WYDZIAŁ MATEMATYKI
i INFORMATYKI
Uniwersytet Łódzki

Mussa Justin Muhindo

Student number: 411991

Field of study: Computer Science

Development of the Legal Connect System with React JS and Node.js

Master's thesis
written in the Department of Computer Science
Supervisor: Dr Adam Bartoszek

Łódź 2024

Abstract

A React JS and Node.js based Legal Connect System for seamless lawyer-client interactions. Enables case management, consultation scheduling, and document handling online. Designed for accessibility and ease of use, it transforms how legal services are delivered.

Abstrakt

System Legal Connect oparty na React JS i Node.js umożliwia bezproblemową komunikację między prawnikami a ich klientami. Umożliwia zarządzanie sprawami, planowanie konsultacji oraz obsługę dokumentów online. Zaprojektowany z myślą o dostępności i łatwości obsługi, zmienia sposób świadczenia usług prawnych.

List of Abbreviations

LCS	: Legal Connect System
JS	: JavaScript
SQL	: Structured Query Language
UI	: User Interface
ERD	: Entity-Relationship Diagram
ORM	: Object-Relational Mapping
API	: Application Programming Interface
JWT	: JSON Web Token
DOM	: Document Object Model
HTTP	: Hypertext Transfer Protocol
REST	: Representational State Transfer
DP_PROJECT_MMJ	: Directory Project for Muhindo Mussa Justin
VS: Visual Studio ID	: Identification
PDF	: Portable Document Format
NR1	: Appendix Number 1
INT	: Integer
VARCHAR	: Variable Character String
TEXT	: Text Block
DATETIME	: Date-Time Stamp
Npm	: Node Package Manager
TRC	: Temporally Residence Card

Spis treści

Introduction

In the heart of Europe, Poland stands as a pivotal destination for international students attracted by its educational offerings, which combine quality with affordability. Despite this attraction, foreign students often face legal challenges, seek contact with lawyers, or navigate an unfamiliar legal system without satisfactory support. This thesis introduces the "Legal Connect System" as an innovative platform designed to bridge the significant gap between those in need of legal advice, including foreign students and residents, and the legal professionals capable of providing such support. This system aims to make legal services more accessible, overcoming language barriers, limited knowledge of Polish legal proceedings, and other obstacles to obtaining timely legal support. The following sections will delve into the critical need for this platform, the approach taken to develop it, and its anticipated impact on making legal assistance more inclusive and accessible in Poland.

Application data

Internet address where the application is available: https://legalc.net/

Information (identifiers, passwords, etc.) needed to run all parts of the application:

Users

Client accounts

1.	muhissakings@gmail.com	Password: Kigali@2023
2.	client.one@hotmail.com	Password: Lodz@2024
3.	client.two@gmail.com	Password: Poznan@2024
4.	client03@gmail.com	Password: Opole@2024
5.	joedoe@gmail.com	Password: Kigali@2023

Lawyers account

1.	jmussamuhindo@gmail.com	Password: Kigali@2023
2.	lawyer.one@gmail.com	Password: Lodz@2024
3.	lawyer.two@yahoo.fr	Password: Poznan@2024
4.	jeansouvenirumutoni@gmail.com	Password: Kigali@2023

Admin:

superadmin@me.com	Password: test@123

1 Programming tools

This chapter gives a broke down overview of the programming tools and technologies used in the development of the Legal Connect System. This system is designed to simplify the legal processes by providing an efficient and user-friendly platform for connecting legal professionals with clients. The selection of technologies was motivated by the requirement for a sturdy, scalable, and flexible solution that could meet the diverse requirements of the legal industry.

1.1 React (Frontend Development)

React is an expressive, efficient, and flexible JavaScript library for building user interfaces. It was chosen for the development of the Legal Connect System's frontend due to its component-based architecture, which enables reusable UI components. This approach significantly reduces development time and enhances the user experience by ensuring a seamless and interactive interface. React's virtual DOM implementation offers efficient updates and rendering of the UI, which is crucial for achieving optimal performance in real-time applications.

1.2 Node.js (Backend Development)

Node.js is important for running JavaScript on servers, not just in browsers. It was key to designing the backend of the Legal Connect System. Node.js handles many tasks at once efficiently, which helps the system perform well even under heavy use. Its large set of tools and libraries available through npm made it easier to add many features quickly.

1.3 Express (Web Application Framework)

Express was chosen as the web framework for the Legal Connect System because it's fast and works well. It was very important for building the system's server-side setup and creating RESTful APIs. Express is easy to use and flexible, which helped quickly set up essential parts of the system. This choice improved the backend's strength and efficiency, making it easier to manage complex tasks and routing. Express proved to be the perfect framework for our development needs.

2 Source code file structure.

2.1 System Architecture

The Legal Connect System, a full-stack web application, is intricately designed with a bifurcation into backend and frontend components, each encapsulating specific functionalities aligned with the system's requirements. This chapter delineates the key directories and files that constitute the core of the application, providing insight into the system's structure and modular composition.

Due to constraints in the number of pages allotted for this thesis, it is not feasible to detail every file and function within the project's codebase. However, emphasis has been placed on discussing the most critical files and functionalities that are central to understanding the architecture and operation of the application. This selection criterion ensures that the most relevant and impactful aspects of the system are comprehensively explored, providing a meaningful insight into the application's design and implementation while adhering to the prescribed length limitations.

2.2 Backend architecture

This is the root directory of the project, containing all the source code, configuration files, and documentation.

2.3 DP_PROJECT_MMJ

2.3.1 Directory: „DP-M_MMJ_Bc"

Directory contains the backend components of the project, including server configurations and application source code.

2.3.1.1 Directory: „src"

Directory holds the source code for the project's backend logic, including models, routes, and controllers.

2.3.1.1.1 Directory: „app"

The app directory contains the core application logic, organizing the request handling controllers, middleware functions for request processing, routing definitions, and data validation rules.

2.3.1.1.1.1 Directory: „controllers"

The scripthandlesin the controller's directory and handles the application's business logic by processing incoming requests and determining the appropriate outgoing responses.

2.3.1.1.1.1.1 Directory: „app"

The scripts in the app directory's controller's subdirectory are responsible for handling specific business logic and HTTP requests related to various entities such as appointments, cases, chats, and user profiles.

2.3.1.1.1.1.1.1 File: „app.controller.ts"

Contains logic for handling incoming HTTP requests and sending responses.

2.3.1.1.1.1.1.1.1 Function: „getApp"

Sends a welcome message to client when the root endpoint is accessed.

2.3.1.1.1.1.1.1.2 Function: „getFaqs"

Retrieves all FAQ questions using the FAQ question using the FAQ service and sends them to the client. Including error handling if the retrieval fails.

2.3.1.1.1.1.1.2 File: „app.skills.match.controller.ts"

The script in this file defines a function matchMySkills that processes user input to match skills with lawyers' practice areas and returns relevant lawyer information, handling both successful matches and various error scenarios.

2.3.1.1.1.1.1.2.1 Function: „matchMySkills"

The matchMySkills function processes user-provided skills to find and return a list of lawyers whose practice areas match those skills, along with error handling for cases where no matches are found, or an exception occurs.

2.3.1.1.1.1.1.3 File: " appointment.controller.ts"

The scripts in this file define functions for booking appointments, retrieving client and lawyer appointments, and handling appointment acceptance, all with corresponding validation and error handling.

2.3.1.1.1.1.1.3.1 Function: „bookAppointment"

Validates the requesting client and lawyer, then attempts to create and save a new appointment.

2.3.1.1.1.1.1.3.2 Function: „getAppointmentsAsClien"

Retrieves and formats a list of appointments for a client after validating their account.

2.3.1.1.1.1.1.3.3 Function: „getAppointmentAsLawyer"

Fetches and structures a list of appointments for a lawyer following account verification.

2.3.1.1.1.1.1.3.4 Function: „acceptAppointment"

Checks the existence of an appointment and updates its status to accept.

2.3.1.1.1.1.1.4 File: „case.controller.ts"

The scripts in this file manage the creation and retrieval of legal cases, handling requests to create new cases, fetch cases associated with clients or lawyers, and filter cases by status, ensuring all operations are validated and properly formatted before sending responses.

2.3.1.1.1.1.1.4.1 Function: „newCase"

Creates a new legal case after validating account, client, law practice, lawyer, and category details.

2.3.1.1.1.1.1.4.2 Function: „myCases”
Retrieves all cases associated with a specific client based on their account ID.

2.3.1.1.1.1.1.4.3 Function: „lawyerCases”
Obtains all cases assigned to a specific lawyer through the lawyer's user ID.

2.3.1.1.1.1.1.4.4 Function: „caseById”
Retrieves detailed information for a specific case identified by its case ID.

2.3.1.1.1.1.1.4.5 Function: „getCasesByLawyer”
Fetches all cases managed by a specific lawyer, identified through the request parameters.

2.3.1.1.1.1.1.4.6 Function: „activeAllCases”
Gathers all active cases within the system, regardless of the client or lawyer.

2.3.1.1.1.1.1.4.7 Function: „pendingAllCases”
Retrieves all cases that are currently pending, offering a view of unresolved legal matters.

2.3.1.1.1.1.1.5 File: „caseaction.controller.ts”
The scripts in this file are designed to manage and record actions within a case tracking system, including updating case statuses, and retrieving a list of all actions associated with a specific case.

2.3.1.1.1.1.1.5.1 Function: „trackerAction”
Updates the status of a specific case and records a new action associated with that case after validating the account and lawyer details.

2.3.1.1.1.1.1.5.2 Function: „getAllCaseActions”
Retrieves all actions recorded for a specific case by its case ID and presents them after validating the existence of the case.

2.3.1.1.1.1.1.6 File: „case.controller.ts”
The scripts in this file facilitate the retrieval of messages exchanged between clients and lawyers, ensuring users can access their relevant communications by validating user and account details before fetching and transforming message data for response.

2.3.1.1.1.1.1.6.1 Function: „getMyClientMessages”
Retrieves messages between a specific client and their lawyer, validating both client and lawyer accounts before returning the messages.

2.3.1.1.1.1.1.6.2 Function: „getMyLawyerMessages”
Fetches messages exchanged between a lawyer and their client, ensuring both parties' accounts are valid before providing the message data.

2.3.1.1.1.1.1.7 File: „caseaction.controller.ts”

2.3.1.1.1.1.1.7.1 Function „trackerAction”
trackerAction adds a new action to a case's tracker based on user inputs and updates the case status accordingly.

2.3.1.1.1.1.1.7.2 Function: „getAllCaseActions”

getAllCaseActions retrieves all actions associated with a specific case ID and returns them in a formatted response.

2.3.1.1.1.1.1.8 File: „chat.controller.ts”

2.3.1.1.1.1.1.8.1 Function: „getMyClientMessage”

getMyClientMessages retrieves the conversation history between a specific client and their lawyer, formatting the output before sending it back to the client.

2.3.1.1.1.1.1.8.2 Function: „getMyLawyerMessages”

getMyLawyerMessages fetches the message exchange between a specific lawyer and their client, also providing a formatted response for the client.

2.3.1.1.1.1.1.9 File: „city.controller.ts”

city.controller.ts is responsible for retrieving all cities from the database and sending them back to the client, with an option to format the data which is currently commented out.

2.3.1.1.1.1.1.9.1 Function: „getAllCities”

The getAllCities function retrieves the complete list of cities from the database and serves them to the client, potentially with formatting, facilitating location-based queries within the application's user interface.

2.3.1.1.1.1.1.10 File: „connection.controller.ts”

The scripts in this file manage connections between clients and lawyers, enabling the creation of new connections, listing existing connections for both lawyers and clients, and approving connection requests, all while verifying the validity of the requester's account.

2.3.1.1.1.1.1.10.1 Function: „newConnect”

Initiates a new connection between a client and a lawyer after verifying their validity and ensuring no existing connection.

2.3.1.1.1.1.1.10.2 Function: „lawyerConnections”

Lists all connections associated with a lawyer, transforming, and returning the data.

2.3.1.1.1.1.1.10.3 Function: „clientConnections”

Retrieves all connections for a client, transforming and returning the connection data.

2.3.1.1.1.1.1.10.4 Function: „approveConnection”

Approves a connection request by updating its status to 'accepted'.

2.3.1.1.1.1.1.11 File: „ law.controller.ts”

The scripts in the file handle web requests to fetch and respond with lists of law practices and case categories, transforming the data for the client.

2.3.1.1.1.1.1.11.1 Function: „getAllPractices"
Fetches a list of all law practices, transforms the data, and sends it back in the response.

2.3.1.1.1.1.1.11.2 Function: „categories"
Retrieves a list of all case categories, transforms the data, and sends it back in the response.

2.3.1.1.1.1.1.12 File: „lawyer.controller.ts"
Manages operations related to lawyer information processing.

2.3.1.1.1.1.1.12.1 Function: „getLawyerInPractice"
fetches and returns a list of lawyers active in a specified practice area.

2.3.1.1.1.1.1.12.2 Function: „myStatus"
checks and returns the active status of a lawyer associated with the user making the request.

2.3.1.1.1.1.1.13 File: „payment.controller.ts"
The scripts in the file manage payment requests within a legal services platform, including creating, updating, and fetching payment requests for cases, as well as handling transactions for clients and lawyers.

2.3.1.1.1.1.1.13.1 Function: „requestPayment"
Creates a new payment request for a case after validating the case exists.

2.3.1.1.1.1.1.13.2 Function: „getClientPaymentRequest"
Retrieves and transforms all payment requests made by a client, validating the client's account.

2.3.1.1.1.1.1.13.3 Function: „lawyerPaymentRequests"
Fetches and transforms all payment requests associated with a lawyer, verifying the lawyer's account.

2.3.1.1.1.1.1.13.4 Function: „allPayments"
Retrieves and transforms all payment transactions within the system.

2.3.1.1.1.1.1.13.5 Function: „paymentTransactionById"
Fetches and transforms a specific payment transaction by its ID, ensuring the transaction exists.

2.3.1.1.1.1.1.13.6 Function: „payTransaction"
Marks a specific payment transaction as paid and updates the associated case status to active, verifying the transaction's existence.

2.3.1.1.1.1.1.14 File: „payment.controller.ts"
The scripts in the file manage user profiles and client statuses on a platform, including fetching, updating user profiles, and checking if a client account is active.

2.3.1.1.1.1.1.14.1 Function: „getMyProfile"
Retrieves the profile information for the currently logged-in user based on their user ID.

2.3.1.1.1.1.1.14.2 Function: „getProfileById"
Fetches the profile information for a user specified by their user ID in the request parameters.

2.3.1.1.1.1.1.14.3 Function: „updateProfile”
Updates the profile information for the currently logged-in user based on the provided request body data.

2.3.1.1.1.1.1.14.4 Function: „clientStatus”
Checks and returns the active status of a client associated with the currently logged-in user's account ID.

2.3.1.1.1.1.1.15 File: „review.controller.ts”
The scripts in this file manage review processes for a legal services platform, including initializing review requests, fetching reviews for lawyers and clients, editing reviews, and changing review visibility statuses.

2.3.1.1.1.1.1.15.1 Function: „initReviewRequest”
Initializes a review request for a case after validating the account and case exist.

2.3.1.1.1.1.1.15.2 Function: „lawyerReviews”
Fetches and transforms reviews submitted for a specific lawyer, verifying the lawyer's account.

2.3.1.1.1.1.1.15.3 Function: „clientReviews”
Retrieves and transforms reviews made by a specific client, ensuring the client's account is valid.

2.3.1.1.1.1.1.15.4 Function: „editReview”
Allows editing of a specific review by updating its title and text, after confirming the review exists.

2.3.1.1.1.1.1.15.5 Function: „getReviewByLawyerId”
Fetches and transforms all reviews associated with a given lawyer by their ID, verifying the lawyer's existence.

2.3.1.1.1.1.1.15.6 Function: „changeReviewStatus”
Changes the visibility status of a specific review from public to private or vice versa, after ensuring the review exists.

2.3.1.1.1.1.1.16 File: „schedules.controller.ts”
availability this file facilitates the management of event schedules for lawyers, including adding new schedule availabilities, retrieving all event schedules, fetching event availabilities for a specific lawyer, and getting available schedules for a lawyer by their ID.

2.3.1.1.1.1.1.16.1 Function: „addScheduleAvailability”
Registers a new schedule availability for a lawyer after validating the lawyer's account.

2.3.1.1.1.1.1.16.2 Function: „allEvents”
Fetches and transforms all scheduled events across the platform.

2.3.1.1.1.1.1.16.3 Function: „myEventAvailabilities”
Retrieves and transforms all schedule availabilities for the logged-in lawyer, ensuring the lawyer's account is valid.

2.3.1.1.1.1.1.16.4 Function:

„getLawyerAvailableSchedules”
Fetches and transforms schedule availabilities for a specific lawyer by their ID, confirming the lawyer exists.

2.3.1.1.1.1.2 Directory: „auth”
The auth directory likely contains scripts for handling authentication processes, including user login, registration, and role management.

2.3.1.1.1.1.2.1 File: „auth.login.controller.ts”
The scripts in this file manage user authentication processes including logging in, logging out, generating a new password, and changing a user's password with security measures such as bcrypt for password hashing.

2.3.1.1.1.1.2.1.1 Function: „login”
Authenticates a user by verifying their email and password, generates a token, and possibly creates a new profile if it doesn't exist.

2.3.1.1.1.1.2.1.2 Function: „logout”
Removes a user's authentication token to log them out of the system.

2.3.1.1.1.1.2.1.3 Function: „getNewPassword”
Resets a user's password to a default one and sends an email notification to the user.

2.3.1.1.1.1.2.1.4 Function: „changePassword”
Changes a user's password after validating their old password.

2.3.1.1.1.1.2.2 File: „auth.register.controller.ts”
The file auth.register.controller.ts contains server-side logic for registering users and lawyers by verifying unique credentials, encrypting passwords, and creating their profiles in the database.

2.3.1.1.1.1.2.2.1 Function: „registerUser”
Function registers a new user after ensuring the uniqueness of email, phone, and ID.

2.3.1.1.1.1.2.2.2 Function: „registerLawyer”
function registers a new lawyer, also verifying the uniqueness of their license number.

2.3.1.1.1.1.2.3 File: „auth.role.controller.ts”
The file auth.role.controller.ts defines server-side functionality to fetch and send a list of user roles from the database to the client, handling the response with appropriate success or error messages.

2.3.1.1.1.1.2.3.1 Function: „ allRoles”
The function allRoles retrieves all the role records from the database, transforms the data through transformRoles, and sends it back to the client with a status code of 200 if successful, or handles errors appropriately if any occur.

2.3.1.1.1.1.2.4 File: „ user.controller.ts”
The script in user.controller.ts manages various user-related functionalities such as listing clients and lawyers, handling lawyer approvals and practice areas, and providing

user profiles, by interfacing with services and sending formatted responses.

2.3.1.1.1.1.2.4.1 Function: „ allClients"
The function allClients retrieves and sends a list of all clients after formatting the data.

2.3.1.1.1.1.2.4.2 Function: „ allLawyers"
The function allLawyers fetches and sends a list of all lawyers with formatted output.

2.3.1.1.1.1.2.4.3 Function: „allActiveLawyers"
The function allActiveLawyers obtains and sends a list of all active lawyers, specifically formatted.

2.3.1.1.1.1.2.4.4 Function: „lawyersByCityId"
The function lawyersByCityId fetches and sends lawyers based on a specific city, with formatted output.

2.3.1.1.1.1.2.4.5 Function: „approveLawyer"
The function approveLawyer updates a lawyer's status to active and confirms the approval.

2.3.1.1.1.1.2.4.6 Function: „newLawyerPractice"
The function newLawyerPractice adds a new practice area to a lawyer's profile.

2.3.1.1.1.1.2.4.7 Function: „allLawyerPractices"
The function allLawyerPractices retrieves and formats a list of all practice areas for a specific lawyer.

2.3.1.1.1.1.2.4.8 Function: „lawyerPracticesByLawyerId"
The function lawyerPracticesByLawyerId fetches and sends practice areas for a lawyer identified by ID.

2.3.1.1.1.1.2.4.9 Function: „lawyerById"
The function lawyerById retrieves and sends detailed information for a specific lawyer by their ID.

2.3.1.1.1.1.2.4.10 Function: „pendingLawyers"
The function pendingLawyers fetches and sends a list of lawyers awaiting approval.

2.3.1.1.1.1.2.4.11 Function: „getUsers"
The function getUsers retrieves and sends a list of all users with formatted data.

2.3.1.1.1.1.2.4.12 Function: „removeLawyerPracticeArea"
The function removeLawyerPracticeArea deletes a specific practice area from a lawyer's profile.

2.3.1.1.1.1.2.4.13 Function: „getMyUserData"
The function getMyUserData fetches and sends the profile data of the user making the request.

2.3.1.1.1.1.3 Directory: „middlewares"
The directory middlewares directory contains scripts for authenticating user sessions, checking user roles and account completion, and handling the uploading of PDF and image files to the server.

2.3.1.1.1.1.3.1 File: „ authenticate.ts"
The file authenticate.ts verifies if a user is logged in.

2.3.1.1.1.1.3.1.1 Function: „isLoggedIn"
The file isLoggedIn function checks if the user making a request is authenticated by

validating their JSON Web Token and allows the request to proceed if the token is verified.

2.3.1.1.1.1.3.2 File: „hasAccountSet.ts”

The file hasAccountSet.ts checks if the user's account details are complete.

