**Full Stack Developer Assignment**

**Time Limit: 3 Days**

**Project Overview**

You are tasked with developing a Full Stack ReactJS NodeJS application with enhanced file management capabilities. The application will consist of four main sections:

1. Top Section: File upload and folder creation with folder breadcrumbs and a context menu.
2. Left Section: Folder and file hierarchy with an ongoing upload progress section.
3. Middle Section: Expanded folder and file structure.
4. Right Section: Document viewer for the selected file.

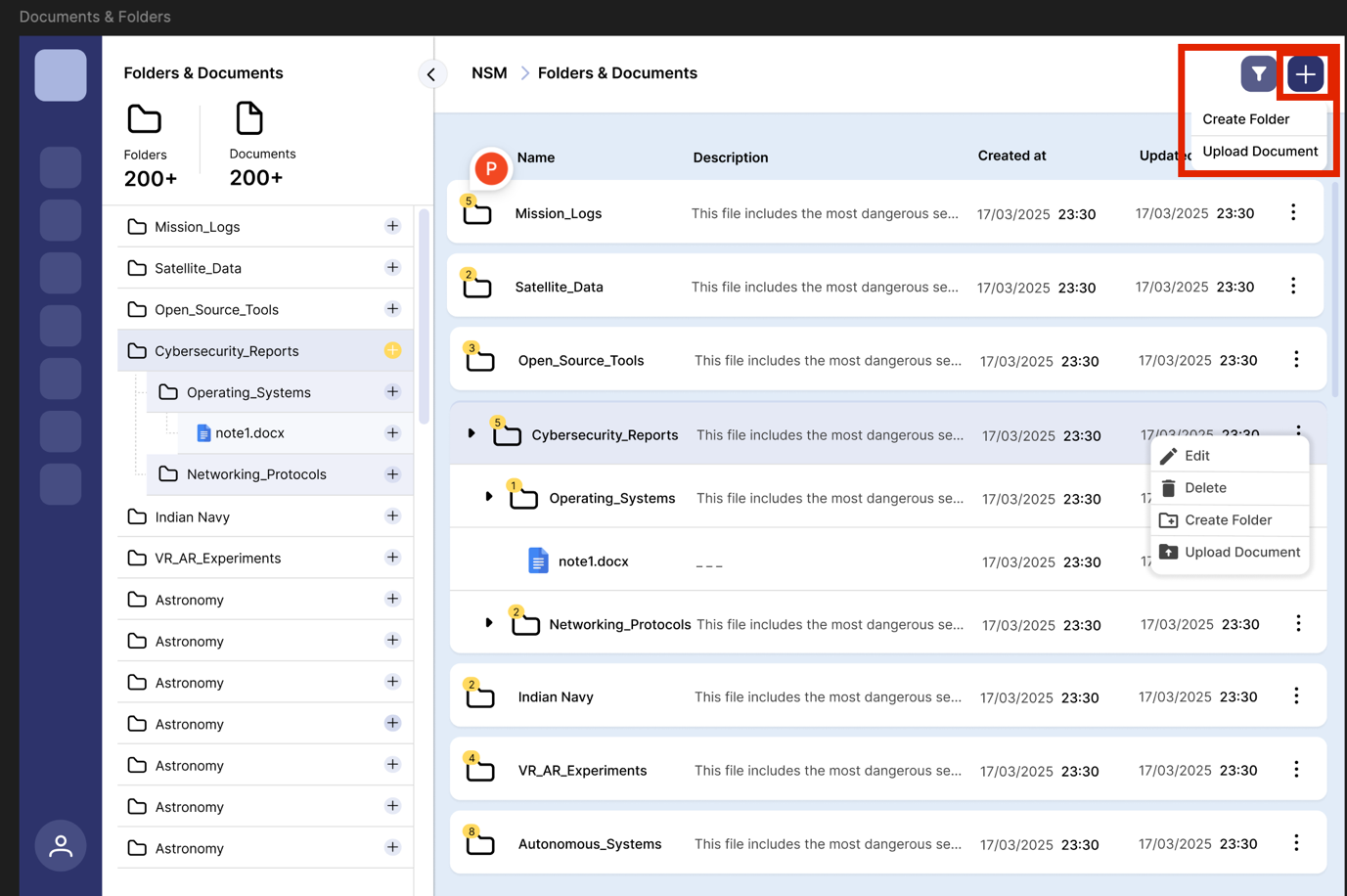
Figma file link: <https://www.figma.com/design/pRDbNDq47eeTX6XSCk3zem/NSM?node-id=3-2910&t=QwNrlztGwpVfg0R7-0>

