Customer Registration & Profile Module Design Specification

This document outlines the design and implementation details for the Customer Registration and Profile Management module. It serves as a guide for junior developers.

1. Module Purpose

The primary purpose of this module is to allow new customers (companies/individuals) to register for an account, provide their essential business and contact information, and manage their profile details after registration. It ensures secure account creation with email verification and captures necessary data for future interactions (e.g., test requests, invoicing).

2. Key Features

- **Customer Registration:**
  - Capture company name, ID card/tax ID, primary address, and shipping address.
  - Allow upload of relevant registration documents (e.g., business license).
  - Secure password creation.
  - Email verification for account activation.
- **Customer Login/Logout:**
  - Secure authentication using email/password.
  - Session management via JWT.
- **Customer Profile Management:**
  - View and edit registered profile information.
  - Manage attached registration files.
  - Change password.

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

A. Pages & Components

1. **/auth/register-customer (Customer Registration Page):**
   - **Layout:** A simple, centered form within an AuthLayout.
   - **Components:**
     - RegistrationForm component:
       - Input fields (Shadcn UI Input): Company Name, Email, Password, Confirm Password, Tax ID/ID Card.
       - Address fields (Shadcn UI Input): Address Line 1, Address Line 2, City,

State/Province, Zip Code, Country (dropdown/select if constrained).
        - Shipping Address fields (can be same as primary or separate inputs).
        - File upload area (Custom FileUpload component using Shadcn UI Button and handling input type="file" internally).
        - Submit button (Shadcn UI Button).
    - **State Management:**
        - React Hook Form: Manage form state, validation, and submission.
        - Zod: Define schema for client-side validation.
        - React Query (useMutation): Handle the submission of registration data to the backend.
    - **Interactions:**
        - Show loading state on button during submission.
        - Display validation errors next to fields.
        - Display success message or redirect to a "Verify Email" page upon successful registration.
        - Display error messages (e.g., "Email already registered").
2. **/auth/login (Login Page):**
    - **Layout:** A simple, centered form within an AuthLayout.
    - **Components:**
        - LoginForm component: Email input, Password input, Login button (Shadcn UI Input, Button).
        - Links for "Forgot Password?" and "Register".
    - **State Management:**
        - React Hook Form & Zod: For form handling and validation.
        - React Query (useMutation): To send login credentials to the backend.
    - **Interactions:**
        - Show loading state.
        - Redirect to customer dashboard (/customer/dashboard) on successful login.
        - Display error messages (e.g., "Invalid credentials").
3. **/customer/profile (Customer Profile Page):**
    - **Layout:** Main application layout with navigation (CustomerLayout).
    - **Components:**
        - ProfileDisplay component: Show current customer details (non-editable initially).
        - ProfileEditForm component: (Optional, can be a modal or separate section) Form with pre-filled customer data for editing.
        - FileUpload component: For managing registration_attachments.
        - ChangePasswordForm component: Separate form for password updates.

- **State Management:**
  - React Query (useQuery): Fetch current customer profile data.
  - React Query (useMutation): For updating profile information or changing password.
  - React Hook Form & Zod: For edit forms.
- **Interactions:**
  - Populate form fields with existing data.
  - Handle updates and show success/error feedback.
  - Display file list for registration_attachments with options to view/delete/upload new.

B. Routing (React Router)

- path: '/auth/register-customer'
- path: '/auth/login'
- path: '/customer/profile' (Protected route, requires customer role)

C. Styling

- **Tailwind CSS:** Apply classes for responsive design, spacing, typography, and component styling.
- **Shadcn UI:** Leverage its components for a consistent and accessible UI.
- **Responsive Design:** Ensure forms and layouts adapt well to mobile, tablet, and desktop screens using Tailwind's responsive prefixes (sm:, md:, lg:).

D. Validation

- **Client-side (Zod + React Hook Form):**
  - Email format, password strength (min length, special chars).
  - Required fields for company name, addresses.
  - File size and type limits for attachments.
  - Password and confirm password match.
- **Server-side (Zod/Joi in Express):** Re-validate all incoming data to ensure 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 table: Stores email, password_hash, role, is_email_confirmed.
- customers table: Stores company_name, tax_id_or_id_card, address_line1, etc., and registration_attachments.
- document_attachments table: Stores file URLs for attachments.

- audit_trail table: For logging user registration and profile updates.

B. API Endpoints

1. **POST /api/v1/auth/register (Customer Registration)**
    - **Purpose:** Register a new customer account.
    - **Request:** multipart/form-data (for files) or application/json (for data, then a separate file upload).
        - Body: email, password, companyName, taxIdOrIdCard, addressLine1, city, zipCode, country, shippingAddressLine1, etc., registrationAttachments (array of files).
    - **Response (201 Created):** { message: 'Registration successful. Please confirm your email.', userId: '...' }
    - **Errors (400 Bad Request):** If email already exists, validation errors.
    - **Middleware:** Input validation.
    - **Logic:**
        - Validate input.
        - Hash password (bcrypt).
        - Create User entry with role: 'customer', is_email_confirmed: false.
        - Create Customer entry linked to the User.
        - Upload files to Google Cloud Storage; store URLs in document_attachments and link to customer.id (or store JSONB array of URLs directly in customers.registration_attachments).
        - Generate email verification token and send confirmation email.
        - Log action to audit_trail.
2. **GET /api/v1/auth/confirm-email/:token (Email Confirmation)**
    - **Purpose:** Confirm user's email address.
    - **Request:** URL parameter :token.
    - **Response (200 OK):** { message: 'Email confirmed successfully.' }
    - **Errors (400 Bad Request):** Invalid/expired token.
    - **Middleware:** None.
    - **Logic:**
        - Verify the token.
        - Find user associated with the token.
        - Update users.is_email_confirmed to true.
        - Invalidate/delete the token.
3. **POST /api/v1/auth/login (Login)**
    - **Purpose:** Authenticate user and issue JWT.
    - **Request:** application/json
        - Body: email, password.

