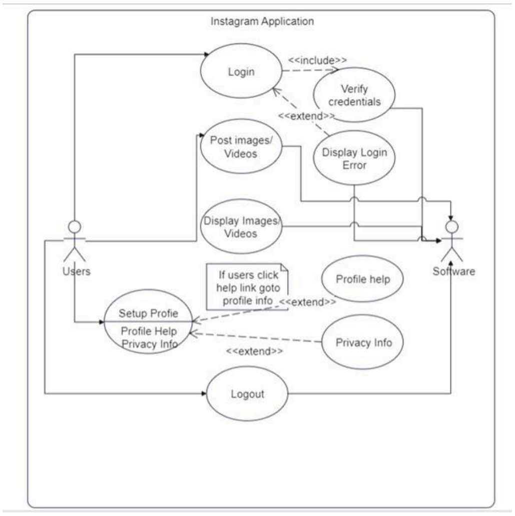
*5430 TB/(24 * 60* 60) = 5430 TB/86400 sec ~= 62.84 GB/s ~= 502.8 Gbps*

Incoming bandwidth *~= 502.8 Gbps*

Let's say the ratio of readers to writers is 100:1.

Required outgoing bandwidth *~= 100 * 502.8 Gbps ~=* ***50.28 Tbps***

4. Use Case Diagram for Instagram System Design

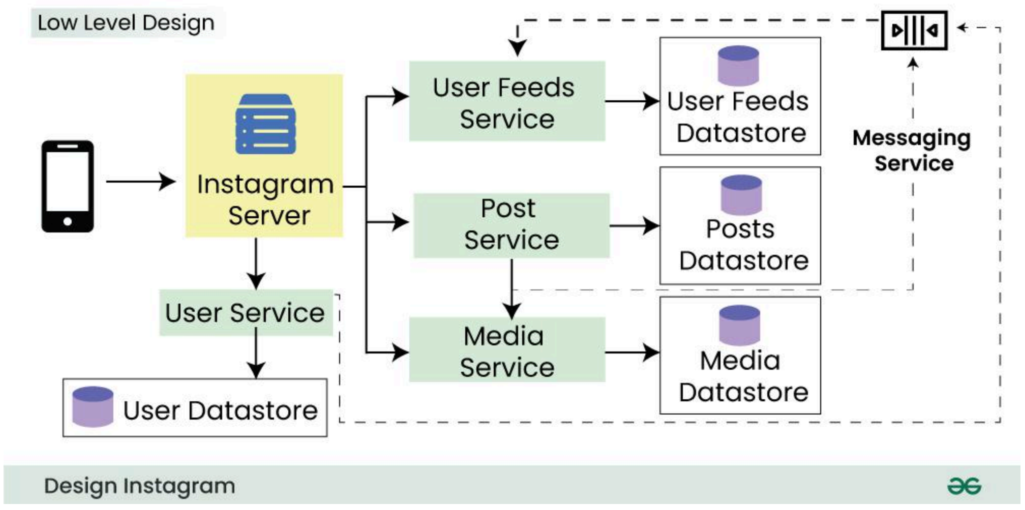


Use Case Diagram Instagram

In the above Diagram we have discussed about the use case diagram of Instagram:

- If the user is new, they will register firstly it will be store in database, they will verifiy the profile.
- If user is already signup, they will provide the email-Id and Password.
- On the home page they will get the photos and videos, as well as the story page.
- The post which is posted now, it will come at the top. User can follow or unfollow the person. User can get live. It's all depend on them.
- There will be setting, in which user can see there past story or the post which has been archive. User can unblock the person they can get verified account, after paying.

5. Low-Level Design(LLD) for Instagram System Design



Here's a breakdown of the key components and interactions for Instagram's low-level design:

- **User Service:**
    - Handles user registration, login, authentication, and profile management.
    - Stores user data like username, email, bio, profile picture, etc.
    - Integrates with social authentication providers (e.g., Facebook, Google).
- **Post Service:**
    - Handles photo and video uploads, editing, and deletion.
    - Stores post metadata like caption, hashtags, location, timestamp, etc.Processes uploaded media for resizing, filtering, and thumbnail generation.
    - Manages photo and video transcoding for different devices and resolutions.
- **Feed Service:**
    - Generates personalized news feeds for each user based on their follows, likes, activity, and engagement.
    - Leverages a distributed system like Apache Kafka or RabbitMQ for real-time updates and notifications.

