Design similar to google drive / dropbox

Planning

1. Requirement gathering
    a. Functional
    b. Non-functional
2. Component Architecture
3. Data APIs and protocol
4. Data Entities
5. Data Store
6. Performance and Optimizataion
7. Accessibility
8. Security

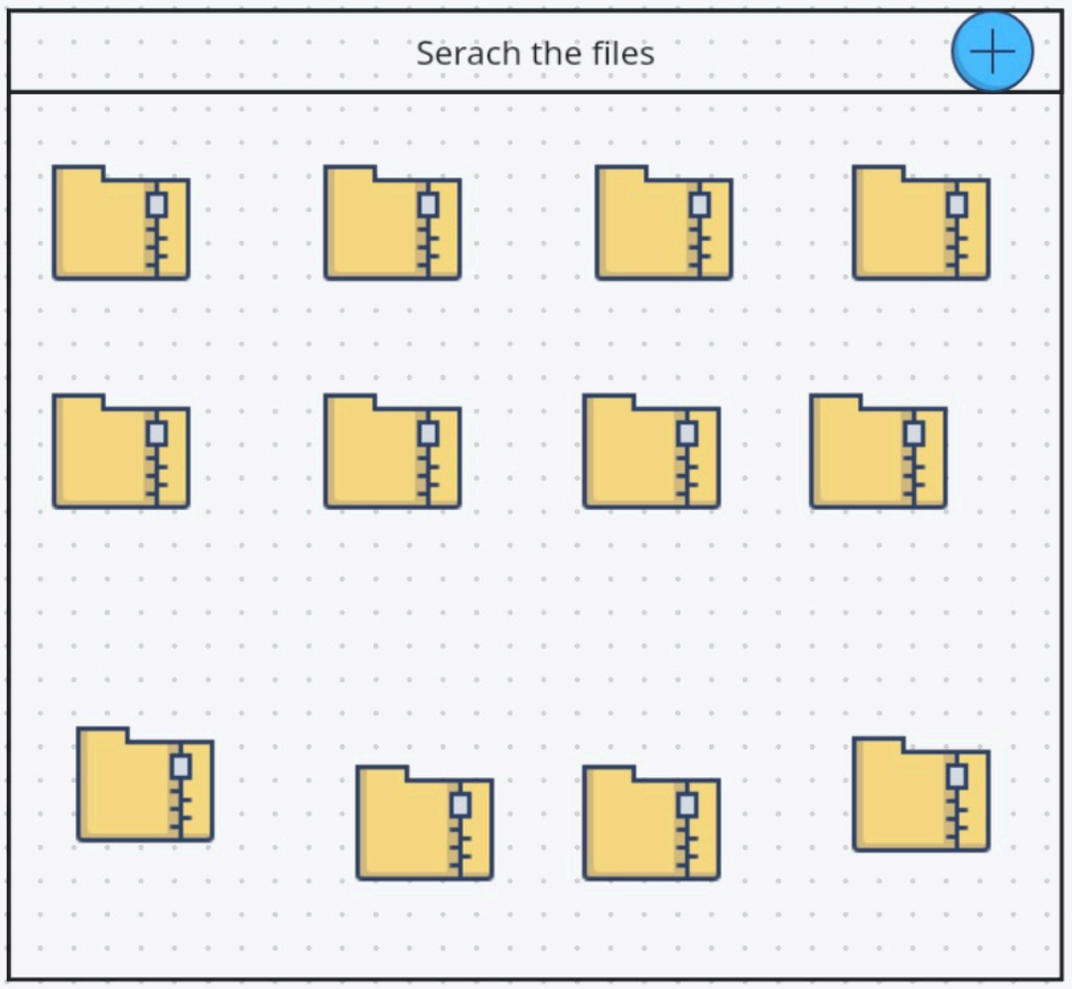
Functional Requirement

1. User can able to upload files
    a. images
    b. videos
    c. files
    d. pdf etc
2. User can able to download the files
3. User can able to share with other.
4. User can ablet to search teh files
5. User can able to sort and change the view.

Non-Functional Requirement

1. It should be consistent across browsers
2. It should support a variety of devices
3. Accessbility
4. Logging the runtime errors
5. Offline support

Component Architecture



Drag-and-drop support
• Search functionality
• File preview (PDF, images, video, etc.)
• Sharing (link-based, email-based, permission levels)
• Real-time updates (if others rename/add files)
• Offline access (optional)
• Concurrency conflict handling (last-write-wins, operational transformation, etc.)

3. **Frontend Architecture**
Talk about component design and structure:
• **Component hierarchy:** Sidebar (folders), Main area (files), Toolbar, Modals
• **State management:**
    • Global (Redux / Zustand / Context API)
    • Local (useState, useReducer)
    • Query management (React Query / SWR)
• **Routing:** For folder navigation (/folder/:id)
• **Lazy loading:** Components and file previews

4. **Handling Large File Uploads**
• Chunked file uploads with progress
• Use of FileReader, Blob, and possibly Web Workers
• Use of headers like Content-Range
• Retries and resumable upload support (like Google Drive)
• File type and size validations on client side

For providing the drag and drop these are the event listener which we can use

dropArea.addEventListener("**dragenter**", handleDragEnter);
dropArea.addEventListener("**dragover**", handleDragOver);
dropArea.addEventListener("**dragleave**", handleDragLeave);
dropArea.addEventListener("**drop**", handleDrop);

draggableItem.addEventListener("**dragstart**", handleDragStart);
draggableItem.addEventListener("**dragend**", handleDragEnd);

FOr Preview
1. FIles or PDF we can use "FileReader()"
2. For video we can use "html video"
3. For audio we can use "html audio"

7. **Performance Optimizations**
• Virtualization (react-window, react-virtualized) for large file lists
• Code splitting (React.lazy)
• Debounced search
• Caching and memoization
• Web Workers for background processing

9. **Security Considerations**
• CSP headers
• XSS and CSRF protection
• Token handling and renewal
• File type validations

10. **Versioning / History (Bonus)**
• Keeping file version history (especially if asked)
• UI for listing and restoring versions

How we are going to render the UI ?

