Lab Internal Operations Module Design Specification

This document outlines the design and implementation details for the Internal Lab Operations module. This module provides the core functionalities for lab staff (Admin, Lab Technician, Doctor) to manage and process customer test requests from reception to result delivery.

1. Module Purpose

This module serves as the operational hub for the lab. It allows administrators to oversee all requests, technicians to receive physical samples and record results, and provides a clear workflow for processing tests. It ensures proper tracking of samples and experiments within the lab's internal processes.

2. Key Features

A. Admin Dashboard & Request Overview

- **Centralized View:** A comprehensive dashboard for Admins and Lab Managers to oversee all test requests.
- **Search & Filtering:** Powerful search capabilities by Request Number, Document Status (customer-facing), Objective, Requester, Company, and Request Date Range.
- **Request Listing:** Display all customer requests with key information: Request Date, Request No., Company, Objective, Requester, and their current Document Status.
- **Document Status (Internal Lab Lifecycle View):**
  - Waiting Approval Lab: Request is submitted by customer, waiting for lab to review.
  - Submitted: (Same as customer submitted status, internal lab acknowledging receipt of the *request*, not physical sample yet).
  - Acknowledged: Lab has acknowledged the request and physically received samples.
  - Lab Result Entry: Samples are being processed, and results are being entered.
  - Waiting Doctor Approval: Lab results are entered and awaiting review/approval by a Doctor.
  - Approved: Lab results have been approved by a Doctor.
  - Paid: Invoice for the request has been paid by the customer.
  - Rejected: Request or results have been rejected at any stage.
- **Actions (Contextual based on role & status):**
  - Acknowledge (Receive Sample): For Lab Technicians/Admins to initiate sample

reception.
    - View Lab Result: To view the detailed lab results (for Admin/Doctor).
    - Approve/Reject: For Admin/Doctor to approve or reject the final lab document.
    - Mark as Paid: For Admin to manually mark an invoice as paid (after verifying payment slip).

B. Receive Sample & Acknowledge Request (Lab Technician)

- **Request Details Display:** Show the full customer request information, including all samples and their requested quantities.
- **Quantity Adjustment:** Ability for the Lab Technician to adjust the received_qty for each sample, if the actual received quantity differs from the requested_qty.
- **Storage Assignment:** Option to assign samples to specific storage locations (e.g., freezer, rack, box, well).
- **Acknowledgement:** A button/action to confirm sample reception, which updates the test_request.document_status to Acknowledged and lab_internal_status to Lab Result Entry. This action also updates test_request_samples.current_status to received.

C. Lab Result Entry (Lab Technician)

- **Request & Case Info:** Display the Test Request Number, Lab internal Case Number, Case Date, Customer Company, Sender Name, and the Admin/Technician who received the samples.
- **Result Input Interface:** A dynamic interface to enter detailed test results for each sample/panel. This may include:
    - Parameter Name
    - Result Value (can be text, number, or structured JSON for complex results)
    - Unit of Measurement
    - Reference Range (optional)
    - Is Abnormal (checkbox/toggle)
    - Notes
- **Attachment:** Ability to attach final lab result documents (e.g., PDF reports, raw data files) that will be accessible to the customer upon approval.
- **Result Status:** Technicians can set the status of the lab result (e.g., Pending Review, Completed).
- **Request Approval Button:** A dedicated button to submit the entered results for Doctor's approval, changing lab_internal_status to Waiting Doctor Approval.

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

A. Pages & Components

1. ***/lab/requests* (Admin Dashboard & Request Overview Page):**
   - **Layout:** AdminLayout with sidebar navigation.
   - **Components:**
     - RequestOverviewTable component:
       - Displays a paginated list of TestRequest objects.
       - Columns: Request Date, Request No., Company, Objective, Requester, Document Status (internal view), Actions.
       - Search/Filter bar (Shadcn UI Input for text search, multiple Select for Document Status and lab_internal_status, DatePicker for date range, Select for Company/Requester).
       - Contextual action buttons per row (Shadcn UI Button or DropdownMenu for more actions):
         - "Acknowledge Sample" (visible if lab_internal_status is Waiting Approval Lab or Submitted).
         - "View Lab Result" (visible if lab_internal_status is Lab Result Entry or Waiting Doctor Approval or Approved).
         - "Approve/Reject" (visible if lab_internal_status is Waiting Doctor Approval, for Admin/Doctor roles).
         - "Mark as Paid" (visible if document_status is not Paid, for Admin role).
     - Table component (Shadcn UI Table).
     - Pagination component (Shadcn UI Pagination).
     - Custom AlertDialog for confirmation (e.g., "Mark as Paid").
   - **State Management:**
     - React Query (useQuery): Fetch list of TestRequests for the overview. Query key includes all search/filter parameters.
     - Local state for search queries, active filters, pagination.
   - **Interactions:**
     - Clicking action buttons triggers navigation or opens modals/dialogs for specific workflows.
     - Show loading states for data fetching and mutations.
     - Display success/error toasts.
2. ***/lab/requests/:requestId/receive-sample* (Receive Sample & Acknowledge Request Page):**
   - **Layout:** LabLayout.
   - **Components:**
     - RequestDetailsDisplay component: Shows customer request details.