    - Utilizes a cache layer like Redis for fast feed retrieval and reduced database load.
- **Storage Service:**
    - Stores uploaded photos and videos efficiently and reliably.
    - Utilizes a scalable object storage solution like Amazon S3, Google Cloud Storage, or Azure Blob Storage.
    - Implements redundancy and disaster recovery mechanisms for data protection.
- **Search Service:**
    - Enables searching for users, hashtags, and locations.
    - Indexes users, posts, and hashtags based on relevant parameters.
    - Employs efficient indexing and search algorithms for fast and accurate results.
- **Comment Service:**
    - Handles adding, editing, and deleting comments on posts.
    - Tracks comment threads and parent-child relationships.
    - Notifies users of new comments on their own posts or comments they participated in.
- **Notification Service:**
    - Informs users about relevant events like likes, comments, mentions, and follows.
    - Pushes notifications to mobile devices through platforms like Firebase Cloud Messaging or Amazon SNS.
    - Leverages a queueing system for asynchronous notification delivery.
- **Analytics Service:**
    - Tracks user engagement, post performance, and overall platform usage.
    - Gathers data on views, likes, comments, shares, and clicks.
    - Provides insights to improve user experience, optimize content recommendations, and target advertising.

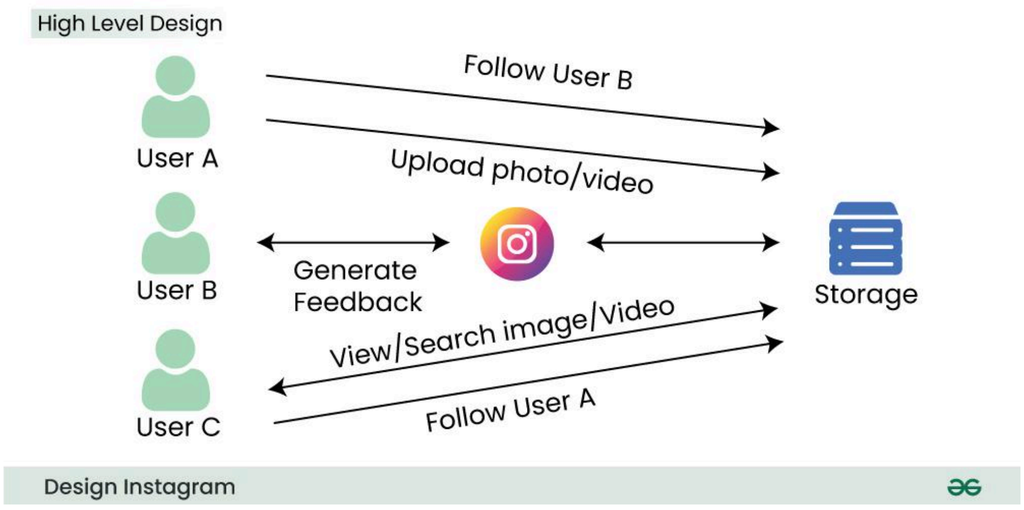
Why we need caching for storing the data?

- Cache the data to handle millions of reads. It improves the user experience by making the fetching process fast. We'll also opt for lazy loading, which minimizes the client's waiting time.
- It allows us to load the content when the user scrolls and therefore save the bandwidth and focus on loading the content the user is currently viewing. It improves the latency to view or search a particular photo or video on Instagram.

6. High-Level Design(HLD) for Instagram System Design

Our system should allow us to upload, view, and search images and videos at a high level. To upload images and videos, we need to store them, and upon fetching, we need to retrieve the data from the storage. Moreover, the users should also be allowed to follow each other.

At a high level, Instagram can be viewed as a system with the following key components and interactions:



Components:

- **Client:** Mobile apps, web app, and APIs providing interface for users to interact with the system.
- **Authentication & Authorization:** Handles user login, registration, and access control.
- **Content Management:** Manages user-generated content like photos, videos, live streams, stories, and messages.
- **Feed Generation:** Personalizes news feeds for each user based on their follows, activity, and engagement.
- **Social Graph:** Tracks relationships between users (follows, followers, friends).
- **Discovery & Search:** Enables searching for users, hashtags, locations, and content.

