System Design Planning

1. Requirement gathering
    a. Functional
    b. Non-functional
2. Component Architecture (High level)
3. Data API's and Protocol
4. Data Entities
5. Data Store
6. Optimization and Performance
7. Accessbility
8. Security

Functional Requirement

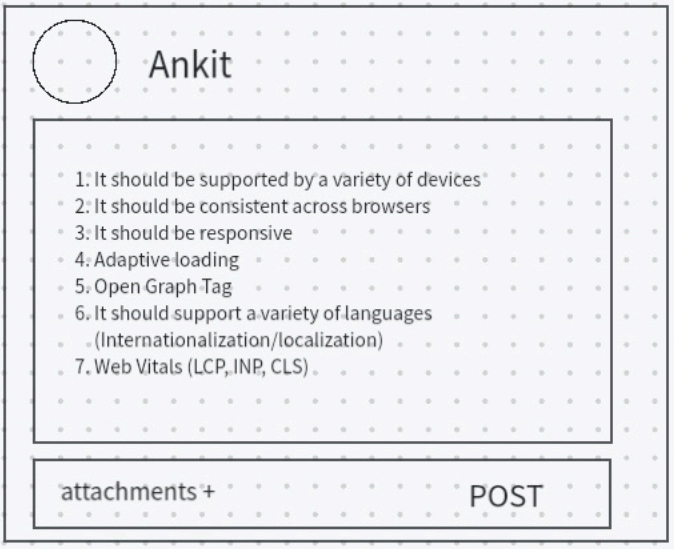
1. User can able to post (text, images, emojis)
2. User can able to delete the post
3. User can comments on Post
4. User can delete his comment
5. User can able to see his old Post in profile section
6. User can update his details.

Non-Functional Requirement

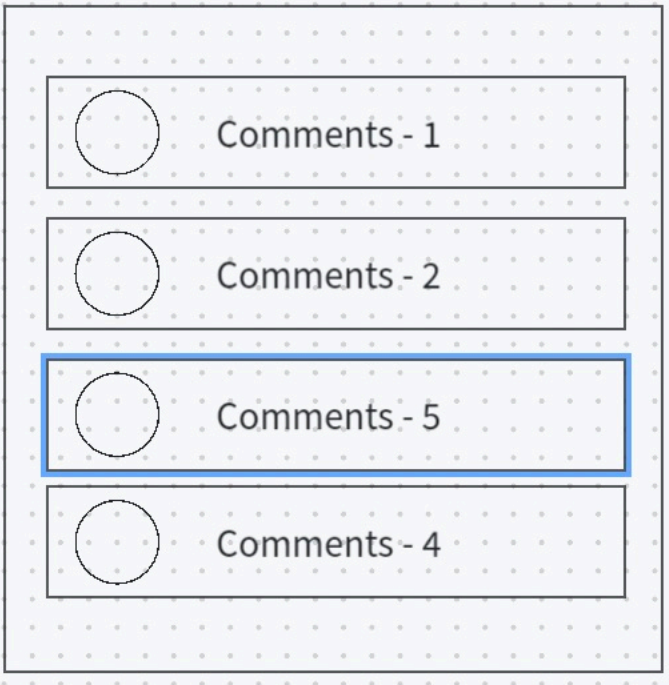
1. It should be supported by a variety of devices
2. It should be consistent across browsers
3. It should be responsive
4. Adaptive loading
5. Open Graph Tag
6. It should support a variety of languages (Internationalization/localization)
7. Web Vitals (LCP, INP, CLS)

