**I. Project Structure:**

You'll likely want a structure like this:

your-app/

├── components/ # Reusable UI components

│ ├── dashboard/

│ │ ├── ProductCard.js

│ │ └── CreateProductLaunch.js

│ ├── product/

│ │ ├── PhaseList.js

│ │ ├── CreatePhase.js

│ │ └── ProductDetails.js

│ ├── phase/

│ │ ├── TaskList.js

│ │ ├── CreateTask.js

│ │ └── PhaseDetails.js

│ ├── task/

│ │ ├── TaskItem.js

│ │ ├── UploadDocument.js

│ │ ├── ApproveButton.js

│ │ └── AskForApprovalButton.js

│ ├── auth/ # Authentication related components (if needed)

│ ├── ui/ # General UI elements (buttons, modals, etc.)

│ └── layout/ # Layout components (e.g., for the dashboard)

├── pages/ # Next.js pages (routing)

│ ├── index.js # Dashboard

│ ├── products/

│ │ ├── [productId].js # Product details page

│ │ └── new.js # Create new product page (optional, could be a modal)

│ ├── phases/

│ │ └── [phaseId].js # Phase details page

│ └── tasks/

│ └── [taskId].js # Task details page (might be modal instead)

├── public/ # Static assets

├── styles/ # CSS modules or global styles

├── lib/ # Utility functions, API interaction

│ └── api.js # Functions to interact with your backend

├── models/ # Data models (if using TypeScript)

└── ...

**II. Core Functionality and UI Elements:**

Let's go through each of your requirements and how you might implement them with Next.js and a focus on a beautiful UI.

**1) Dashboard:**

* **Displaying Products:**
  + Fetch a list of ongoing products from your backend API (/api/products?status=ongoing).
  + Use useEffect in your pages/index.js to fetch this data when the component mounts.
  + Map through the product data and render a ProductCard component for each.
  + **ProductCard.js:** This component will display basic product information (name) and be styled as a "big card." You can use CSS modules or a styling library like Tailwind CSS or Styled Components for the visual appeal. Make it clickable to navigate to the product details page (/products/[productId]).
  + Implement horizontal scrolling for the ProductCard list if there are many products. CSS properties like overflow-x: auto and flexbox can achieve this.
* **Create New Product Launch:**
  + Include a prominent button or section on the dashboard labeled "Create New Product Launch."
  + This could either navigate to a dedicated /products/new page or open a modal using a library like react-modal or a custom implementation.
  + The form for creating a new product would be on this page/modal.

**2) Creating Products, Phases, and Tasks:**

* **Creating a Product:**
  + On the "Create New Product Launch" page/modal, provide an input field for the product name.
  + Upon submission, send a POST request to your backend API (/api/products) with the product name.
  + Upon successful creation, you can redirect back to the dashboard or the newly created product's page.
* **Creating a Phase:**
  + On the pages/products/[productId].js page (the product details page), fetch and display the phases associated with that product.
  + Include a button to "Create New Phase." This could open a modal or have an inline form.
  + The form should have an input field for the phase name.
  + When submitting, send a POST request to your backend API (/api/products/[productId]/phases) including the productId in the URL or request body. The backend should handle associating the new phase with the correct product.
  + After successful creation, refresh the phase list on the page.
* **Creating a Task:**
  + Similarly, on the pages/phases/[phaseId].js page (the phase details page), fetch and display the tasks associated with that phase.
  + Include a button to "Create New Task." This could also use a modal or an inline form.
  + The form should include:
    - Task Name (text input)
    - Estimated Date-Time (DatePicker component - you can use a library like react-datepicker or a custom one)
    - Needs Approval (checkbox)
    - An option to upload a document (see below).
  + When submitting, send a POST request to your backend API (/api/phases/[phaseId]/tasks) including the phaseId.

**3) Task Details:**

* **DatePicker:** Integrate a React date and time picker library. These libraries provide user-friendly interfaces for selecting dates and times.
* **Needs Approval (Checkbox):** A simple <input type="checkbox" /> in your task creation/editing form.
* **Upload Document:**
  + Use an <input type="file" /> element.
  + When the user selects a file, you'll need to handle the file upload. This typically involves:
    - Client-side: Reading the file using the FileReader API or a library like react-dropzone for a better UI.
    - Server-side: Sending the file data (often as multipart/form-data) to your backend API endpoint for storage (e.g., on a cloud storage service like AWS S3, Google Cloud Storage, or your server's file system). You'll need to store the file path or URL in your database associated with the task.
    - Displaying the uploaded document: Show a link to download or view the document.
* **Approve Button:**
  + This button should only be visible to users with the "admin" role and only when the task's "needs approval" attribute is true and it hasn't been approved yet.
  + When clicked, it should send a request to your backend API (/api/tasks/[taskId]/approve) to update the task's status.
* **Ask for Approval Button:**
  + This button should be visible to users who created the task (or have relevant permissions) when the task "needs approval" and hasn't been requested for approval yet.
  + Clicking it should send a request to your backend API (/api/tasks/[taskId]/request-approval) to update the task's status.