- **Notifications:** Informs users about relevant events like likes, comments, mentions, and follows.
- **Analytics & Reporting:** Tracks user engagement, content performance, and overall platform usage.

Interactions:

- **User creates content:**
    - Client uploads photo/video.
    - Content Management stores media and metadata.
    - Feed Generation updates user's and relevant followers' feeds.
    - Notifications inform interested users.
- **User interacts with content:**
    - Client sends like/comment/share actions.
    - Content Management and Social Graph update relevant data.
    - Feed Generation potentially reshuffles feeds based on new interactions.
    - Notifications inform interested users.
- **User discovers new content:**
    - Client uses search functionalities.
    - Discovery & Search identifies relevant content.
    - Client displays search results.
- **User manages connections:**
    - Client sends follow/unfollow requests.
    - Social Graph updates connections.
    - Feed Generation adjusts based on changed relationships.
- **User monitors activity:**
    - Client checks notifications feed.
    - Notifications provide updates on relevant events.

Key Design Considerations:

- **Scalability:** System should handle millions of users and massive data volumes.
- **Performance:** Deliver fast response times for user interactions and content delivery.
- **Reliability:** Ensure high availability and prevent data loss.
- **Security:** Protect user data and privacy.
- **Engagement:** Design features that encourage user interaction and content creation.

7. API Design for Instagram System Design

7.1 Post photos and videos

Here's a potential API design for uploading photos and videos to Instagram:

Endpoints:

- **POST /media:** Submits a new photo or video.
- **PUT /media/{media_id}:** Updates existing metadata for a media item.
- **DELETE /media/{media_id}:** Deletes a media item.

Request

```python
import requests

url = 'https://api.instagram.com/media'  # Replace with the actual API endpoint URL
access_token = 'your_access_token'  # Replace with the user's valid access token

# Define file path, caption, hashtags, and location (adjust as needed)
file_path = 'path/to/your/photo_or_video.jpg'  # Or .mp4 for videos
caption = 'Your caption for the media'
hashtags = 'photography,nature,beautiful'
location = {"latitude": 37.421998, "longitude": -122.084269}

# Prepare headers and files for the request
headers = {
    'Authorization': f'Bearer {access_token}',
    'Content-Type': 'multipart/form-data'
}
files = {
    'file': open(file_path, 'rb'),
    'caption': caption,
    'hashtags': hashtags,
    'location': str(location)  # Location needs to be serialized as a string
}

# Send the POST request
response = requests.post(url, headers=headers, files=files)

# Handle the response
if response.status_code == 201:
    data = response.json()
    print('Media uploaded successfully!')
    print('Media ID:', data['media_id'])
    print('URL:', data['url'])
else:
    print('Upload failed:', response.text)
    print('Error details:', response.json())  # Display any error details
```

Response

```json
{
  "media_id": "1234567890abcdef",
  "url": "https://instagram.com/p/12345678",
  "created_at": "2024-01-09T18:34:00Z",
  "updated_at": "2024-01-09T18:34:00Z"
}
```

7.2 Follow and unfollow users

Here's a potential API design for following and unfollowing users on Instagram:

Endpoints:

- **POST /users/{user_id}/follow:** Follows the specified user.
- **DELETE /users/{user_id}/follow:** Unfollows the specified user.
- **GET /users/{user_id}/following:** Retrieves a list of users followed by the specified user (paginated).
- **GET /users/{user_id}/followers:** Retrieves a list of users following the specified user (paginated).

Response

```
{
  "message": "User followed successfully"
}
```

Requests

```python
import requests

url = f'https://api.instagram.com/users/{user_id_to_follow}/follow'
headers = {'Authorization': f'Bearer {access_token}'}
response = requests.post(url, headers=headers)

if response.status_code == 200:
    print('User followed successfully!')
else:
    print('Follow failed:', response.text)
    print('Error details:', response.json())
```

7.3 Like or Dislike posts

Designing an API for liking and disliking posts involves multiple considerations. Here's a breakdown of key aspects to think about:

API Endpoints:

- **GET /posts/{post_id}**: Retrieves details of a post, including the number of likes/dislikes.
- **POST /posts/{post_id}/like**: Registers a like for the post by the authenticated user.

