🚀 SWE Intern Assessment: The "LegalFlow" Engine

Role: Software Engineer Intern (React Native / MERN Focus)
Duration: 48 Hours
Difficulty: Medium (Logic-Intensive)

---

1. The Scenario

Legal cases require a strict order of operations. You cannot **"Sign Settlement"** before you **"Draft Settlement."** You cannot **"Bill Client"** until the **"Case is Closed."**

We need a workflow engine that prevents lawyers from doing things out of order. Your job is to build a "Smart Task Manager" that enforces these dependencies and prevents logical errors (like circular loops).

---

2. The Tech Stack (Strict)

You must use the following technologies:

- **Authentication:** Clerk (Next.js Middleware integration)
- **Frontend & Backend:** Next.js (App Router, Server Actions/API Routes)
- **Language:** TypeScript
- **Database:** MongoDB (via Mongoose)
- **State Management:** Zustand
- **UI/Styling:** Tailwind CSS + shadcn/ui
- **Deployment:** Vercel (Optional - See Deliverables)
- **Code Editor:** Use the code editor of your choice, but Antigravity or Cursor is recommended for faster prototyping

---

3. The Core Challenge: "Topological Sort"

Most to-do apps just list tasks by `created_at`. Ours is different.
If **Task B** depends on **Task A**, then **Task A** must always appear *above* Task B in the list.

The Algorithm Requirements

You must implement **Kahn's Algorithm (Topological Sort)** in your backend API to:

1. **Sort the Workflow:** Return tasks in their valid execution order.
2. **Prevent Deadlocks:** If a user tries to link `Task A -> Task B` and `Task B -> Task A`, the system must **reject** the request with a "Cycle Detected" error.

---

4. Functional Requirements

Phase 1: Authentication & Onboarding

- User logs in via **Clerk**.
- User lands on a private dashboard.
- Each user sees only their own tasks (Data isolation).

Phase 2: Task Management (The UI)

Task Creation Interface

- **Input Field:** A clean text input with placeholder text (e.g., "Enter task name...").
- **Add Button:** Primary button (using shadcn/ui Button component) labeled "Add Task" or "+ New Task".
- **Validation:** Show error state if user tries to add an empty task.

Task List Display

- **Task Cards:** Each task should be displayed as a card component with:
  - **Task Name** (left-aligned, primary text)
  - **Status Indicator:** Visual badge showing "Completed" (green) or "Pending" (gray/yellow)
  - **Checkbox:** Allow users to mark tasks as complete/incomplete
  - **Delete Button:** Small icon button (trash icon) to remove the task
  - **Lock Icon:** Display a lock icon (🔒) for tasks that are locked due to incomplete dependencies
- **Visual States:**
  - **Unlocked & Incomplete:** Full color, checkbox enabled
  - **Unlocked & Complete:** Checkbox checked, slightly muted color or strikethrough text
  - **Locked (Dependencies Incomplete):** Grayed out appearance, checkbox disabled, lock icon visible
  - **Hover State:** Subtle elevation/shadow on hover for unlocked tasks

Dependency Management Interface

- **Dependency Section:** A separate card or section labeled "Add Dependency"
- **Two Dropdown Menus:**
  - **Dropdown 1:** "Select dependent task" (the task that needs another task completed first)
  - **Dropdown 2:** "Select provider task" (the task that must be completed first)
- **Relationship Indicator:** Show the relationship as: `[Dependent Task] depends on → [Provider Task]`
- **Add Dependency Button:** Secondary button to create the link

- **Dependency List:** Below the add section, display existing dependencies as:
  - `"Review Motion" → depends on → "Draft Motion"` with a small delete icon to remove the dependency

Error Handling UI

- **Toast Notifications:** Use shadcn/ui Toast component for:
  - Success: "Task created successfully"
  - Success: "Dependency added successfully"
  - Error: "⚠️ Cycle Detected! Cannot create this dependency as it would create an infinite loop."
  - Error: "Cannot delete task with active dependencies"
- **Inline Validation:** Red border on input fields for validation errors

Overall Layout

- **Header:** Top navigation with app logo/name and user profile (Clerk UserButton)
- **Main Content:**
  - Left side or top: Task creation form
  - Center: Sorted task list (main focus)
  - Right side or bottom: Dependency management panel
- **Responsive Design:** Mobile-friendly layout that stacks vertically on smaller screens
- **Loading States:** Skeleton loaders or spinners when fetching/sorting tasks

Phase 3: The API Logic (The Brains)

- `GET /api/tasks`:
  - Must **not** return raw data.
  - Must run **Kahn's Algorithm** to sort tasks based on dependencies.
  - *Example:* If I create "Step 2" first, then "Step 1", the API must return `[Step 1, Step 2]`.
- `POST /api/dependency`:
  - Accepts `{ providerId, dependentId }`.
  - **CRITICAL:** Before saving to MongoDB, you must simulate the graph in memory. If adding this link creates a **Cycle** (Infinite Loop), return `400 Bad Request` and do not save.

