Customer Portal Features Module Design Specification

This document outlines the design and implementation details for the Customer Portal Features module. This module provides customers with functionalities to manage their test requests, submit new requests, and view/manage invoices.

1. Module Purpose

The Customer Portal is the primary interface for external customers to interact with the lab. It allows them to initiate test requests, track the status of their submissions, manage their associated samples, and handle financial aspects like viewing invoices and submitting payment proofs.

2. Key Features

A. Document Request List

- **Listing & Search:** Customers can view a comprehensive list of all their submitted test requests. This list should be searchable and filterable.
- **Key Information Display:** For each request, display: Request Date, Request Number, Company Name (customer's), Requester (contact person at customer's company), and current Document Status.
- **Document Status Lifecycle (Customer View):**
  - Submitted: Request has been sent to the lab.
  - Acknowledged and Received Sample: Lab has received the request and physical samples.
  - Paid: Invoice for the request has been paid by the customer.
  - Approved: Lab results have been approved by the doctor.
  - Rejected: Request or results have been rejected by the lab/doctor.
- **Actions:**
  - View: Open a detailed view of the request.
  - Edit: Modify the request (only if status is Submitted or Draft).
  - Delete: Remove the request (only if status is Draft or Submitted and not yet Acknowledged).
  - Print Request Summary: Generate a printable summary of the request.

B. Add/Edit Requesting Test

- **Auto-Generated Request No.:** The system automatically generates a unique request number upon saving a draft or submitting the request. This number can be based on company identifier, sender details, and date.
- **Sample List Management:** A dynamic, interactive list to add, edit, or remove

individual samples associated with the request.
    - **Sample Details:** For each sample, capture: Sample ID (customer's own identifier), Sent Sample Date, Animal Type, Sample Specimen (e.g., blood, tissue), Panel (of tests to be performed), Method (specific analysis method), and Sample Quantity.
- **Saving Options:**
    - Save Draft: Allows customers to save an incomplete request to continue working on it later. Drafts are not sent to the lab.
    - Submit to Approval Labs: Finalizes the request and sends it to the lab for processing. This action changes the request status from Draft or Submitted (if editing) to Submitted for the lab.

C. Invoice Page

- **Detailed Invoice Display:** Present a full invoice including:
    - Lab's Company Information (configured globally).
    - Customer's Tax Company Information (from customer profile).
    - Invoice Number.
- **Itemized Details:** A table listing each service/test with: Running Item ID, Detail (description), Quantity, Unit Price, and Total for that item.
- **Summary:** Clearly display Sub-total, Tax (7%), and Net Total.
- **Actions:**
    - Print Invoice: Generate a printable version of the invoice.
    - Attach Pay Slip: Allow the customer to upload proof of payment (e.g., bank transfer screenshot).

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

A. Pages & Components

1. **/customer/requests (Document Request List Page):**
    - **Layout:** CustomerLayout with sidebar navigation.
    - **Components:**
        - RequestTable component:
            - Displays a paginated list of TestRequest objects.
            - Columns: Request Date, Request No., Company Name, Requester, Document Status, Actions (View, Edit, Delete, Print).
            - Search/Filter bar (Shadcn UI Input for text search, Select for status filtering, DatePicker for date range).
            - "Add New Request" button (Shadcn UI Button).

        - Table component (Shadcn UI Table).
        - Pagination component (Shadcn UI Pagination).
        - Custom AlertDialog (Shadcn UI AlertDialog) for delete confirmation.
    - **State Management:**
        - React Query (useQuery): Fetch list of customer TestRequests. Query key should include search/filter parameters for automatic re-fetching.
        - Local state for search queries, active filters, pagination.
    - **Interactions:**
        - Clicking "Add New Request" navigates to /customer/requests/add.
        - Clicking "View" on a row navigates to /customer/requests/:requestId.
        - "Edit" button enabled only if document_status is Draft or Submitted. Navigates to /customer/requests/:requestId/edit.
        - "Delete" button enabled only if document_status is Draft or Submitted (and optionally, not yet acknowledged_sample_received). Triggers confirmation.
        - Show loading states for data fetching and mutations. Display success/error toasts.
2. **/customer/requests/add or /customer/requests/:requestId/edit (Add/Edit Requesting Test Page):**
    - **Layout:** CustomerLayout.
    - **Components:**
        - TestRequestForm component:
            - Input fields (Shadcn UI Input): Objective, Requester Name.
            - SampleListEditor component:
                - Uses useFieldArray from React Hook Form to manage an array of sample objects.
                - For each sample: Input fields for Sample ID, DatePicker for Sent Sample Date, Select for Animal Type, Sample Specimen, Panel, Method, Quantity.
                - "Add Sample" button.
                - "Remove" button per sample item.
            - "Save Draft" button (Shadcn UI Button).
            - "Submit to Approval Labs" button (Shadcn UI Button).
    - **State Management:**
        - React Hook Form: Manages the entire form state, including the dynamic sample list.
        - Zod: Define schema for client-side validation of the request and each sample.
        - React Query (useQuery): If editing, fetches the existing TestRequest data.