Component Architecture

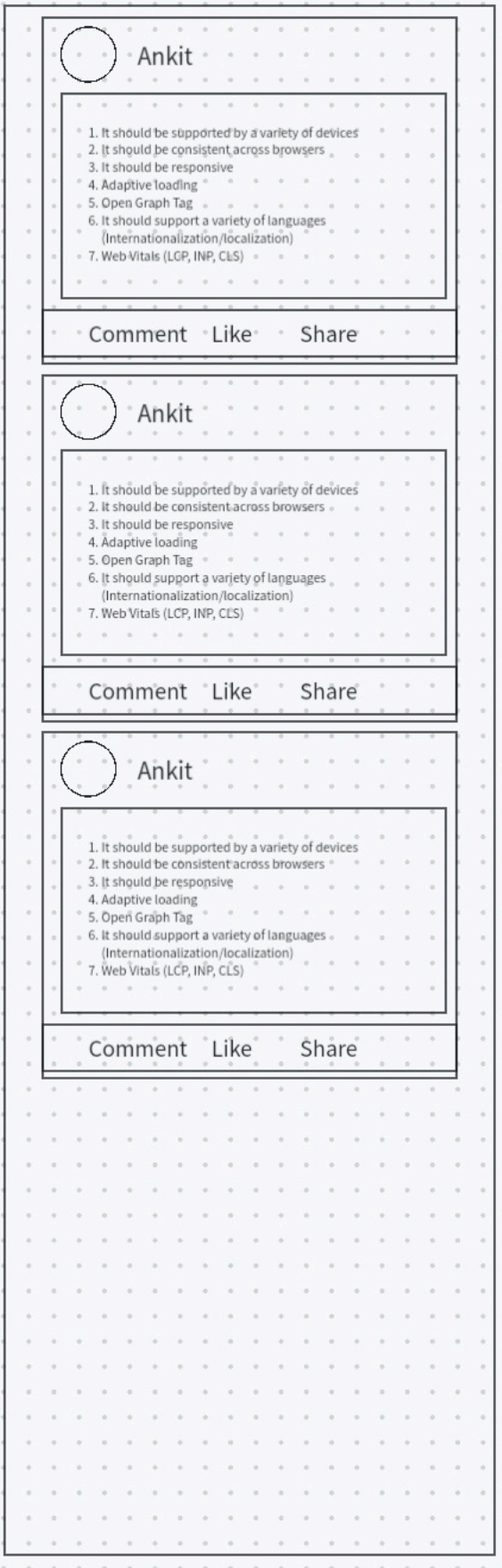
Post Component

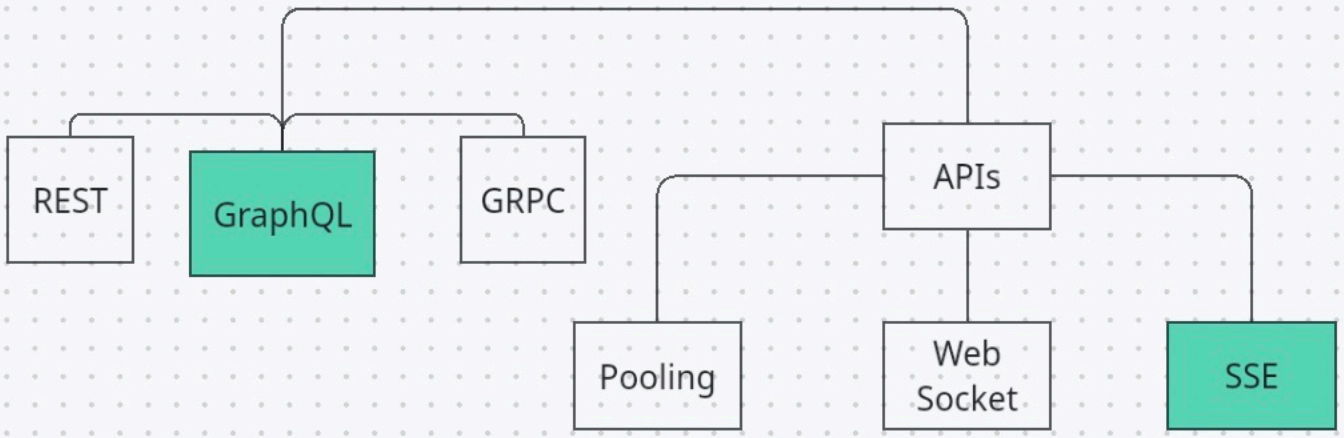


Comment Component



News Feed Component






```
type Post = {
    id: String
    origin: String
    metaData: String
    attachments: [Links]
    comments: [Comments]
    interactivity: {
      likesCount: Number
      shareCount: Number
    }
}
```

```
type Comment = {
    id: String
    origin: String
    metadata: String
    userId: String
}
```

```
type User = {
    id: String
    nickName: String
    address: Address
    mobNumber: String
    emailId: String
    joinedDate: DateTime;
}
```

```
type Address = {
    address1: String
    address2: String
    landmark: String
}
```