2.3.1.1.1.1.3.2.1 Function: „hasAccountSet”

The hasAccountSet checks if a user has completed their profile by verifying address details, denying access if not, and allowing the request to proceed if they have.

2.3.1.1.1.1.3.3 File: „isSuperAdmin.ts”

The file isSuperAdmin.ts determines if the user has super administrator privileges.

2.3.1.1.1.1.3.3.1 Function: „isSuperAdmin”

The isSuperAdmin function verifies whether the user associated with the provided JWT token has super administrator privileges and allows them to proceed if so, otherwise it denies access.

2.3.1.1.1.1.3.4 File: „pdfupload.ts”

The file pdfupload.ts configures the upload of PDF files.

2.3.1.1.1.1.3.5 File: „ upload.ts”

The file sets up the upload of image files.

2.3.1.1.1.1.4 Directory: „routers”

The routes directory scripts map API endpoints to controller functions that execute CRUD operations and various actions for resources like users, appointments, cases, messages, and more, each file serving a distinct application feature.

2.3.1.1.1.1.4.1 File: „authRoutes.ts”

Manages routes for user and lawyer authentication processes like login and registration.

2.3.1.1.1.1.4.1.1 Function: „registerUser and registerLawyer”

handle the registration of users and lawyers, while login and logout manage the authentication flow.

2.3.1.1.1.1.4.2 File: „appointmentRoutes.ts”

Handles routes for booking, retrieving, and managing appointments.

2.3.1.1.1.1.4.2.1 Function: „as bookAppointment, getAppointmentsAsClien, and getAppointmentsAsLawyer”

deal with appointment creation and retrieval based on user roles.

2.3.1.1.1.1.4.3 File: „caseRoutes.ts”

Defines routes for case management, including case creation, retrieval, and case action tracking.

2.3.1.1.1.1.4.3.1 Function: „ newCase, getActiveCases, lawyerCases, and getAllCaseActions”

These functions allow for the creation, filtering, and tracking of legal cases.

2.3.1.1.1.1.4.4 File: „chatMessageRoutes.ts”

Establishes routes for messaging functionality between clients and lawyers.

2.3.1.1.1.1.4.4.1 Function: „getMyClientMessages and getMyLawyerMessages”

fetch message histories for clients and lawyers.

2.3.1.1.1.1.4.5 File: „cityRoutes.ts"

Sets up routes for retrieving city information. cityRoutes.ts simply uses the getAllCities function to list all cities.

2.3.1.1.1.1.4.6 File: „connectionRoutes.ts"

Controls routes for creating and managing connections between different user types.

2.3.1.1.1.1.4.6.1 Function: „newConnect, clientConnections, and approveConnection"

The functions to establish and manage professional connections.

2.3.1.1.1.1.4.7 File: „ index.ts"

Aggregates and exports all route modules to be used by the main application. It doesn't contain functions but is used to combine all route modules.

2.3.1.1.1.1.4.8 File: „lawRoutes.ts"

Manages routes related to law practices, including adding and removing practice areas.

2.3.1.1.1.1.4.8.1 Function: „getAllPractices, newLawyerPractice, and removeLawyerPracticeArea"

They are managing law practice information and lawyer specialties.

2.3.1.1.1.1.4.9 File: „passwordRoutes.ts"

Routes for user password management including reset and change functionalities.

2.3.1.1.1.1.4.10 File: „profileRoutes.ts"

Routes for user profile retrieval and updates, uses functions such as getMyProfile and updateProfile for users to view and edit their profile information.

2.3.1.1.1.1.4.11 File: „scheduleRoutes.ts"

Routes for managing user schedules and appointments, and it has functions like addScheduleAvailability and getLawyerAvailableSchedules to manage and retrieve available time slots for users.

2.3.1.1.1.1.4.12 File: „ staticRoutes.ts"

Routes for serving static files from the server. contains a function to retrieve uploaded files from a static directory.

2.3.1.1.1.1.4.13 File: „super-adminRoutes.ts"

Routes dedicated to super-admin users for managing the platform at a high level. Includes functions like allLawyers and allPayments, providing super-admin users with access to comprehensive data across the platform.

2.3.1.1.1.1.5 Directory: „validation"

Includes files that validate request data, ensuring that the data adheres to predefined schemas and constraints before processing.

2.3.1.1.1.1.5.1 File „actionAction.validation.ts."

This TypeScript file uses Joi to validate new case data in an Express.js app, ensuring required fields like title, description, and status are filled. If data is invalid, it returns an error; otherwise, it moves to the next function.

2.3.1.1.1.1.5.2 File „profileRoutes.ts."

The function validates that necessary fields (title, description, status, caseId) are present and correctly formatted in a request to create a new case, using Joi, and either proceeds to the next step or returns an error if the validation fails.

2.3.1.1.1.1.5.3 File „appointment.validation"

This file defines an asynchronous middleware function newAppointmentValidator that uses Joi to validate required fields (appointment_date, reason) in an appointment creation request for an Express.js application, and either returns an error response or proceeds to the next middleware function based on the validation result.

2.3.1.1.1.1.5.4 File „case.validation.ts"

This file defines an asynchronous middleware in an Express.js application that uses Joi to enforce validation rules for case-related data fields like title, description, and law practice, ensuring data integrity before proceeding or returning error responses.

2.3.1.1.1.1.5.5 File „payment.validation.ts"

This file defines an asynchronous middleware function requestPaymentValidation in an Express.js application that utilizes Joi for stringent validation of payment request data, such as amount, case_id, and currency, ensuring all fields meet predefined standards before proceeding or issuing an error response.

2.3.1.1.1.1.5.6 File „reviewValidation.ts."

This file defines an asynchronous middleware function newReviewValidator, an asynchronous middleware in an Express.js application that employs Joi to validate essential fields of a review submission, such as review_id, review_title, and review_text, ensuring compliance with required standards before advancing or returning error feedback.

2.3.1.1.1.1.5.7 File „schedule.validation.ts"

This file defines an asynchronous middleware function newAppointmentValidator, in an Express.js application that uses Joi to ensure that the appointment date is provided and valid before proceeding with the request or returning an error response.

2.3.1.1.1.1.5.8 File „schedule.validation.ts"

This file defines multiple asynchronous middleware functions in an Express.js application that use the Joi library to validate various user data inputs like names, email, password, and specific attributes related to user profiles and legal professionals, ensuring that all provided data meet the required standards before advancing in the request pipeline or returning error messages if validations fail.

2.4 Frontend architecture

The front end, designed for interactivity and user experience, employs components, hooks, and state management, among other paradigms.

2.4.1 Directory „ DP-M_MMJ_Fr”

2.4.1.1 Directory „src”

2.4.1.1.1.1 Directory „api”

2.4.1.1.1.1.1 File „ actions.ts.”

The React application's actions file effectively manages API interactions, data operations, and error handling, promoting best practices in asynchronous communication and improving user experience.

2.4.1.1.1.1.1.1 Function: „ saveCaseAction”

This function uses appAxios.post to asynchronously send action data to a server endpoint, utilizing Axios configured with specific application settings.

2.4.1.1.1.1.2 File „admin.ts.”

File contains functions designed for administrative tasks within the application, interacting with the server to manage lawyer-related data through Axios requests.

2.4.1.1.1.1.2.1 Function: „getLawyersByAdmin”

Retrieves a list of all lawyers through an Axios GET request to the /superadmin/lawyers/all endpoints.

2.4.1.1.1.1.2.2 Function: „getPendingLawyers”

Fetches a list of all pending lawyers by making an Axios GET request to the /superadmin/lawyers/pending/all endpoint, returning an array of lawyer data or an empty array on error.

2.4.1.1.1.1.2.3 Function: „approveLawyerById”

Sends a PUT request using Axios to the /users/lawyers/approve/{lawyerId} endpoint to approve a specific lawyer, returning a success status and message upon successful approval.

2.4.1.1.1.1.3 File „ auth.ts”and Functions.

The auth.ts file manages authentication and user profiles in a React application using Axios for server communication. It sends POST, GET, and POST requests, handles responses.

appropriately, and manages error scenarios for smooth user experience.

2.4.1.1.1.1.4 File „ axios.ts”

The axios.ts file configures two Axios instances, appAxios and appAxiosUpload, for API calls and file uploads, using interceptors for secure requests and user ID information.

3 Code documentation.

The Legal Connect System is a web-based application that connects clients with legal professionals, schedules consultations, and provides efficient services. Utilizing modern technologies like React JS, Node.js with Express, and SQL, it digitizes legal services.

3.1 Backend `DP-M_MMJ_Bc`

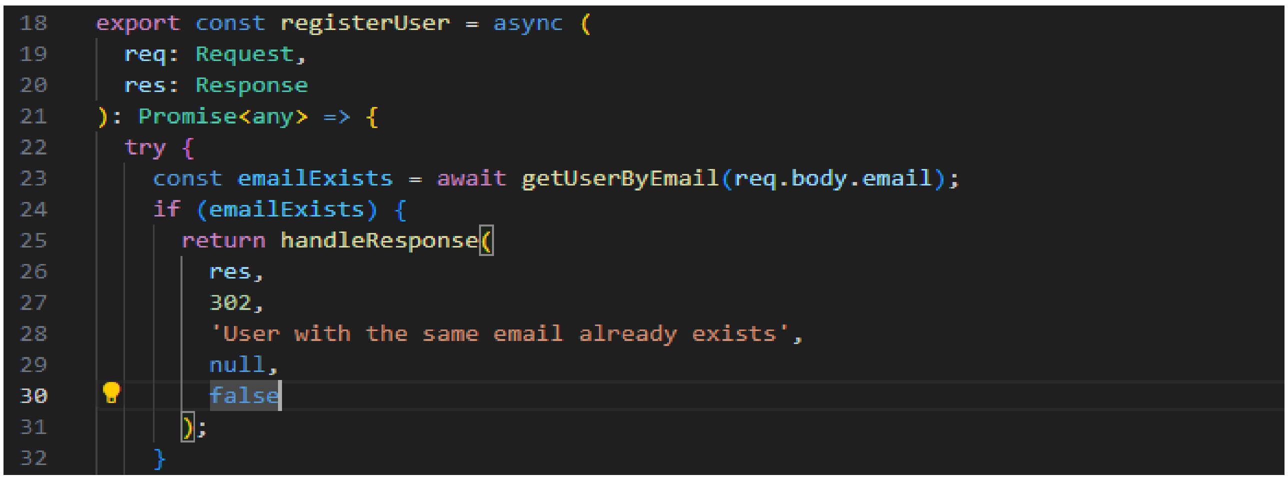
3.1.1 User Authentication and logic Process

3.1.1.1 Path: `src/app/controllers/auth/`

3.1.1.2 File `auth.register.controller.ts`

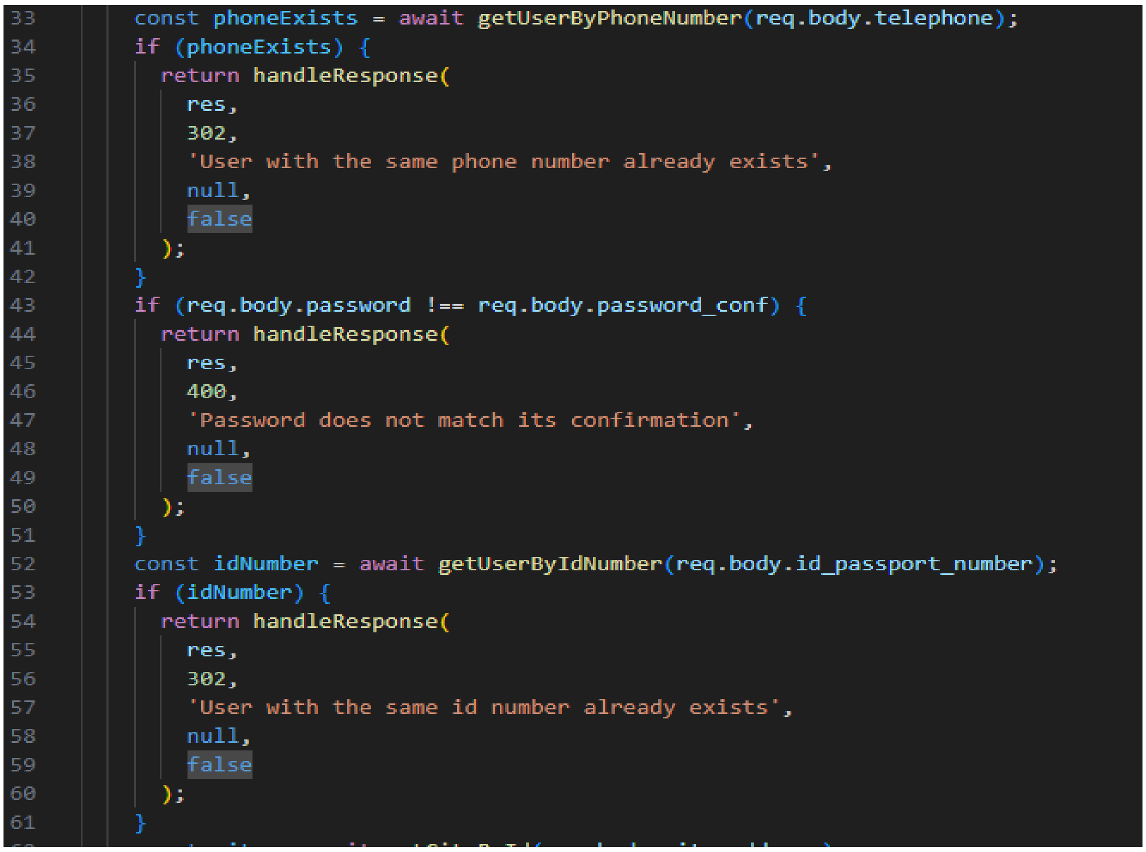
The file `src/app/controllers/auth/auth.register.controller.ts` is for registering new users, making sure they don't use an email, phone number, or ID that's already in the system. It also checks that the password is typed correctly twice to avoid mistakes.

3.1.1.2.1 Function `registerUser`



```
export const registerUser = async (
  req: Request,
  res: Response
): Promise<any> => {
  try {
    const emailExists = await getUserByEmail(req.body.email);
    if (emailExists) {
      return handleResponse(
        res,
        302,
        'User with the same email already exists',
        null,
        false
      );
    }
```

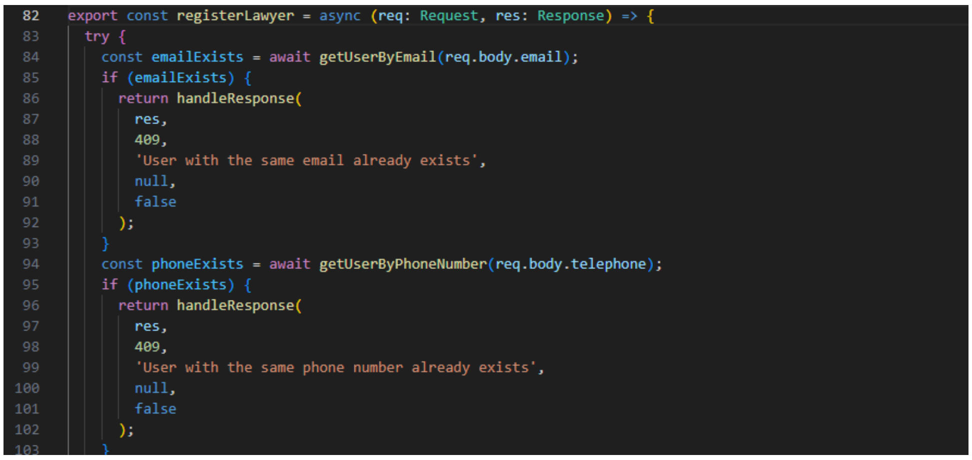
The `registerUser` function checks if a user with the same email, phone, or ID number already exists; if not, it creates a new user with a hashed password and associates them with a city. If everything goes well, it creates a client profile for the user.



```
    const phoneExists = await getUserByPhoneNumber(req.body.telephone);
    if (phoneExists) {
      return handleResponse(
        res,
        302,
        'User with the same phone number already exists',
        null,
        false
      );
    }
    if (req.body.password !== req.body.password_conf) {
      return handleResponse(
        res,
        400,
        'Password does not match its confirmation',
        null,
        false
      );
    }
    const idNumber = await getUserByIdNumber(req.body.id_passport_number);
    if (idNumber) {
      return handleResponse(
        res,
        302,
        'User with the same id number already exists',
        null,
        false
      );
    }
```

3.1.1.2.2 Function `registerLawyers`

The `registerLawyer function performs similar checks for an email, phone number, and ID number, as well as a unique lawyer license number. If all checks are clear, it registers a new lawyer with hashed password details, creates a lawyer profile, and sends a notification message.



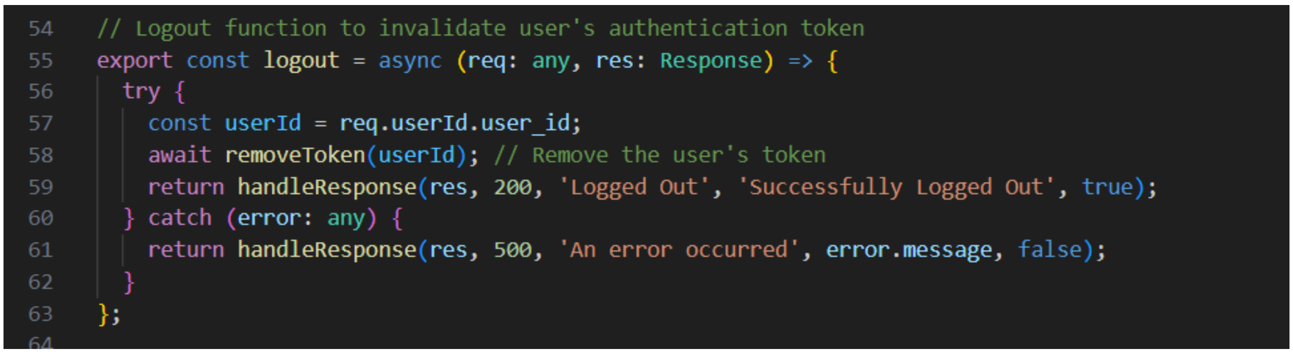
```
export const registerLawyer = async (req: Request, res: Response) => {
  try {
    const emailExists = await getUserByEmail(req.body.email);
    if (emailExists) {
      return handleResponse(
        res,
        409,
        'User with the same email already exists',
        null,
        false
      );
    }
    const phoneExists = await getUserByPhoneNumber(req.body.telephone);
    if (phoneExists) {
      return handleResponse(
        res,
        409,
        'User with the same phone number already exists',
        null,
        false
      );
    }
```

3.1.1.3 File ` auth.login.controller.ts`

3.1.1.3.1 Function `login`

This code defines a login function that authenticates users. First, it checks if the user's email and password are provided. If not, it responds with an error. If the credentials are provided, it tries to find the user by their email. If the user doesn't exist, it responds that the email doesn't exist. If the user is found, it compares the provided password with the one stored in the database using bcrypt for security. If the password matches, it generates an authentication token for the user. It then checks if the user's profile exists and creates one if it doesn't. Finally, if the login is successful, it responds with a success message and the user's data. If any errors occur during this process, it catches them and responds with an appropriate error message.

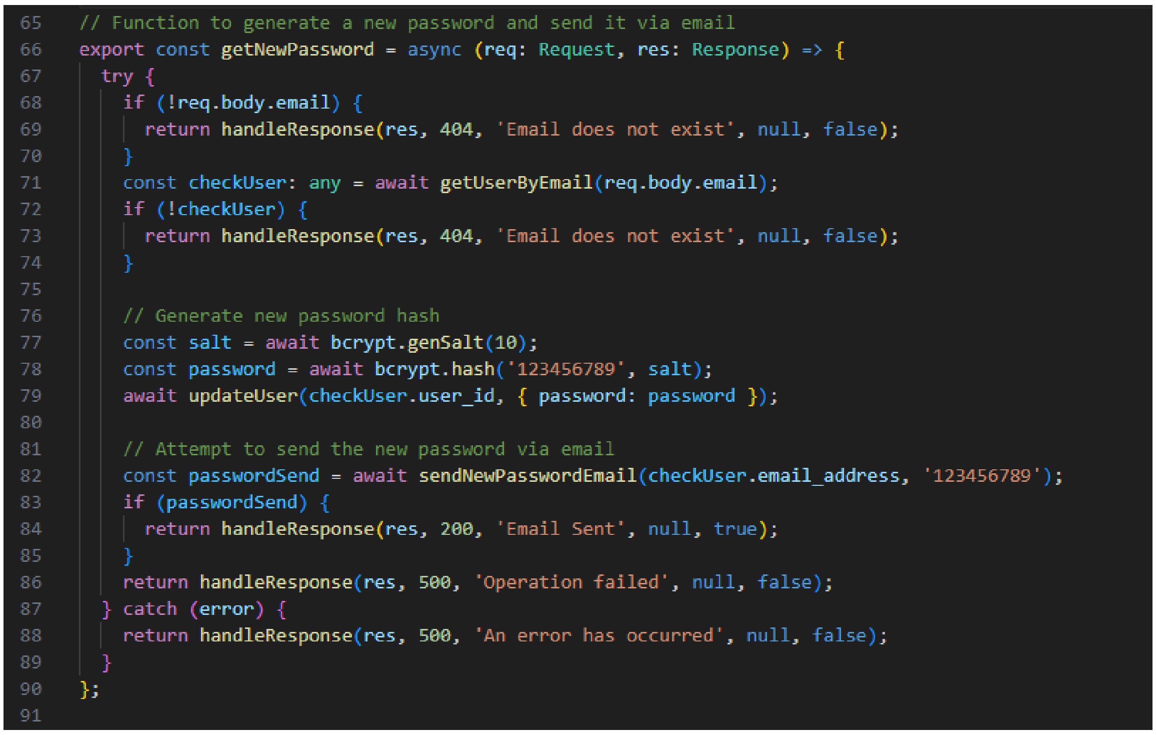
3.1.1.3.2 Function `logout`



```
// Logout function to invalidate user's authentication token
export const logout = async (req: any, res: Response) => {
  try {
    const userId = req.userId.user_id;
    await removeToken(userId); // Remove the user's token
    return handleResponse(res, 200, 'Logged Out', 'Successfully Logged Out', true);
  } catch (error: any) {
    return handleResponse(res, 500, 'An error occurred', error.message, false);
  }
};
```

This code outlines a logout function that deactivates a user's authentication token, effectively logging them out. It works by first identifying the user through their ID, then proceeding to remove their current token from the system to invalidate their session. Upon successful token removal, the user receives confirmation of their logout. If any errors are encountered during this process, such as issues with token removal or user identification, the function catches these errors and sends back an error message to the user.