        - SampleReceptionForm component:
            - A table or list displaying each TestRequestSample with customer_sample_id, requested_qty, unit.
            - Input field (Shadcn UI Input with type number): For received_qty per sample.
            - Select/Dropdown (Shadcn UI Select): For storage_location_id for each sample.
            - "Confirm Sample Reception" button (Shadcn UI Button).
    - **State Management:**
        - React Query (useQuery): Fetch the specific TestRequest with its test_request_samples.
        - React Hook Form: Manage form state for received_qty and storage_location_id for each sample.
        - Zod: Client-side validation for quantities (e.g., non-negative).
        - React Query (useMutation): For submitting the sample reception data.
    - **Interactions:**
        - Pre-fill received_qty with requested_qty by default.
        - Show loading state during submission.
        - Display validation errors.
        - Redirect back to /lab/requests on successful acknowledgement.
3. **/lab/results/entry/:requestId (Lab Result Entry Page):**
    - **Layout:** LabLayout.
    - **Components:**
        - LabInfoDisplay component: Shows Request No., Case No., Case Date, Company, Sender, Admin who received sample.
        - LabTestResultsEditor component:
            - Uses useFieldArray from React Hook Form for dynamic list of LabResult entries per LabTest.
            - For each result: Input fields for parameter, value, unit, reference_range, is_abnormal (checkbox), notes.
            - "Add Result Parameter" button.
            - "Remove" button per result item.
        - FileUpload component (Custom): For attaching final lab result documents.
        - ResultStatusSelect component (Shadcn UI Select): To set lab_test.lab_result_status.
        - "Save Draft" button (Shadcn UI Button).
        - "Submit for Doctor Approval" button (Shadcn UI Button).
    - **State Management:**
        - React Query (useQuery): Fetch relevant TestRequest, LabTest, and

existing LabResult data.
        - React Hook Form: Manages the entire form state for result entry.
        - Zod: Client-side validation for result data.
        - React Query (useMutation): For saving draft results or submitting for approval.
    - **Interactions:**
        - Pre-fill form fields if editing existing results.
        - Show loading state during submission.
        - Display validation errors.
        - Redirect back to /lab/requests (or a dedicated lab-results list) on successful save/submit.

B. Routing (React Router)

- path: '/lab/requests' (Protected, admin, lab_manager, lab_technician roles)
    - loader to fetch TestRequest list for overview.
- path: '/lab/requests/:requestId/receive-sample' (Protected, lab_technician, admin roles)
    - loader to fetch specific TestRequest for sample reception.
    - action to handle sample acknowledgement.
- path: '/lab/results/entry/:requestId' (Protected, lab_technician, admin roles)
    - loader to fetch specific TestRequest and associated LabTest for result entry.
    - action to handle lab result saving/submission.

C. Styling

- **Tailwind CSS:** Consistent application for all layouts, forms, tables, and buttons.
- **Shadcn UI:** Leverage its components for a polished and accessible look. Ensure consistent spacing, typography, and component styling.
- **Responsive Design:** All tables should be responsive (e.g., horizontal scroll, or conditional rendering for smaller screens). Forms should adapt to various screen sizes.

D. Validation

- **Client-side (Zod + React Hook Form):**
    - received_qty: Non-negative, not exceeding requested quantity (if applicable).
    - storage_location_id: Required.
    - LabResult: Parameter name, value, unit are required. Value format based on type.
    - File upload validation (max size, allowed types for lab reports).
- **Server-side (Zod in Express):** Re-validate all incoming data rigorously for

security and data integrity.