CSR

1. Inital page load is slow
2. SEO is lesss
3. Server load is less
4. More number of JS files or UI code can degrade the performance
5. Tech - React.js, Angular, Vue etc.

SSR

1. Inital page load is fast
2. SEO is high
3. Server load is high
4. We need hydration
5. Tech - Next.js or Nuxt.js.

Why SSR has higher SEO ?

As we know, every search engine has their on web crawler which reads the content of website and do the ranking
but In case of CSR
** ** as we know initally we have this div and once the JS files gets downloaded after that we put all the elements inside.
Suppose web crawler came to cehck the content, but that time or JS is not yet downlowded So, our website content he will not be able to read that's why CSR has less SEO.

API architecture

REST Approach

1. It follows HTTP/HTTPs protocol
2. It is easy to implement
3. It has multiple endpoints
4. Problem of overfetching and undefetching
5. Versioning-related problem
6. Method - GET, POST, PUT, DELETE.

GraphQL Approach

1. It follows HTTP/HTTPs protocol But it also follow wss protocol
2. It is not very much easy to implement because we have to implement the graphql server also and write their own schema and so on.
3. It has one endpoints
    a. /graphql
4. No Problem of overfetching and undefetching
5. No Versioning-related problem
6. Method - POST.

Data Entities

```
Type Folder = {
    id: String
    name: String
    type: String
    userId: String
    uploadedAt: DateTime
    access: [User]
    size: String
}
```

```
Type File = {
    id: String
    name: String
    type: String
    userId: String
    uploadedAt: DateTime
    access: [User]
    size: String
}
```

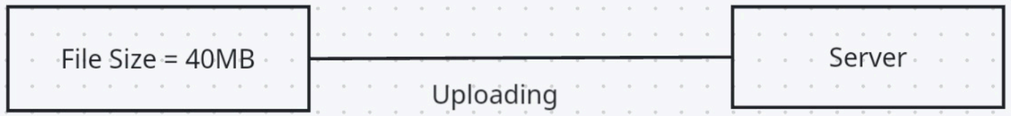
```
Type User = {
    id: String
    name: String
    email: String
}
```

1. Local Storage
2. Session Stoarge
3. IndexedDB

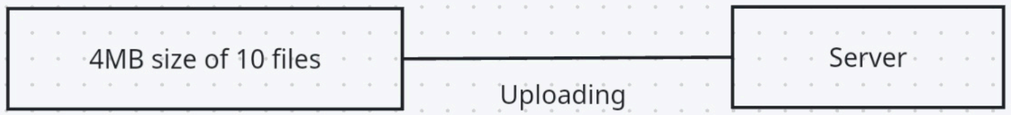
Data Store
1. Local Storage
2. Session storage
3. IndexDB

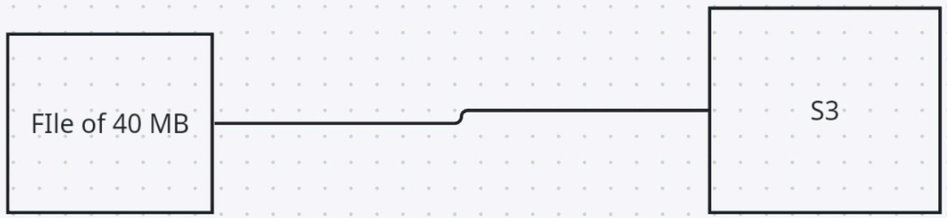
How we are going to upload the videos to server?

How we are going to upload the videos to server?

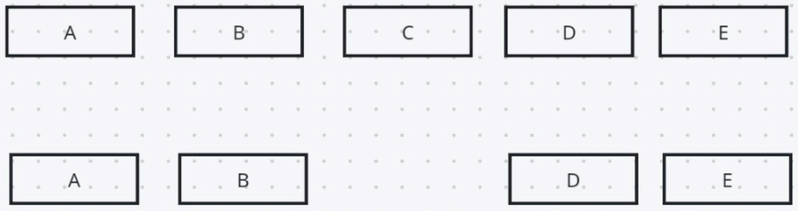


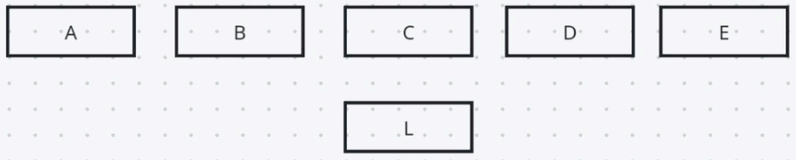
IT will take more bandwith
It will increase Latency





Here to problem is for one files In the server we are taking the size of 160MB which is very expensive





When we will upload larges files, we can use "web worker" to run that process in background.

After this we can talk about, client side performance and optimisation
After that Accessibility and Security