1. HTTP benifits
2. Easy to implement
3. Uni directional

Making un-necessay calls
Incraese server loads
Not for scaling purpose

1. ws:// or wss:// protocol
    a. ws -> web socket protocol
    b. wss -> web socket secure protocol
2. Bi-directional

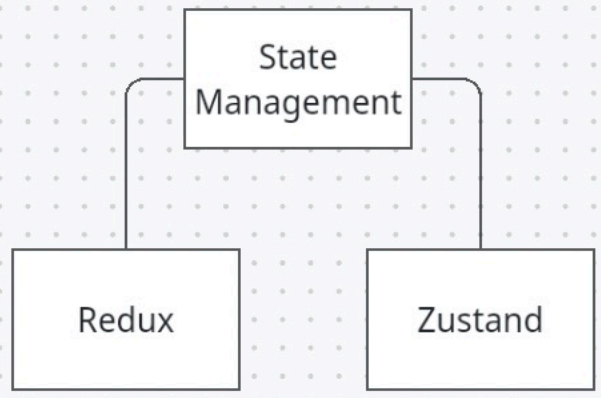
1. Need to open the connection from both side
2. It will use device resource
3. Firewalls and Proxy , they are looking the web socket connection

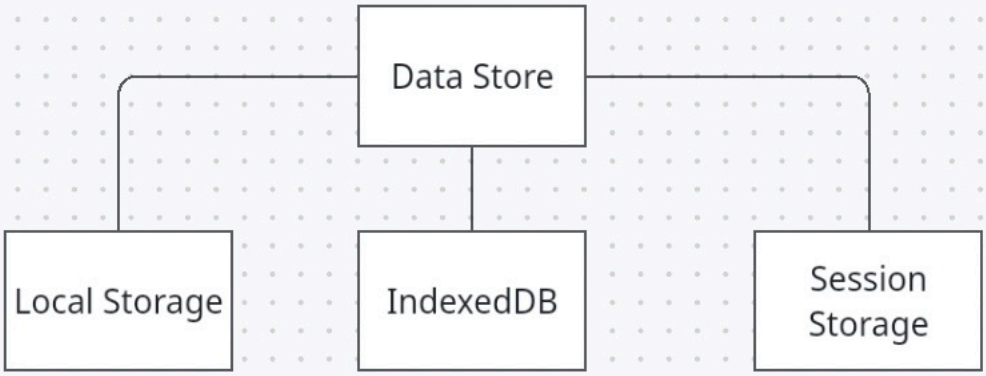
1. HTTP benifits
2. Unidirectional
3. Server -> Client
4. Server automatically rety if connection get lost
5. Easy to load balance

We can't close the connection

High Level APIs

1. Response createPostAPI(token, PostDetails)
2. Response deletePostAPI(token, post_id)
3. Response makeComment(token, post_id, Comments)
4. Response deleteComment(token, post_id, comment_id)
5. Response editComment(token, comment_id, Comments)
6. List<Post> getAllPost(token)
7. Response UpdateUserProfile(token)
8. Response likePostAPI(token, post_id)





1. Persistant across tabs
2. It stays in your browser for long period, until you are not deleting by yourself
3. Easy to manage local storage

