ChatGPT

Limi Lighting EPDP/EPDVP Training Webapp Design

Overview and Objectives

Limi Lighting's Engineering & Product Development Procedure (EPDP) and Engineering Prototyping & Design Validation Procedure (EPDVP) outline a comprehensive 7-phase, gate-controlled process for product development – from initial concept to mass production [1] [2]. The core philosophy of EPDP is to **"achieve speed through clarity"**, eliminating confusion by providing a foolproof roadmap [3]. This training webapp will embody that philosophy, serving as an interactive guide through the procedure to help engineers and operations team members understand **what to do at each phase, who is responsible, and why it matters**. By presenting the process in a clear, visual, and bilingual format, the tool aims to **bridge language barriers** and ensure consistent understanding across the team [4]. It is intended for use in team training sessions or self-service onboarding, consolidating all key information (phases, roles, deliverables, documents, etc.) into one accessible interface.

Key Objectives:
- Reinforce understanding of all seven phases of the EPDP/EPDVP, including decision gates and the critical **PM Gateway** control point [5].
- Provide an engaging, **bilingual (English/Chinese)** learning experience so that team members of different languages can follow along easily.
- Highlight **roles and responsibilities** (PM, Lead Engineer, QC, Purchasing, etc.) throughout the process, using RACI principles to clarify who is Responsible, Accountable, Consulted, or Informed at each step [6].
- Offer quick access to **templates, documents, FAQs, and a glossary** to support the procedural steps (e.g., Product Brief template in Phase 1, Pre-Production Design Package contents in later phases).
- Include practical tools like a **NAS path generator** to encourage proper documentation storage (as the procedures require storing outputs in designated NAS folders [7]).
- Ensure the app is **easy to navigate, visually appealing, and responsive**, making the learning process smoother and faster (thus supporting the procedure's goal of speed through clarity).

Technology Stack & Modular Structure

This webapp will run entirely **locally** in a web browser, with no server or database required. It will be built as a set of static files (HTML, CSS, JS) that can be opened from a computer or hosted on an internal network drive. The structure will be modular for maintainability and clarity:

- **HTML:** The main interface will be defined in HTML (e.g., an `index.html` file). This file will contain the overall layout – header, navigation tabs, the phase timeline, content sections, modals, etc. Content for each phase and section can be included in the HTML or dynamically loaded via JavaScript for organization. For example, separate HTML snippets or template `<script>` blocks could store the content for each phase in both languages, which the script can insert into the page when needed.

- **CSS (Tailwind CSS):** The design uses **Tailwind CSS** (a utility-first CSS framework) for layout, styling, and transitions. Tailwind can be included via the official CDN for simplicity, or a local build can be used if offline operation is needed. We will also include minimal **custom CSS** for specific effects that Tailwind utilities cover less directly (e.g., custom keyframes or detailed control of an interactive element's style). In the HTML `<head>`, we'll link Tailwind (ensuring the appropriate meta tags for responsive design are present) and add a small `<style>` or separate CSS file for custom classes (the provided code already shows custom classes like `.phase-card`, `.gate`, `.highlight-role`, etc. for our unique UI elements [8] [9]).
- **JavaScript:** A separate JS file (e.g., `app.js`) will handle all interactivity: language toggling, tab switching, phase navigation, role-based filtering, form processing for the NAS path generator, etc. We aim to keep the JS modular as well, perhaps by organizing functions by feature (one section for language toggle logic, one for handling the flowchart clicks, one for role highlight, etc.). The script will be referenced at the bottom of the HTML body for proper loading. No external JS frameworks are strictly required; vanilla JS is sufficient given the scope, but we will include libraries as needed for enhancements (for instance, Chart.js is referenced in the initial code, possibly for rendering a custom timeline graphic or charts [10], and could be used for illustrative graphs if desired).

This modular setup ensures the tool is easy to maintain. Each aspect (style, structure, behavior) is separated, and content updates (especially translations or procedure changes) can be made in one place without altering the entire codebase.

Visual Design with Tailwind CSS

Using Tailwind CSS will allow a **clean, modern design** that is easily adjustable via utility classes:

- **Layout:** The app will likely use a combination of Flexbox and Grid (Tailwind provides utilities like `flex`, `grid`, `flex-col`, `justify-between`, etc.) to arrange elements. For example, the header and tab menu can be a flex container with spacing, the phase timeline can be a responsive flex or grid of cards, and the content area can be a scrollable container. Tailwind's **responsive design utilities** (like `sm:`, `md:`, `lg:` prefixes) will be used to adapt layouts for different screen sizes.
- **Styling:** A cohesive color scheme will be applied, probably leveraging the company's brand colors. The provided code uses a teal (`#0d9488`) for active highlights [11] [12], which can be the primary accent color (matching perhaps Limi Lighting's corporate style). Neutral grays and white will keep the background and text clean (as seen with classes like `bg-white`, `text-teal-800`, `bg-gray-300` in the code [13] [14]). Tailwind's default font (or a loaded font like Inter in the code [15]) ensures legible typography. We will use Tailwind classes for spacing (`p-4`, `m-2`, etc.), borders (`border`, `rounded`), and text styles (`text-xl`, `font-bold`, etc.) to quickly apply a consistent look.
- **Animations & Transitions:** Smooth hover and state transitions will make the app feel modern and interactive. For example, the **phase cards** will have a hover effect to slightly raise and shadow them (using Tailwind's `transition` and `transform` utilities or custom CSS). In the code, `.phase-card:hover` adds a subtle shadow [16], and `.phase-card.active` raises the card with a translateY transform [11]. Similar feedback will be used for clickable elements like buttons or role cards (indeed `.role-card:hover` in the code scales it up slightly [17]). Transitions on the language toggle (sliding the switch knob) are defined in CSS for smoothness [18]. We will leverage these classes to ensure that when the user interacts (hover, click, toggle), the response is visually smooth (e.g. using `duration-300 ease-in-out` classes for Tailwind or the predefined styles as in our snippet).

- **Icons and Graphics:** The design will incorporate simple icons or shapes to enhance understanding. For example, **phase gates** are represented by diamond shapes – accomplished by a small div with a rotated square (the `.gate` class rotates a square 45° to form a diamond [19]). The gate number or checkmark can be overlaid with an inner text span rotated back (`.gate-text` rotates content -45° to read upright [20]). Role labels or icons might be used (e.g., using initials or FontAwesome icons for PM, Engineer, etc.) to visually tag content related to roles. Tailwind makes it easy to style these (e.g., small round badges with background colors for each role).
- **Theme and Polish:** Overall, the look will be **professional yet engaging** – matching the serious nature of process training but avoiding dry text-only presentation. Light backgrounds, highlighted sections for emphasis, and perhaps occasional illustrations (if available, e.g. a small schematic or product image in a modal) will keep users interested. By using Tailwind, we ensure consistency in spacing and sizing, which contributes to a polished feel without a huge custom CSS overhead.

Bilingual Support (English & Chinese Toggle)

To make the tool accessible to both English and Chinese-speaking team members, all content will be provided in **two languages**. The user can switch between languages seamlessly via a toggle switch in the header (for instance, a checkbox-style toggle labeled "EN / 中文"). The implementation plan is:

- **Toggle UI:** As shown in the code snippet, we will include a switch input in the header for language selection [21]. When toggled, this will trigger a JS function to swap all displayed text to the alternate language. We might visually indicate the active language on the toggle (e.g., left for English, right for Chinese) and possibly use flags or abbreviations. For clarity, text labels "EN" and "中" (or "中文") can be placed on each side of the switch.
- **Content Storage:** We will maintain all interface text and procedural content in both languages, likely in a structured format (such as parallel objects or dictionaries in the JS, or data attributes in the HTML). For example, we could have a `content.phases` object with `content.phases["1"].en.title`, `content.phases["1"].zh.title`, etc., for each piece of text. Alternatively, we might duplicate the content HTML for each language and show/hide based on the toggle. The choice will balance ease of maintenance (keeping translations aligned) vs. performance (loading only needed text). Given no server, a reasonable approach is to load both languages at startup and just swap what's visible.
- **Dynamic Text Swap:** The JS will respond to toggle changes by either toggling a CSS class on the `<html>` (e.g., `.lang-zh` vs `.lang-en`) which in turn could show/hide elements via CSS, *or* by programmatically replacing text nodes from the content object. For maintainability, using data attributes might be clean: e.g., `<span data-en="Project Initiation" data-zh="项目启动"></span>` and the JS sets `innerText` to the appropriate attribute on toggle. Another approach is to keep the content sections in HTML with language-specific classes and simply hide the non-active one (like `<div class="content-en">English text</div><div class="content-zh hidden">中文内容</div>` and toggle the `hidden` class on switch). We will choose an approach that ensures **both languages are fully present and easy to update**.
- **Translation Scope:** All user interface elements (headings, button labels, tab names, tooltips) and all procedural content (phase descriptions, roles, FAQs, glossary definitions) will be bilingual. The only exceptions might be internal references like NAS folder paths which likely remain in English (since folder structure is typically not translated). The glossary can cover translations of key terms if needed.

- **Rationale:** This bilingual feature is crucial for our diverse team. It directly addresses the process goal of bridging communication gaps – clear documentation in multiple languages reduces miscommunication [4]. By allowing instant toggling, the app can also be used in mixed-language meetings or training sessions, showing content in one language and then the other for clarification. It also helps employees improve cross-language understanding of technical terms (since they can switch and see the equivalent term).