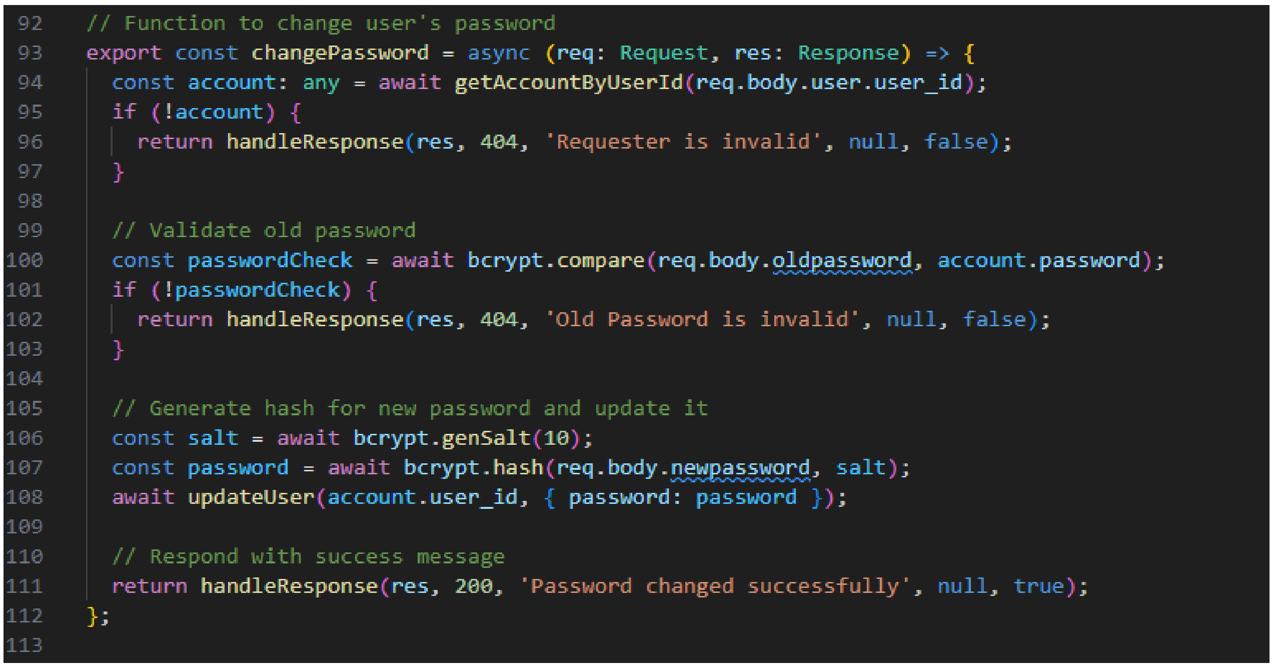
3.1.1.3.3 Function `getNewPassword`



```
// Function to generate a new password and send it via email
export const getNewPassword = async (req: Request, res: Response) => {
  try {
    if (!req.body.email) {
      return handleResponse(res, 404, 'Email does not exist', null, false);
    }
    const checkUser: any = await getUserByEmail(req.body.email);
    if (!checkUser) {
      return handleResponse(res, 404, 'Email does not exist', null, false);
    }

    // Generate new password hash
    const salt = await bcrypt.genSalt(10);
    const password = await bcrypt.hash('123456789', salt);
    await updateUser(checkUser.user_id, { password: password });

    // Attempt to send the new password via email
    const passwordSend = await sendNewPasswordEmail(checkUser.email_address, '123456789');
    if (passwordSend) {
      return handleResponse(res, 200, 'Email Sent', null, true);
    }
    return handleResponse(res, 500, 'Operation failed', null, false);
  } catch (error) {
    return handleResponse(res, 500, 'An error has occurred', null, false);
  }
};
```

The code defines a function for resetting a user's password. It checks if the user's email is provided and sends an error message if not. If found, it creates a new password hash using bcrypt, updates the user's database, and sends the new password to the user's email. If errors occur, the operation is canceled.

3.1.1.3.4 Function: `changePassword`



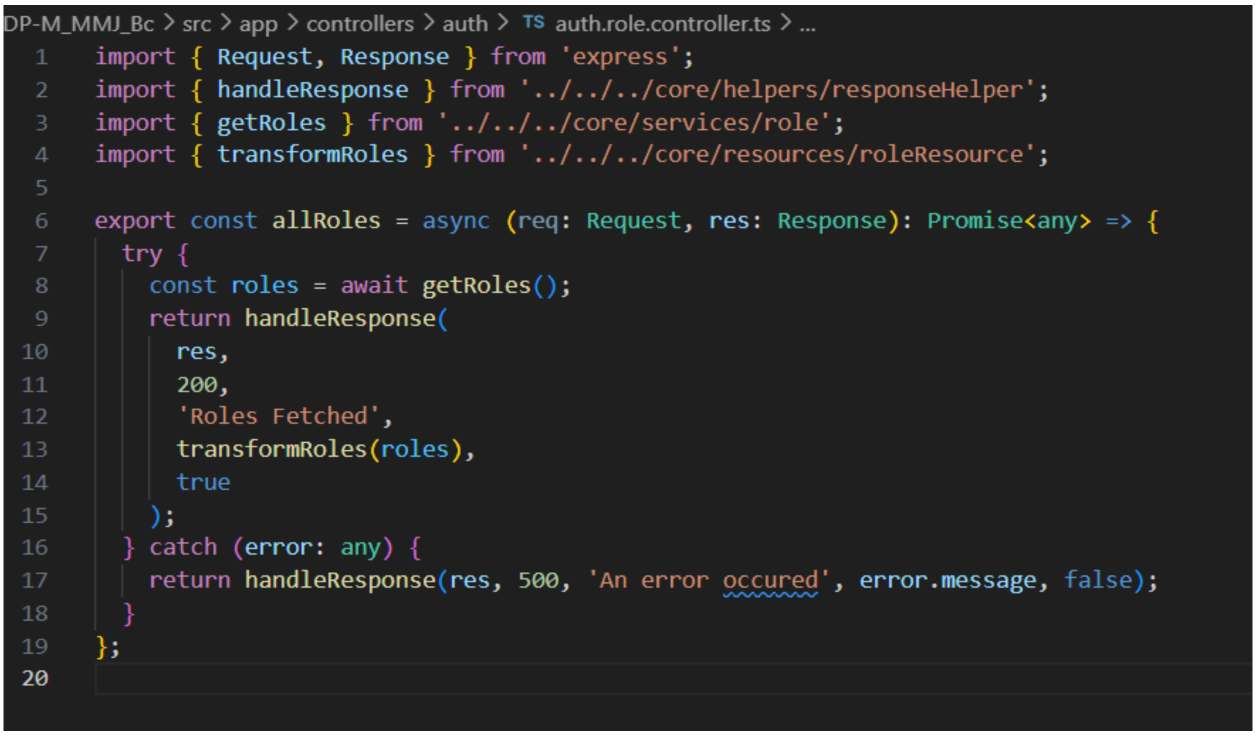
```
// Function to change user's password
export const changePassword = async (req: Request, res: Response) => {
  const account: any = await getAccountByUserId(req.body.user.user_id);
  if (!account) {
    return handleResponse(res, 404, 'Requester is invalid', null, false);
  }

  // Validate old password
  const passwordCheck = await bcrypt.compare(req.body.oldpassword, account.password);
  if (!passwordCheck) {
    return handleResponse(res, 404, 'Old Password is invalid', null, false);
  }

  // Generate hash for new password and update it
  const salt = await bcrypt.genSalt(10);
  const password = await bcrypt.hash(req.body.newpassword, salt);
  await updateUser(account.user_id, { password: password });

  // Respond with success message
  return handleResponse(res, 200, 'Password changed successfully', null, true);
};
```

The changePassword function is designed to securely update a user's password. It first confirms the user's identity, then checks if the old password matches the one on record. If everything is correct, it creates a new password hash and updates the user's account. Finally, it confirms the password change with a success message.

3.1.1.4 File `auth.role.controller.ts`

The file ` **src/app/controllers/auth/auth.role.controller.ts**` fetches a list of user roles from the database and sends them back to the user. If there's a problem, it reports an error.

3.1.1.4.1 Function: `allRoles`

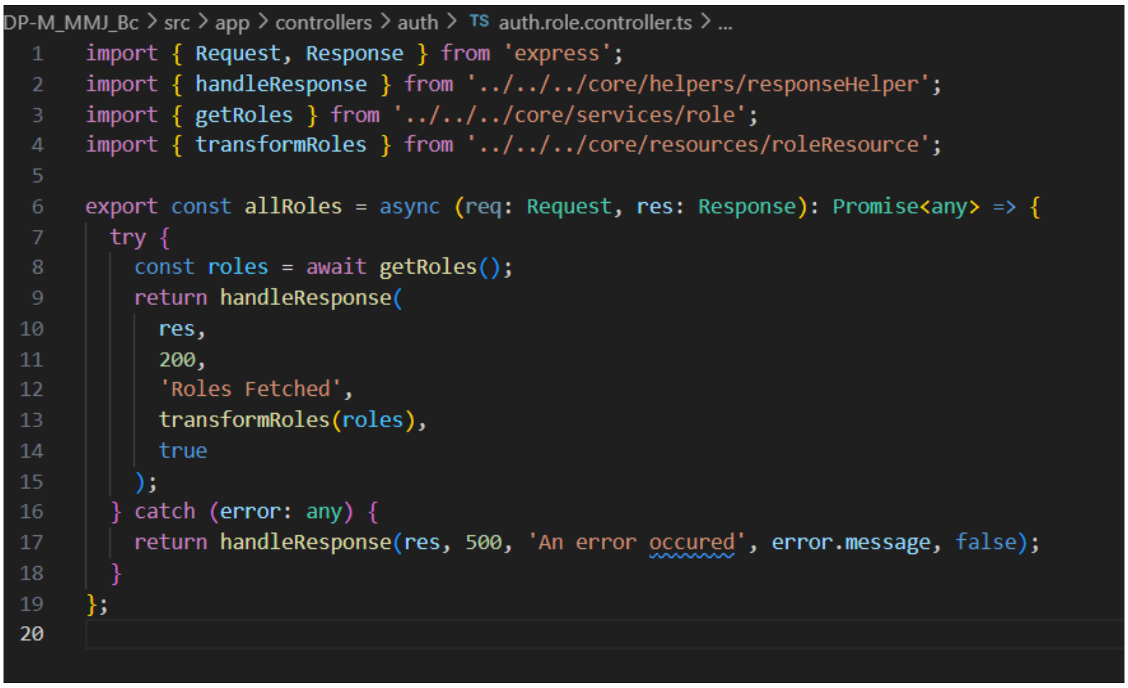

DP-M_MMJ_Bc > src > app > controllers > auth > TS auth.role.controller.ts > ...

```
import { Request, Response } from 'express';
import { handleResponse } from '../../../core/helpers/responseHelper';
import { getRoles } from '../../../core/services/role';
import { transformRoles } from '../../../core/resources/roleResource';

export const allRoles = async (req: Request, res: Response): Promise<any> => {
  try {
    const roles = await getRoles();
    return handleResponse(
      res,
      200,
      'Roles Fetched',
      transformRoles(roles),
      true
    );
  } catch (error: any) {
    return handleResponse(res, 500, 'An error occured', error.message, false);
  }
};

```

This code defines a function that retrieves a list of all user roles from the system. When called, it tries to get the roles, processes them for the response, and sends them back to the requester with a success status. If an error occurs during this process, it catches the error and sends back an error message.

3.1.1.5 File `user.controller.ts`

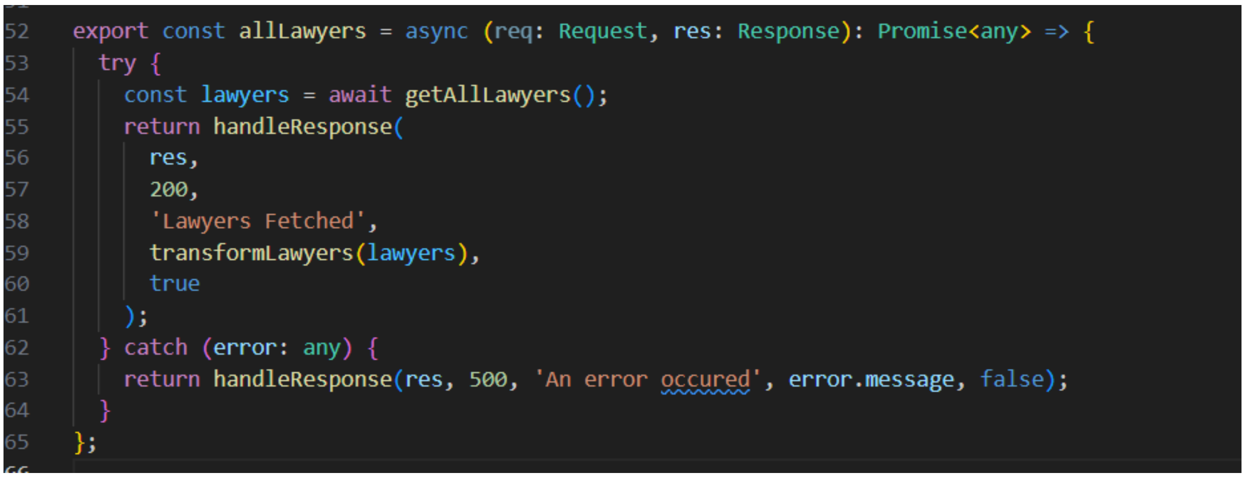
3.1.1.5.1 Function `allClients`


DP-M_MMJ_Bc > src > app > controllers > auth > TS auth.role.controller.ts > ...

```
import { Request, Response } from 'express';
import { handleResponse } from '../../../core/helpers/responseHelper';
import { getRoles } from '../../../core/services/role';
import { transformRoles } from '../../../core/resources/roleResource';

export const allRoles = async (req: Request, res: Response): Promise<any> => {
  try {
    const roles = await getRoles();
    return handleResponse(
      res,
      200,
      'Roles Fetched',
      transformRoles(roles),
      true
    );
  } catch (error: any) {
    return handleResponse(res, 500, 'An error occured', error.message, false);
  }
};

```

This function retrieves all clients from the database using the getAllClients service and transforms the data using transformClients. It returns the transformed data through a standardized response format via handleResponse. Error handling is included to manage any issues during the fetch process.

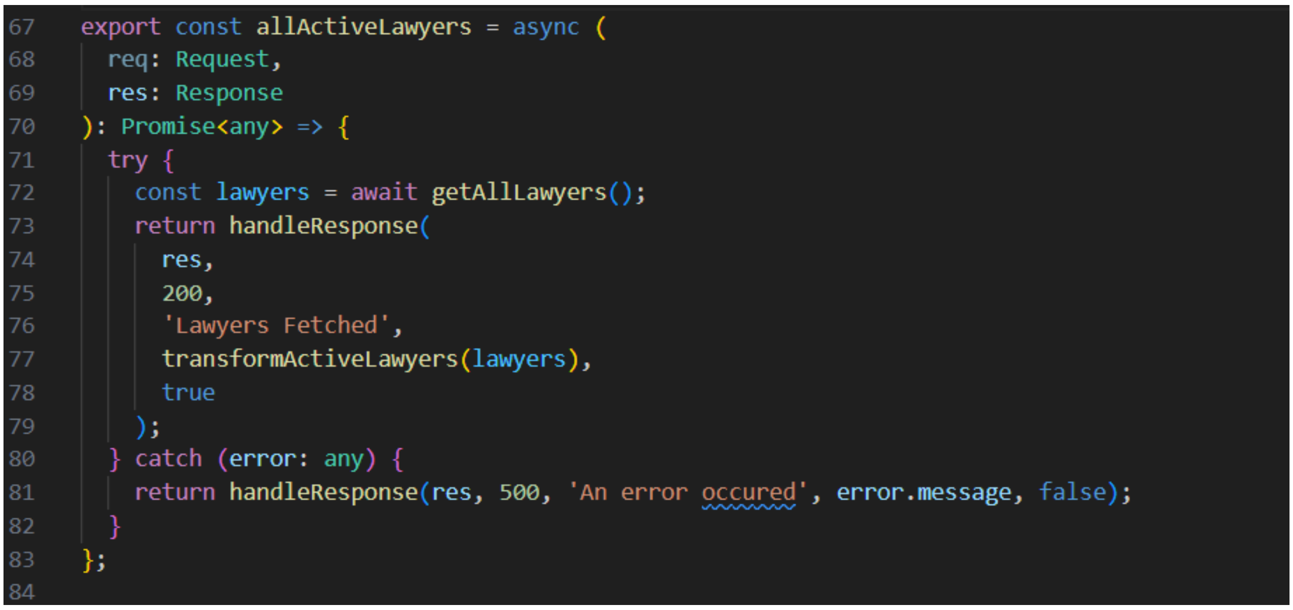
3.1.1.5.2 Function `allLawyers`



```
export const allLawyers = async (req: Request, res: Response): Promise<any> => {
  try {
    const lawyers = await getAllLawyers();
    return handleResponse(
      res,
      200,
      'Lawyers Fetched',
      transformLawyers(lawyers),
      true
    );
  } catch (error: any) {
    return handleResponse(res, 500, 'An error occured', error.message, false);
  }
};
```

Retrieves a list of all lawyers by invoking the getAllLawyers service. The data is transformed using transformLawyers and returned to the client using the handleResponse function. It handles potential errors by responding with an appropriate error message.

3.1.1.5.3 Function `allActiveLawyers`



```
export const allActiveLawyers = async (
  req: Request,
  res: Response
): Promise<any> => {
  try {
    const lawyers = await getAllLawyers();
    return handleResponse(
      res,
      200,
      'Lawyers Fetched',
      transformActiveLawyers(lawyers),
      true
    );
  } catch (error: any) {
    return handleResponse(res, 500, 'An error occured', error.message, false);
  }
};
```

Like allLawyers, but it specifically processes and returns only active lawyers using transformActiveLawyers for data transformation. It ensures data consistency and error management through standard response handling.

3.1.1.5.4 Function `lawyersByCityId`

This function fetches lawyers based on a city ID provided in the request query. It first verifies the city's existence, then retrieves lawyers associated with that city, transforms the data, and returns it using a standard response structure.

3.1.1.5.5 Function `approveLawyer`

Approves a lawyer by updating their status to 'active' using the lawyer's ID from the request parameters. It confirms the lawyer’s existence and saves the updated data, handling both success and error scenarios with appropriate responses.

3.1.1.5.6 Function `newLawyerPractice`

Allows the addition of a new practice area to a lawyer’s profile. It verifies the lawyer and the practice's existence, creates a new lawyer practice recod, and updates the database, providing feedback through a standardized response.

3.1.1.5.7 Function `allLawyerPractices`

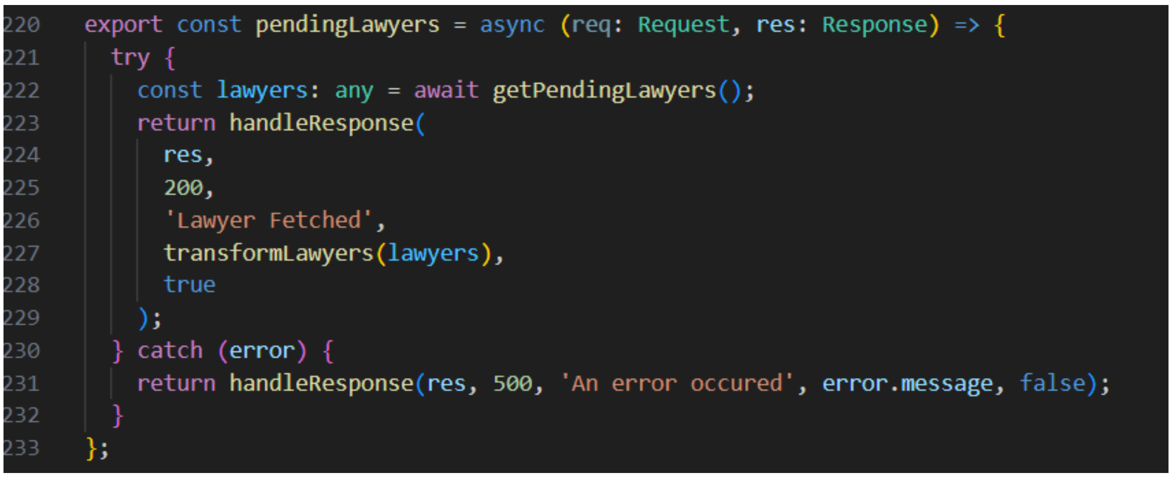
Retrieves all practice areas for a lawyer identified by the user ID in the request body. It ensures both the account and lawyer's existence before fetching and returning the practice details, handling errors appropriately.

