Doctor Approval Workflow Module Design Specification

This document outlines the design and implementation details for the Doctor Approval Workflow module. This module provides Doctors (or Approvers) with the functionality to review and formally approve or reject lab result documents.

1. Module Purpose

The Doctor Approval module is a critical gatekeeping component in the lab's operational workflow. Its purpose is to allow authorized medical professionals (Doctors/Approvers) to perform a final review of entered lab results and attached reports. Upon their decision, the lab document's status is updated, triggering subsequent actions like customer notification and invoice processing.

2. Key Features

A. Document Approval Page

- **List Pending Documents:** Doctors will see a dedicated list of lab result documents that are awaiting their review and approval. This list should be filterable by request number, date, company, or lab internal status.
- **Review Lab Document:**
  - Ability to view comprehensive details of the TestRequest and its associated LabTest and LabResult entries.
  - Access to all attached final lab result documents (e.g., PDF reports) for review.
  - View audit trail relevant to result entry (who entered, when).
- **Actions:**
  - **Approve:**
    - Mark the associated LabTest record as approved.
    - Update the test_request.lab_internal_status to approved.
    - Update test_request.document_status to Approved.
    - Trigger a notification to the customer (e.g., email) indicating that results are available.
    - Enable the release of the invoice for customer payment.
  - **Reject:**
    - Mark the associated LabTest record as rejected.
    - Update the test_request.lab_internal_status to rejected.
    - Update test_request.document_status to Rejected.
    - Provide a mandatory reason for rejection.
    - Potentially change the status back to Lab Result Entry (or a similar status) to allow technicians to re-test or re-enter results.

        - Trigger a notification to the responsible lab technician for review.

3. Frontend Design Specifications (React with React Router & Tailwind CSS)

A. Pages & Components

1. **/doctor/approval (Document Approval Page):**
    - **Layout:** DoctorLayout with a simplified navigation focused on approval tasks.
    - **Components:**
        - PendingDocumentsTable component:
            - Displays a paginated list of TestRequest objects where lab_internal_status is Waiting Doctor Approval.
            - Columns: Request Date, Request No., Company, Requester, Lab Internal Status, Actions.
            - Search/Filter bar (Shadcn UI Input for text search, DatePicker for date range, Select for Company).
        - Table component (Shadcn UI Table).
        - Pagination component (Shadcn UI Pagination).
        - Contextual DropdownMenu or Button per row for "Review" and "Reject" actions.
        - DocumentReviewModal component:
            - Displayed when "Review" is clicked.
            - Shows detailed TestRequest, TestRequestSample, LabTest, LabResult data.
            - Includes an area to display attached DocumentAttachment files (e.g., embed PDFs, display images).
            - "Approve Document" button (Shadcn UI Button).
            - "Reject Document" button (Shadcn UI Button).
            - Textarea for rejection reason (visible only on Reject click).
        - ConfirmationDialog (Custom AlertDialog using Shadcn UI AlertDialog): For "Approve" and "Reject" actions, requesting confirmation from the doctor.
    - **State Management:**
        - React Query (useQuery): Fetch the list of TestRequests awaiting doctor approval.
        - React Query (useMutation): For approving or rejecting lab documents.
        - Local state for search queries, filters, pagination, and modal visibility.
    - **Interactions:**
        - Clicking "Review" opens the DocumentReviewModal with the selected document's details.

        - Inside the modal, "Approve Document" and "Reject Document" buttons trigger mutations after confirmation.
        - If "Reject" is clicked, a text area for the rejection reason becomes visible and mandatory.
        - Show loading states on buttons and for data fetching.
        - Display success/error toasts for approval/rejection outcomes.

B. Routing (React Router)

- path: '/doctor/approval' (Protected route, requires doctor or admin role)
    - loader to fetch the list of TestRequests awaiting doctor approval.
- path: '/doctor/approval/:requestId/review' (Optional: if review is a separate page instead of modal)
    - loader to fetch a single TestRequest for detailed review.
    - action to handle approval/rejection.

C. Styling

- **Tailwind CSS:** Consistent application for all layouts, tables, and modal components.
- **Shadcn UI:** Leverage its components for a polished and accessible UI, ensuring consistent spacing, typography, and component styling.
- **Responsive Design:** Tables should be responsive. The DocumentReviewModal should be scrollable and adapt to different screen sizes.

