Google Doc:

1. How the system should be used? Use Cases
   1. File storage
   2. Online editing and formatting
   3. Collaboration (Concurrency)
      1. There are solutions like (OT) Operational Transformation: The basic idea is to transform each person’s mutation based on its revision and revisions from other collaborators. Examples: ‘abc’: ‘xabc’; ‘ab’🡪’xab’
   4. Access Control (owner, read-only, allow comment etc.)
2. Scaling and Constraints

When scaling system to millions of users, there can be a lot of issues?

* 1. Speed (example: update permission of a folder, the update should be propagated to all its children. Speed can be a concern here.)
  2. Consistency. (Multiple replications…)
  3. Propagation

Twitter

1. Use Cases:
   1. Send Tweets
   2. Following
      1. For example, there are features like suggestions about who to follow.

This is a core feature that plays an important role in user onboarding and engagement.

* 1. Serve feeds
  2. Search feature
     1. Also involves with ranking algorithm:

The most straightforward approach is to give each feature/signal a weight and then compute a ranking score for each tweet. Then we can just rank them by the score. Features can include reply/retweet/favorite numbers, relevance, freshness, users popularity etc.

* 1. Trending topic
     1. Ranking of topic based on frequency, reply/retweet/favorite numbers

1. Data Modeling - use a relational database like MySQL
   1. User object
   2. Feed Object
2. How to scale the system when there are millions/billions of users?
3. How to evaluate the system?
4. How to design the same feature for Facebook (bi-directional relationship)

Instagram (Photo Sharing app)

Two major objects: (Relational database)

* User object

User name, email, registration date

Relations: user follow relations (not a bi-directional relation/user-picture relation

* Picture object

Potential scale issues (what if we have millions of users)

1. Response time: one costly operation is to render users feed

* One approach: upgrade the ranking algorithm (ranking by date and only read the top n most recent pictures with the infinite scroll feature)
* accelerating the picture fetching and ranking is the core

1. scale architecture

For example, we can have database separate from web apps (in different servers) with [load balancers](https://en.wikipedia.org/wiki/Load_balancing_(computing)).

1. Scale database

For this specific problem, we can either do the [vertical splitting (partitioning)](https://en.wikipedia.org/wiki/Partition_(database)) by splitting the database into sub-databases like user database, comment database etc. or [horizontal splitting (sharding)](https://en.wikipedia.org/wiki/Shard_(database_architecture)) by splitting based on attributes like US users, European users.

1. Feed Ranking

* come up with a scoring mechanism, rank everything in chronological order, like/comment numbers, whether the user has liked many photos of the owner and so on. A [linear combination](https://en.wikipedia.org/wiki/Linear_regression) can be used as a starting point due to simplicity.
* Later, more advanced machine learning algorithms like [collaborative filtering](https://en.wikipedia.org/wiki/Collaborative_filtering) is worth to try.

1. Image compression

[Amazon S3](https://aws.amazon.com/s3/)

The point is that images are usually of large size and seldom get updated. So a separate system for image storage has a lot of advantages. For instance, cache and replication can be much simpler when files are static.