- **Response (200 OK):** { token: '...', user: { id: '...', email: '...', role: '...' } }
- **Errors (401 Unauthorized):** Invalid credentials, unconfirmed email.
- **Middleware:** Input validation.
- **Logic:**
  - Validate input.
  - Find user by email.
  - Compare password hash (bcrypt.compare).
  - Check is_email_confirmed.
  - Generate JWT with userId and role.
  - Return JWT.

4. **GET /api/v1/customers/profile (Get Customer Profile)**
   - **Purpose:** Retrieve the logged-in customer's profile.
   - **Request:** None (uses authenticated user's ID).
   - **Response (200 OK):** Customer object (excluding sensitive data like password hash).
   - **Errors (404 Not Found):** If profile doesn't exist for authenticated user.
   - **Middleware:** authMiddleware (verify JWT), roleMiddleware (ensure customer role).
   - **Logic:**
     - Extract userId from JWT.
     - Fetch Customer record linked to userId.
5. **PUT /api/v1/customers/profile (Update Customer Profile)**
   - **Purpose:** Update the logged-in customer's profile information.
   - **Request:** application/json
     - Body: Updatable fields like companyName, addressLine1, shippingAddressLine1, etc.
   - **Response (200 OK):** Updated Customer object.
   - **Errors (400 Bad Request):** Validation errors.
   - **Middleware:** authMiddleware, roleMiddleware (ensure customer role), Input validation.
   - **Logic:**
     - Extract userId from JWT.
     - Validate input.
     - Update Customer record.
     - Log action to audit_trail.
6. **PATCH /api/v1/customers/change-password (Change Password)**
   - **Purpose:** Allow logged-in customers to change their password.
   - **Request:** application/json
     - Body: currentPassword, newPassword, confirmNewPassword.

    - **Response (200 OK):** { message: 'Password updated successfully.' }
    - **Errors (400 Bad Request):** Invalid current password, new password validation errors.
    - **Middleware:** authMiddleware, roleMiddleware (ensure customer role), Input validation.
    - **Logic:**
        - Extract userId from JWT.
        - Verify currentPassword against stored hash.
        - Hash newPassword.
        - Update users.password_hash.
        - Log action to audit_trail.

C. Prisma Operations

- PrismaClient.user.create()
- PrismaClient.customer.create()
- PrismaClient.user.findUnique()
- PrismaClient.customer.findUnique()
- PrismaClient.user.update()
- PrismaClient.customer.update()
- PrismaClient.auditTrail.create()
- PrismaClient.documentAttachment.create()

D. Middleware Requirements

- authMiddleware: Verifies JWT and attaches req.user (containing id, role).
- roleMiddleware('customer'): Checks if req.user.role is 'customer'.
- validateMiddleware(schema): Validates request body/params/query using a Zod schema.

E. Business Logic / Service Functions

- UserService: Contains methods for user creation, password hashing/comparison, finding users.
- CustomerService: Methods for customer registration, profile retrieval, and updates.
- AuthService: Handles login logic, JWT generation, and email confirmation.
- FileService: For uploading/managing files in Google Cloud Storage.
- AuditService: For logging actions.

F. Error Handling

- Custom error classes (e.g., AuthenticationError, ValidationError, NotFoundError).
- Centralized error handling middleware in Express to catch these errors and send

appropriate HTTP status codes and messages.

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

Define interfaces for:

- User: id, email, role, isEmailConfirmed.
- Customer: id, userId, companyName, address, shippingAddress, taxIdOrIdCard, registrationAttachments (array of AttachmentInfo objects).
- AttachmentInfo: id, fileName, fileUrl, mimeType.
- RegisterCustomerPayload: Input types for registration form.
- LoginPayload: Input types for login form.
- UpdateCustomerProfilePayload: Input types for profile update.
- ChangePasswordPayload: Input types for password change.

6. Edge Cases & Considerations

- **Email Sending:** Integrate with an email service (e.g., SendGrid, Mailgun) for sending confirmation emails.
- **File Storage:** Ensure Google Cloud Storage bucket permissions are correctly configured for uploads and retrieval. Implement proper naming conventions for uploaded files to avoid collisions.
- **Security:**
  - Always use HttpOnly cookies for JWT storage if possible (though React Query might make localStorage more common for single-page apps, be aware of XSS risks).
  - Rate limit registration and login endpoints to prevent brute-force attacks.
  - Strict input validation on both client and server.
  - Ensure sensitive data (passwords, tax IDs) are never exposed in logs or frontend responses.
- **User Experience:** Provide clear loading indicators, success messages, and error messages to guide the user.
- **Forgotten Password:** Plan for a separate "Forgot Password" flow (sending reset link to email) in the authentication module.
- **Internationalization:** Consider address formats, country lists, and potential tax ID variations if supporting multiple regions.