3.1.1.5.8 Function `lawyerPracticesByLawyerId`

Fetches practice areas associated with a specific lawyer using their ID. It ensures the lawyer's existence and gathers practice details for response, maintaining error handling and standardized responses.

3.1.1.5.9 Function `lawyerById`

3.1.1.5.10 Function `pendingLawyers`



```
export const pendingLawyers = async (req: Request, res: Response) => {
  try {
    const lawyers: any = await getPendingLawyers();
    return handleResponse(
      res,
      200,
      'Lawyer Fetched',
      transformLawyers(lawyers),
      true
    );
  } catch (error) {
    return handleResponse(res, 500, 'An error occured', error.message, false);
  }
};
```

Fetches lawyers whose accounts are pending approval. It uses getPendingLawyers for data retrieval, transforms it, and provides the result or error message through standardized responses.

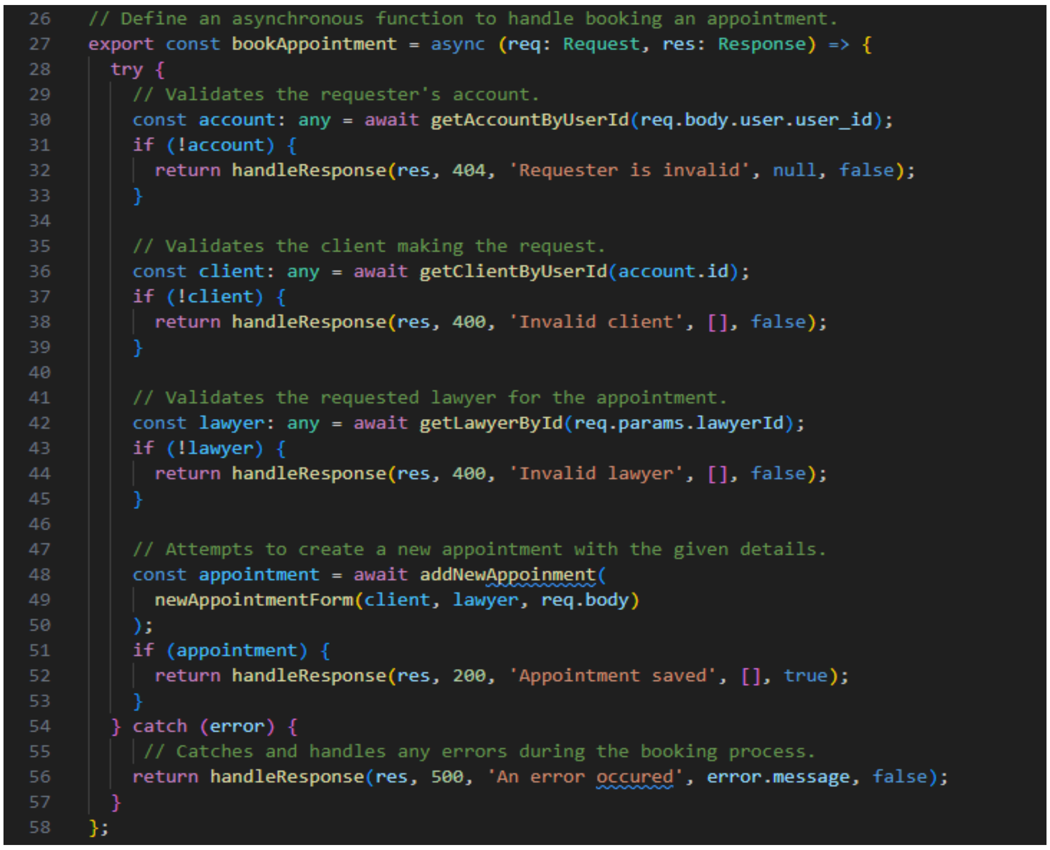
3.1.1.5.11 Function `getUsers`

This function lists all users in the system by using the getAllUsers service. It transforms the user data for presentation and manages both success and error scenarios through standardized response handling.

3.1.1.6 File `appointment.controller.ts`

This TypeScript file serves as a controller for appointment-related actions in an Express.js application. It handles operations such as booking appointments, retrieving appointments for clients and lawyers, and accepting appointments.

3.1.1.6.1 Function `bookAppointment`



```
// Define an asynchronous function to handle booking an appointment.
export const bookAppointment = async (req: Request, res: Response) => {
  try {
    // Validates the requester's account.
    const account: any = await getAccountByUserId(req.body.user.user_id);
    if (!account) {
      return handleResponse(res, 404, 'Requester is invalid', null, false);
    }

    // Validates the client making the request.
    const client: any = await getClientByUserId(account.id);
    if (!client) {
      return handleResponse(res, 400, 'Invalid client', [], false);
    }

    // Validates the requested lawyer for the appointment.
    const lawyer: any = await getLawyerById(req.params.lawyerId);
    if (!lawyer) {
      return handleResponse(res, 400, 'Invalid lawyer', [], false);
    }

    // Attempts to create a new appointment with the given details.
    const appointment = await addNewAppoinment(
      newAppointmentForm(client, lawyer, req.body)
    );
    if (appointment) {
      return handleResponse(res, 200, 'Appointment saved', [], true);
    }
  } catch (error) {
     // Catches and handles any errors during the booking process.
    return handleResponse(res, 500, 'An error occured', error.message, false);
  }
};
```

bookAppointment: Validates user and lawyer information, creates a new appointment based on the request data, and returns a confirmation response. Error handling is included to manage invalid requests and system errors.

3.1.1.6.2 Function ` getAppointmentsAsClient`

getAppointmentsAsClient: Retrieves all appointments associated with the client making the request, transforming the data for the response. It ensures the validity of the requester's account before proceeding.

3.1.1.6.3 Function `getAppointmentAsLawyer`

getAppointmentsAsLawyer: Fetches appointments related to the requesting lawyer, similarly, ensuring account validity and transforming the appointment data for response. It handles potential errors during the data retrieval process.

3.1.1.6.4 Function `acceptAppointment`

acceptAppointment: This function updates the status of an appointment to 'accepted'. It checks if the appointment exists and returns a confirmation upon success. Error handling is included for both non-existent appointment references and system errors.
This function facilitates the creation of a new legal case. It first verifies the identity and validity of the user attempting to create the case, then confirms the details of the client and the lawyer involved, as well as the specific legal practice and category related to the case. Upon successful validation and creation, it sends a successful response; otherwise, it handles errors such as invalid inputs or server issues.

3.1.1.6.5 Function `caseById`

caseById is designed to fetch detailed information about a single case using its unique ID. This is useful for obtaining comprehensive details about a case, which could include the status, parties involved, and other relevant case data. It's particularly helpful for clients or lawyers who need to review the specifics of a case.

3.1.1.6.6 Function `getCasesByLawyer`

This function serves a purpose like lawyerCases, but it specifically queries for cases by a given lawyer ID passed in the request parameters. It's useful for fetching a caseload when the lawyer's ID is known, such as for administrative purposes or when lawyers have multiple accounts.

3.1.1.6.7 Function `activeAllCases`

The function **activeAllCases** offers a system-wide view of all active cases, irrespective of the client or lawyer. It's typically used by administrators or for reporting purposes to get a sense of all ongoing legal activities within the firm or legal service.

3.1.1.6.8 Function `pendingAllCases`

Conversely, **pendingAllCases** compiles a list of all cases that are marked as 'pending', which usually signifies that these cases are awaiting some form of initiation or completion of an action. This is another administrative tool that can help with managing workflows and understanding bottlenecks or pending workloads.

3.1.1.6.9 Function `getMyLawyerMessages`

This function **getMyLawyerMessages** Mirroring the structure of the previous function, getMyLawyerMessages serves the purpose from the perspective of the lawyer. It authenticates the lawyer's account against the user's session and then retrieves messages from the perspective of the lawyer communicating with a specific client. The outcome mirrors that of getMyClientMessages, either returning the conversation history in a client-friendly format or providing an appropriate error response in the absence of a valid communication thread.

3.1.1.7 File `connection.controller.ts`

The connection.controller.ts file of an Express.js application manages the networking features between clients and lawyers. It provides endpoints to establish new connections, retrieve existing connections, and approve connection requests.

3.1.1.7.1 Function ` newConnect `

Function newConnect Initiates a new professional connection between a client and a lawyer after ensuring both parties exist, and no prior connection is in place. It responds with success upon creating a new connection or an error if any validation fails.

3.1.1.7.2 Function ` lawyerConnections `

Function lawyerConnections Retrieves all connections associated with a lawyer, transforming them into a format suitable for response. It checks for the validity of the lawyer's account and confirms successful data retrieval.

3.1.1.7.3 File ` payment.controller.ts `

The payment.controller.ts file within an Express.js application contains a collection of functions that manage the financial transactions associated with legal services provided by the application. These functions address various aspects of the payment process, from initiating payment requests to updating payment statuses.

3.1.1.7.3.1 Function ` requestPayment `

This function is for initiating a new payment request related to a specific case. It ensures that the case exists before creating the payment request and responds with a success message upon successful creation or an error if the case is invalid.

3.1.1.7.3.2 Function `getClientPaymentRequests `

Function getClientPaymentRequests Retrieves all payment requests made by a specific client. It validates the client's account and aggregates their payment requests, which are then formatted and sent back in the response.

3.1.1.7.3.3 Function `lawyerPaymentRequests`

Function lawyerPaymentRequests Similar to getClientPaymentRequests, this function fetches all payment requests for a specific lawyer, ensuring that the lawyer's account is valid before retrieving the payment data.

3.1.1.7.3.4 Function `allPayments`

Function allPayments Provides an overview of all payments within the system, which can be useful for administrative purposes to monitor transactions.

3.1.1.7.3.5 Function `paymentTransactionById `

Function paymentTransactionById Fetches a single payment transaction by its ID, providing detailed information about that payment request, which is useful for tracking individual transactions.

3.1.1.7.3.6 Function `payTransaction`

Function payTransaction Processes the action of marking a transaction as paid, which includes updating the payment status to 'paid' and modifying the associated case status accordingly. This is an important function for maintaining the integrity of financial records.

3.1.1.8 File ` payment.controller.ts `profile.controller.ts`

The profile.controller.ts module is responsible for handling user profile-related operations Review Visi. bility Toggle (changeReviewStatus): The ability to switch a review's visibility between public and private states not only empowers users with control over their content but also illustrates the platform's adaptability to user privacy preferences.

3.1.1.9 File `schedules.controller.ts`

The schedules.controller.ts module is a core element that allows lawyers to manage their availability and clients to view event schedules within a legal services application. It facilitates various scheduling-related functionalities, essential for the organization and planning of legal consultations and services:

3.1.1.9.1 Function `addScheduleAvailability`

Add Schedule Availability (addScheduleAvailability): Lawyers can declare their available times through this function. It takes a lawyer's information and the desired schedule to create new available time slots. This is critical for clients to find and book time with a lawyer.

3.1.1.9.2 Function `allEvents`

View All Schedules (allEvents): This function retrieves all scheduled events within the platform, giving an overview that could be used for administrative monitoring or to help clients find available slots across all lawyers.

3.1.1.9.3 Function `myEventAvailabilities`

View Personal Event Availabilities (myEventAvailabilities): Lawyers can use this function to check their own schedules. This personal view ensures that lawyers can manage and plan their commitments effectively.

3.1.1.9.4 Function `getLawyerAvailableSchedules`

Retrieve Available Schedules for a Specific Lawyer (getLawyerAvailableSchedules): Clients can use this function to view the availability of a specific lawyer, helping them to schedule appointments based on the lawyer's free time.

3.2 Frontend `DP-M_MMJ_Fr`

3.2.1 The `src` directory where the main application code resides.

3.2.1.1 Path ` DP-M_MMJ_Fr /src/api`

This folder may contain files related to API calls, often organized as services or utilities that interface with backend services.

3.2.1.1.1 File `actions.ts`

This section of the thesis discusses the implementation of key functions in the software using Axios for HTTP requests, which are crucial for managing case actions and user queries within the application. The functions are designed to enhance user interaction by providing backend support for various operations:

3.2.1.1.1.1 Function `saveCaseAction`

This function is responsible for submitting case actions to a server endpoint. Upon a successful post request, the function verifies the server response and returns a success message. In case of an error, particularly one related to network issues as identified by an instance of AxiosError, the function provides an appropriate error message, thereby enhancing error management in the application.

3.2.1.1.1.2 Function `getActionsById`

This function retrieves case actions based on a specific identifier from the server. The successful retrieval of data results in returning the actual data, while failure due to any error results in an empty array. This function demonstrates error handling where no complex error message is required, reflecting simplified error feedback for operations that fetch data.

3.2.1.1.1.3 Function `uploadActionFile`

Dedicated to uploading files associated with case actions, this function prepares a FormData object, appending necessary details and the file, and posts it to a designated server endpoint. Success or failure of the upload is communicated through respective messages, illustrating the application's capability to handle file uploads alongside standard data requests.

3.2.1.1.1.4 Function `faqQuestions`

This function fetches FAQs from the server. If the request is successful and the server's response contains a status of success, it returns the FAQs data; otherwise, it returns an empty array.

3.2.1.1.2 File `admins.ts`

The admins.ts TypeScript module serves as an interface for administrative actions related to managing lawyer and client accounts within an application. The functions utilize axios for HTTP requests to interact with a backend API and handle asynchronous operations.

3.2.1.1.2.1 Function `getLawyersByAdmin`

This function asynchronously retrieves all lawyer accounts registered in the system. If the operation is successful and the server returns a status indicating so, the function provides the list of lawyers; otherwise, it defaults to returning an empty array, likely used to signify no data or an error without throwing an exception.

3.2.1.1.2.2 Function `getPendingLawyers`

Like the previous function, this one fetches all lawyer accounts that are in a pending state, awaiting administrative approval. Success and error handling mimic the approach taken in getLawyersByAdmin, with the provision of data or an empty array.

3.2.1.1.2.3 Function `approveAlawyerById`

This function is responsible for approving a lawyer account by a given ID. It sends a PUT request to update the status of a lawyer's account to 'approved'. Upon successful approval, it returns an object with a true status and a success message; in the case of an error, especially one associated with Axios, it provides a detailed error message or a general failure notice.

3.2.1.1.2.4 Function `approveAlawyerById `

This function fetches all client accounts through an administrative endpoint. It handles responses and errors similarly to the lawyer retrieval functions, providing data or an empty array based on the presence of a success status in the server's response.

3.2.1.1.3 File `axios.ts`

The axios.ts module centralizes HTTP request configurations for a web application, creating two distinct axios instances for handling general API interactions and file uploads. These instances are equipped with interceptors that automatically append authentication tokens from local storage to each request, ensuring secure and authorized communication with the backend. The module's design emphasizes modularity, security, and maintainability in the network layer of the application.

4 Data base

For the Legal Connect System, an SQL database was chosen for use. This database keeps all the system's important information, like user details and case information, safe and well-organized. We needed an SQL database because it allows us to keep track of lots of data and helps us make sure that the data is accurate and secure.

We decided on an SQL database because it is great for handling complex data and relationships, like those between clients and lawyers or lawyers and their cases. It's a popular choice which means there's a lot of support and it's reliable. It's also designed to handle lots of operations and can grow as more people use our system. This is important for making sure that as our system gets bigger, it continues to work well and keeps information safe.

4.1 Schema Overview

The database is set up with several tables, each serving a specific function to make the system work effectively. There are multiple tables in the database, and each table has a primary role. For example, some tables store details about the users, like lawyers and clients, including their names and contact information. Other tables could hold information about legal cases, documents related to these cases, and the interactions between clients and lawyers.

Overall, the number of tables in the database is Twenty-one (21), but they work together to keep the system organized and efficient. Each table connects with others in a way that helps retrieve and manage data quickly, supporting the functionality of the whole system.

4.2 Description of the database schema.

The Legal Connect System's database is composed of multiple tables, each serving a specific purpose and containing various data types. Here's a concise summary of Legal Connect System:

- **appointments:** Manages information on scheduled appointments, with fields for unique IDs, client and lawyer identifiers, and appointment details.
- **appointments_schedules:** Holds lawyers' available slots for appointments.
- **case_categories:** Categorizes legal cases into types.
- **case_trackers:** Monitors progress and actions in legal cases.
- **cases:** Contains detailed information about legal cases.
- **chat_messages:** Records messages sent between users of the system.
- **cities:** Lists cities likely related to the location of cases or practices.
- **clients:** Stores client profiles and related information.

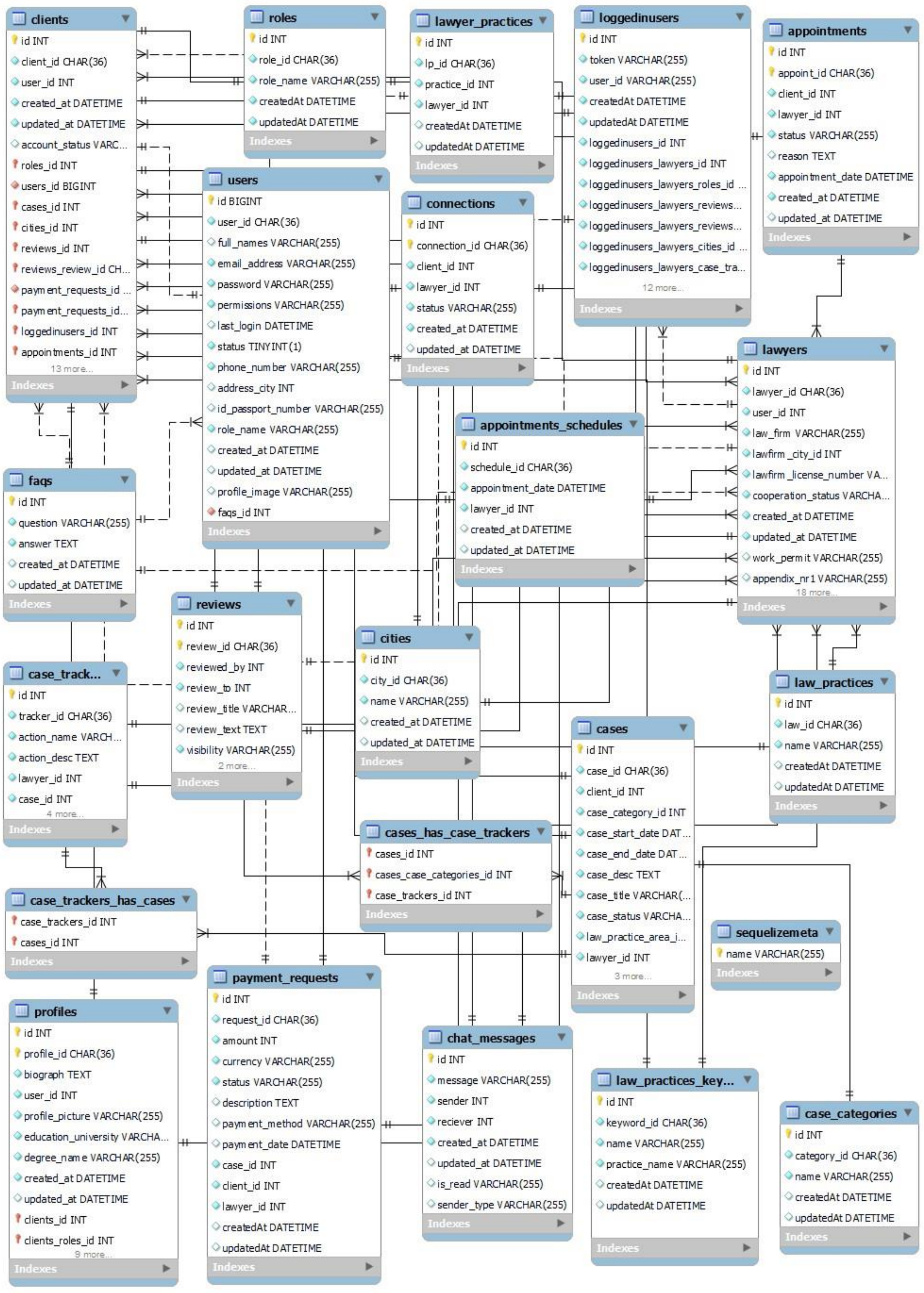
- **connections:** Tracks connections, possibly between clients and lawyers or between cases and legal actions.
- **faqs:** Frequently Asked Questions section, providing quick information to users.
- **law_practices:** Likely holds information about various law practices within the system.
- **law_practices_keywords:** Manages keywords associated with law practices.
- **lawyer_practices:** Relates lawyers with their areas of practice.
- **lawyers:** Details profiles of lawyers.
- **logged_in_users:** Keeps track of users currently logged into the system.
- **payment_requests:** Manages billing and payment details for services rendered.
- **profiles:** User profiles, which may include extended information beyond basic login details.
- **reviews:** User-generated reviews for lawyers or services.
- **roles:** Defines roles within the system to manage access and permissions.
- **sequelize_meta:** Typically used for system operations like tracking migrations.
- **users:** The main table for user information, likely holding login credentials and basic profile information.