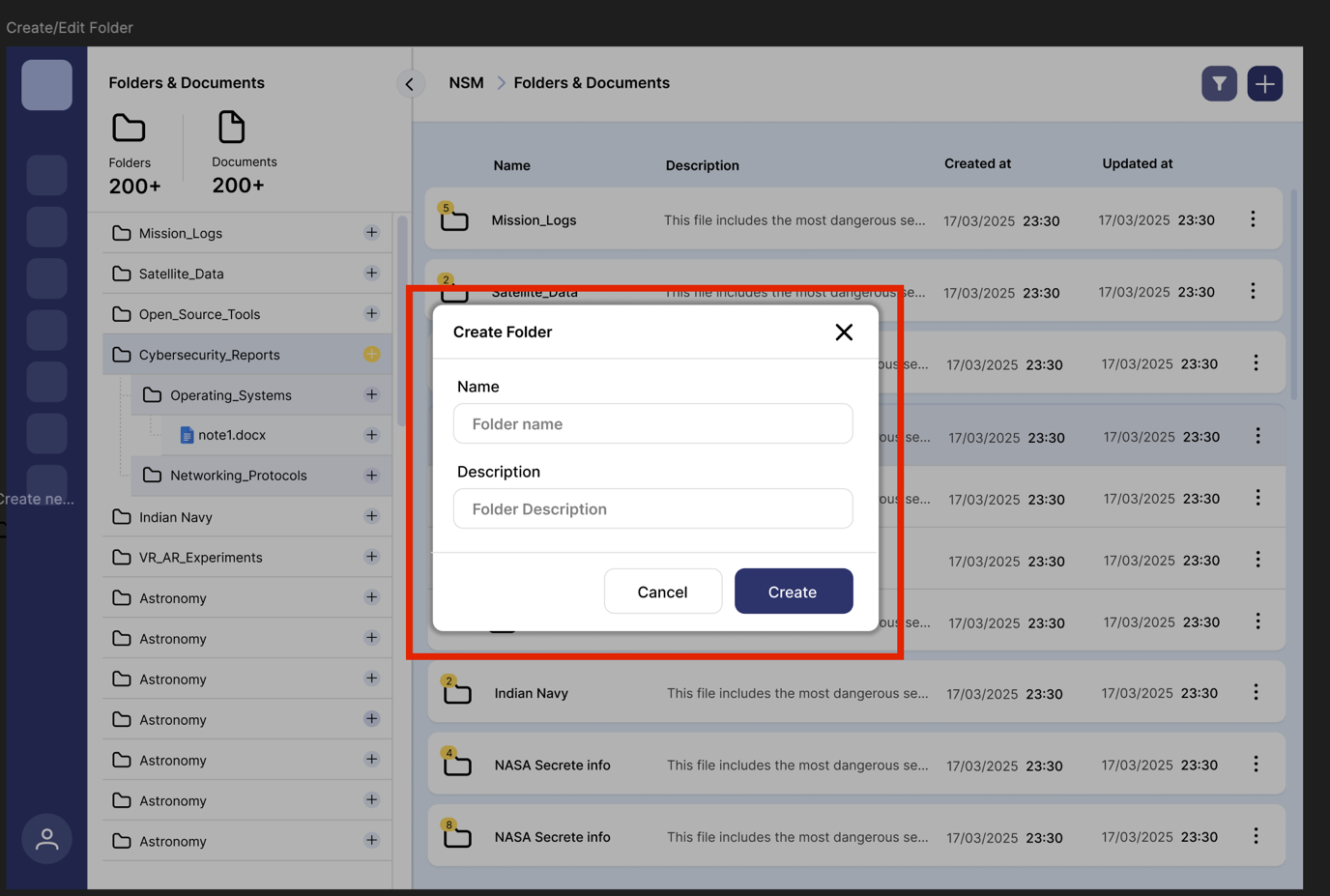
This assignment requires both frontend and backend implementation, with a focus on using a database (MongoDB with Mongoose or any SQL database) to manage folder and file structures.