        - React Query (useMutation): For saving (POST) drafts or submitting (PUT) requests.
    - **Interactions:**
        - On "Add": Form is empty.
        - On "Edit": Form is pre-filled with existing TestRequest data and samples.
        - Show loading state during submission.
        - Display validation errors.
        - Redirect to /customer/requests on successful save/submit.

3. **/customer/invoices (Invoice List Page):**
    - **Layout:** CustomerLayout.
    - **Components:**
        - InvoiceTable component:
            - Displays a paginated list of Invoice objects related to the customer.
            - Columns: Invoice No., Request No., Invoice Date, Net Total, Payment Status, Actions (View, Print, Attach Pay Slip).
            - Search/Filter bar.
        - Table component (Shadcn UI Table).
        - Pagination component (Shadcn UI Pagination).
    - **State Management:** React Query (useQuery) for fetching invoices.
    - **Interactions:**
        - "View" on a row navigates to /customer/invoices/:invoiceId.
        - "Attach Pay Slip" button (per row) opens a modal with a file upload component.
4. **/customer/invoices/:invoiceId (Invoice Detail Page):**
    - **Layout:** CustomerLayout.
    - **Components:**
        - InvoiceDetail component: Displays all invoice details (Lab Info, Customer Info, Invoice No., Itemized List, Summary).
        - PrintButton (Shadcn UI Button): Triggers browser print function.
        - AttachPaySlipButton (Shadcn UI Button): Opens file upload modal for payment slip.
        - FileUpload component (Custom): For attaching payment slips.
    - **State Management:**
        - React Query (useQuery): Fetch single Invoice data.
        - React Query (useMutation): For attaching payment slip.
    - **Interactions:**
        - Display invoice data.
        - Handle payment slip upload and show success/error feedback.

B. Routing (React Router)

- path: '/customer/requests' (Protected, customer role)
  - loader to fetch TestRequest list.
- path: '/customer/requests/add' (Protected, customer role)
  - action to handle new TestRequest creation.
- path: '/customer/requests/:requestId' (Protected, customer role)
  - loader to fetch single TestRequest details.
- path: '/customer/requests/:requestId/edit' (Protected, customer role)
  - loader to fetch single TestRequest details for editing.
  - action to handle TestRequest updates.
- path: '/customer/invoices' (Protected, customer role)
  - loader to fetch Invoice list.
- path: '/customer/invoices/:invoiceId' (Protected, customer role)
  - loader to fetch single Invoice details.
  - action to handle payment slip upload.

C. Styling

- **Tailwind CSS:** Consistent application for all layouts, forms, tables, and buttons.
- **Shadcn UI:** Use components for a polished and accessible look. Ensure consistent spacing and typography.
- **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):**
  - TestRequest: Required fields (e.g., objective, requesterName).
  - TestRequestSample: Required fields (e.g., customerSampleId, panel, requestedQty), date formats, quantity validation (positive numbers).
  - File upload validation (max size, allowed types for payment slips/attachments).
- **Server-side (Zod in Express):** Re-validate all incoming data for security and data integrity. This is paramount.

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
- invoices
- invoice_line_items
- document_attachments
- audit_trail

B. API Endpoints

1. **GET /api/v1/customer-requests (List Customer Requests)**
   - **Purpose:** Retrieve a list of test requests for the authenticated customer.
   - **Request:** Optional query parameters for pagination (page, limit), search (searchQuery), and filtering (documentStatus, requestDateFrom, requestDateTo).
   - **Response (200 OK):** { data: TestRequest[], total: number, page: number, limit: number } (TestRequest objects should include samples and other relevant fields).
   - **Errors (403 Forbidden):** If user is not a customer.
   - **Middleware:** authMiddleware, roleMiddleware('customer'), Input validation for query params.
   - **Logic:**
     - Extract userId from JWT, fetch associated customerId.
     - Query test_requests table filtered by customerId, applying pagination, search, and status/date filters.
     - Include related test_request_samples.
     - Log action to audit_trail (e.g., 'customer_viewed_requests').
2. **GET /api/v1/customer-requests/:id (Get Single Customer Request)**
   - **Purpose:** Retrieve details of a specific test request.
   - **Request:** URL parameter :id (UUID of the request).
   - **Response (200 OK):** TestRequest object with nested TestRequestSample array.
   - **Errors (404 Not Found):** If request doesn't exist or doesn't belong to the customer.
   - **Middleware:** authMiddleware, roleMiddleware('customer').
   - **Logic:**
     - Extract userId from JWT, fetch associated customerId.
     - Fetch TestRequest by :id, ensuring customer_id matches the authenticated customer's ID.
     - Include related test_request_samples.
     - Log action to audit_trail (e.g., 'customer_viewed_request_details').