D. Validation

- **Client-side (Zod + React Hook Form):**
    - Rejection reason is mandatory when rejecting.
    - Basic validation on any input fields within the modal.
- **Server-side (Zod in Express):** Crucially re-validate all incoming data for security and data integrity, especially the approval/rejection actions and associated reasons.

4. Backend Design Specifications (Express.js with TypeScript, PostgreSQL, Prisma)

A. Relevant Database Tables (from lab-tracking-webapp-plan immersive)

- users (for doctor's role and ID)
- test_requests (to update document_status and lab_internal_status)
- lab_tests (to update lab_result_status)
- lab_results (for review, no direct update here)
- document_attachments (to retrieve attached reports)

- audit_trail (for logging approval/rejection actions)
- invoices (status check/generation trigger)

B. API Endpoints

1. **GET /api/v1/doctor/pending-approvals (List Pending Documents for Doctor Approval)**
    - **Purpose:** Retrieve all test requests that are in the Waiting Doctor Approval status.
    - **Request:** Optional query parameters for pagination (page, limit), search (searchQuery on request_no, company_name, requester_name), and filtering (requestDateFrom, requestDateTo).
    - **Response (200 OK):** { data: TestRequest[], total: number, page: number, limit: number } (TestRequest objects, including minimal customer and related lab_test info for display).
    - **Errors (403 Forbidden):** If the authenticated user is not a doctor or admin.
    - **Middleware:** authMiddleware, roleMiddleware(['doctor', 'admin']), Input validation for query params.
    - **Logic:**
        - Query test_requests where lab_internal_status is waiting_doctor_approval.
        - Include necessary relations (e.g., customer, lab_tests).
        - Apply filters and pagination.
        - Log action to audit_trail (e.g., 'doctor_viewed_pending_approvals').
2. **GET /api/v1/doctor/approvals/:requestId (Get Document Details for Review)**
    - **Purpose:** Retrieve full details of a specific test request (including lab results and attachments) for the doctor's review.
    - **Request:** URL parameter :requestId (UUID of the test_request).
    - **Response (200 OK):** TestRequest object with nested TestRequestSample, LabTest, LabResult, DocumentAttachment (for lab reports), and Customer details.
    - **Errors (404 Not Found):** If request does not exist.
    - **Middleware:** authMiddleware, roleMiddleware(['doctor', 'admin']).
    - **Logic:**
        - Fetch TestRequest by :requestId, including all relevant relations needed for review (customer, test_request_samples, lab_tests with lab_results, and document_attachments where entity_type is lab_result).
        - Log action to audit_trail (e.g., 'doctor_reviewed_document').
3. **POST /api/v1/doctor/approvals/:requestId/approve (Approve Lab Document)**
    - **Purpose:** Mark a lab document as approved.
    - **Request:** application/json (optional, can be empty or include small details).

        - URL parameter :requestId (UUID of the test_request).
    - **Response (200 OK):** { message: 'Document approved successfully.' }
    - **Errors (400 Bad Request):** If request not in waiting_doctor_approval status.
    - **Middleware:** authMiddleware, roleMiddleware(['doctor', 'admin']).
    - **Logic:**
        - Start a database transaction.
        - Fetch TestRequest by :requestId, ensure lab_internal_status is waiting_doctor_approval.
        - Update test_request.lab_internal_status to approved.
        - Update test_request.document_status to Approved.
        - For each associated LabTest record, update lab_test.lab_result_status to approved.
        - Commit transaction.
        - Log action to audit_trail (e.g., 'lab_document_approved', details: request ID, approved by user).
        - **Trigger Notification:** Send email to customer (customer_notification_service.sendResultsApprovedEmail(requestId))
        - **Trigger Invoice Release:** (This might be a background task or handled by the InvoiceService generateInvoice(requestId))
4. **POST /api/v1/doctor/approvals/:requestId/reject (Reject Lab Document)**
    - **Purpose:** Mark a lab document as rejected.
    - **Request:** application/json
        - URL parameter :requestId (UUID of the test_request).
        - Body: rejectionReason: string.
    - **Response (200 OK):** { message: 'Document rejected successfully.' }
    - **Errors (400 Bad Request):** If request not in waiting_doctor_approval status, rejectionReason missing.
    - **Middleware:** authMiddleware, roleMiddleware(['doctor', 'admin']), Input validation.
    - **Logic:**
        - Start a database transaction.
        - Fetch TestRequest by :requestId, ensure lab_internal_status is waiting_doctor_approval.
        - Update test_request.lab_internal_status to lab_result_entry (or a specific 're_entry_required' status).
        - Update test_request.document_status to Rejected.
        - For each associated LabTest record, update lab_test.lab_result_status to rejected.
        - Store rejectionReason in test_requests.notes or a new rejection_reason