Each table is equipped with an array of fields, including primary keys (PK) for unique identification and foreign keys (FK) for relational linkages between tables, ensuring data integrity and relational mapping. The data types range from integers (INT), variable character strings (VARCHAR), text blocks (TEXT), and date-time stamps (DATETIME), facilitating a robust structure for storing and managing legal and user-related data.

4.3 Entity-Relationship Diagram (ERD)

The Entity-Relationship Diagram (ERD) for the Legal Connect System visualizes the database's structure, showcasing how data is organized and interlinked. This ERD is crucial for developers and stakeholders to understand the system's data model, relationships, and flow, thereby supporting efficient design, development, and future enhancements of the Legal Connect System.

Find below the picture of Entity-Relationship Diagram


clients
id INT
client_id CHAR(36)
user_id INT
created_at DATETIME
updated_at DATETIME
account_status VARC...
roles_id INT
users_id BIGINT
cases_id INT
cities_id INT
reviews_id INT
reviews_review_id CH...
payment_requests_id ...
payment_requests_id...
loggedinusers_id INT
appointments_id INT
13 more...
Indexes
roles
id INT
role_id CHAR(36)
role_name VARCHAR(255)
createdAt DATETIME
updatedAt DATETIME
Indexes
lawyer_practices
id INT
lp_id CHAR(36)
practice_id INT
lawyer_id INT
createdAt DATETIME
updatedAt DATETIME
Indexes
loggedinusers
id INT
token VARCHAR(255)
user_id VARCHAR(255)
createdAt DATETIME
updatedAt DATETIME
loggedinusers_id INT
loggedinusers_lawyers_id INT
loggedinusers_lawyers_roles_id ...
loggedinusers_lawyers_reviews...
loggedinusers_lawyers_reviews...
loggedinusers_lawyers_cities_id ...
loggedinusers_lawyers_case_tra...
12 more...
Indexes
appointments
id INT
appoint_id CHAR(36)
client_id INT
lawyer_id INT
status VARCHAR(255)
reason TEXT
appointment_date DATETIME
created_at DATETIME
updated_at DATETIME
Indexes
users
id BIGINT
user_id CHAR(36)
full_names VARCHAR(255)
email_address VARCHAR(255)
password VARCHAR(255)
permissions VARCHAR(255)
last_login DATETIME
status TINYINT(1)
phone_number VARCHAR(255)
address_city INT
id_passport_number VARCHAR(255)
role_name VARCHAR(255)
created_at DATETIME
updated_at DATETIME
profile_image VARCHAR(255)
faqs_id INT
Indexes
connections
id INT
connection_id CHAR(36)
client_id INT
lawyer_id INT
status VARCHAR(255)
created_at DATETIME
updated_at DATETIME
Indexes
lawyers
id INT
lawyer_id CHAR(36)
user_id INT
law_firm VARCHAR(255)
lawfirm_city_id INT
lawfirm_license_number VA...
cooperation_status VARCHA...
created_at DATETIME
updated_at DATETIME
work_permit VARCHAR(255)
appendix_nr1 VARCHAR(255)
18 more...
Indexes
appointments_schedules
id INT
schedule_id CHAR(36)
appointment_date DATETIME
lawyer_id INT
created_at DATETIME
updated_at DATETIME
Indexes
faqs
id INT
question VARCHAR(255)
answer TEXT
created_at DATETIME
updated_at DATETIME
Indexes
reviews
id INT
review_id CHAR(36)
reviewed_by INT
review_to INT
review_title VARCHAR...
review_text TEXT
visibility VARCHAR(255)
2 more...
Indexes
cities
id INT
city_id CHAR(36)
name VARCHAR(255)
created_at DATETIME
updated_at DATETIME
Indexes
case_track...
id INT
tracker_id CHAR(36)
action_name VARCH...
action_desc TEXT
lawyer_id INT
case_id INT
4 more...
Indexes
law_practices
id INT
law_id CHAR(36)
name VARCHAR(255)
createdAt DATETIME
updatedAt DATETIME
Indexes
cases
id INT
case_id CHAR(36)
client_id INT
case_category_id INT
case_start_date DAT...
case_end_date DAT...
case_desc TEXT
case_title VARCHAR(...
case_status VARCHA...
law_practice_area_i...
lawyer_id INT
3 more...
Indexes
cases_has_case_trackers
cases_id INT
cases_case_categories_id INT
case_trackers_id INT
Indexes
case_trackers_has_cases
case_trackers_id INT
cases_id INT
Indexes
sequelizemeta
name VARCHAR(255)
Indexes
payment_requests
id INT
request_id CHAR(36)
amount INT
currency VARCHAR(255)
status VARCHAR(255)
description TEXT
payment_method VARCHAR(255)
payment_date DATETIME
case_id INT
client_id INT
lawyer_id INT
createdAt DATETIME
updatedAt DATETIME
Indexes
profiles
id INT
profile_id CHAR(36)
biograph TEXT
user_id INT
profile_picture VARCHAR(255)
education_university VARCHA...
degree_name VARCHAR(255)
created_at DATETIME
updated_at DATETIME
clients_id INT
clients_roles_id INT
9 more...
Indexes
chat_messages
id INT
message VARCHAR(255)
sender INT
reciever INT
created_at DATETIME
updated_at DATETIME
is_read VARCHAR(255)
sender_type VARCHAR(255)
Indexes
law_practices_key...
id INT
keyword_id CHAR(36)
name VARCHAR(255)
practice_name VARCHAR(255)
createdAt DATETIME
updatedAt DATETIME
Indexes
case_categories
id INT
category_id CHAR(36)
name VARCHAR(255)
createdAt DATETIME
updatedAt DATETIME
Indexes

5 Project creation documentation

In the development of the Legal Connect System, using tools such as Visual Studio, the chronological creation of the project can be summarized in the following steps:

5.1 Database Initialization and setup

The project commenced with requirements gathering to understand the legal system's needs, aligning the functionalities with the user's requirements.

WHAT: The goal was to initialize the database structure for Legal Connect System to store user data, case information, and other relevant data points securely and efficiently.

HOW: Leveraged Visual Studio's integrated SQL Server Object Explorer to establish a new SQL database, defining initial tables according to the predefined schema. Used SQL commands to create tables with primary keys, foreign keys, and indexes to ensure data integrality and optimized access. For example, the command `CREATE TABLE` was executed to set up the `Users` table with fields such as `Username`, `Password`, and relations to other tables.

5.2 Design and user interface.

The Legal Connect System, aimed at providing a robust platform for legal services, required a user interface that was both intuitive and professional. To achieve this, utilized Figma, a cloud-based design tool known for its collaborative features and efficiency in creating dynamic user interfaces. The choice of Figma was instrumental due to its real-time collaboration capabilities, allowing team members and stakeholders to provide immediate feedback which was crucial for the agile development process.

WHAT: Aimed to create a user-friendly and intuitive interface that allows users to navigate the system easily, manage their legal cases, and communicate effectively.

HOW: Implemented the design phase in Figma, starting with low-fidelity wireframes and advancing to high-fidelity prototypes. Utilizes Figma's collaboration features to iterate on the design with input from team members, resulting in a set of UI components and layouts that were both aesthetically pleasing and functional. For example, designed the login page to include form fields for username and password, as well as links for password recovery and new registration.

5.3 Implementation Authentication Flow

WHAT: Needed to ensure that users could securely log in and out of the Legal Connect System with their credentials being protected.

HOW: Developed a real-time chat application with ReactJS on the front end for an interactive user experience. The chat components were programmed to update immediately as messages were sent and received. On the back end, NodeJS and the Express framework were employed to set up RESTful APIs, while Socket.IO was integrated to manage websocket connections for real-time bidirectional event-based communication. For the database, structured SQL tables were created to hold message and chat session data. Sequelize, a NodeJS ORM compatible with SQL databases, was utilized to interact with the database, enabling seamless data handling and retrieval operations for storing messages and maintaining chat histories.

5.4 Establishing Client-Lawyer Communication Channel

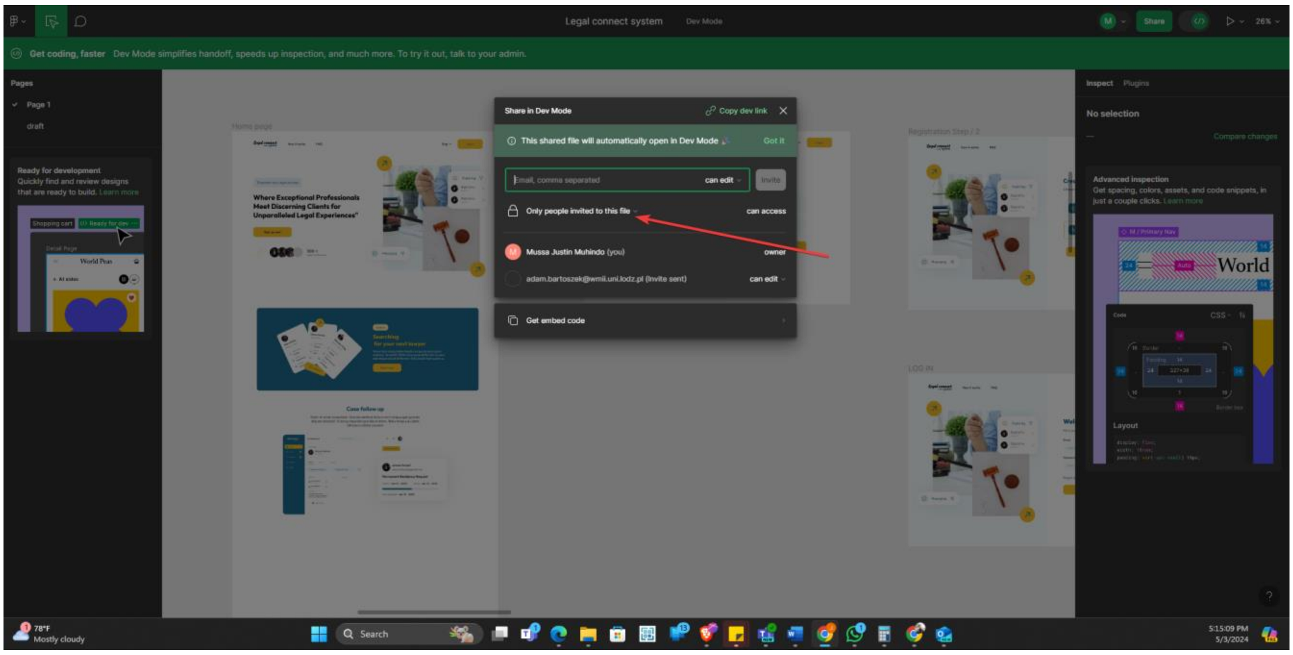
WHAT: The objective was to facilitate direct communication between clients and lawyers through an integrated messaging system within the Legal Connect System.

HOW: The system's architecture was designed using Figma design tool to create high-fidelity wireframes and interactive prototypes. This allowed for the visualization of the user interface and user experience workflow before development began. The use of Figma fostered collaboration between designers, developers, and stakeholders, ensuring that the system's interface was both user-friendly and aligned with the system's intended functionality, including the drafting of the initial database schema. Entity-Relationship Diagrams (ERDs) were used to visualize the data structure and relationships.

5.5 Designing the Legal Connect System Using Figma

5.5.1 Introduction to Figma and its Impact on the project

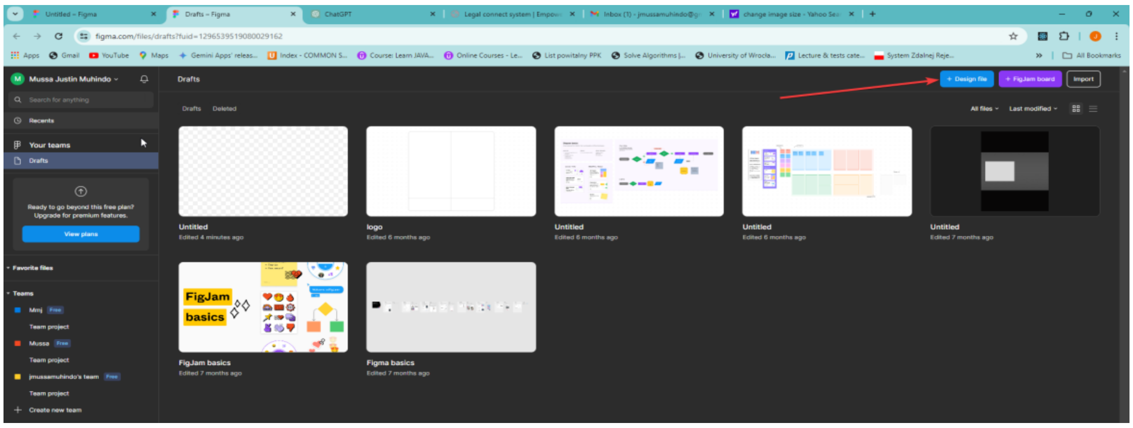
The Legal Connect System, aimed at providing a robust platform for legal services, required a user interface that was both intuitive and professional. To achieve this, used of Figma, a cloud-based design tool known for its collaborative features and efficiency in creating dynamic user interfaces. The choice of Figma was instrumental due to its real-time collaboration capabilities, allowing team members and stakeholders to provide immediate feedback which was crucial for the agile development process. The screenshot provided below shows how to invite people and collaborate.


Legal connect system
Dev Mode
Share
Get coding, faster Dev Mode simplifies handoff, speeds up inspection, and much more. To try it out, talk to your admin.
Pages
Page 1
Share in Dev Mode
Copy dev link
This shared file will automatically open in Dev Mode
Got it
Email, comma separated
can edit
Invite
Only people invited to this file
can access
Mussa Justin Muhindo (you)
owner
adam.bartoszek@wmii.uni.lodz.pl (invite sent)
can edit
Get embed code
Inspect
Plugins
No selection
Advanced inspection
Get spacing, colors, assets, and code snippets, in just a couple clicks. Learn more
78°F
Mostly cloudy
Search
5:15:09 PM
5/3/2024

5.5.2 Setting Up the Design Environment

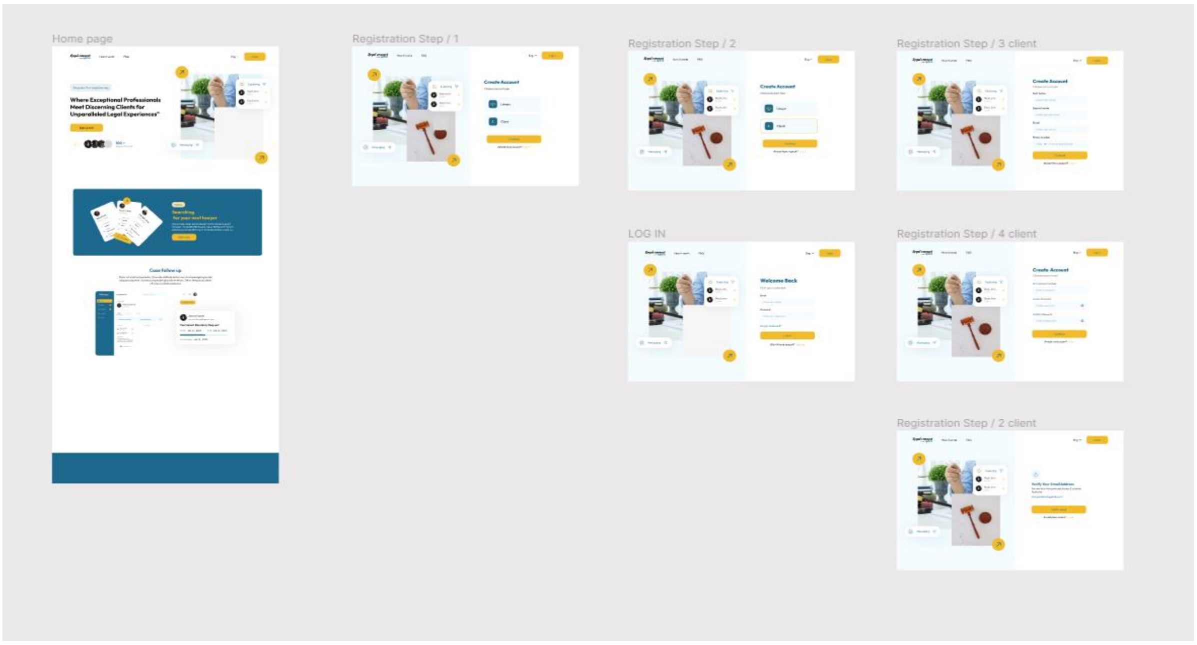
Initially, the project setup in Figma involved creating a new file designated for the Legal Connect System and establishing a consistent design system. This system included defined color palettes, typographies, and reusable components such as buttons, input fields, and modal windows. The use of these predefined components ensured consistency and coherence throughout the application, which facilitated a smoother transition from design to development.

The screenshot below shows how to start a new file, the red arow mention the first step to create the new file design.



5.5.3 Wireframing Phase

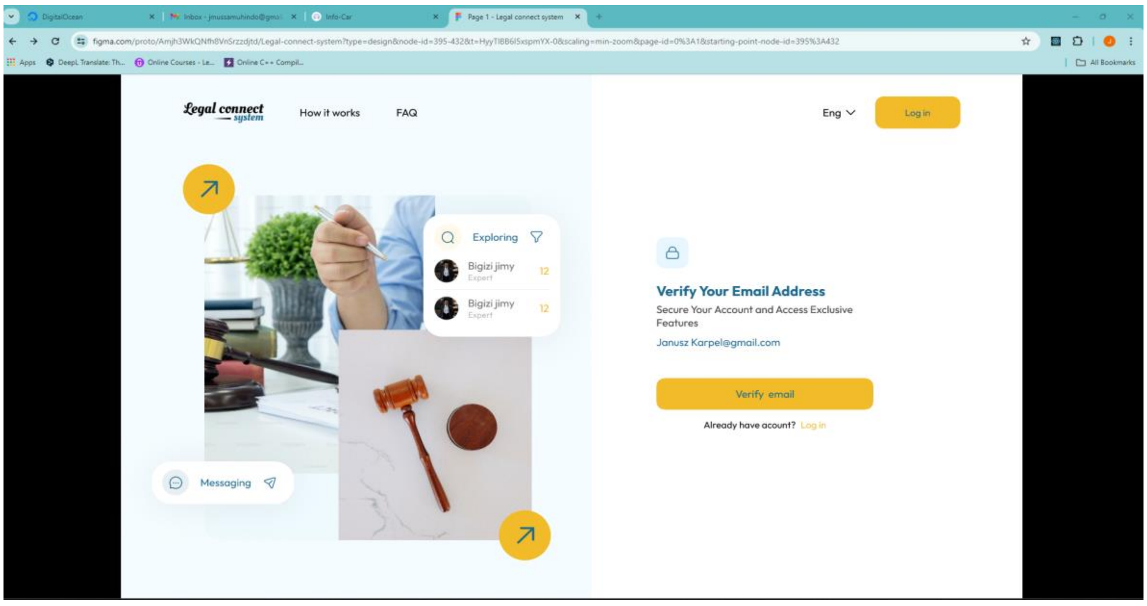
The design process began with the creation of wireframes for the core screens of the Legal Connect System. These wireframes served as a blueprint for the layout and interaction flow within the application. During this phase, we focused on the arrangement of elements to optimize user experience, particularly how users would navigate through legal documents, connect with Lawyers, and access various legal services. Each wireframe was iteratively refined based on feedback from legal professionals and potential users to ensure it met practical needs and usability standards.



Home page
Registration Step / 1
Registration Step / 2
Registration Step / 3 client
LOG IN
Registration Step / 4 client
Registration Step / 2 client

5.5.4 Developing High-Fidelity Prototypes

Once the wireframes included detailed visual were approved, we transitioned to developing high-fidelity prototypes that and interaction designs. These prototypes were fully interactive, simulating the user interface and experiences of the final product. Key screens were designed to demonstrate the workflow of booking a consultation, managing legal documents, and receiving notifications. The high-fidelity prototypes were pivotal for visualizing the product and conducting user testing. The screenshot below shows the view of prototype, The red arrow shows the flows.



5.5.5 User Testing and Feedback Integration

User testing sessions were conducted using the high-fidelity prototypes created in Figma. Participants included legal professionals and potential users who were asked to complete specific tasks while observers recorded their interactions. The feedback obtained was invaluable, leading to multiple iterations that focused on enhancing usability and functionality. Common adjustments included simplifying the navigation, increasing the legibility of legal texts, and improving the responsiveness of interactive elements. To access the legal connect prototype open the following link: https://www.figma.com/proto/Amjh3WkQNfh8VnSrzzdjtd/Legal-connect-system?type=design&node-id=395-432&t=HyyTIBB6I5xspmYX-0&scaling=min-zoom&page-id=0%3A1&starting-point-node-id=395%3A432&show-proto-sidebar=1.

5.6 Visual Studio

Development environment setup was done using Visual Studio, along with the configuration of the necessary development and database management tools.