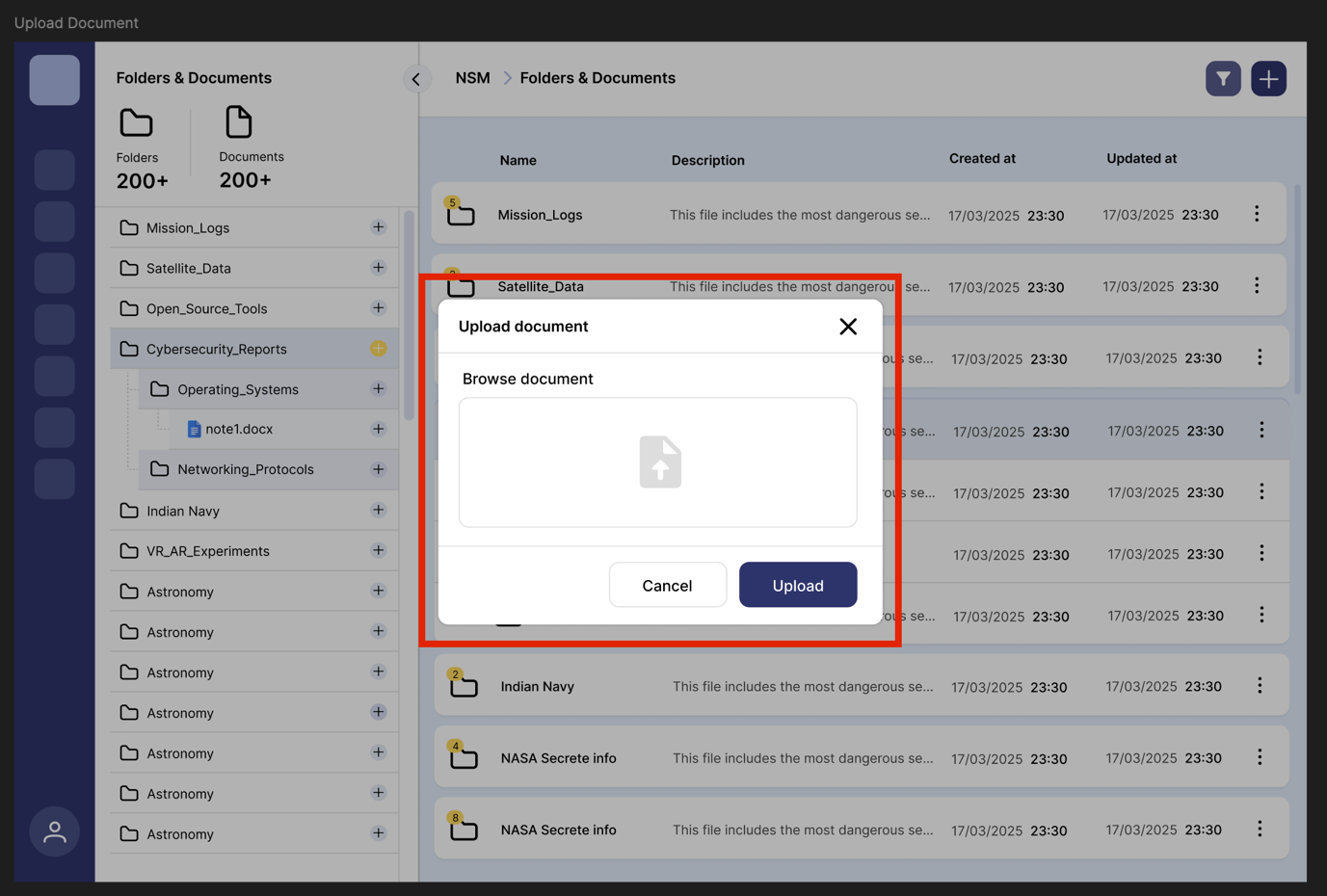
**Requirements**

1. Top Section: File Upload and Folder Creation

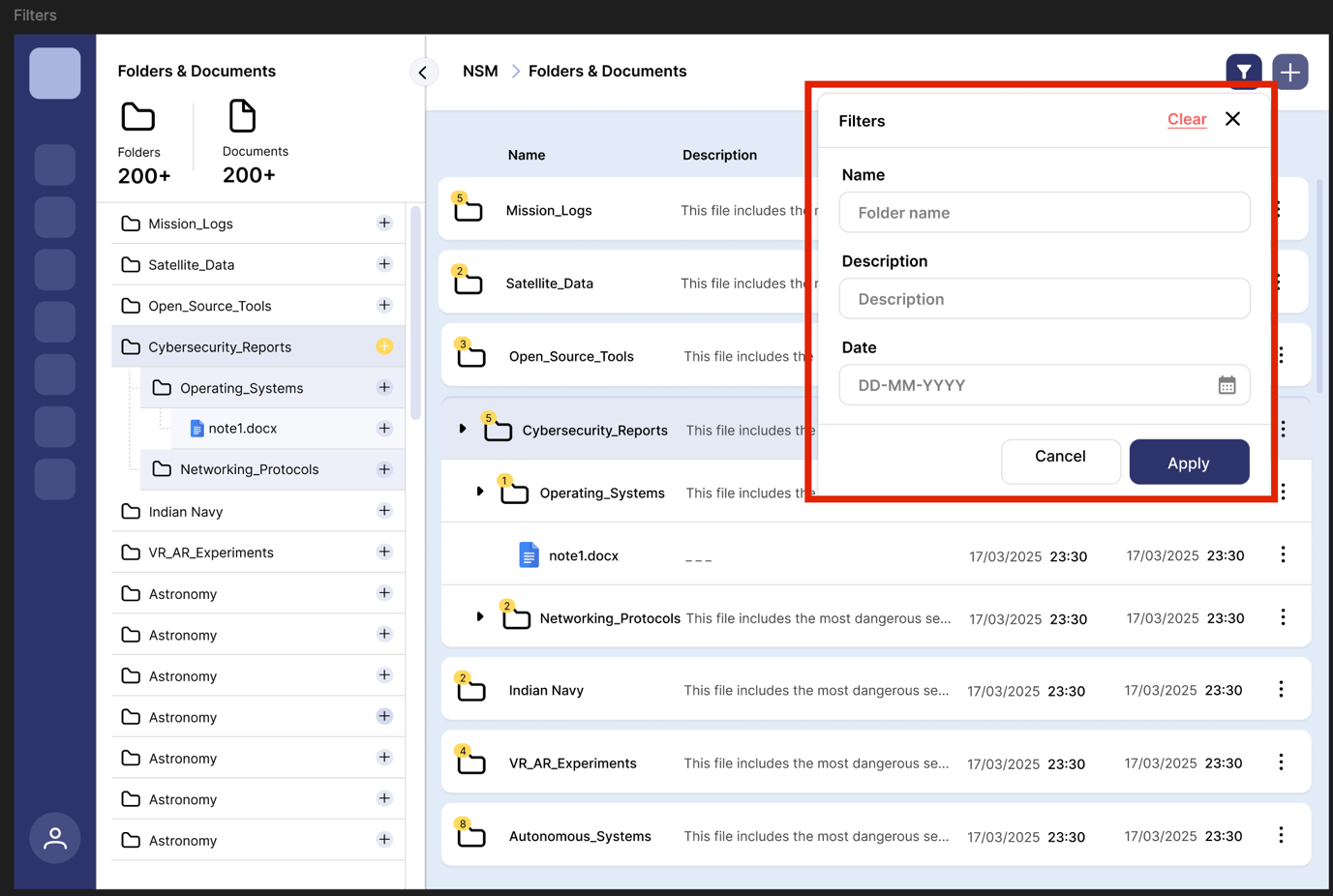
* Breadcrumb Navigation: Display a breadcrumb showing the current folder path.
* Context Menu (+ Button): A "+" button next to the breadcrumb opens a dropdown with two options:
  + Create Folder: Opens a modal prompting for a folder name. Upon saving, the folder is created in the current directory (backend integration required).
  + Upload File: Opens a modal prompting for a file name. Upon upload, the file is uploaded to the current directory (backend integration required).