Interactive 7-Phase Flowchart Navigation

At the heart of the app is an **interactive flowchart/timeline** that visually represents the 7 phases of the EPDP/EPDVP process and their interconnecting gates. This will likely be the first thing a user sees below the header. Key design points for this component:

- **Phase Timeline Layout:** The phases will be laid out in chronological order. On a wide screen, a **horizontal timeline** will be used: e.g., a row of phase "cards" or nodes, connected by lines or gate icons. Each **Phase card** will display the phase number and title (e.g., "Phase 1: Concept & Planning"), and possibly a very brief tagline about its goal. The cards are interactive – hovering will highlight them (with a glow or lift, as per `.phase-card:hover` styling [16]), and clicking will select that phase. The selected (active) phase card is visually distinguished (e.g., a colored border and raised position [11] to indicate it's active). Between each phase card, we will display a **Gate** icon (a diamond shape) to represent the review/approval gate at the end of the phase. These gate icons can be labeled (e.g., "Gate 1", "Gate 2"… or perhaps icons like checkmarks) – the documentation shows that at Phase 1's end, an approval leads to proceeding to Phase 2 [22]. We will mirror this: clicking a gate could show the criteria needed to pass it.
- **Phase Cards Content:** Each phase card might contain a concise summary that appears on hover or as part of the card. For example, Phase 1 card could on hover show a tooltip "Establish project requirements and plan (Output: Product Brief, Project Plan) [23]", Phase 2 might show "Detailed design and internal prototyping (Output: CAD models, low-fi prototypes) [24]", etc. This gives a quick overview. We may implement this with Tailwind's tooltip plugin or a custom small `<div>` that appears on hover.
- **Click to Navigate:** When a phase is clicked, the main content area will scroll or load the details for that phase. This could be a smooth scroll to the section or dynamically swapping content. For a single-page app feel, we might not do page reloads; instead, we show/hide phase details in place. For example, clicking Phase 3 card hides Phase 1's content and shows Phase 3's content container (maintaining both in DOM for quick toggling). This can be animated (e.g., a fade or slide-in using Tailwind classes or JS). The active card stays highlighted to remind the user which phase they are viewing.
- **Responsive Behavior:** On smaller screens, if a horizontal timeline doesn't fit, we will adapt to a **vertical stacked timeline**. Each phase can be a row with a connector down to the next. Tailwind's responsive utilities allow us to switch layout (`flex-row` vs `flex-col`) based on screen width. We will ensure the design remains clear: even vertically, each phase will have a card or header, and gates can be smaller icons to the side or in between. Users can still tap a phase to expand its details.
- **Phase/Gate Emphasis:** The EPDP document explicitly suggested a flowchart highlighting the **PM Gateway** as a critical control point [25]. To honor that, our timeline will mark the gateway (which occurs between Phase 2 and Phase 3 in the process) with special emphasis – perhaps a distinct color or an icon indicating the PM's approval step. For example, between Phase 2 and 3 we might label Gate 2 as "PM Authorization" and color it differently (since the **Mandatory PM Gateway** occurs after

internal prototyping before external engagement [5]). This alerts users to the significance of that transition. Other gates can be generic “Gate X” indicators requiring management or cross-functional approval as described in the docs.

- **Visual Aids:** We may incorporate mini-icons on the phase nodes to hint at content (e.g., a lightbulb icon for concept phase, a CAD icon for design phase, a factory icon for production phase, etc.), making the timeline more intuitive at a glance. Also, including a legend or color-coding (for example, phases grouped into larger stages if any) can help. The timeline combined with the bilingual labels ensures any user can quickly grasp the overall structure: **Phases 1–2 are development and internal validation, Phase 3 onward involve external prototyping, and Phase 7 is production launch**, all in sequence with go/no-go gates between [2] [22] .

This interactive flowchart not only serves as navigation but also reinforces the process structure in the user’s mind, aligning with the idea that a clear visual roadmap prevents confusion and delays [26] .

Phase Details: Content and Presentation

For each phase selected in the timeline, the app will display a detailed view of that phase’s information, essentially creating a **mini training module** per phase. The content for each phase will be drawn from the EPDP and EPDVP documents, ensuring accuracy. Key elements of the phase detail view:

- **Phase Overview:** At the top, we’ll show the phase title and a short description of its purpose and importance. For example, Phase 1’s purpose is to *“establish crystal-clear project requirements…”* because unclear requirements cause project failures [23] . Phase 2’s description highlights *“using internal resources to validate designs before involving external suppliers”* [24] , and so on. We will quote or paraphrase these to maintain the intent. A brief **“Why This Phase Matters”** summary (as given in the EPDP text) will also be included to reinforce the significance of doing the phase properly [27] . This overview sets the context for learners.
- **Subsections & Activities:** Each phase will be broken down into its key activities or sub-steps, much like the sections (1.1, 1.2, 1.3, etc.) in the documents. For instance, in Phase 1: *Product Brief creation*, *Engineering review & clarification*, *Project plan & concept selection* are distinct steps [28] [29] . We will present each such sub-step with a title and details on:
- **Who** is involved (roles responsible) – e.g., “Product Manager creates the brief; Engineering team reviews” [28] . This will later tie into the role filter feature.
- **When** it occurs (e.g., “at project initiation” or “after previous step” – the docs specify timing like “After Gate 1 approval” for Phase 2 tasks [30]).
- **Outputs/Deliverables** produced (e.g., “Product Brief Document” [31] , “CAD models and drawings” [32] , “Design Review Report” etc.). We will list these so trainees know what artifacts to expect from each phase.
- **Key tasks or criteria** within that sub-step. The EPDVP doc is very action-oriented, with checklists like *“Review the Product Brief thoroughly”, “Conduct preliminary research”* [33] [34] . We can incorporate these as bullet points or even interactive checkboxes to simulate the checklist. For clarity, these might be presented in collapsible accordions: the user can click “1.1 Product Brief” to expand the tasks involved. This prevents the page from being too text-heavy initially.
- **Gate Criteria & Outcomes:** At the end of each phase’s content, we will clearly outline the gate review criteria and the possible outcomes (Approved to move forward, or Not Approved requiring rework). The EPDP doc provides bullet checklists for gate readiness (e.g., end of Phase 1 gate criteria: “All clarification questions resolved; Chosen concept aligns with business objectives; …” [35]). We will

display these as a checklist for the user to review. This serves as a learning reinforcement ("these must be done to pass Gate 1"). The outcomes (approve or not) can be shown perhaps as a colored callout or simply as part of the text (e.g., **Approved:** proceed to next phase; **Not approved:** revisit earlier steps [22]). This highlights the decision point before unlocking the next phase in the interface (we might even disable clicking Phase 2 content until Phase 1 gate is "approved" in a training simulation mode).

- **Embedded Document Content:** Throughout the phase details, we'll embed relevant content from the actual procedures. This could include small excerpts or images: for instance, a **template snippet** of a Product Brief or a **sample photo** of a low-fidelity prototype (if those are available from the team). The docs suggest visuals like example brief or prototype photos [36] [37] – we can allocate space for these. If no actual images are provided, we might use placeholders or simple diagrams. The content sections will thus mix text and any media to keep it engaging.
- **Roles Emphasized:** In each phase section, whenever a role is mentioned (e.g., "**Lead Engineer leads the design review**" [38] or "**Purchasing team reviews for sourcing feasibility**" [39]), we will style the role name distinctly (maybe a badge or bold color) so that it stands out. This not only prepares for the role filter feature but also makes scanning the document for one's responsibilities easier.
- **Language Toggle Effects:** Importantly, all the above content will be available in both languages as previously described. The phase detail view should switch all text when language toggles. Given the volume of text per phase, we must ensure translations are accurate and layout is flexible for different text lengths (Tailwind's utility classes will help adjust spacing as needed for the Chinese text which may be more compact or more verbose in places).

By structuring each phase's info in a **consistent format** (overview, tasks, gate checklist, deliverables, etc.), the app ensures clarity and repeatability. Users will know what to expect as they move from Phase 1 to Phase 7, which aligns with the procedure's goal of making sure "anyone can follow it" clearly [40] .

Role-Based Views and RACI Highlights

One of the powerful features of this training tool is the ability to view the process **through the lens of specific roles**. Different team members (PM, Lead Engineer, Engineers, QC Lead, Purchasing Lead, etc.) have different responsibilities at each stage, and highlighting those will help individuals focus on what's relevant to them.

- **Role Selection UI:** We will provide a set of role buttons or toggles in the interface (perhaps at the top of the phase content or in the header). For example, buttons labeled "PM", "Lead Engineer", "Engineer", "QC", "Purchasing", etc. The user can click on a role to activate the filter/highlight for that role. We might allow multiple selection (to compare roles) or just one at a time – likely one at a time for simplicity. The active role button will be visually distinct (e.g. a different background color or a checkmark on it, similar to how `.role-card.active` is styled in the code with a border-color and background change [12]).
- **Content Highlighting:** When a role is selected, the script will scan the currently displayed content and **highlight all responsibilities or mentions for that role**. Concretely, we will mark up the content in advance by tagging elements with roles (e.g., a `<div>` for a task could have `data-role="PM"` if the PM is responsible, or multiple roles in a list if shared). Alternatively, wrap role-specific text in spans with classes like `role-PM`, `role-ENG` etc. The JS can then add a CSS class (like `.highlight-role`) to all elements matching the selected role. The provided CSS `.highlight-role` might, for instance, add a light green background to draw attention [41] . We

may also dim non-highlighted content for contrast. For example, if "PM" is selected, every bullet or paragraph where the Product Manager has a responsibility will be highlighted, while others might be grayed out. This way, a PM in training can effectively skim "what do I do in this phase?" without distraction.