5.6.1 Database Creation in VS Code

The database was implemented in SQL, with tables and relationships established as per the ERD. Sequelize, a Node.js ORM, was utilized for database operations, ensuring smooth migration and schema evolution.

5.7 Development Phases

5.7.1 Backend Development

The server, APIs, and database connectivity were developed, ensuring robust data handling and security measures.

5.7.2 Frontend Development

The user interface was built, focusing on usability and accessibility allowing for intuitive navigation and interaction.

6 User documentation

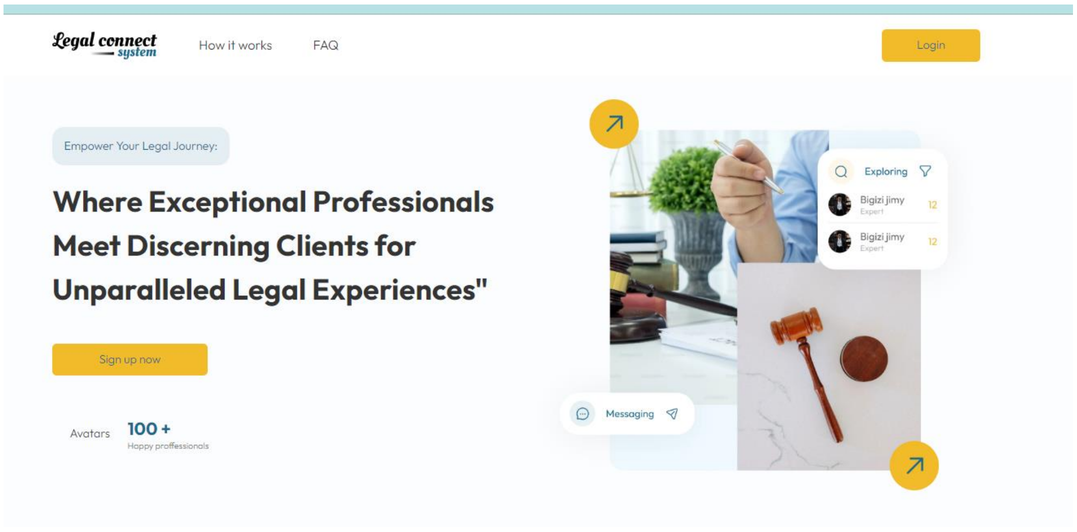
Legal Connect System is a web-based platform designed to streamline legal professional-client connections. It offers intuitive functionality and a simple interface. The guide provides detailed descriptions of features and clear screenshots to ensure an efficient and pleasant virtual experience. It is suitable for those seeking legal advice, managing legal documents, or connecting with potential clients.

To access the Legal Connect System's features, users must register and follow specific steps to create an account and start their journey towards finding legal professionals or presenting their expertise.

6.1 Client

6.1.1 Registration

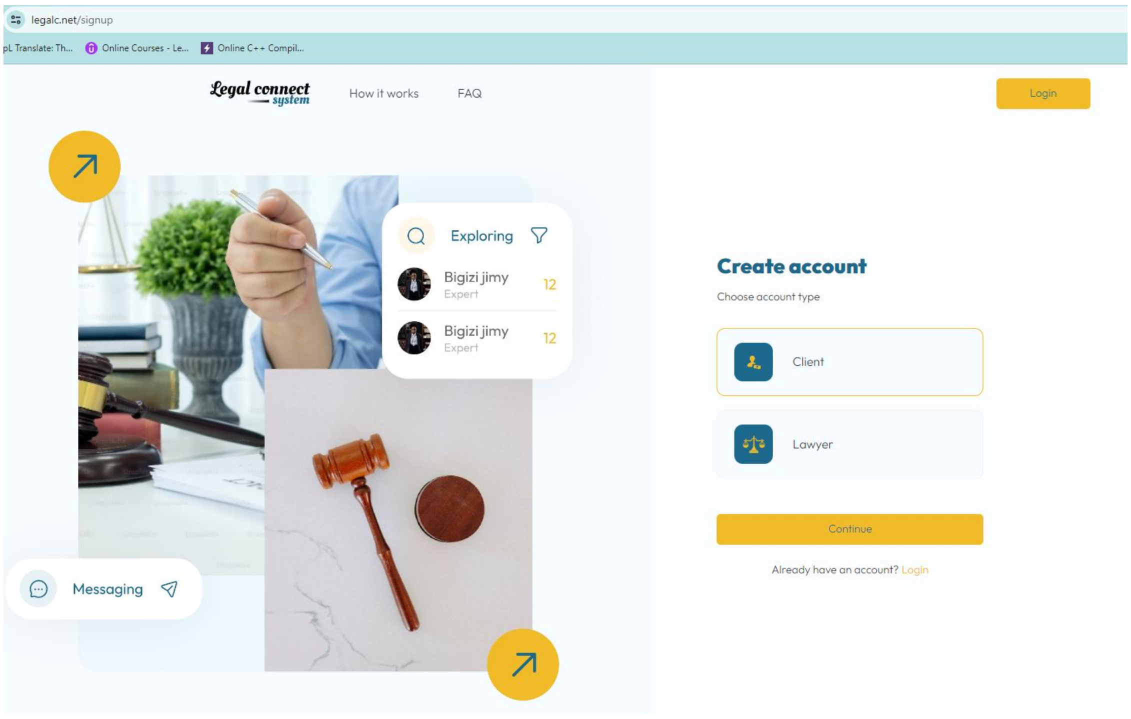
To access the Sign-Up Page, click the 'Sign up now' button on the homepage, located under the banner promoting exceptional professionals and exceptional legal experiences.



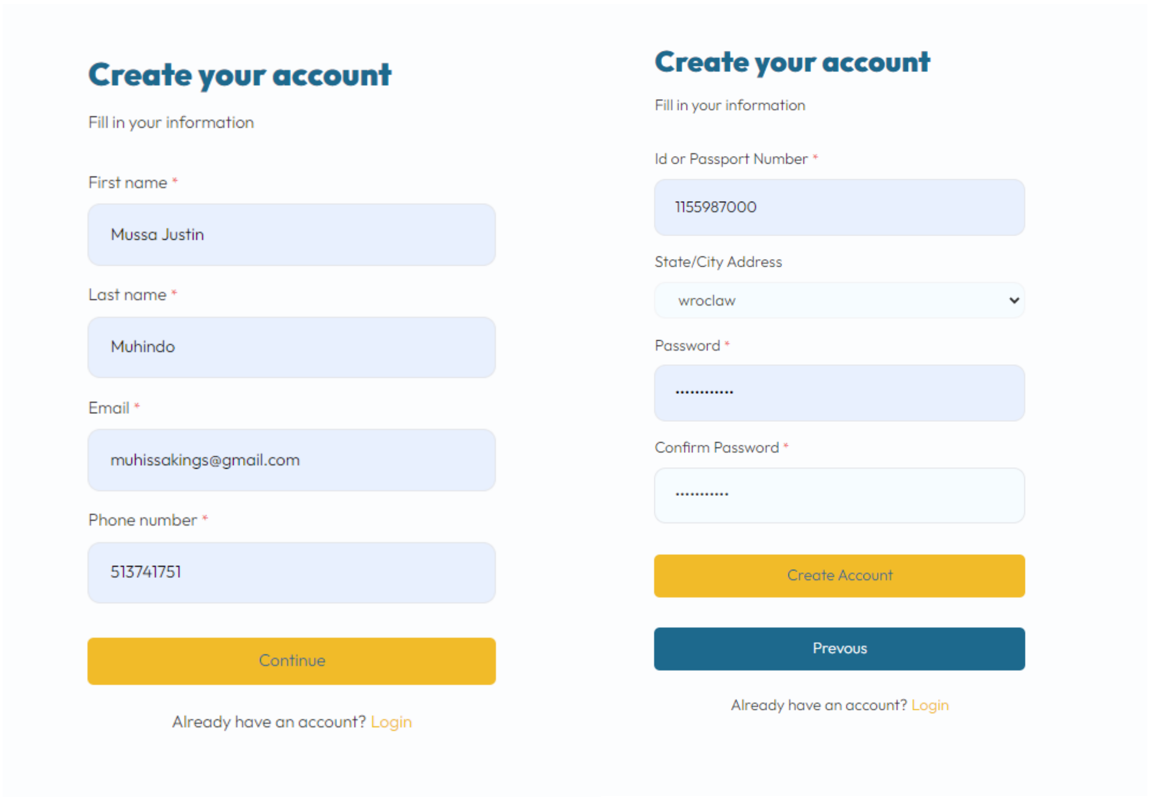
After initiating the signup process by clicking the **'Sign up now'** button on the homepage, you will be prompted to specify the type of account you wish to create. Legal Connect System offers tailored experiences for both clients seeking legal services and lawyers offering their expertise. Here's how to proceed:

6.1.2 Selecting account type

Client Account: If you are looking to find a legal professional, click on the 'Client' button. This option is for users who require legal services and wish to connect with lawyers through our platform.



After selecting the 'Client' account type, you will be brought to the "Create your account" page. Here you will provide your basic personal information to set up your new account. Below are the steps and fields you will encounter:



Create your account

Fill in your information

First name *	Mussa Justin
Last name *	Muhindo
Email *	muhissakings@gmail.com
Phone number *	513741751

Continue

Already have an account? Login

Create your account

Fill in your information

Id or Passport Number *	1155987000
State/City Address	wroclaw
Password *	············
Confirm Password *	···········

Create Account

Prevous

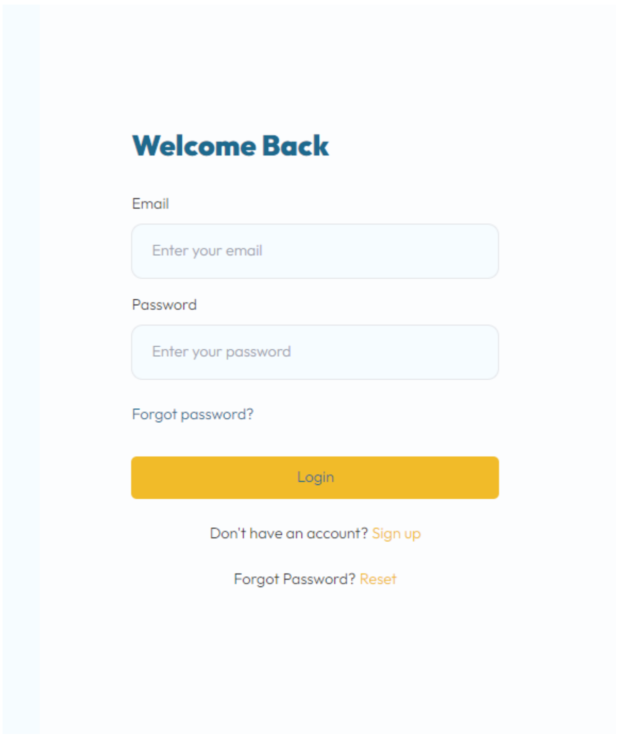
Already have an account? Login

Personal Information: Users begin by entering their personal information, including first name, last name, email, and phone number. Upon completion, the '**Continue**' button is clicked to proceed to the next step.

Identification and Security: The second step requires users to input additional identification details such as an ID or passport number. Users must also provide their location by selecting their city address from a dropdown menu. For security, users create a password and then confirm it in the next field to ensure accuracy.

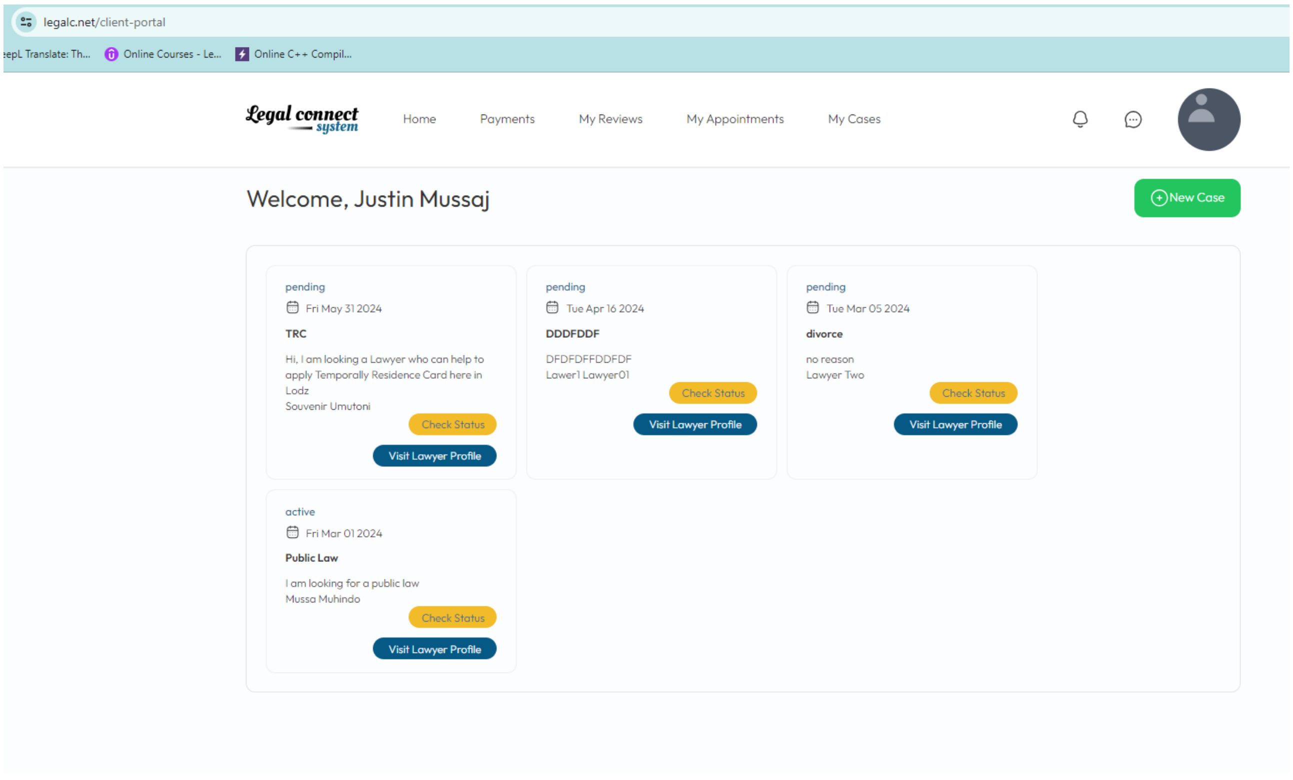
Account Creation Confirmation: The final step is to review all provided information. Users finalize the process by clicking **'Create Account'** to establish their new account. If they need to revise any information, they can click **'Previous'** to return to earlier steps.

6.1.3 Login



Login: The login interface displayed above is utilized by both clients and lawyers, without any differentiation between the two user groups. Both clients and lawyers use the same form to access the site, indicating that there is no distinction in the login process for these two categories of users.

6.1.3.1 Home Page Client



legalc.net/client-portal

eepL Translate: Th... Online Courses - Le... Online C++ Compil...

Legal connect system Home Payments My Reviews My Appointments My Cases

Welcome, Justin Mussaj

New Case

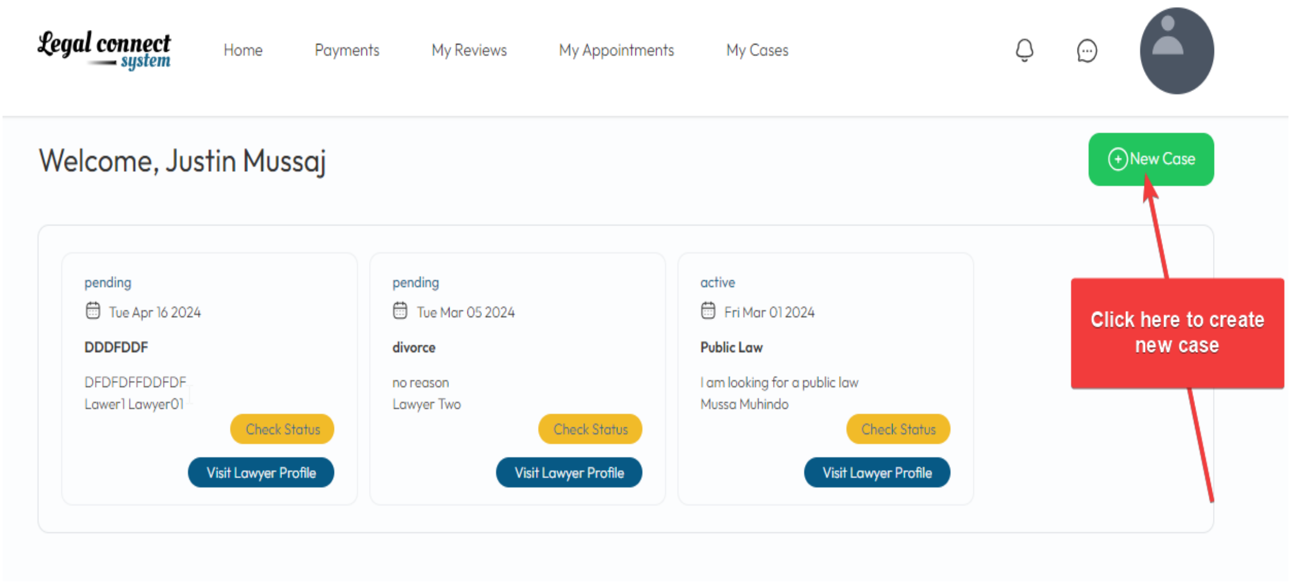
pending
Fri May 31 2024
TRC
Hi, I am looking a Lawyer who can help to apply Temporally Residence Card here in Lodz
Souvenir Umutoni
Check Status
Visit Lawyer Profile

pending
Tue Apr 16 2024
DDDFDDF
DFDFDFFDDFDF
Lawer1 Lawyer01
Check Status
Visit Lawyer Profile

pending
Tue Mar 05 2024
divorce
no reason
Lawyer Two
Check Status
Visit Lawyer Profile

active
Fri Mar 01 2024
Public Law
I am looking for a public law
Mussa Muhindo
Check Status
Visit Lawyer Profile

6.1.3.2 Create Case's Client



Legal connect system Home Payments My Reviews My Appointments My Cases

Welcome, Justin Mussaj

New Case

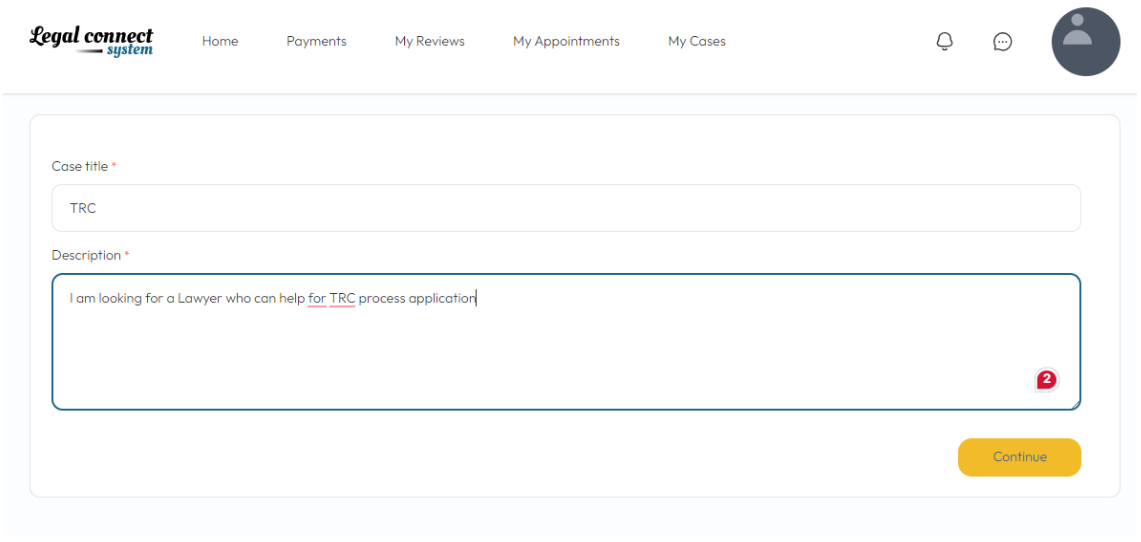
Click here to create new case

pending
Tue Apr 16 2024
DDDFDDF
DFDFDFFDDFDF
Lawer1 Lawyer01
Check Status
Visit Lawyer Profile

pending
Tue Mar 05 2024
divorce
no reason
Lawyer Two
Check Status
Visit Lawyer Profile

active
Fri Mar 01 2024
Public Law
I am looking for a public law
Mussa Muhindo
Check Status
Visit Lawyer Profile

On the right side of the screen, there's a prominent green button labeled "New Case" with an arrow pointing to it, indicating that it's likely an important or commonly used feature where a user can initiate a new case within the system.