*Option to create File/Folder*

*Create Folder Pop-up*

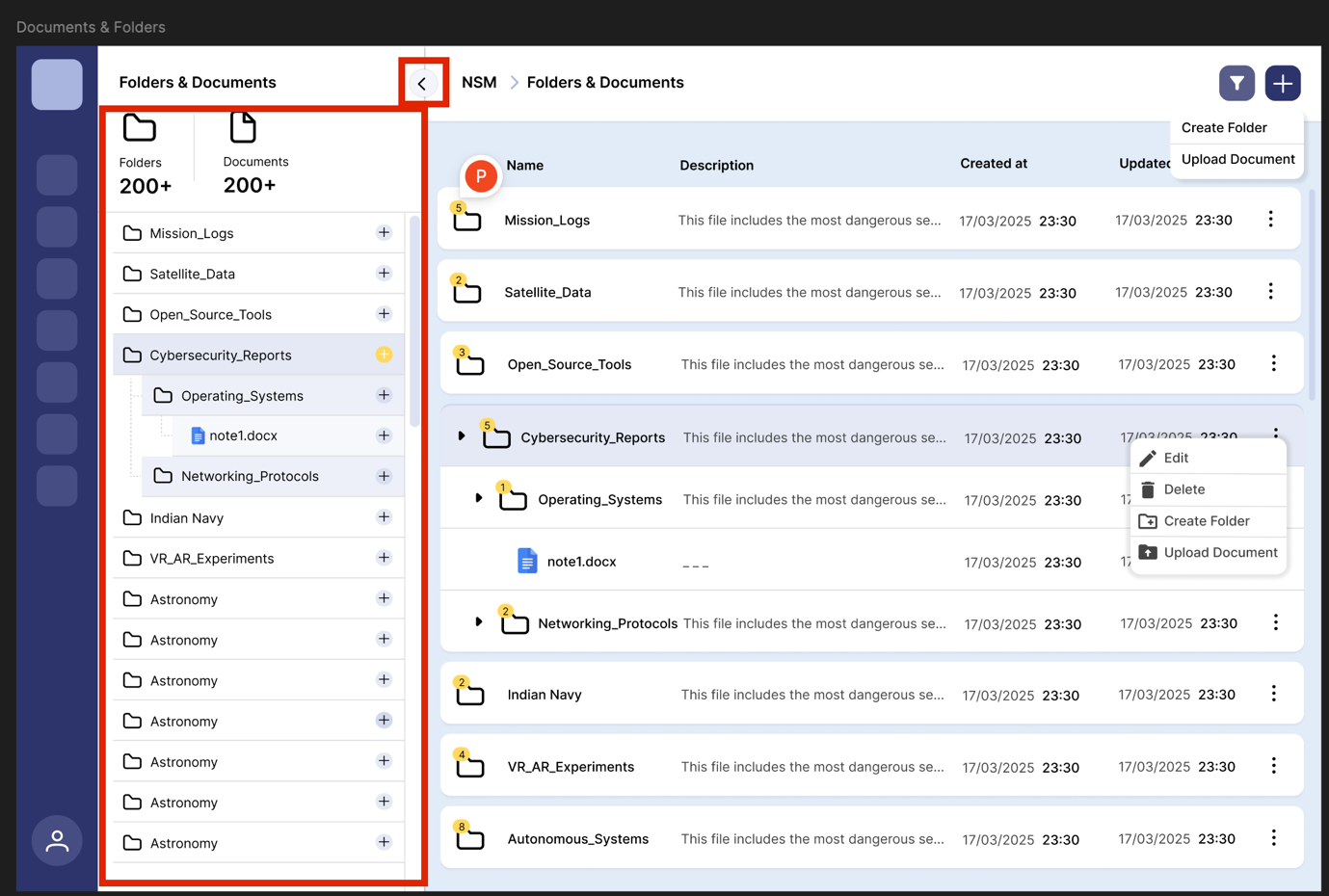
*Upload document pop-up*

* Database Integration: Newly created folders and uploaded files are saved in the database and dynamically added to the folder structure in the left and middle sections.
* File/Folder Filtering: An input field is provided in the top section to filter files and folders *within the current directory* based on name, description, created date. Filtering should happen dynamically as the user types for search inputs.