- **RACI Matrix Approach:** We can also incorporate the **RACI matrix** concept here. RACI (Responsible, Accountable, Consulted, Informed) is a framework that clarifies different levels of involvement [6]. In our documentation, roles are mainly described as responsible or leading certain tasks. We could map that to RACI by noting, for instance, that the **Lead Engineer** is accountable for technical quality, the **PM** is accountable for meeting business requirements, etc. Within each task, a role might be Responsible (doing the task) or Accountable (owner of the task's outcome). The training app might not explicitly label each task with R/A/C/I letters, but the **RACI roles tab** (covered in the next section) will give an overview. In the phase content, the simplest approach is to highlight who is responsible (primary doer) for each action item. If needed, we could use different highlight styles (color or icon) to distinguish Responsible vs Accountable, but this might complicate the UI. A consistent highlight for any primary role involvement should suffice for training purposes.
- **Examples:** In Phase 1, selecting "PM" would highlight the Product Manager's duties like creating the Product Brief and authorizing any external engagement (PM is effectively the gateway approver) [28] [5]. Selecting "Lead Engineer" would highlight tasks like leading the engineering clarification meeting and design reviews [42] [38]. For "Purchasing", their involvement starts later (e.g., reviewing design for manufacturability in internal review [43] and then leading supplier engagement in Phase 4), so earlier phases might have little highlighted, visually indicating when their role becomes critical. This dynamic view helps each department understand **when and how they fit in the process**.
- **Role Legends:** We will ensure there is a legend or explanation for this feature so users know what the highlight means. Perhaps a small note like "*Select a role to highlight relevant actions and responsibilities in the process.*" Additionally, hovering over a role label in text could show a tooltip of that role's full name (since acronyms like PM or QC might be used) and their general remit. This ties into the glossary and RACI info as well.

By integrating role-based views, the app not only teaches the process steps but also reinforces accountability. It reflects the **"Key Roles & Responsibilities"** outlined in the procedure document [44], ensuring that each participant is aware of their duties (the docs emphasize that following the procedure is a professional standard and everyone's accountability [45]). The interactive highlighting makes this learning personalized and directly relevant to each team member.

Tabbed Navigation: RACI Roles, Governance & Docs, FAQs, Glossary

Beyond the core phase content, the webapp will offer additional reference sections through a **tabbed navigation** interface. These sections provide supporting information and resources in a structured way. The tabs likely to be included are:

- **1. RACI Roles & Responsibilities:** This tab will detail the various roles involved in the EPDP/EPDVP and their overall responsibilities (not just phase-specific). It may present a **RACI chart** or a summary table. For example, a table with columns for each phase and rows for each role, indicating whether that role is Responsible (R), Accountable (A), Consulted (C), or Informed (I) in that phase's major tasks. This would give a one-glance understanding of involvement across the project. Additionally, a brief description of each role (as in the EPDP doc's introduction, e.g., "**Product Manager (PM)** – initiates project, defines requirements [46]; **Lead Engineer** – technical lead ensuring feasibility and quality [47] ; **Purchasing Lead** – sources suppliers, manages cost and production coordination [48] ; etc.) will be provided. This section essentially complements the role-based highlight feature by giving the theoretical overview of roles. It reinforces the idea from the docs that *"The PM serves as the mandatory gateway between Engineering and Purchasing"* [49] , and similar governance principles tied to roles.