---

5. Deliverables (Evaluation Criteria)

Your submission will be evaluated based on the following mandatory deliverables:

1. Source Code (GitHub) - MANDATORY

- Public GitHub repository with:
  - Clean, well-structured code
  - Comprehensive `README.md` with:
    - Project overview
    - Setup instructions (local development)
    - Environment variables needed
    - How to run the application
  - Clear folder structure
- **Code Quality Focus Areas:**
  - Your `/api` route handlers and Topological Sort implementation
  - Zustand store architecture and state management
  - TypeScript type definitions
  - Component structure and reusability

2. Demo Video (YouTube/Loom) - MANDATORY

Upload a video (Unlisted) demonstrating the complete workflow:

Duration: Maximum 10 minutes

Required Demonstrations:

1. **Authentication Flow** (1-2 min):
   - Sign up or Login via Clerk
   - Show successful redirect to dashboard
2. **Task Creation & Management** (2-3 min):
   - Create at least 4-5 tasks in random order
   - Show the UI clearly displaying each task
   - Mark a task as complete and show state change
3. **Dependency Linking** (2-3 min):
   - Add multiple dependencies (create a chain: A → B → C)
   - Refresh the page to prove automatic sorting
   - Show locked state: demonstrate that dependent tasks are locked until providers are completed
   - Complete a provider task and show dependent task unlocking
4. **Edge Cases & Error Handling** (2-3 min):
   - **Cycle Detection Attack:** Attempt to create a circular dependency (e.g., C → A when A → B → C already exists)
   - Show the error message/toast notification

- **Try creating multiple cycles** to prove robust validation
- Show handling of disconnected graphs (if implemented)
- Demonstrate any other edge cases you've handled

5. **Code Walkthrough** (1-2 min):
   - Open your code editor
   - Navigate to your `topologicalSort` function
   - **Explain your implementation:**
     - How you built the adjacency list
     - How you implemented Kahn's Algorithm
     - How you detect cycles
     - Walk through the key logic briefly

Video Quality Requirements:

- Clear screen recording with audio narration
- Show both UI interactions and code
- Explain your thought process
- Upload to YouTube (Unlisted) or Loom and provide the link

3. Live Deployment on Vercel - OPTIONAL (Bonus Points)

- Deployed application with all environment variables configured
- Functional authentication and database connection
- Provide the live URL
- **Bonus Consideration:** A working live deployment demonstrates production-readiness and earns extra credit

---

6. How to Submit

Submit your work via email to **canadawaleyarindia@gmail.com** with:

1. ✅ GitHub Repository URL
2. ✅ Demo Video URL (YouTube Unlisted / Loom)
3. ⭐ Vercel Deployment URL (Optional)

Deadline: 11:59 PM, February 15, 2026

---

7. Success Criteria

To pass this assessment, your submission must demonstrate:

✅ **Functional Topological Sort:** Tasks display in correct dependency order
✅ **Cycle Detection:** System prevents and alerts users about circular dependencies
✅ **UI/UX Quality:** Clean, intuitive interface with proper visual states (locked/unlocked)
✅ **Code Quality:** Well-structured, typed, and documented code
✅ **Complete Demo:** Video covers all required scenarios and edge cases
✅ **Kahn's Algorithm:** Proper implementation with clear explanation in video

What Will Cause Failure:

- Using simple `.sort()` by date instead of Topological Sort
- Missing cycle detection (app crashes or allows circular dependencies)
- Incomplete demo video (missing edge cases or code walkthrough)
- No GitHub repository or video submission

---

8. Tips for Success

- **Start with the Algorithm:** Get Kahn's Algorithm working first before building the UI
- **Test Edge Cases Early:** Don't wait until the last minute to test cycle detection
- **Keep It Simple:** Focus on core functionality over fancy animations
- **Document Your Code:** Add comments explaining your Topological Sort logic
- **Practice Your Demo:** Record a draft video to ensure smooth presentation

Good luck. Show us you can handle logic, not just UI.