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

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

- users
- customers
- test_requests
- test_request_samples
- storage_locations
- lab_tests
- lab_results
- document_attachments
- audit_trail
- invoices (for marking as paid)

B. API Endpoints

1. **GET /api/v1/lab/requests (List All Requests for Lab)**
    - **Purpose:** Retrieve a list of all test requests for lab staff overview.
    - **Request:** Optional query parameters for pagination (page, limit), search (searchQuery on request_no, company_name, requester_name, objective), and filtering (documentStatus, labInternalStatus, requestDateFrom, requestDateTo, companyId, requesterId).
    - **Response (200 OK):** { data: TestRequest[], total: number, page: number, limit: number } (TestRequest objects including customer and related fields).
    - **Errors (403 Forbidden):** If the authenticated user does not have an appropriate lab role (admin, lab_manager, lab_technician, doctor).
    - **Middleware:** authMiddleware, roleMiddleware(['admin', 'lab_manager', 'lab_technician', 'doctor']), Input validation for query params.
    - **Logic:**
        - Query test_requests table, including customer information.
        - Apply all requested filters and pagination.
        - Log action to audit_trail (e.g., 'lab_user_viewed_requests_overview').
2. **GET /api/v1/lab/requests/:id (Get Single Request Details for Lab)**
    - **Purpose:** Retrieve full details of a specific test request for internal lab use.
    - **Request:** URL parameter :id (UUID of the request).
    - **Response (200 OK):** TestRequest object with nested TestRequestSample array, customer details, and potentially associated LabTest and LabResult

    data.
  - **Errors (404 Not Found):** If request does not exist.
  - **Middleware:** authMiddleware, roleMiddleware(['admin', 'lab_manager', 'lab_technician', 'doctor']).
  - **Logic:**
    - Fetch TestRequest by :id, including all relevant relations (customer, test_request_samples, lab_tests, lab_results).
    - Log action to audit_trail (e.g., 'lab_user_viewed_request_details').
3. **POST /api/v1/lab/requests/:id/acknowledge-sample (Acknowledge Sample Reception)**
   - **Purpose:** Mark a request's samples as received and update status.
   - **Request:** application/json
     - URL parameter :id (UUID of the test_request).
     - Body: samples: [{ id: UUID, receivedQty: number, storageLocationId: UUID }].
   - **Response (200 OK):** { message: 'Samples acknowledged and status updated.' }
   - **Errors (400 Bad Request):** Invalid quantities, request not in correct status, sample not found.
   - **Middleware:** authMiddleware, roleMiddleware(['lab_technician', 'admin']), Input validation.
   - **Logic:**
     - Validate incoming data (e.g., receivedQty is valid).
     - Start a database transaction.
     - Update test_request.document_status to acknowledged_sample_received.
     - Update test_request.lab_internal_status to lab_result_entry.
     - For each sample in the request body:
       - Update test_request_samples.received_qty and test_request_samples.storage_location_id.
       - Update test_request_samples.current_status to received.
     - Commit transaction.
     - Log action to audit_trail (e.g., 'sample_received_acknowledged', details: request ID, samples received, received by user).
     - Consider sending an email notification to the customer.
4. **POST /api/v1/lab/results/entry/:requestId (Submit Lab Results)**
   - **Purpose:** Submit lab test results for a given request.
   - **Request:** multipart/form-data (for attachments) or application/json (for results data, then separate file upload).

    - URL parameter :requestId (UUID of the test_request).
    - Body: labTests: [{ testRequestSampleId: UUID, caseNo: string, caseDate: date, assignedLabTechnicianId: UUID, testPanel: string, testMethod: string, labResultStatus: 'pending' | 'completed', results: LabResult[], attachments: File[] }].
  - **Response (201 Created):** { message: 'Lab results submitted successfully.', labTestId: '...' }
  - **Errors (400 Bad Request):** Validation errors, request not in lab_result_entry status.
  - **Middleware:** authMiddleware, roleMiddleware(['lab_technician', 'admin']), multer (if handling files directly here), Input validation.
  - **Logic:**
    - Validate input, ensuring all required result fields are present.
    - Start a database transaction.
    - For each labTest entry in the request:
      - Create/Update LabTest record (if updating existing test, or creating new ones).
      - Create/Update associated LabResult entries.
      - Upload attachments to Google Cloud Storage and create DocumentAttachment records.
    - Update test_request.lab_internal_status to waiting_doctor_approval.
    - Commit transaction.
    - Log action to audit_trail (e.g., 'lab_results_entered', details: request ID, results entered by user).
5. **PUT /api/v1/lab/results/entry/:labTestId (Update Specific Lab Test Results)**
  - **Purpose:** Update an existing lab test and its results.
  - **Request:** application/json (or multipart/form-data if attachments are updated).
    - URL parameter :labTestId.
    - Body: caseNo, caseDate, testPanel, testMethod, labResultStatus, results: LabResult[], attachments: File[].
  - **Response (200 OK):** { message: 'Lab test results updated successfully.' }
  - **Errors (400 Bad Request):** Validation errors, trying to update an approved test.
  - **Middleware:** authMiddleware, roleMiddleware(['lab_technician', 'admin']), Input validation.
  - **Logic:**
    - Fetch existing LabTest by :labTestId.
    - Validate status (e.g., cannot update if already approved).