3. **POST /api/v1/customer-requests (Create Test Request - Draft/Submit)**
   - **Purpose:** Create a new test request (as a draft or submitted).
   - **Request:** application/json
     - Body: objective, requesterName, samples: TestRequestSample[], status: 'draft' | 'submitted'.
   - **Response (201 Created):** { message: 'Request created successfully.', requestId: '...' }
   - **Errors (400 Bad Request):** Validation errors.
   - **Middleware:** authMiddleware, roleMiddleware('customer'), Input validation (Zod schema for request and samples).
   - **Logic:**
     - Extract userId from JWT, fetch associated customerId.
     - Validate input.
     - If status is submitted, auto-generate request_no based on company/sender and date.
     - Create TestRequest entry.
     - Create associated TestRequestSample entries.
     - Log action to audit_trail (e.g., 'customer_created_request_draft', 'customer_submitted_request').
4. **PUT /api/v1/customer-requests/:id (Update Test Request - Draft/Submit)**
   - **Purpose:** Update an existing test request.
   - **Request:** application/json
     - Body: objective, requesterName, samples: TestRequestSample[], status: 'draft' | 'submitted'.
   - **Response (200 OK):** { message: 'Request updated successfully.' }
   - **Errors (400 Bad Request):** Validation errors, trying to update a request that's not Draft or Submitted.
   - **Middleware:** authMiddleware, roleMiddleware('customer'), Input validation.
   - **Logic:**
     - Extract userId from JWT, fetch associated customerId.
     - Fetch existing TestRequest by :id, ensure customer_id matches and document_status is Draft or Submitted.
     - Validate input.
     - Update TestRequest fields.
     - Handle TestRequestSample updates (add new, modify existing, delete removed ones). This might require transactional logic.
     - If status changes from Draft to Submitted, auto-generate request_no if not already present.
     - Log action to audit_trail (e.g., 'customer_updated_request_draft',

        'customer_submitted_request').

5. **DELETE /api/v1/customer-requests/:id (Delete Test Request)**
    - **Purpose:** Delete a test request.
    - **Request:** None (uses URL parameter :id).
    - **Response (204 No Content):** (Successful deletion)
    - **Errors (400 Bad Request):** Trying to delete a request that's not Draft or Submitted.
    - **Middleware:** authMiddleware, roleMiddleware('customer').
    - **Logic:**
        - Extract userId from JWT, fetch associated customerId.
        - Fetch existing TestRequest by :id, ensure customer_id matches and document_status is Draft or Submitted.
        - Delete TestRequest (Prisma's cascading deletes will remove associated samples).
        - Log action to audit_trail (e.g., 'customer_deleted_request').
6. **GET /api/v1/invoices (List Customer Invoices)**
    - **Purpose:** Retrieve a list of invoices for the authenticated customer.
    - **Request:** Optional query parameters for pagination, search.
    - **Response (200 OK):** { data: Invoice[], total: number, page: number, limit: number } (Invoice objects, excluding labTaxInfo if not customer-facing).
    - **Errors (403 Forbidden):** If user is not a customer.
    - **Middleware:** authMiddleware, roleMiddleware('customer').
    - **Logic:**
        - Extract userId from JWT, fetch associated customerId.
        - Query invoices table filtered by customer_id.
        - Log action to audit_trail (e.g., 'customer_viewed_invoices').
7. **GET /api/v1/invoices/:id (Get Single Invoice Details)**
    - **Purpose:** Retrieve details of a specific invoice.
    - **Request:** URL parameter :id (UUID of the invoice).
    - **Response (200 OK):** Invoice object with nested InvoiceLineItem array.
    - **Errors (404 Not Found):** If invoice doesn't exist or doesn't belong to the customer.
    - **Middleware:** authMiddleware, roleMiddleware('customer').
    - **Logic:**
        - Extract userId from JWT, fetch associated customerId.
        - Fetch Invoice by :id, ensuring customer_id matches.
        - Include related invoice_line_items.
        - Log action to audit_trail (e.g., 'customer_viewed_invoice_details').
8. **POST /api/v1/invoices/:id/payment-slip (Attach Payment Slip)**