**4) Phase and Product Completion:**

* **Phase Completion:**
  + On the backend, when all tasks associated with a phase are marked as "completed," you can automatically update the phase's status to "completed."
  + Alternatively, you could have a "Mark Phase as Complete" button (potentially visible only to admins or relevant roles) that triggers a backend check.
* **Product Completion:**
  + On the pages/products/[productId].js page, display the list of phases.
  + Include a "Mark as Launched" or "Mark as Complete" button.
  + When clicked, this button should:
    - Validate on the client-side (or more importantly, on the backend) if all phases associated with the product are marked as "completed."
    - If all phases are complete, send a request to your backend API (/api/products/[productId]/complete) to update the product's status.
    - Provide feedback to the user if not all phases are complete.

**5) Permissions Management:**

* You'll need a way to manage user roles and permissions. This is typically handled on the backend.
* When a user logs in (if you implement authentication), their role should be determined.
* On the frontend, you'll conditionally render UI elements (like the "Approve" button or create/delete buttons) based on the user's role. You might use context or props to pass the user's role down to components.
* **Example:**

JavaScript

import { useUser } from '../lib/auth'; // Assuming you have an auth context

function TaskItem({ task }) {

const { user } = useUser();

return (

<div>

{task.name}

{task.needsApproval && !task.isApproved && user?.role === 'admin' && (

<button onClick={() => handleApprove(task.id)}>Approve</button>

)}

{/\* ... other task details \*/}

</div>

);

}

**6) Backend (Conceptual):**

You'll need a backend to handle:

* Storing your data (products, phases, tasks, users, roles, permissions). A database like PostgreSQL, MongoDB, or Firebase could be used.
* API endpoints for creating, reading, updating, and deleting products, phases, and tasks.
* Authentication and authorization (if you need user accounts and roles). Libraries like NextAuth.js can help with authentication.
* Handling file uploads and storage.
* Implementing the logic for phase and product completion and approval workflows.

**7) Beautiful UI:**

* **Styling Libraries:** Consider using a CSS-in-JS library like Styled Components or Emotion, or a utility-first CSS framework like Tailwind CSS. These can help you create a consistent and visually appealing design.
* **Component Libraries:** Explore React UI component libraries like Material UI, Chakra UI, or Ant Design. These provide pre-built, well-styled components that can speed up development and ensure a polished look.
* **Design System:** If your application grows, consider creating a basic design system with consistent colors, typography, spacing, and component styles.
* **User Experience (UX):** Focus on clear navigation, intuitive forms, and helpful feedback to the user.

**Next Steps:**

1. **Set up your Next.js project:**

Bash

npx create-next-app your-product-app

cd your-product-app

1. **Choose a styling approach:** Decide if you'll use CSS modules, Tailwind CSS, Styled Components, or a component library. Install the necessary dependencies.
2. **Design your database schema:** Plan how you will structure your data for products, phases, tasks, users, and roles.
3. **Start building your backend API:** Create the endpoints needed for the core functionality (CRUD operations for products, phases, and tasks).
4. **Implement the dashboard (pages/index.js and components/dashboard)**: Focus on displaying the ProductCard components and the "Create New Product Launch" option.
5. **Implement the product details page (pages/products/[productId].js and components/product)**: Display phases and the "Create New Phase" option.
6. **Implement the phase details page (pages/phases/[phaseId].js and components/phase)**: Display tasks and the "Create New Task" option.
7. **Build the task item component (components/task/TaskItem.js)**: Include the DatePicker, approval checkbox, and placeholders for document upload and approval buttons (you'll implement the logic for these later).
8. **Implement the document upload functionality (components/task/UploadDocument.js and backend)**.
9. **Implement the approval workflow (components/task/ApproveButton.js, AskForApprovalButton.js, and backend logic)**.
10. **Implement the phase and product completion logic (frontend and backend)**.
11. **Implement basic role-based permission rendering on the frontend.**

This is a significant undertaking, but by breaking it down into smaller steps, you can build your simplified product management application iteratively. Remember to focus on creating a clean and intuitive UI as you go! Good luck!