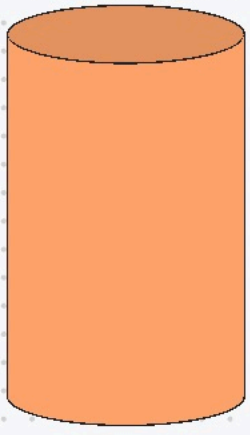
We can't store large set of data

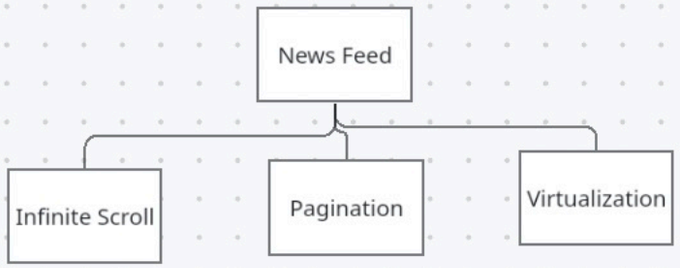
1. Persistant across tabs
2. It stays in your browser for long period
3. Easy to manage IndexedDB
4. We can store the large data set also.

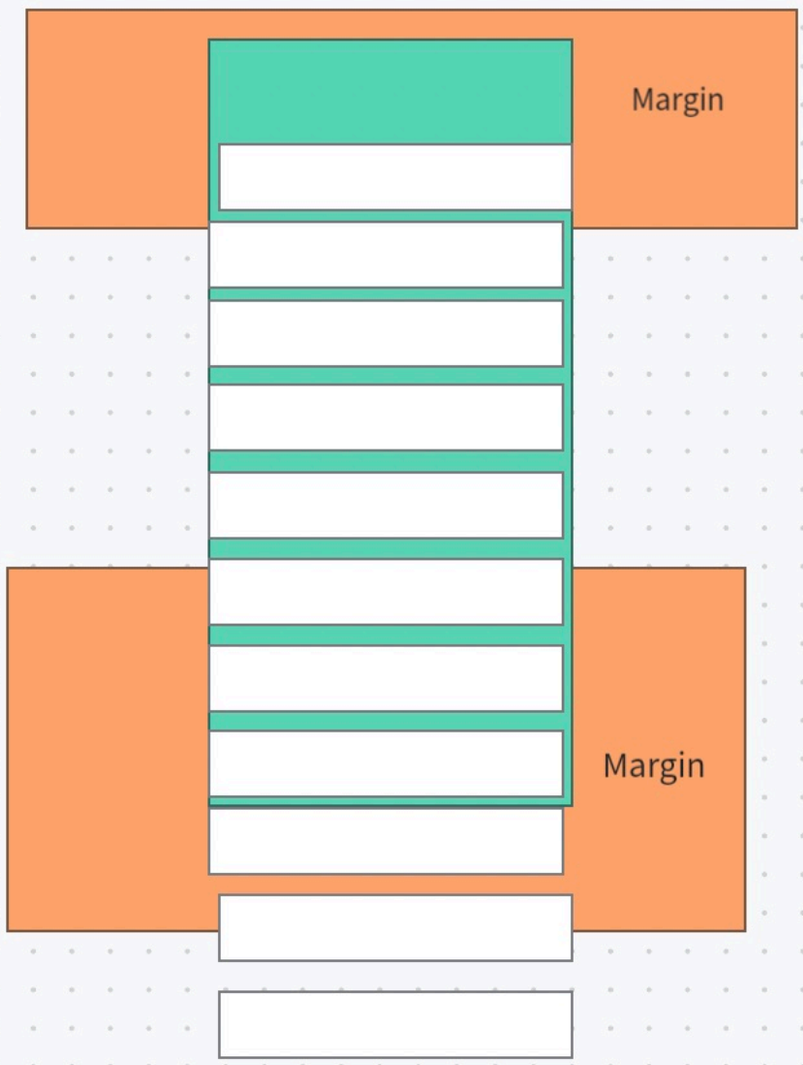
1. Not persistent across tabs
2. Data goes off once we close the tab
3. Memory size is small
4. We can't store the large data set.