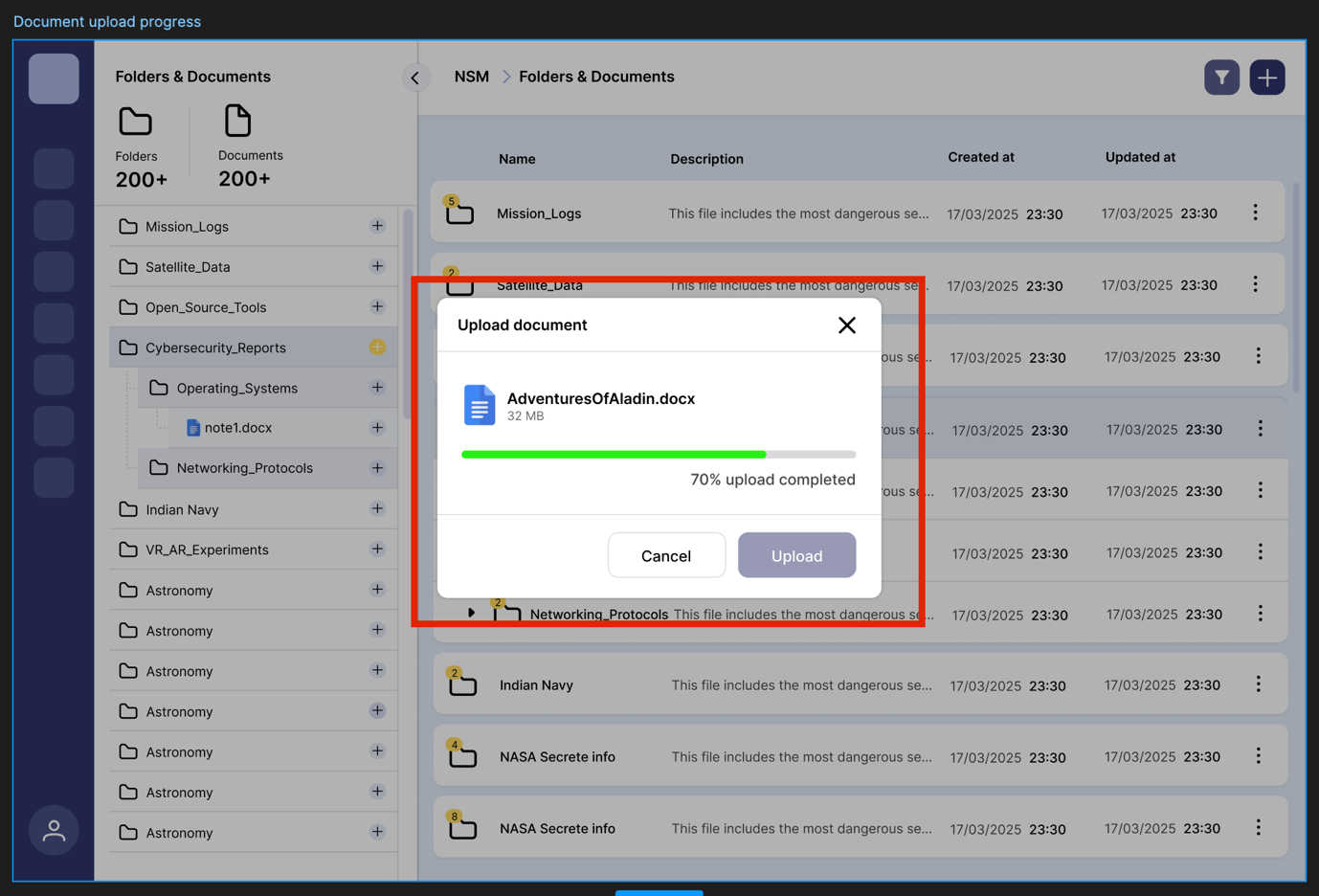
*Filters*

1. Left Section: Folder and File Hierarchy

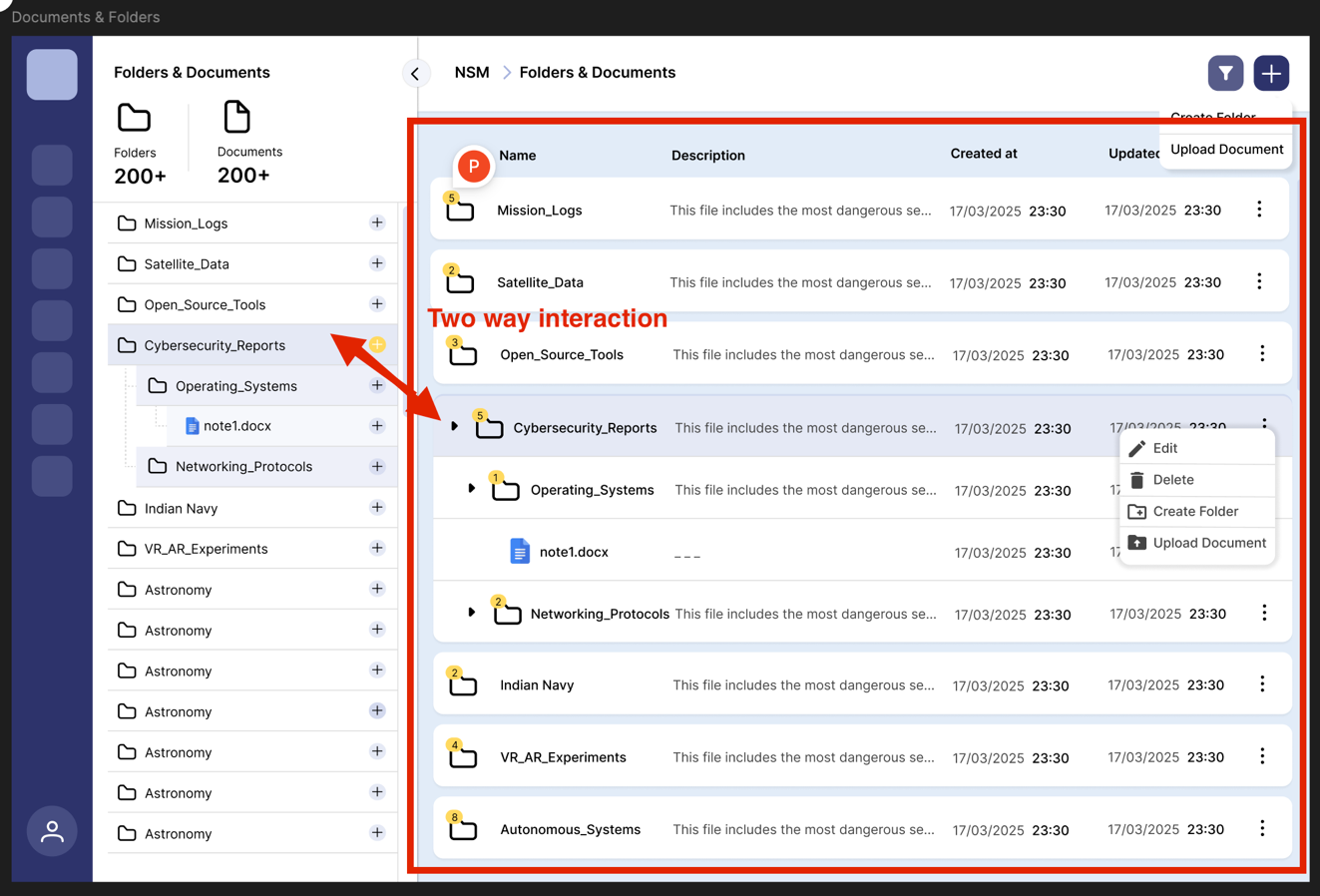
* Hierarchical View: Display a hierarchical view of folders and files fetched from the database.
* Expand/Collapse Folders: Clicking on a folder expands/collapses its contents.