column.
    - Commit transaction.
    - Log action to audit_trail (e.g., 'lab_document_rejected', details: request ID, rejected by user, reason).
    - **Trigger Notification:** Send email to responsible lab technician (lab_technician_notification_service.sendRejectionNotification(requestId, reason))

C. Prisma Operations

- PrismaClient.testRequest.findMany(), findUnique(), update().
- PrismaClient.labTest.updateMany().
- PrismaClient.documentAttachment.findMany() (for retrieving lab reports).
- PrismaClient.$transaction(): Crucial for atomic updates during approval/rejection.
- PrismaClient.auditTrail.create().

D. Middleware Requirements

- authMiddleware: Verifies JWT and populates req.user.
- roleMiddleware(['doctor', 'admin']): Ensures the authenticated user has the necessary role.
- validateMiddleware(schema): For validating request bodies (e.g., rejectionReason).

E. Business Logic / Service Functions

- DoctorApprovalService:
    - getPendingApprovals(filters): Retrieves TestRequests for the approval queue.
    - getApprovalDetails(requestId): Fetches comprehensive data for a single document review.
    - approveDocument(requestId, doctorId): Handles status updates, triggers notifications/invoice.
    - rejectDocument(requestId, doctorId, rejectionReason): Handles status updates, stores reason, triggers notifications.
- TestRequestService: (Collaborates with DoctorApprovalService for status updates)
- LabTestService: (Collaborates with DoctorApprovalService for lab_test_status updates)
- NotificationService: For sending emails to customers and technicians.
- InvoiceService: (Responsible for generating/releasing invoices upon approval).
- AuditService: For logging all doctor actions.

F. Error Handling

- Custom error classes (e.g., NotFoundError, ForbiddenError, ValidationError, InvalidStatusTransitionError).
- Centralized Express error handling middleware to catch these errors and return appropriate HTTP status codes and messages.

5. Shared Types (from packages/shared/types.ts)

Define interfaces for:

- TestRequest (and its statuses: document_status and lab_internal_status).
- LabTest (and its lab_result_status).
- LabResult.
- DocumentAttachment (for lab reports).
- Payloads for:
  - RejectDocumentPayload: { rejectionReason: string }.
- Filter/pagination types for pending approvals (PendingApprovalFilters).
- Enums for DocumentStatus and LabInternalStatus.

6. Edge Cases & Considerations

- **Status Transitions:** Enforce that approval/rejection can only happen when the lab_internal_status is waiting_doctor_approval. Prevent double approvals/rejections.
- **Invoice Generation:** Clarify when the invoice is *generated* versus *released*. Typically, it's generated once results are available and accurate, but only *released* for payment to the customer after doctor approval. The backend logic for approveDocument should either trigger invoice generation or change the invoice status to ready_for_payment.
- **Notification Content:** Ensure notifications are clear, professional, and contain relevant links (e.g., to the customer portal for approved results).
- **Rejection Workflow:** If a document is rejected, the system needs a clear path for the lab technician to access it, make corrections, and resubmit for approval. The lab_internal_status might be set to lab_result_entry or a new re_entry_required status to facilitate this.
- **Audit Trail:** Capture the doctor's ID, the action taken (approve/reject), the request ID, and the rejection reason (if applicable) in the audit trail.
- **Reviewing Large Documents:** If lab reports are large PDF files, consider optimizing their display in the frontend (e.g., using a PDF viewer library, server-side rendering of pages for faster load).
- **Concurrency:** If multiple doctors might try to approve/reject the same document, implement pessimistic or optimistic locking on the backend to prevent race

conditions.