Data Normailization

```
{
  posts: {
    [post_id]: PostDetails
    [post_id_2]: PostDetails
  }
  comments: {
    [comment_id]: Comment
    [comment_id2]: Comment
  },
  User: {
    user_id: User
    user_id2: User
  }
}
```

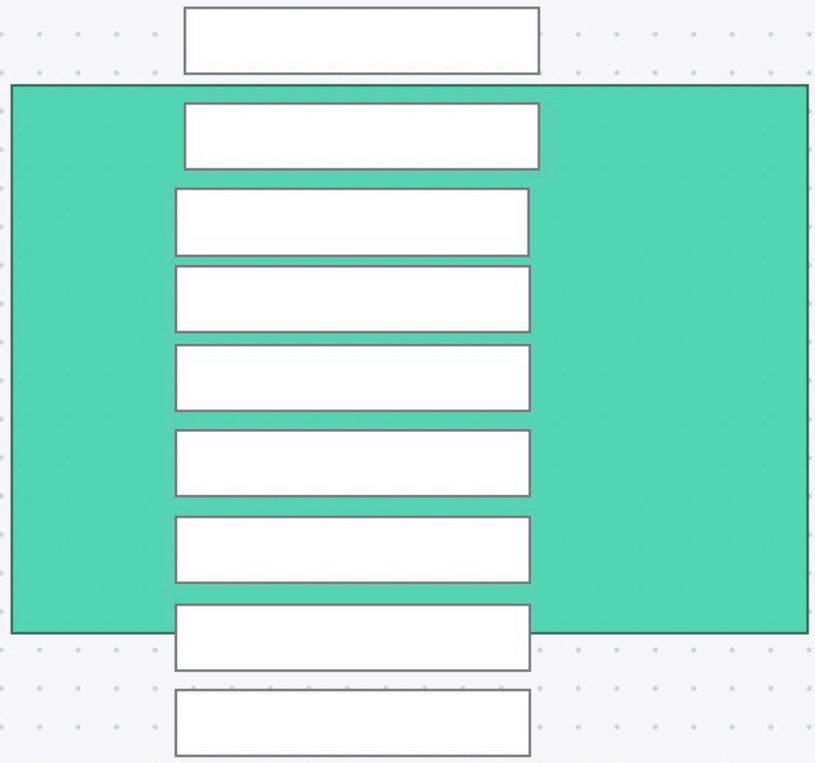






Pagination
1. Off set based (page number and count per page)
2. Cursor Based Pagination

Virualization



```
const infiniteScrollObsever =  new IntersectionObserver(elements) {
   const lastElement = elements[0];
   const { isInterSecting} = lastElements;
   if(isIntesrSection) {
        fetchNewStories()
        observerNewLastElement()
     }
}
```

Optimization

Asset Optmizataion

Network Optimization

1. Compression Technique
2. Loading in async fashion
3. Chaching
4. Tree Shaking
5. Bundling
6. we can leverage the resource hinting (pre-connect, preload, predns-connect)
7. User debouncing and throttling for network calls

JavaScript

1. Lazy loading
2. Code splitting
3. Tree shaking
4. Load files in async fashing with the help of "async" or "defer"
5. Web worker - to perform heavy task
6. Prevent memory leaks
7. Try to remove event listener if works id done

CSS

1. Try to load in Async fashion media="print"
2. We can load CSS with help of media queries
3. Serving critical CSS with the help of rel="preload"
4. We can load CSS with the help of JavaScript
5. Tree Shaking
6. Compression

Images

1. Compress Images
2. Fallback Images
3. Blur effecet technique
4. CSS Sprites
5. Lazy loading
    a. lazy="loading"
    b. lazy="eager"
6. Adaptive loading
    a. Based on the network speed will load , different quality of images
7.

Fonts

1. Woff / WOff2 format
2. Fallback fonts
3. Prefer to use those fonts which is supported by Browser
4. rel="preload" or "pre-connect"
5. Try to avoid FOIT and use FOUT
    a. FOIT - flashing of invisible text
    b. FOUT - flashing of unstyled text

Accessibility

1. Using semantic HTML
2. Provindg Screen Reader
3. Keyboard Shortcuts
4. Colors contrast
5. Zoom in / Zoom out effect