- **2. Governance & Documents:** This tab will list all **phase deliverables and documentation** requirements in one place. It acts as a reference checklist of what needs to be produced or signed off at each phase and gate. For example:
- Phase 1: Product Brief, Project Plan, Concept Selection Document [50] (all documents needed by Gate 1).
- Phase 2: CAD models, Drawings, BOM, Prototype Test Results, etc., and the **Design Review Report** at Gate 2.
- Phase 3: Sourcing Strategy Document [51] , etc., culminating in the **Pre-Production Design Package (PPDP)** by Phase 4 [52] [53] .
- Phase 5: Golden Sample approval records [54] , etc., and so on up to Phase 7: final Quality Reports, Production SOPs, etc.
  This tab can be organized by phase or document type. It will also reiterate any **governance policies** (e.g., "No external supplier engagement before PM approval" which was a key policy [55]). We might incorporate download links or placeholders for actual templates (e.g., a link to a blank Product Brief template or a PPDP checklist PDF). If the company has standard forms, we can link them here for easy access during training.
- **3. FAQs:** A Frequently Asked Questions section to address common confusions or edge cases in the process. This content might be developed based on real questions the team has asked. Example FAQs could be: "*What if a project doesn't get approved at Gate 1?*" (Answer: reiterate that it goes back to clarification or concept refinement [22]); "*How formal does the low-fidelity prototype need to be?*" (Answer: emphasize focus on function over finish [56]); "*Who can authorize skipping a phase?*" (likely no skipping without management approval, per adherence rules [45]); or "*How do we handle an emergency change outside this process?*" (the scope section indicates a separate emergency protocol [57]). We will format FAQs as an accordion list: each question is a collapsible header, clicking it reveals the answer. This keeps the section compact and user-friendly.
- **4. Glossary:** A glossary of terms and acronyms used in the process documents. The engineering process documentation is rich with terms like *DFM/DFA* (Design for Manufacturing/Assembly), *BOM*, *Lo-Fi prototype*, *Golden Sample*, *PPDP*, *RACI*, etc. New hires might not know all these. We will list terms alphabetically with concise definitions. For instance, **"PPDP (Pre-Production Design Package):** The complete documentation package (final CAD, drawings, BOM, etc.) prepared after design validation for handoff to Purchasing [53] ."; **"Golden Sample:** A sample unit that is fully approved and serves as the quality benchmark for production [54] ."; **"DFM/DFA:** Design for Manufacturing / Design for Assembly – reviews to ensure the design can be efficiently manufactured and assembled."; etc. Bilingual translations of each term can be given here as well for technical vocabulary.

The tabbed navigation will be implemented using a combination of Tailwind classes and simple JS to switch content. The code already includes styles for tabs (`.tab-button` with active states) [58] . When a tab is clicked, we'll add an active class to highlight the selected tab and show the corresponding content div while hiding others. This way, the main page doesn't become one extremely long scroll – users can jump to these sections on demand.

By providing these tabs, we ensure the app is **not just a step-by-step walkthrough, but a comprehensive reference**. A user can easily find supporting info without leaving the app – enhancing the “one-stop-shop” convenience for training.

NAS Folder Path Generator Utility

Proper documentation storage is a crucial part of the EPDP/EPDVP (to maintain version control and clear handoffs [59] [60]). The app will include a small utility to help users generate the correct **NAS (Network Attached Storage) folder paths** for project documents, as per the standard convention. This feature turns what is often a rote, error-prone task into an interactive, foolproof step.

- **Purpose:** Both the EPDP and EPDVP documents repeatedly instruct team members to store outputs in a structured NAS directory (e.g., *“Store in NAS: [Project Name]/Phase 1 - Concept/ 01_Product_Brief_Clarifications.docx”* [7]). New employees might find it confusing to remember the exact naming scheme or might not know the project code conventions. The generator will ensure they input the right pieces and it outputs the properly formatted path. This not only trains them on the convention but can be used in practice to copy-paste paths, reducing mistakes.
- **UI Layout:** The tool will likely be an interactive form accessible via perhaps a "Utilities" tab or as part of the Governance/Docs tab. It will contain input fields and dropdowns. For example:
- A text input for **Project Name or Code** (the base folder name on NAS). Possibly also fields for project code or numbers if those are separate.
- A dropdown or set of radio buttons for **Phase** (1 through 7, with labels like "Phase 1 - Concept", "Phase 2 - Detailed Design & Low-Fi Proto", ... "Phase 7 - Production Launch"). Selecting one could auto-fill the phase portion of the path.
- Another dropdown for **Document type/section** relevant to that phase. For instance, if Phase 1 is selected, document options might be "01_Product_Brief_Clarifications.docx", "02_Preliminary_Research_Ideation", "03_Conceptual_Designs", etc., based on the EPDVP section breakdown [61] [62]. We will derive these options from the actual procedure documentation, which often lists how files should be named and stored. For Phase 2, options might include "01_Design_V0.1", "02_LowFi_Proto_Plan.docx", "03_LowFi_Prototypes_and_Iterations", etc. [63] [64]. We can hard-code these patterns or load from a config.
- Possibly a field for version or date if applicable (some paths include version numbers like V0.1). But we might keep it simple and focus on the folder hierarchy.
- A **Generate/Display button** which when clicked (or in real-time) shows the full concatenated path in a read-only text box. For example, if Project Name = "Apollo", Phase 2 and document "01_Design_V0.1" selected, it might show: `Apollo/Phase 2 - Detailed Design & LowFi Proto/01_Design_V0.1/`. We will match exactly the formatting from the docs (notice the phase folder name in the example includes a descriptive label after the number).
- **Output and Copy:** The generated path will be displayed in a monospaced font (for clarity, matching how code/paths look) and possibly with a “copy to clipboard” icon button. So a user can click to copy the path string and use it when navigating the NAS or writing documentation. The output box can also be made to look visually distinct (maybe a light gray background, slight border – the code’s `.nas-path` class suggests a styled monospace label [65]).
- **Validation and Guidance:** We can add hints in the form, like placeholder text or tooltips, to guide input. For example, if the project naming convention is specific (like use Project Code and Name), we can illustrate that. We can also automatically replace any illegal path characters the user might accidentally input, ensuring the output is a valid path string.

