Assignment for Full-Stack Intern

Simplified PRD: Approval Workflow for Financial Transactions with Audit Logs

Project Overview

This project aims to create a financial transaction submission and approval system with simple role-based access control (RBAC). Employees submit transactions, and managers can approve or reject them. A basic audit log will record these actions. The project will use **Next.js**, **Tailwind CSS**, **NextAuth**, and **TanStack Table**.

---

1. Objectives

- Implement a basic transaction submission and approval workflow with role-based permissions.
- Keep track of key actions via audit logs.
- Gauge candidate's proficiency with Next.js, NextAuth, and UI/UX design with Tailwind and shadcn.

---

2. User Roles & Permissions

0 - Must Have

- **Employee**:
  - Can submit transactions.
  - Can view only their own submitted transactions and statuses.
- **Manager**:
  - Can view all submitted transactions.
  - Can approve or reject pending transactions.

1 - Good to Have

- **Admin** (optional stretch):
  - Can view all transactions.
  - Can view audit logs for all actions.
  - Has permissions of both employees and managers.

---

3. Functional Requirements

Transaction Submission (Employee)

- **0 - Must Have**:
  - A form for employees to submit new transactions with fields for type, amount, and description.
  - Submitted transactions should have a default "Pending" status.

View Transactions

- **0 - Must Have**:
  - Display transactions using **TanStack Table**.
  - **Employee**: Can view only their own transactions.
  - **Manager**: Can view all transactions, with filters for status (Pending, Approved, Rejected).
- **1 - Good to Have**:
  - Basic filters for transaction status and pagination (using TanStack Table features).

Approval Workflow (Manager)

- **0 - Must Have**:
  - Buttons to approve or reject transactions directly in the table for pending transactions.

Audit Logs

- **0 - Must Have**:

    - Basic logging of actions (submit, approve, reject) with timestamps and user details.
- **1 - Good to Have**:
    - Display logs in a modal or separate table view that managers and admins can access.
- **2 - Killer Thing**:
    - Include the previous state and changes in each log entry (e.g., amount changed from X to Y).
    - Ability to download the audit logs as CSV.

Authentication & RBAC

- **0 - Must Have**:
    - Basic authentication using **NextAuth** (Google or GitHub provider).
    - Protect routes based on user roles (Employee, Manager).
- **1 - Good to Have**:
    - Session management with refresh tokens for long-lived sessions and logout flow.
- **2 - Killer Thing**:
    - Implement role-based access at a more granular level (e.g., only specific managers can approve transactions for certain employees).

---

4. Non-Functional Requirements

0 - Must Have:

- Basic performance handling: Ensure that the app responds quickly for users with a small dataset.

1 - Good to Have:

- Pagination for the transaction table if there are a large number of entries.

2 - Killer Thing:

- Optimize for high traffic: Implement caching mechanisms (like **SWR** or **React Query**) for better data fetching and client-side caching.

---

5. User Interface

0 - Must Have:

- A simple dashboard layout with a transaction table using **TanStack Table**.
  - The table should display key fields: transaction type, amount, status, and action buttons.
  - Use **Tailwind CSS** for responsive design and a clean, minimal UI.

1 - Good to Have:

- Use **shadcn** components (e.g., buttons, modals) for a cohesive, polished UI.

2 - Killer Thing:

- Implement dark/light mode toggle using Tailwind and Next.js.

---

6. Technical Stack

0 - Must Have:

- **Next.js** for both frontend and backend API routes.
- **Tailwind CSS** for styling.
- **NextAuth** for authentication and role-based access control.
- **TanStack Table** for table management.
- **MongoDB** or **SQLite** for a lightweight database.

1 - Good to Have:

- **zod** for validation of forms and API payloads.

2 - Killer Thing:

- Use **TypeScript** throughout for type safety.

---

7. APIs

0 - Must Have:

- **POST** `/api/transactions`: Submit a new transaction (Employee).
- **GET** `/api/transactions`: Retrieve transactions filtered by user role.
- **PUT** `/api/transactions/:id/approve`: Approve a transaction (Manager).
- **PUT** `/api/transactions/:id/reject`: Reject a transaction (Manager).

1 - Good to Have:

- **GET** `/api/transactions/:id/audit-log`: Retrieve audit logs for a specific transaction.

---

8. Project Timeline

Day 1:

- Set up the project using **Next.js**, **Tailwind CSS**, and **NextAuth** for authentication.
- Implement basic transaction submission and RBAC.
- Display transactions using **TanStack Table**.

Day 2:

- Implement the approval/rejection workflow for managers.
- Add audit logging functionality for transactions.

Day 3:

- Refine the UI using **shadcn** components.
- Test role-based access control and basic filters.
- Stretch: Add pagination, filters, or session management.

Simplifications:

- **Must Haves (0)**: Focus on transaction submission, approval workflow, basic RBAC, and minimal audit logs.
- **Good to Haves (1)**: Improve UI, add pagination and audit log display, and add some filters.
- **Killer Things (2)**: Advanced audit logging, granular RBAC, and performance optimization.