Dynamic Document Builder Website

You are tasked with creating a web application called "Dynamic Document Builder" that allows users to create, manage, and preview structured documents with dynamic sections. The application should generate a markdown file as output and support loading existing markdown files to populate the form. Below are the detailed requirements and expectations for this assignment.

1. Technology

- **UI/Styling:** Use of Tailwinds, Bootstrap, UIkit, Pure for UI templating.
- **Framework**: Use a framework like React, Vue.js, Svelte, HTMX or Angular.
- **Back-End (Optional)**: If you choose to implement file handling or parsing on the server side, you may use a language/framework of your choice (e.g., Node.js, Python Flask/Django) or NO-SQL database.
- **File Handling**: Handle image uploads and markdown file parsing either client-side (e.g., using the File API in JavaScript) and/or server-side.
- **Error Handling**: Include basic validation (e.g., section names cannot be empty) and user feedback (e.g., "File loaded successfully").

2. Functional Requirements

2.1. Form with Dynamic Sections

- Create a form where users can dynamically add or remove text sections.
- Each section should have a text input field to allow the user to assign it a custom name (e.g., "Introduction", "Chapter 1").
- Provide a mechanism (e.g., buttons or drag-and-drop) to make a section a subsection of another section, creating a hierarchical structure.
- Sections can optionally include diagrams uploaded as image files (e.g., `.png`, `.jpg`). Users should be able to upload an image for a section.

2.2. Document Preview

- Include a "Preview" button or option that displays a real-time preview of the document. The preview should reflect the current state of the form, including:
  - Section names as headings.
  - Hierarchical structure (e.g., subsections indented or styled differently).
  - Uploaded images embedded in the appropriate sections.

2.3. Form Submission and Output

- When the user submits the form, the application should generate:
  - A markdown file (e.g., `output.md`) containing the structured content:
    - Section names as markdown headings (e.g., `# Introduction`, `## Subsection`).
    - Images referenced in markdown syntax (e.g., `![Diagram](image1.png)`).
    - Proper nesting of subsections.
  - A metadata file (e.g., `metadata.json`) containing additional details, such as:
    - Total number of sections.
    - Hierarchy structure (e.g., a tree-like JSON object representing parent-child relationships).
    - File paths or references to uploaded images.
- The generated files should be downloadable by the user.

2.4. Loading Markdown Files

- Provide an option for users to upload a previously generated markdown file (and optionally its metadata).
- Parse the markdown file and populate the form with the sections, subsections, and images as they were originally structured.
- Ensure the form reflects the hierarchy and content accurately.

3. Deliverables

1. **Source Code**: Host the complete source code for the web application in a Git repository (e.g., GitHub, GitLab).
2. **README**: Include a README file with:
   - Instructions to set up and run the project locally.
   - A brief explanation of your design choices (e.g., frameworks used, approach to hierarchy management).
   - Any assumptions made during development.
3. **Demo**: Plan to demonstrate the tool in the next session

4. Example Workflow

- User opens the website and sees an empty form with an "Add Section" button.
- User adds a section named "Introduction" and uploads an image (`intro_diagram.png`).
- User adds another section named "Details" and marks it as a subsection of "Introduction".
- User clicks "Preview" and sees:
- `# Introduction`
- `![Diagram](intro_diagram.png)`
  - `## Details`

- User submits the form, downloads `output.md` and `metadata.json`.
- User uploads `output.md` later, and the form repopulates with "Introduction" and "Details" in the correct hierarchy.

5. Evaluation Criteria

- **Functionality**: Does the application meet all the specified requirements?
- **Code Quality**: Is the code well-structured, readable, and maintainable?
- **UI/UX**: Is the interface intuitive and visually appealing?
- **Creativity**: Did you add any thoughtful features or optimizations beyond the requirements?
- **Problem-Solving**: How did you handle the hierarchical structure and file parsing?

Feel free to reach out with clarifying questions before submission. Good luck, and we look forward to seeing your solution!