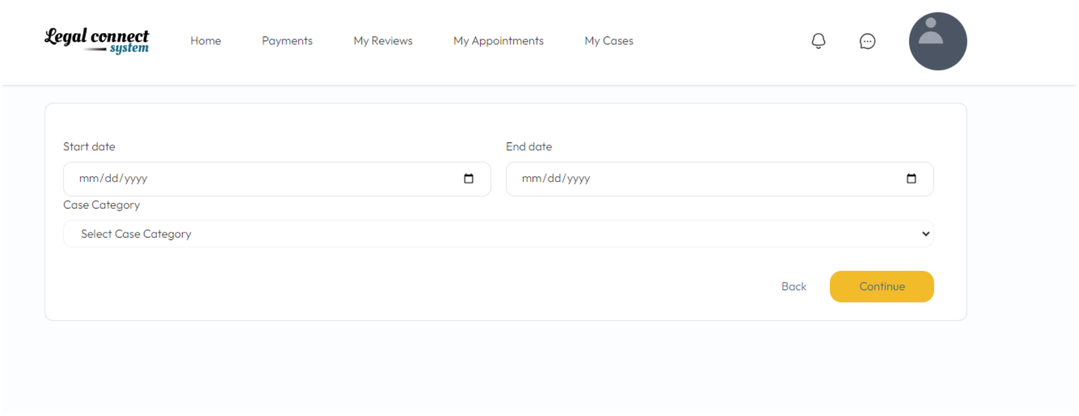
Case title *

TRC

Description *

I am looking for a Lawyer who can help for TRC process application

This screenshot depicts a form within the "Legal Connect System" web interface where a user can submit a new case. The form includes fields for the "Case title" and "Description" of the case. The user has filled in the case title with "TRC" and provided a brief introduction in the description box. seeking assistance with TRC process. At the bottom of the form, there is a "Continue" button, likely to submit the information or to proceed to the next step in the case submission process. The form fields marked with an asterisk (*) indicate that they are required. The layout retains the same top navigation bar as the previous screenshot, maintaining consistency across the platform.



Start date

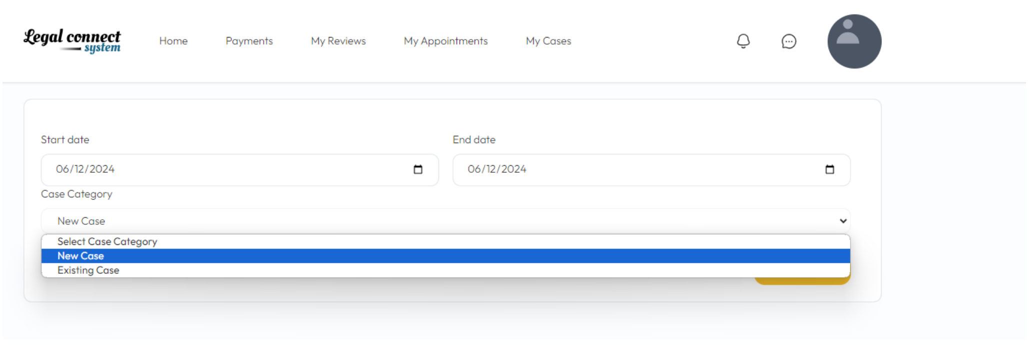
mm/dd/yyyy

End date

mm/dd/yyyy

Case Category

Select Case Category



Start date

06/12/2024

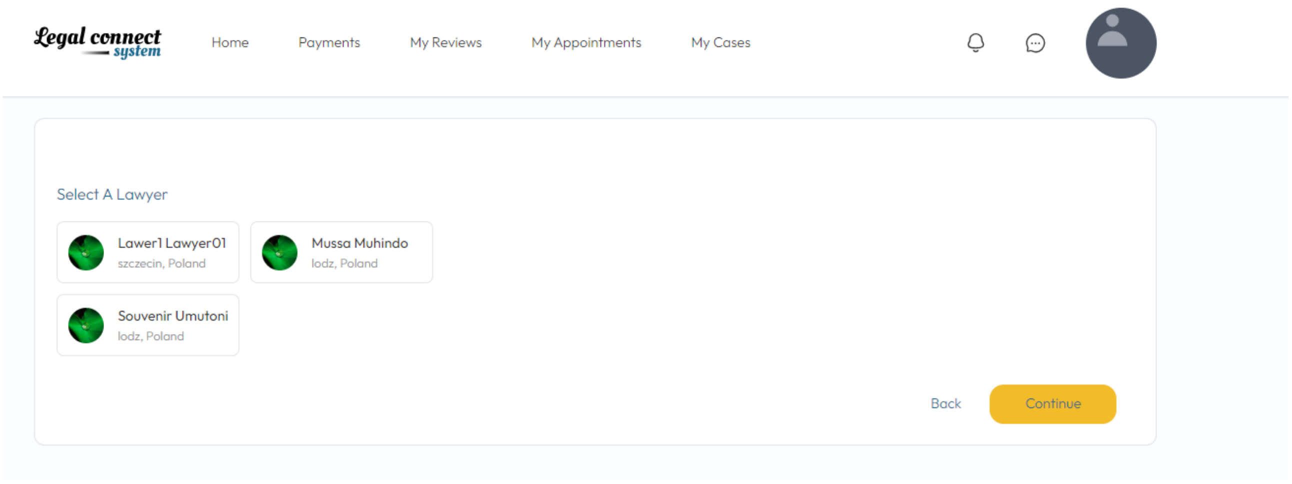
End date

06/12/2024

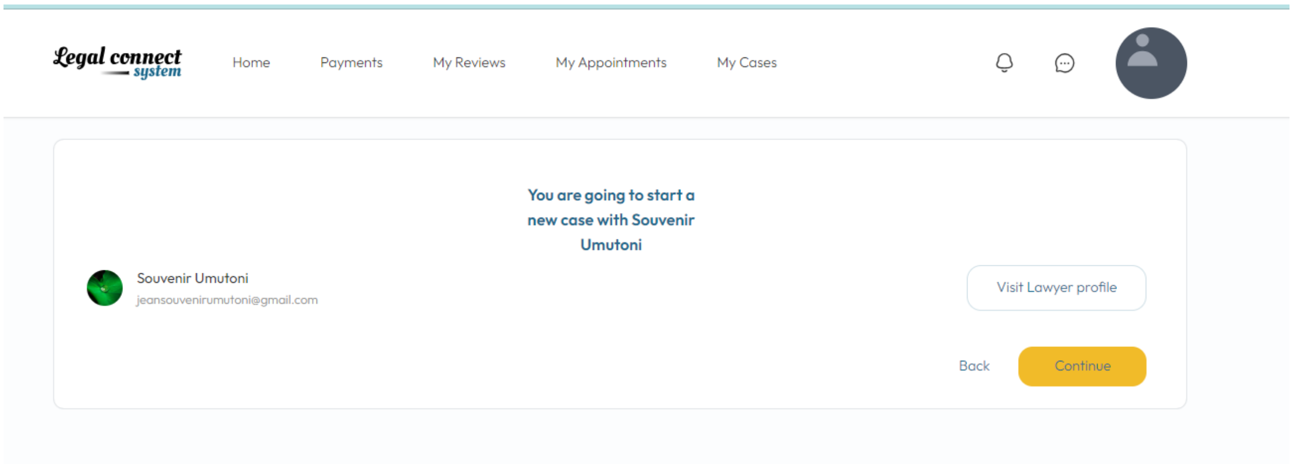
Case Category

New Case

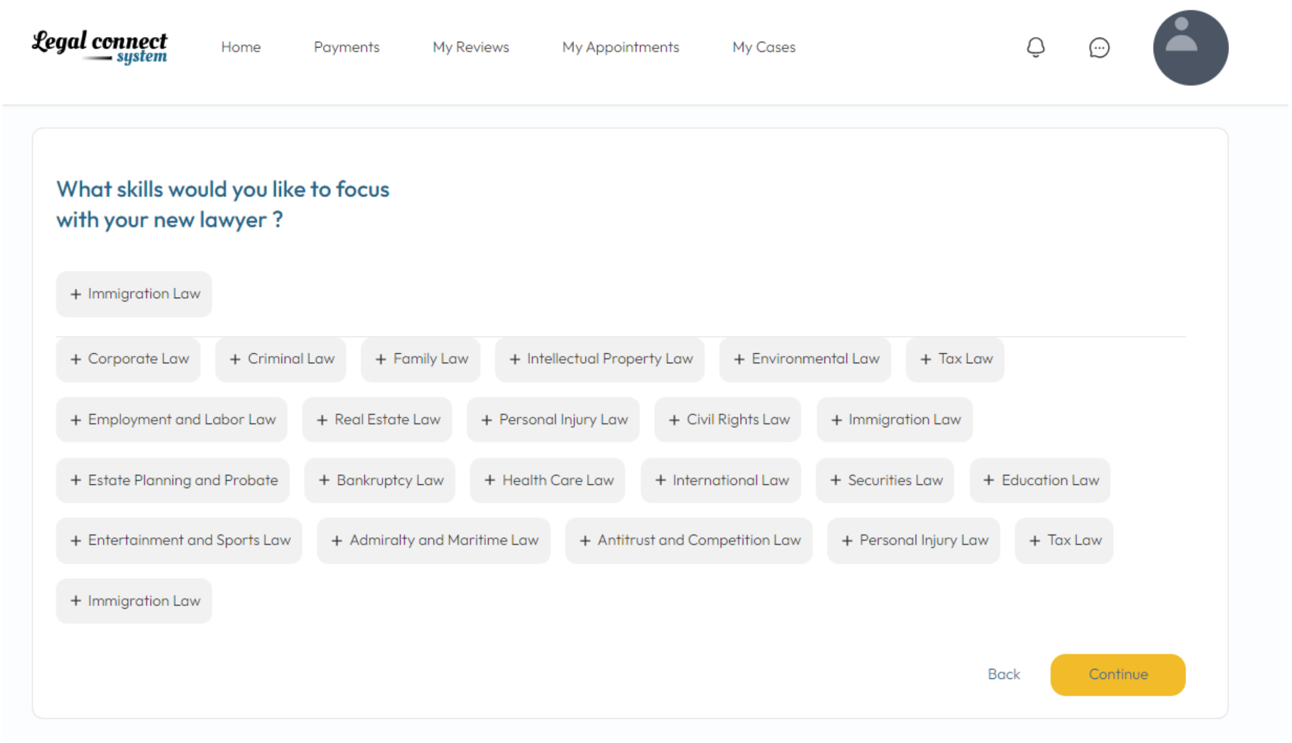
There is a dropdown menu for "Case Category" which is currently set to "New Case." This could be a placeholder, or the actual category selected by the user. The terms used suggest that the system may handle different categories or types of legal cases like Currently.



The screenshot above shows a user interface where the user is prompted to select a lawyer from a list of available options, with names and locations displayed for each lawyer system fetched them accordingly to the Algorithm set depending on the description provided.



The above screenshot displays the initial step where a user is prompted to start a new case with a lawyer named "Souvenir Umutoni" along with a button to 'Continue'.



What skills would you like to focus with your new lawyer ?

+ Immigration Law

+ Corporate Law + Criminal Law + Family Law + Intellectual Property Law + Environmental Law + Tax Law

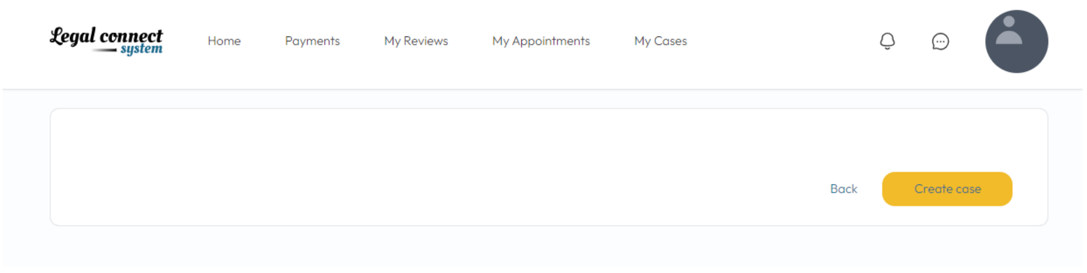
+ Employment and Labor Law + Real Estate Law + Personal Injury Law + Civil Rights Law + Immigration Law

+ Estate Planning and Probate + Bankruptcy Law + Health Care Law + International Law + Securities Law + Education Law

+ Entertainment and Sports Law + Admiralty and Maritime Law + Antitrust and Competition Law + Personal Injury Law + Tax Law

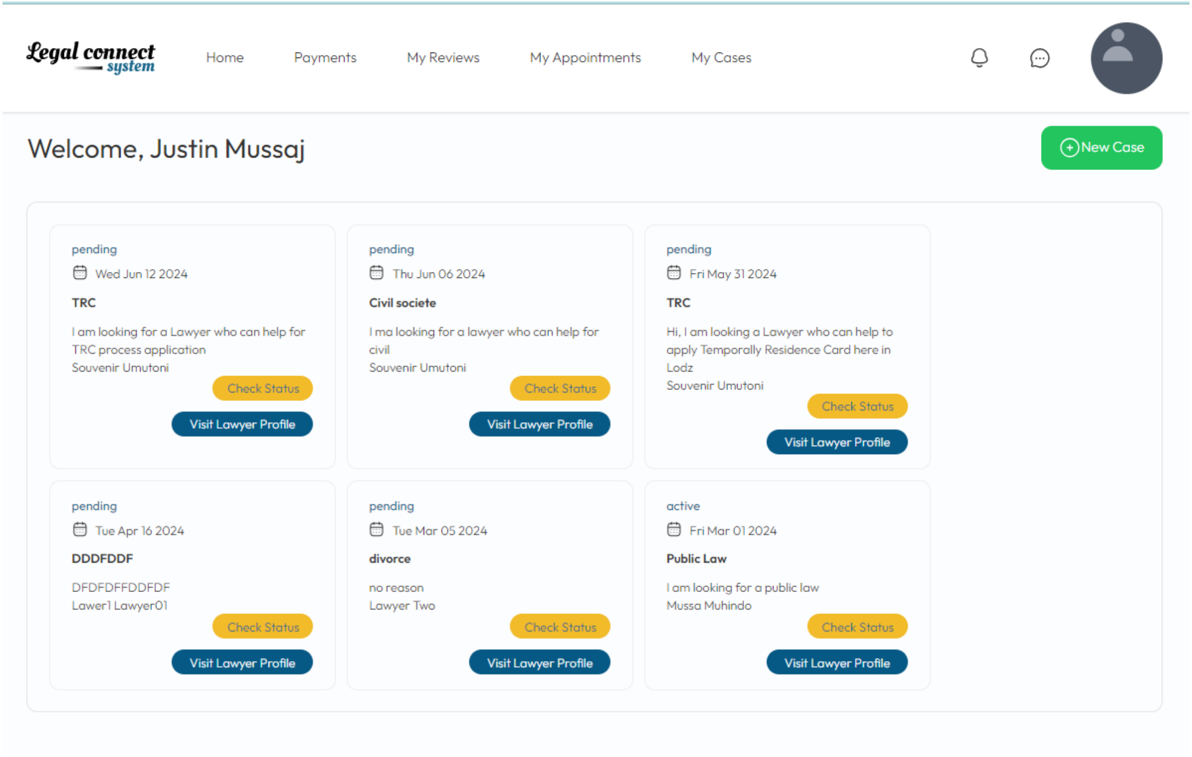
+ Immigration Law

The above image shows a subsequent step where the user can select the legal skills or areas of law they are seeking in a lawyer, with various fields such as "Immigration Law", available for selection, and another button to 'Continue' to the next step.



The screenshot above, showing a simplified interface with a "Create case" button highlighted by a yellow color, indicates the action to be taken to proceed. The user is at the final step of the case creation process, with the option to either go 'Back' or to 'Create case'. For this case the user will create a case.

6.1.3.3 Visit Lawyer's Profile and connect.


Legal connect system

Home · Payments · My Reviews · My Appointments · My Cases

Welcome, Justin Mussaj

New Case

pending
Wed Jun 12 2024
TRC
I am looking for a Lawyer who can help for TRC process application
Souvenir Umutoni
Check Status
Visit Lawyer Profile

pending
Thu Jun 06 2024
Civil societe
I ma looking for a lawyer who can help for civil
Souvenir Umutoni
Check Status
Visit Lawyer Profile

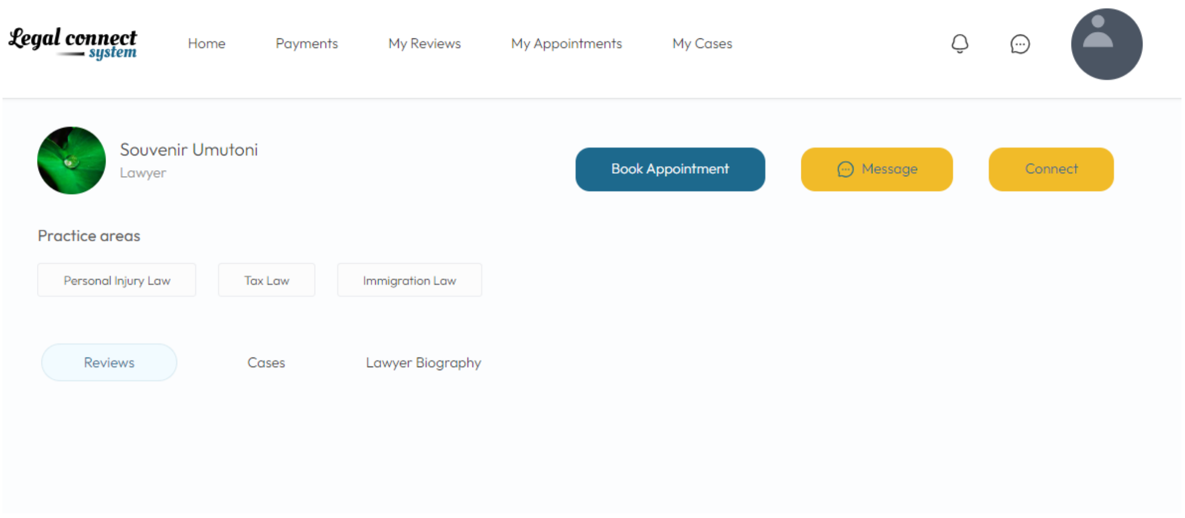
pending
Fri May 31 2024
TRC
Hi, I am looking a Lawyer who can help to apply Temporally Residence Card here in Lodz
Souvenir Umutoni
Check Status
Visit Lawyer Profile

pending
Tue Apr 16 2024
DDDFDDF
DFDFDFFDDFDF
Lawer1 Lawyer01
Check Status
Visit Lawyer Profile

pending
Tue Mar 05 2024
divorce
no reason
Lawyer Two
Check Status
Visit Lawyer Profile

active
Fri Mar 01 2024
Public Law
I am looking for a public law
Mussa Muhindo
Check Status
Visit Lawyer Profile

The screenshot above shows a personalized welcome message for a user on the "Legal Connect System" website. It displays a summary of pending cases with the dates a brief message from users to a lawyer and options to "Check Status" or "Visit Lawyer Profile. Additionally, there is a button for "New Case" in the top right corner, suggesting that the user can initiate a new case if needed again. Users will visit Lawyer Profile.


Legal connect system

Home · Payments · My Reviews · My Appointments · My Cases

Souvenir Umutoni
Lawyer

Book Appointment · Message · Connect

Practice areas

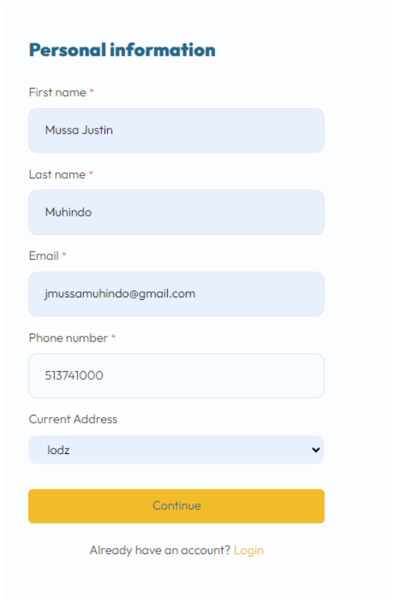
Personal Injury Law · Tax Law · Immigration Law

Reviews · Cases · Lawyer Biography

The screenshot features a lawyer's profile on the "Legal Connect System," with the highlighted 'Connect' button. This functionality implies that clients must send a connection request to the lawyer, and only after its acceptance can they proceed to book an appointment or engage in messaging, reflecting a two-step engagement process designed to manage interactions on the platform.