- **POST /posts/{post_id}/dislike**: Registers a dislike for the post by the authenticated user.
- **DELETE /posts/{post_id}/like**: Removes the like for the post by the authenticated user (if previously done).
- **DELETE /posts/{post_id}/dislike**: Removes the dislike for the post by the authenticated user (if previously done).

Request

```
import requests

url_like = f"https://api.instagram.com/media/likes/{media_id}"
url_unlike = f"https://api.instagram.com/media/likes/{media_id}"
headers = {'Authorization': f'Bearer {access_token}'}

# Like a post
response_like = requests.post(url_like, headers=headers)

# Unlike a post
response_unlike = requests.delete(url_unlike, headers=headers)
```

Response

```
// Request
POST /posts/123/like
Content-Type: application/json

{
  "user_id": 456
}

// Response (if successful)
Status: 200 OK
{
  "message": "Post liked successfully"
}

// Response (if already liked)
Status: 400 Bad Request
{
  "error": "Post already liked"
}
```

7.4 Search photos and videos

- **Search Endpoint:**
    - Typically a POST request to a `/search` endpoint.

    ◦ Query parameters include:`query`: The search term(s), `media_type`: Filters results by photo, video, or both, Additional filters: date range, location, people, etc. (if supported)

Request

```python
import requests

url = "https://photoslibrary.googleapis.com/v1/mediaItems:search"
headers = {
    "Authorization": "Bearer YOUR_ACCESS_TOKEN"  # Replace with your access token
}
data = {
    "filters": {
        "mediaTypes": ["PHOTO", "VIDEO"]  # Search for both photos and videos
    }
}
response = requests.post(url, headers=headers, json=data)

if response.status_code == 200:
    results = response.json()
    # Process the search results
else:
    print("Error:", response.status_code, response.text)
```

Response

```json
{
  "data": [
    {
      "media_type": "IMAGE",
      "media_url": "https://www.instagram.com/p/abc123/",
      "thumbnail_url": "https://scontent-amt2-1.cdninstagram.com/v/t51.2885-15/e35/s1080
      "caption": "My adorable cat!",
      "timestamp": "2024-01-10T13:40:00+0000"
    },
    {
      "media_type": "VIDEO",
      "media_url": "https://www.instagram.com/tv/xyz987/",
      "thumbnail_url": "https://scontent-amt2-1.cdninstagram.com/v/t51.2885-15/e35/s1080
      "caption": "Amazing sunset timelapse!",
      "timestamp": "2024-01-09T20:30:00+0000"
    }
  ]
}
```

8. Database Design for Instagram Database Design

We need the following tables to store our data:

8.1 User:

Table to store user data – **Users**

User

```
{
userId: string[HashKey]
name: string
emailId: string
creationDateInUtc: long
}
```

8.2 User_Follows:

Table to store follower data – **User_follows**

Follow_User_Table

```
{
followingUserId_followerUserId: string [HashKey]
followingUserId: string [RangeKey]
followerUserId: string
creationDateInUtc: long
}
```

8.3 User Uploads

Table to store user uploads – **User_uploads**

Upload Image Table

```
{
uploadId: string[Hashkey]
userId: string[RangeKey]
imageLocation: string
uploadDateInUtc: long
caption: string
}
```

8.4 User Feed

Table to store the user feed data – **User_feed**

Upload Video or Image

```
{
userId: string[Hashkey]
uploadId: string
creationDateInUtc: long[RangeKey]
}
```

8.5 Which Database we should select for Data storing?

It is essential to choose the right kind of database for our Instagram system, but which is the right choice — SQL or NoSQL? Our data is inherently relational, and we need an order for the data (posts should appear in chronological order) and no data loss even in case of failures (data durability). Moreover, in our case, we would benefit from relational queries like fetching the followers or images based on a user ID. Hence, SQL-based databases fulfill these requirements.

Image and Feed generation service used as microservice architecture.

9. Microservices for Instagram System Design

Image and Feed generation service used as microservice architecture.