        - Update LabTest fields.
        - Handle LabResult updates (add, modify, delete).
        - Handle DocumentAttachment updates (add new, delete old).
        - Log action to audit_trail (e.g., 'lab_results_updated').

6. **PATCH /api/v1/lab/requests/:id/mark-paid (Mark Request as Paid)**
    - **Purpose:** For Admin to manually mark an invoice/request as paid.
    - **Request:** application/json (optional paymentDate).
        - URL parameter :id (UUID of the test_request).
    - **Response (200 OK):** { message: 'Request/Invoice marked as paid.' }
    - **Errors (400 Bad Request):** If request/invoice already paid, not found.
    - **Middleware:** authMiddleware, roleMiddleware('admin').
    - **Logic:**
        - Fetch associated Invoice for the TestRequest.
        - Update invoices.payment_status to paid.
        - Update test_requests.document_status to paid.
        - Log action to audit_trail (e.g., 'invoice_marked_paid', details: request ID, user who marked).

C. Prisma Operations

- PrismaClient.testRequest.findMany(), findUnique(), update().
- PrismaClient.testRequestSample.updateMany().
- PrismaClient.labTest.create(), update(), findUnique().
- PrismaClient.labResult.createMany(), deleteMany(), update().
- PrismaClient.invoice.update().
- PrismaClient.documentAttachment.create(), deleteMany().
- PrismaClient.$transaction(): For complex operations involving multiple table updates (e.g., sample acknowledgement, result submission).
- PrismaClient.auditTrail.create().

D. Middleware Requirements

- authMiddleware: Verifies JWT and populates req.user.
- roleMiddleware(['admin', 'lab_manager', 'lab_technician', 'doctor']): Ensures appropriate lab role for access.
- validateMiddleware(schema): For all incoming request data.
- multer middleware: For handling file uploads (.single('file') or .array('files')).

E. Business Logic / Service Functions

- TestRequestService:
    - getLabRequests(filters): Retrieves requests for lab overview.
    - getLabRequestById(id): Retrieves a single request with all lab-relevant details.

    - acknowledgeSamples(requestId, samplesData, userId): Handles received_qty updates and status changes.
- LabTestService:
    - submitLabResults(requestId, labTestsData, userId): Handles creation/update of LabTest and LabResult records, and associated attachments.
    - updateLabTestResults(labTestId, data, userId): For specific lab test updates.
- InvoiceService:
    - markInvoiceAsPaid(requestId, userId): Handles updating invoice status.
- FileService: Manages interactions with Google Cloud Storage for lab report attachments.
- AuditService: For logging all lab operations.

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 (including all its statuses: document_status and lab_internal_status).
- TestRequestSample (with received_qty, storage_location_id).
- LabTest (with case_no, lab_result_status).
- LabResult (with parameter, value, unit, is_abnormal, notes).
- Payloads for:
    - AcknowledgeSamplePayload: { samples: { id: UUID, receivedQty: number, storageLocationId: UUID }[] }.
    - SubmitLabResultsPayload: { labTests: { testRequestSampleId: UUID, caseNo: string, caseDate: date, assignedLabTechnicianId: UUID, testPanel: string, testMethod: string, labResultStatus: 'pending' | 'completed', results: LabResult[], attachments: File[] }[] }.
    - MarkPaidPayload.
- Filter/pagination types for lab requests (LabRequestFilters).
- Enums for DocumentStatus and LabInternalStatus (distinct values).

6. Edge Cases & Considerations

- **Status Transitions:** Enforce strict backend logic for document_status and lab_internal_status transitions. For example, results cannot be entered if samples haven't been acknowledged.

- **Concurrency:** When multiple technicians might be working on requests, ensure atomic updates for status changes and quantity adjustments, potentially using locking or transactions.
- **Case Number Generation:** Decide if case_no is auto-generated by the system or manually entered by the technician. If auto-generated, ensure its uniqueness and format (e.g., "LAB-YYMMDD-SEQ").
- **Lab Result Data Structure:** The value field in LabResult is TEXT or JSONB. JSONB is powerful for complex or varying result structures (e.g., for different test panels). Consider a flexible schema here.
- **File Naming & Management:** Implement a clear naming convention for files uploaded to Google Cloud Storage (e.g., lab-reports/:requestId/:sampleId/:fileName). Ensure secure access.
- **Notifications:** Send internal notifications (e.g., to Doctors when results are ready for approval) and external notifications (to customers when samples are acknowledged or results are approved).
- **User Interface for Complex Results:** The LabTestResultsEditor needs to be highly flexible and user-friendly, potentially allowing different input types based on the panel or parameter.
- **Audit Trail:** Log all critical actions: sample acknowledgment, result entry, result status changes, and any user who performs them.
- **Historical Data:** Ensure that all previous versions or changes to results are traceable if required for compliance.