*Left Hierarchy Tree with collapsable menu button on top left*

* Upload Progress Section: A dedicated section at the bottom displays ongoing upload progress using EventSource (SSE). Progress is shown as a percentage for each file.

*Upload Document Progress*

1. Middle Section: Expanded Folder and File Structure

*Middle Section of Detailed Hierarchy*

* Detailed Hierarchy: Display the detailed hierarchy of the selected folder.
* Dynamic Updates: Clicking a folder in the left section updates this section and vice-versa (Implement opening and closing of the left menu by click on arrow button, and opening the left folder tree menu which should also open the tables row in the middle section using a global state management tool like redux or context api.) There is two-way interaction. So, if a folder is clicked on middle section, it opens the corresponding folder in the tree menu as well.
* File Selection: Clicking a file opens it in the right section using an iframe.
* Pagination: Implement pagination for the file and folder list in this section. Display a fixed number of items per page (e.g., 10) and provide controls to navigate between pages. The backend should handle the pagination logic.

1. Right Section: Document Viewer

* File Display: Display the selected file in an iframe (on click of file icon or title).
* Supported File Types: Support PDF, images, and text files.
* Dynamic Rendering: Updates when a new file is selected.

**Technical Requirements**

**Frontend:**

* Framework: ReactJS (Create React App or Vite).
* Component-Based Architecture: Reusable components (FileUpload, FolderTree, Breadcrumb, ProgressBar, DocumentViewer, Modal).
* State Management: React Context API or Redux.
* Responsive Design: Mobile and desktop views.
* Lazy Loading: For folder contents.
* Search and Filter: In the folder/file list.
* Modals: For folder creation and file upload, triggered from the top section's context menu.

**Backend:**

* Database: MongoDB with Mongoose or SQL database (MySQL, PostgreSQL).
* RESTful APIs:
  + Upload files.
  + Fetch folder/file structures.
  + Stream upload progress (EventSource).
  + Create Folder.
  + Fetch paginated data: Accepts parameters for current page and items per page. Returns requested data and total item count.
  + Fetch filtered data: Accepts a search term and folder ID. Returns filtered data.
* File Storage: Store files on the server and metadata in the database.
* Error Handling: Meaningful error messages.

**Styling:**

* CSS, SCSS, or CSS-in-JS (Styled Components).
* Consistent design matching the provided mockup.

**Testing:**

* Integration tests (user flows).

**Additional Features**

* Real-Time Updates: WebSockets or SSE for folder/file changes.
* Robust Error Handling: File uploads, database, API requests.
* Accessibility: WCAG guidelines.

**Submission Guidelines**

* **GitHub Repo: Include complete source code, Postman/Swagger documentation, and a README file with:**
* Architecture decisions.
* Setup instructions.
* Documentation: Inline comments.

**Evaluation Criteria**

* Code Quality: Clean, modular, maintainable.
* Functionality: All requirements met.
* UI/UX: Intuitive, responsive and exactly matching with figma design.
* Testing: Comprehensive coverage.
* Database Integration: Efficient schema.
* Real-Time Features: Effective implementation.

**Mockup and Assets**

Attached figma file and icons to be used are material icons. The UI needs to be as close as possible to the figma file.

**Key Notes**

* Backend is Mandatory: Node.js, Express, database (MongoDB or SQL).
* Focus on Real-Time Features: EventSource/WebSockets.
* Database Design: Efficient folder/file hierarchy handling.
* Error Handling: Graceful error handling and user feedback.
* Modal Implementation: Implement modals for folder creation and file upload.
* Pagination Implementation: Ensure both front-end and back-end pagination logic is correct.
* Filtering Implementation: Implement filtering by name within the current directory.