Microservices – also known as the microservice architecture – is an architectural style that structures an application as a collection of services that are:

- Independently deployable
- Loosely coupled
- Organized around business capabilities
- Owned by a small team

The microservice architecture enables an organization to deliver large, complex applications rapidly, frequently, reliably and sustainably – a necessity for competing and winning in today's world.

10. Scalability for Instagram System Design

Scalability refers to the ability of an organization (or a system, such as a computer network) to perform well under an increased or expanding workload. A system that scales well will be able to maintain or increase its level of performance even as it is tested by larger and larger operational demands.

We can add more servers to application service layers to make the scalability better and handle numerous requests from the clients. We can also increase the number of databases to store the growing users' data.

These requirements has been covered:

- **Scalability**: We can add more servers to application service layers to make the scalability better and handle numerous requests from the clients. We can also increase the number of databases to store the growing users' data.
- **Latency**: The use of cache and CDNs have reduced the content fetching time.
- **Availability**: We have made the system available to the users by using the storage and databases that are replicated across the globe.
- **Durability**: We have persistent storage that maintains the backup of the data so any uploaded content (photos and videos) never gets lost.
- **Consistency**: We have used storage like blob stores and databases to keep our data consistent globally.
- **Reliability**: Our databases handle replication and redundancy, so our system stays reliable and data is not lost. The load balancing layer routes requests around failed servers.

11. Conclusion

In this article we have discussed about instagram design in which how discussed about the save post and videos, like and dislike the post, show feeds, posting the post or videos, how we can put the hashtags. nstagram's system design is a complex and sophisticated architecture that prioritizes scalability, availability, security, and user experience. The platform's success is not only attributed to its user-friendly interface but also to the robust backend infrastructure that supports its massive user base and dynamic content.

rupa... 4

Next Article

Design Reddit | System Design

Similar Reads

Designing Authentication System | System Design

Keeping your important digital information safe is like building a strong foundation, and the key to that is a good security system. This article will help you understand how to...

10 min read

Designing Parking Lot (Garage) System | System Design

Parking garage plays an important role in today's world, as the number of vehicles is increasing every day, so it has become a major concern to manage them. The parking...

11 min read

How does the process of developing a system differ from designing a system?

System Development vs. System DesignSystem DevelopmentSystem DesignHow Does The Process Of Developing A System Differ From Designing A System?...

7 min read

Complete Reference to Databases in Designing Systems - Learn System Design

Previous Parts of this System Design Tutorial What is System DesignAnalysis of Monolithic and Distributed SystemsImportant Key Concepts and TerminologiesWhat is...

14 min read

Designing Google Maps | System Design

A web mapping tool called Google Maps offers a range of geographic data, such as street maps, satellite photos, streets' aerial views, real-time traffic reports, and...

11 min read

View More Articles

Article Tags : Geeks Premier League System Design Geeks Premier League 2023

Company
About Us
Legal
In Media
Contact Us
Advertise with us
GFG Corporate Solution
Placement Training Program
GeeksforGeeks Community

Languages
Python
Java
C++
PHP
GoLang
SQL
R Language
Android Tutorial
Tutorials Archive

DSA
Data Structures
Algorithms
DSA for Beginners
Basic DSA Problems
DSA Roadmap
Top 100 DSA Interview Problems
DSA Roadmap by Sandeep Jain
All Cheat Sheets

Data Science & ML
Data Science With Python
Data Science For Beginner
Machine Learning Tutorial
ML Maths
Data Visualisation Tutorial
Pandas Tutorial
NumPy Tutorial
NLP Tutorial
Deep Learning Tutorial

Web Technologies
HTML
CSS
JavaScript
TypeScript
ReactJS
NextJS
Bootstrap
Web Design

Python Tutorial
Python Programming Examples
Python Projects
Python Tkinter
Web Scraping
OpenCV Tutorial
Python Interview Question
Django

Computer Science
Operating Systems
Computer Network
Database Management System
Software Engineering
Digital Logic Design
Engineering Maths
Software Development
Software Testing

DevOps
Git
Linux
AWS
Docker
Kubernetes
Azure
GCP
DevOps Roadmap

System Design
High Level Design
Low Level Design

Inteview Preparation
Competitive Programming
Top DS or Algo for CP