News Feed

Variants

- Design Facebook news feed.
- Design Twitter news feed.
- Design Quora feed.
- Design Instagram feed.

Requirements Gathering

- What is the intended platform?
    - Mobile (mobile web or native)? Web? Desktop?
- What features are required?
    - CRUD posts.
    - Commenting on posts.
    - Sharing posts.
    - Trending posts?
    - Tag people?
    - Hashtags?
- What is in a news feed post?
    - Author.
    - Content.
    - Media.
    - Tags?
    - Hashtags?
    - Comments/Replies.
    - Operations:
        - CRUD
        - Commenting/replying to a post.
- What is in a news feed?
    - Sequence of posts.
    - Query pattern: query for a user's ranked news feed.
    - Operations:
        - Append - Fetch more posts.
        - Delete - I don't want to see this.
- Which metrics should we optimize for?
    - User retention.
    - Ads revenue.
    - Fast loading time.
    - Bandwidth.

        - Server costs.

Core Components

TODO

Data modeling

- What kind of database to use?
    - Data is quite structured. Would go with SQL.
- Design the necessary tables, its columns and its relations.
    - `users`
    - `posts`
    - `likes`
    - `follows`
    - `comments`

There are two basic objects: user and feed. For user object, we can store userID, name, registration date and so on so forth. And for feed object, there are feedId, feedType, content, metadata etc., which should support images and videos as well.

If we are using a relational database, we also need to model two relations: user-feed relation and friend relation. The former is pretty straightforward. We can create a user-feed table that stores userID and corresponding feedID. For a single user, it can contain multiple entries if he has published many feeds.

For friend relation, adjacency list is one of the most common approaches. If we see all the users as nodes in a giant graph, edges that connect nodes denote friend relation. We can use a friend table that contains two userIDs in each entry to model the edge (friend relation). By doing this, most operations are quite convenient like fetch all friends of a user, check if two people are friends.

The system will first get all userIDs of friends from friend table. Then it fetches all feedIDs for each friend from user-feed table. Finally, feed content is fetched based on feedID from feed table. You can see that we need to perform 3 joins, which can affect performance.

A common optimization is to store feed content together with feedID in user-feed table so that we don't need to join the feed table any more. This approach is called denormalization, which means by adding redundant data, we can optimize the read performance (reducing the number of joins).

The disadvantages are obvious:

- Data redundancy. We are storing redundant data, which occupies storage space (classic time-space trade-off).
- Data consistency. Whenever we update a feed, we need to update both feed table and user-feed table. Otherwise, there is data inconsistency. This increases the complexity of the system.
- Remember that there's no one approach always better than the other (normalization vs denormalization). It's a matter of whether you want to optimize for read or write.

Feed Display

- The most straightforward way is to fetch posts from all the people you follow and render them sorted by time.
- There can be many posts to fetch. How many posts should you fetch?
- What are the pagination approaches and the pros and cons of each approach?
- Offset by page size
- Offset by time
- What data should the post contain when you initially fetch them?
- Lazy loading approach for loading associated data: media, comments, people who liked the post.
- Media
    - If the post contains media such as images and videos, how should they be handled? Should they be loaded on the spot?

    - A better way would be to fetch images only when they are about to enter the viewport.
    - Videos should not autoplay. Only fetch the thumbnail for the video, and only play the video when user clicks play.
    - If the content is being refetched, the media should be cached and not fetched over the wire again. This is especially important on mobile connections where data can be expensive.

- Comments

    - Should you fetch all the comments for a post? For posts by celebrities, they can contain a few hundred or thousand comments.
    - Maybe fetch the top few comments and display them under the post, and the user is given the choice to "show all comments".

- How does the user request for new content?

    - Infinite scrolling.
    - User has to tap next page.

Feed Ranking

- First select features/signals that are relevant and then figure out how to combine them to calculate a final score.
- How do you show the relevant posts that the user is interested in?

    - Chronological - While a chronological approach works, it may not be the most engaging approach. For example, if a person posts 30 times within the last hour, his followers will have their news feed clogged up with his posts. Maybe set a cap on the number of time a person's posts can appear within the feed.
    - Popularity - How many likes and comments does the post have? Does the user usually like posts by that person?

- How do you determine which are the more important posts? A user might be more interested in a few-hour old post from a good friend than a very recent post from an acquaintance.
- A common strategy is to calculate a post score based on various features and rank posts by its score.
- Prior to 2013, Facebook was using the EdgeRank (https://www.wikiwand.com/en/EdgeRank) algorithm to determine what articles should be displayed in a user's News Feed.
- Edge Rank basically is using three signals: affinity score, edge weight and time decay.

    - Affinity score (u) - For each news feed, affinity score evaluates how close you are with this user. For instance, you are more likely to care about feed from your close friends instead of someone you just met once.
    - Edge weight (e) - Edge weight basically reflects importance of each edge. For instance, comments are worth more than likes.
    - Time decay (d) - The older the story, the less likely users find it interesting.

- Affinity score

    - Various factors can be used to reflect how close two people are. First of all, explicit interactions like comment, like, tag, share, click etc. are strong signals

we should use. Apparently, each type of interaction should have different weight. For instance, comments should be worth much more than likes.
    - Secondly, we should also track the time factor. Perhaps you used to interact with a friend quite a lot, but less frequent recently. In this case, we should lower the affinity score. So for each interaction, we should also put the time decay factor.

- A good ranking system can improve some core metrics - user retention, ads revenue, etc.

Feed Publishing

TODO. Refer to http://blog.gainlo.co/index.php/2016/04/05/design-news-feed-system-part-2/.

Additional Features

Tagging feature

- Have a `tags` table that stores the relation between a post and the people tagged in it.

Sharing feature

- Add a column to `posts` table called `original_post_id`.
- What should happen when the original post is deleted?
    - The shared `posts` have to be deleted too.

Notifications feature

- When should notifications happen?
- Can the user subscribe to only certain types of notifications?

Trending feature

- What constitutes trending? What signals would you look at? What weight would you give to each signal?
- Most frequent hashtags over the last N hours.
- Hottest search queries.
- Fetch the recent most popular feeds and extract some common words or phrases.

Search feature

- How would you index the data?

Scalability

- Master-slave replication.
    - Write to master database and read from replica databases/in-memory data store.
    - Post contents are being read more than they are updated. It is acceptable to have a slight lag between a user updating a post and followers seeing the

updated content. Tweets are not even editable.

- Data for real-time queries should be in memory, disk is for writes only.
- Pre-computation offline.
- Tracking number of likes and comments.
    - Expensive to do a `COUNT` on the `likes` and `comments` for a post.
    - Use Redis/Memcached for keeping track of how many likes/comments a post has. Increment when there's new activity, decrement when someone unlikes/deletes the comment.
- Load balancer in front of your API servers.
- Partitioning the data.

References

- Design News Feed System (Part 1) (http://blog.gainlo.co/index.php/2016/03/29/design-news-feed-system-part-1-system-design-interview-questions/)
- Design News Feed System (Part 1) (http://blog.gainlo.co/index.php/2016/04/05/design-news-feed-system-part-2/)
- Etsy Activity Feeds Architecture (https://www.slideshare.net/danmckinley/etsy-activity-feeds-architecture)
- Big Data in Real-Time at Twitter (https://www.slideshare.net/nkallen/q-con-3770885)