- **Benefit:** By using this tool, team members get used to the structure of storing documents at the correct phase locations. It reinforces discipline in documentation (a key part of the process, since *“Documentation is Key: maintain clear, version-controlled records…”* [66]). During training, instructors can demonstrate using this tool to locate or set up project folders, making the learning experience hands-on.

Overall, the NAS path generator is a small but practical addition that links the theoretical process to real operational tasks. It underlines the importance of organization and standardization in the engineering process.

Optional Enhancements

In addition to the core features above, several optional enhancements were proposed to enrich the training tool. These can be implemented as time and resources permit:

- **Phase-Based Checklists & Progress Tracking:** We have already embedded checklists for gate criteria in each phase. We can extend this to an interactive **progress tracking feature**. For example, as a user goes through training, they could mark each sub-step as "understood" or simulate completion. A progress bar per phase (or overall completion percentage) can gamify the learning a bit. Alternatively, for actual project use, these checklists could be used to track real progress (though without a database, persistence would require using `localStorage` to save checklist states locally). The idea is to ensure all tasks are accounted for – echoing the process aim that nothing is skipped. Every checked item would indicate that requirement (like *“Success criteria are agreed upon”* in Phase 1 [67]) is met, reinforcing thoroughness.
- **PDF Export of Procedure Content:** Providing a way to export content is useful for offline reference or printing. We can implement a **“Generate PDF”** feature for either the entire procedure or selected sections. Using a library like jsPDF or html2canvas, the app can take the currently viewed content (with maybe the user’s selected language and role filters applied) and produce a nicely formatted PDF. For instance, an engineering lead might export the Phase 2 instructions to have a checklist on the workshop floor during prototyping. Or the entire EPDP could be exported as a formatted handbook. This complements the training by allowing easy distribution of updated process info (especially if the official documents get updated, this tool could always export the latest version). If not actual PDF generation, we will at least ensure the page is **printable** with print-specific CSS so that using the browser’s print to PDF yields a clean result.
- **Interactive Modals for Concept Comparisons or Templates:** Throughout the training content, certain topics might be better explained with visuals or comparisons. For example, the concept selection step could have a modal showing side-by-side concept sketches or a mini case study of good vs bad product briefs (as teaching material). We can implement clickable **“Learn More”** links that open a modal dialog on top of the page. Tailwind CSS, especially with component libraries like Flowbite or Headless UI, can simplify modals styling and behavior. These modals might include:
- **Image galleries:** e.g., a carousel of prototype iteration images (to demonstrate how a design improved from one prototype to the next, aligning with *before/after* suggestions [37]).
- **Template previews:** e.g., clicking “Product Brief Document” could show a template or an example brief in a modal, rather than navigating away. Similarly, a preview of a filled-out PPDP table of contents can be shown when that is discussed.

- **Video clips:** If available, a short video of a phase in action (like a quick tour of a trial production line for Phase 6) could be embedded in a modal. This keeps users engaged and breaks up text monotony.
- **Search Function:** As the content grows, having a client-side search to quickly find keywords could be helpful (for instance, typing "BOM" highlights where Bill of Materials is mentioned). This is not trivial without a backend, but a simple JS search that filters or jumps to content could be done. This would aid users in quickly answering specific questions.
- **User Personalization:** Another idea (more complex) is allowing users to save notes or mark favorite sections, but without a backend this would rely on localStorage. Possibly out-of-scope for initial version, but worth mentioning as a future enhancement.

Even without all these extras, the core application will be highly useful. These enhancements are there to illustrate how the tool can evolve into an even more robust training and reference platform. They emphasize interactivity and user engagement – important for adult learning – making the process not just a document to read, but a set of concepts to interact with.

Responsive & User-Friendly Experience

Finally, it's crucial that the webapp provides a smooth, **user-friendly experience across different devices**. Engineers and team members might use it on their desktop during training, or pull it up on a tablet or phone on the shop floor for a quick refresher. Key considerations for responsiveness and usability:

- **Mobile-Friendly Layouts:** Using Tailwind’s responsive classes, we will ensure that on smaller screens, content reflows in a single column, fonts adjust size, and touch targets (buttons, toggles) are suitably large. The timeline may switch to a vertical stepper which is easier to scroll on a phone. Tabs might collapse into an accordion or a dropdown menu for easier tapping. All interactive elements will be tested for tap responsiveness. The design avoids any hover-only interactions for critical features (hover doesn’t exist on touch devices), so we will ensure things like tooltips or hover highlights have an equivalent click or tap mechanism (e.g., tapping a phase card on mobile both shows its content and perhaps triggers a brief highlight effect).
- **Performance:** Because this is a purely front-end app, we need to manage performance especially as it loads a lot of content for seven phases and multiple tabs. We will keep images optimized (use web-friendly formats and reasonable resolutions since high-res images can slow mobile devices). Also, heavy content (like a large PDF or video) won’t be loaded unless requested (e.g., only load a video in a modal when the user opens that modal). The initial load will focus on essential HTML/CSS/JS and maybe the content for the first phase, then load other phases on demand (this can be done by splitting content into separate JSON files and fetching them when needed, which still respects the “local” criterion if files are local). This ensures the app feels snappy.
- **Navigation & State:** The UI will provide clear navigation cues. The currently active phase or tab will be visually distinct (highlighted tab, active phase card) so the user knows where they are. We might also include “Next Phase” / “Previous Phase” buttons at the bottom of a phase page for linear navigation in addition to the timeline bar, which is especially helpful on mobile where the whole timeline might not be visible at once. We will also consider preserving state – for example, if a user switches to the FAQ tab and back to a phase, it would be nice if the phase content they last viewed remains active. We can achieve this by not fully unloading content sections and by tracking the last active phase in a variable.

- **Accessibility:** We will follow basic accessibility practices: proper semantic HTML (using `<header>`, `<nav>`, `<main>`, `<section>`, `<h1>-<h3>` headings in logical order, lists for lists, etc.), alt text for any images, and ARIA attributes where needed (like `aria-expanded` on accordions, labeling the language toggle for screen readers). Ensuring that the app can be navigated by keyboard (tab order for interactive elements) is also important, as some users or situations (like projecting in a meeting) might rely on keyboard control. Tailwind doesn't interfere with accessibility inherently, so it's about using it appropriately.
- **Testing:** We will test the interface with several personas in mind: a new engineer going through step by step, a PM quickly jumping to a reference in the Glossary, a user with Chinese preference ensuring nothing is missed in translation, etc. This will help fine-tune the text clarity (making sure, for instance, that translations still convey the meaning of original terms accurately) and the flow of the app. The ultimate goal is that the app "**empowers team members**" by giving clarity and confidence in following the procedure [68], so user feedback will be key in refining any confusing parts.

Conclusion

In summary, the fully local HTML/CSS/JS webapp for Limi Lighting's EPDP and EPDVP will be a **comprehensive, interactive training tool** that brings the paper procedures to life. It integrates the structured content of the official documents with modern web design techniques to facilitate quicker and deeper understanding. By navigating through an interactive flowchart of all 7 phases, team members learn the sequence of development and the importance of each gate review. By toggling languages and filtering by roles, they can tailor the learning experience to their needs, ensuring nothing is lost in translation or relevance. Additional tabs provide quick access to important reference material – from role definitions to document templates – all in one place.

This approach will help eliminate the confusion and miscommunication that slow down projects, aligning perfectly with the EPDP's purpose of achieving **speed through clarity** [3]. The tool serves not only as training documentation but also as an ongoing reference during project execution (thanks to features like the NAS path generator and checklists). It empowers users to follow the process correctly every time, which in turn **prevents rework, enables speed, protects quality, and bridges communication gaps** – the very benefits the procedure promises [26].

By prioritizing clarity, simplicity, and interactivity, the EPDP/EPDVP training webapp will become an invaluable asset in onboarding new team members and upholding process discipline for seasoned professionals alike. It transforms a lengthy procedure into a dynamic learning journey, ensuring that *"a child can follow it"* style clarity is achieved for all [40] – ultimately helping Limi Lighting deliver successful engineering projects faster and more efficiently.

Sources: The design and content of this webapp are based on Limi Lighting's internal process documents [1] [33], enhanced with industry best practices in web-based training tools and project management concepts like RACI [6]. All procedure details (phases, roles, outputs, criteria) are drawn from the EPDP V2.0 and EPDVP V1.0 documentation to ensure accuracy and alignment with official guidelines.

---

1 3 4 5 22 23 24 25 26 27 28 29 30 31 32 35 36 37 38 39 40 42 43 44 45 46 47 48 49 50 51 55 56 57 67 68 Process doc .txt
file://file-AWS43SQ7PCjh2k72pruuuB

2 7 33 34 52 53 54 59 60 61 62 63 64 66 Limi Lighting Engineering Prototyping & Design Validation Procedure (EPDVP) - V1.0.docx
file://file-KZdqBrHgXNihiZm9ydAH9V

6 RACI Matrix: Responsibility Assignment Matrix Guide
https://project-management.com/understanding-responsibility-assignment-matrix-raci-matrix/

8 9 10 11 12 13 14 15 16 17 18 19 20 21 41 58 65 Webpage .txt
file://file-M28zRLk5avFqpqytZnzzLi