    - **Purpose:** Allow customer to upload proof of payment for an invoice.
    - **Request:** multipart/form-data
        - URL parameter :id (UUID of the invoice).
        - File: paymentSlipFile.
    - **Response (200 OK):** { message: 'Payment slip attached successfully.', fileUrl: '...' }
    - **Errors (400 Bad Request):** Invalid file type/size, invoice not found.
    - **Middleware:** authMiddleware, roleMiddleware('customer'), multer middleware for file upload.
    - **Logic:**
        - Extract userId from JWT, fetch associated customerId.
        - Verify invoice :id exists and belongs to the customer.
        - Upload paymentSlipFile to Google Cloud Storage.
        - Update invoices.payment_slip_attachment_url with the file URL.
        - Create DocumentAttachment entry linked to the invoice.
        - Log action to audit_trail (e.g., 'customer_uploaded_payment_slip').

C. Prisma Operations

- PrismaClient.testRequest.findMany(), findUnique(), create(), update(), delete() (with include for samples).
- PrismaClient.testRequestSample.createMany(), update(), deleteMany().
- PrismaClient.invoice.findMany(), findUnique(), update() (with include for line items).
- PrismaClient.$transaction(): For complex operations like creating/updating TestRequest with nested samples to ensure atomicity.
- PrismaClient.auditTrail.create().
- PrismaClient.documentAttachment.create().

D. Middleware Requirements

- authMiddleware: Verifies JWT and populates req.user.
- roleMiddleware('customer'): Ensures req.user.role is 'customer'.
- validateMiddleware(schema): For validating request bodies, query parameters, and route parameters using Zod schemas.
- multer middleware: For handling file uploads (e.g., multer().single('paymentSlipFile')).

E. Business Logic / Service Functions

- TestRequestService:
    - createTestRequest(data, customerId): Handles request number generation, saving draft/submission.

    - getTestRequestsByCustomer(customerId, filters): Retrieves paginated and filtered requests.
    - getTestRequestById(id, customerId): Retrieves a single request.
    - updateTestRequest(id, data, customerId): Handles updates, including nested samples.
    - deleteTestRequest(id, customerId): Deletes a request.
- InvoiceService:
    - getInvoicesByCustomer(customerId, filters): Retrieves paginated invoices.
    - getInvoiceById(id, customerId): Retrieves single invoice.
    - attachPaymentSlip(invoiceId, customerId, file): Handles file upload and URL update.
- RequestNumberGeneratorService: Utility to generate unique request numbers.
- FileService: Manages interactions with Google Cloud Storage for file uploads.
- AuditService: For logging all customer actions.

F. Error Handling

- Custom error classes (e.g., NotFoundError, ForbiddenError, ValidationError, OperationNotAllowedError for status-based actions).
- Centralized Express error handling middleware to catch these errors and return appropriate HTTP status codes (e.g., 400, 401, 403, 404, 500).

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

Define interfaces for:

- TestRequest (and its statuses: 'submitted' | 'acknowledged_sample_received' | 'paid' | 'approved' | 'rejected' | 'draft').
- TestRequestSample (and its individual sample details).
- Invoice (and InvoiceLineItem).
- AttachmentInfo (for payment slips).
- Payloads for CreateTestRequestPayload, UpdateTestRequestPayload, AttachPaymentSlipPayload.
- Filter/pagination types (TestRequestFilters, InvoiceFilters, PaginatedResponse<T>).

6. Edge Cases & Considerations

- **Request Number Generation:** Ensure the auto-generation logic for request numbers is robust and handles concurrency (e.g., using database sequences or optimistic locking).
- **Print Functionality:** For "Print Request Summary" and "Print Invoice", consider:
    - Generating a clean, CSS-optimized view for printing.

    - Potentially using a PDF generation library on the backend for more complex, consistently formatted PDFs (e.g., pdfkit or puppeteer to render HTML to PDF) if frontend window.print() is insufficient.
- **Status Transitions:** Ensure that "Edit" and "Delete" actions are strictly controlled by the document_status. For example, once a request is Acknowledged and Received Sample, it should no longer be editable or deletable by the customer.
- **File Upload Security:** Implement comprehensive file validation (type, size), virus scanning (if critical), and secure storage with proper access controls in Google Cloud Storage.
- **Payment Status:** The invoice payment_status will be updated by the lab side (once they verify the payment slip). Customers can only upload the slip.
- **Notifications:** Consider sending email notifications to the customer upon:
    - Request Submitted
    - Request Acknowledged and Received Sample
    - Lab results Approved (with a link to view the results)
    - Invoice Generated
    - Payment Confirmed
- **Data Consistency:** When updating a TestRequest with new/modified/deleted samples, ensure the backend logic correctly handles these nested changes, potentially using database transactions to prevent partial updates.