6.2 Lawyer

6.2.1 Registration



Personal information

First name *

Musa Justin

Last name *

Muhindo

Email *

jmussamuhindo@gmail.com

Phone number *

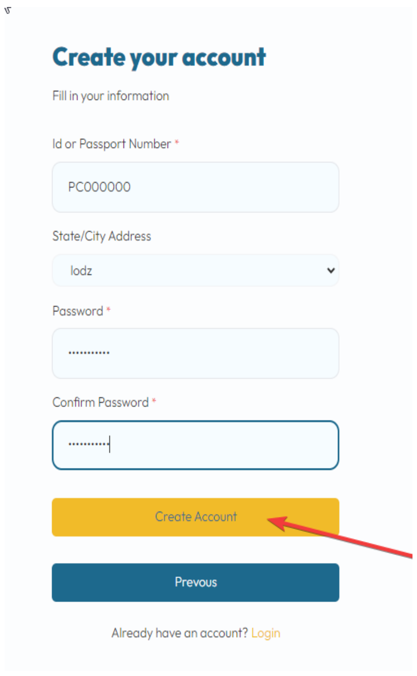
513741000

Current Address

lodz

Continue

Already have an account? Login



Create your account

Fill in your information

Id or Passport Number *

PC000000

State/City Address

lodz

Password *

············

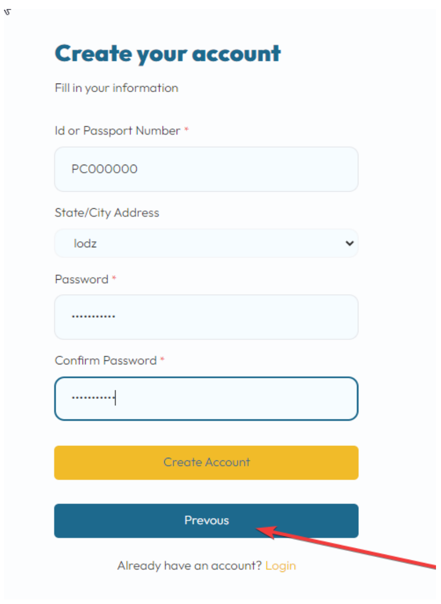
Confirm Password *

············

Create Account

Prevous

Already have an account? Login



Create your account

Fill in your information

Id or Passport Number *

PC000000

State/City Address

lodz

Password *

············

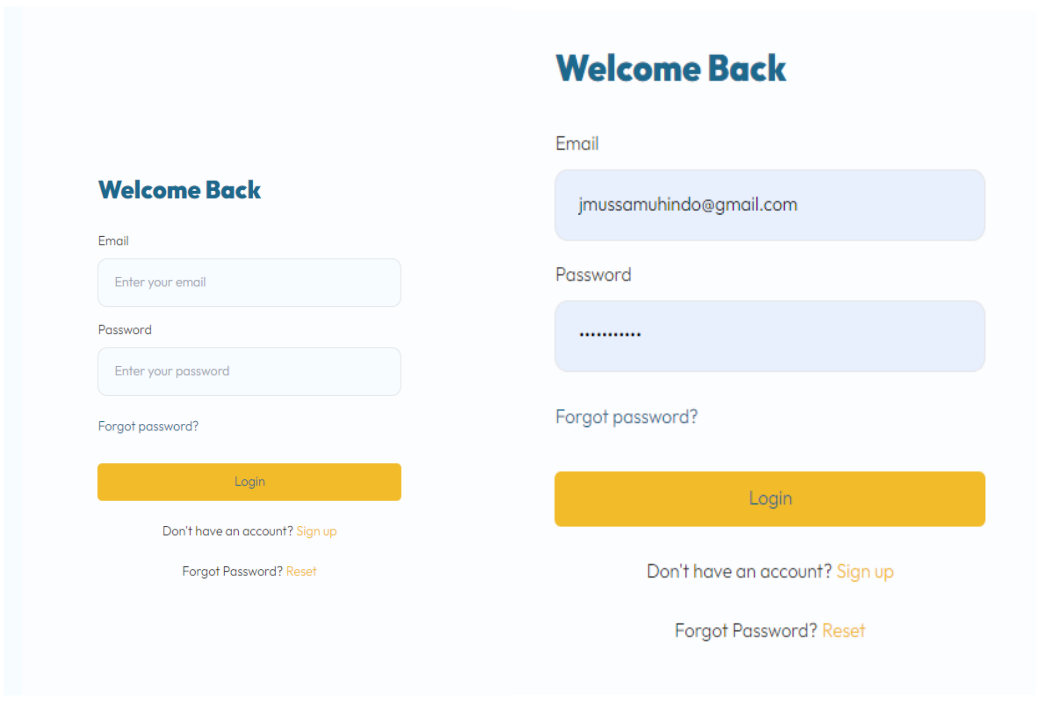
Confirm Password *

············

Create Account

Prevous

Already have an account? Login



Welcome Back

Email

Enter your email

Password

Enter your password

Forgot password?

Login

Don't have an account? Sign up

Forgot Password? Reset

Welcome Back

Email

jmussamuhindo@gmail.com

Password

············

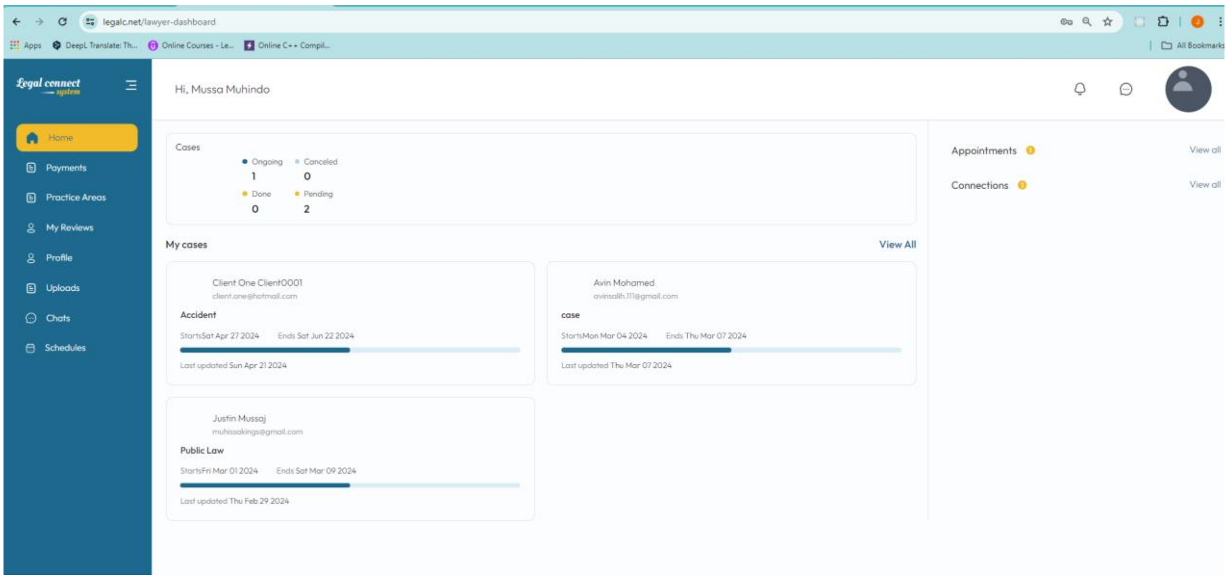
Forgot password?

Login

Don't have an account? Sign up

Forgot Password? Reset

6.2.1.1 Home Page Lawyer



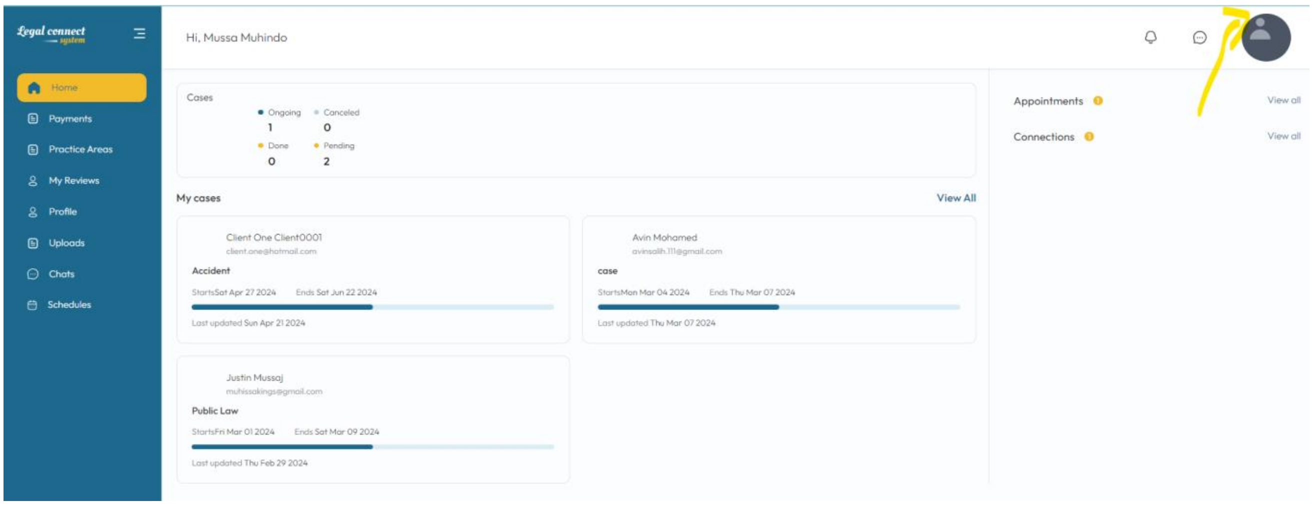
Client One Client0001
client.one@hotmail.com
Accident
StartsSat Apr 27 2024 Ends Sat Jun 22 2024
Last updated Sun Apr 21 2024

Avin Mohamed
avinsalih.111@gmail.com
case
StartsMon Mar 04 2024 Ends Thu Mar 07 2024
Last updated Thu Mar 07 2024

Justin Mussaj
muhissakings@gmail.com
Public Law
StartsFri Mar 01 2024 Ends Sat Mar 09 2024
Last updated Thu Feb 29 2024

Navigation Menu: The main navigation menu includes several tabs—'Home', 'Payments', 'My Reviews', 'My Appointments', and 'My Cases'. These represent different sections of the site where users can manage aspects of their legal journey, from making payments to reviewing past service engagements.

6.2.1.2 Profile

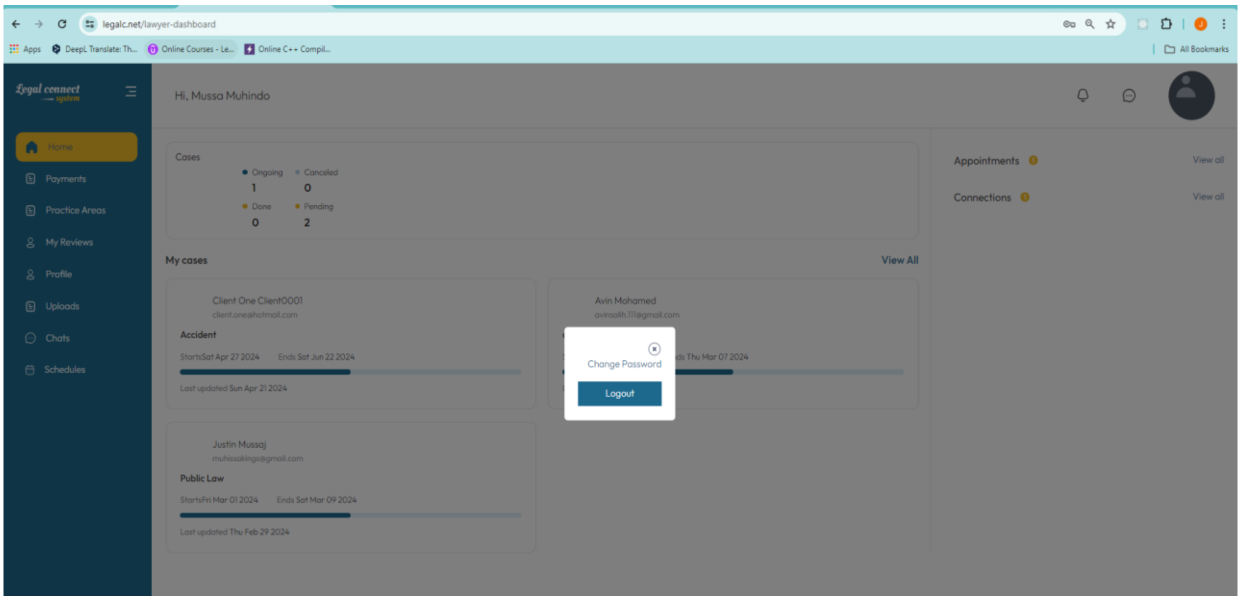


Client One Client0001
client.one@hotmail.com
Accident
StartsSat Apr 27 2024 Ends Sat Jun 22 2024
Last updated Sun Apr 21 2024

Avin Mohamed
avinsalih.111@gmail.com
case
StartsMon Mar 04 2024 Ends Thu Mar 07 2024
Last updated Thu Mar 07 2024

Justin Mussaj
muhissakings@gmail.com
Public Law
StartsFri Mar 01 2024 Ends Sat Mar 09 2024
Last updated Thu Feb 29 2024

Profile Access: The above screenshot shows the user's dashboard with a navigation bar at the top and a profile icon on the right. When the user clicks on their profile photo, a dropdown menu is revealed.



6.2.1.3 Logout

Logout: This is a security feature that lets users sign out of their account to prevent unauthorized access, especially when using shared or public computers.

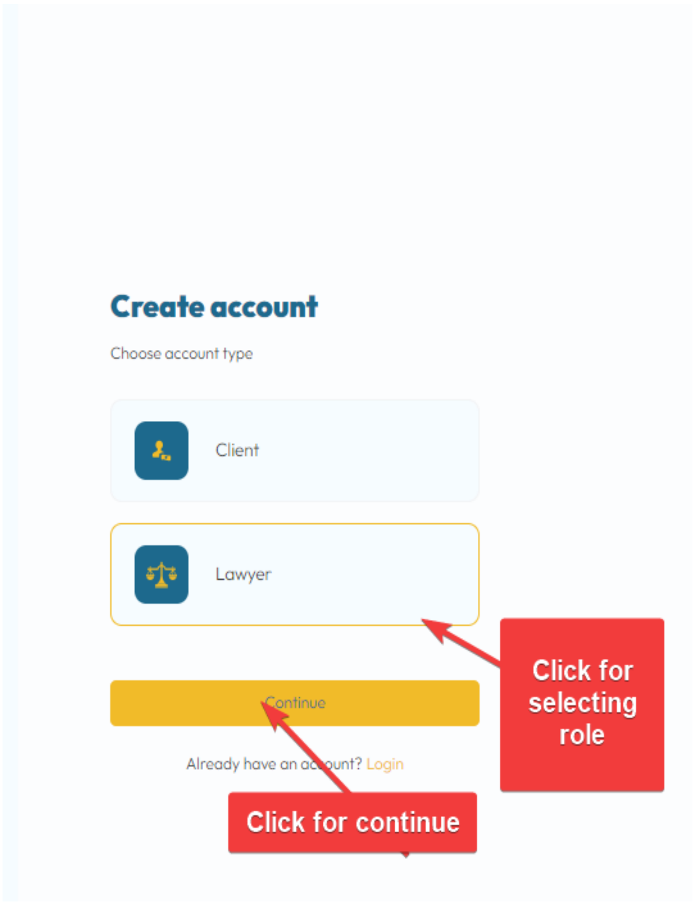
6.2.1.4 Edit Profile

Edit Profile: This allows the user to update their personal and professional information, such as profile picture and change password.

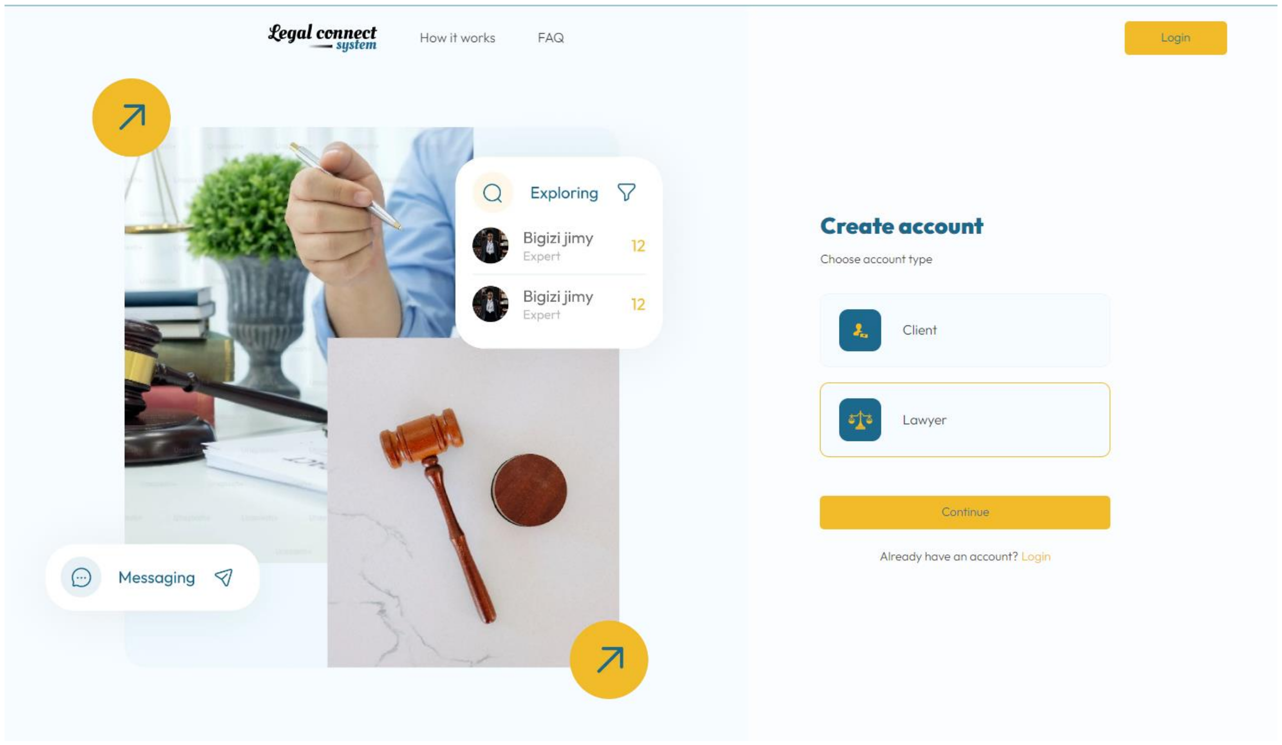
6.3 Lawyer

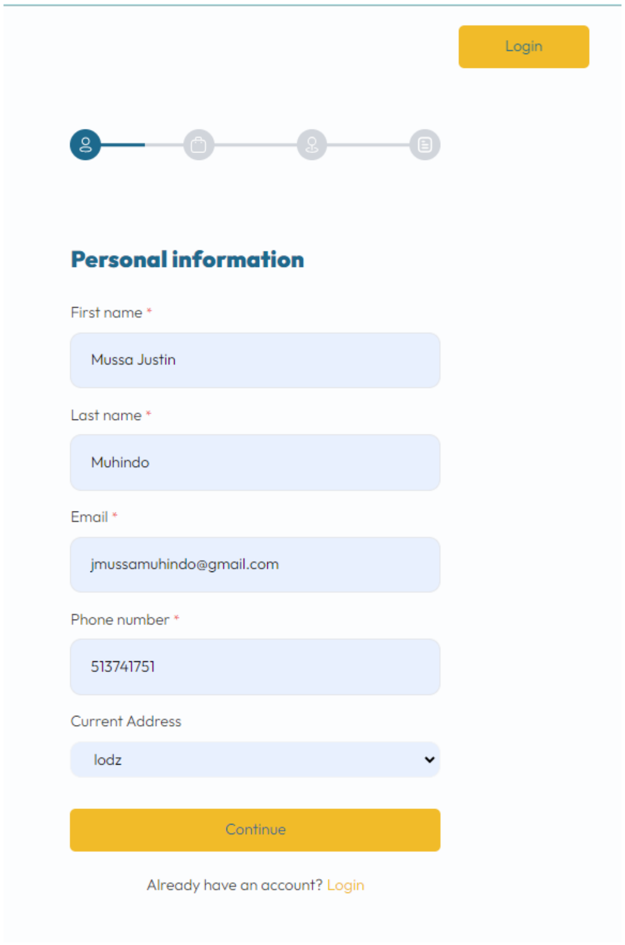
6.3.1 Registration

Lawyer Account: If you are a legal professional looking to offer your services, click on the 'Lawyer' button. This will allow you to create a profile that showcases your expertise to potential clients.



After selecting the 'Lawyer' account type, you will be brought to the "Create your account" page. Here you will provide your basic personal information to set up your new account. Below are the steps and fields you will encounter:




Login

Personal information

First name *

Mussa Justin

Last name *

Muhindo

Email *

jmussamuhindo@gmail.com

Phone number *

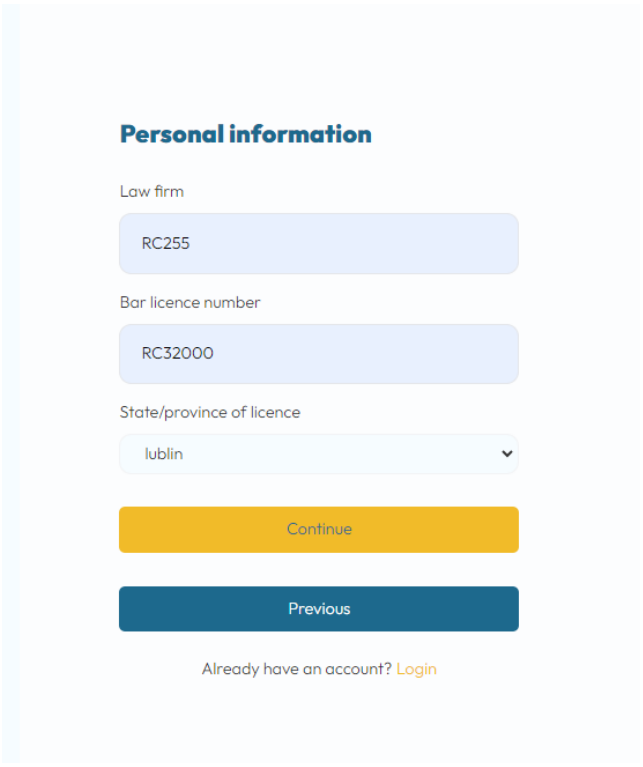
513741751

Current Address

lodz

Continue

Already have an account? Login


Personal information

Law firm

RC255

Bar licence number

RC32000

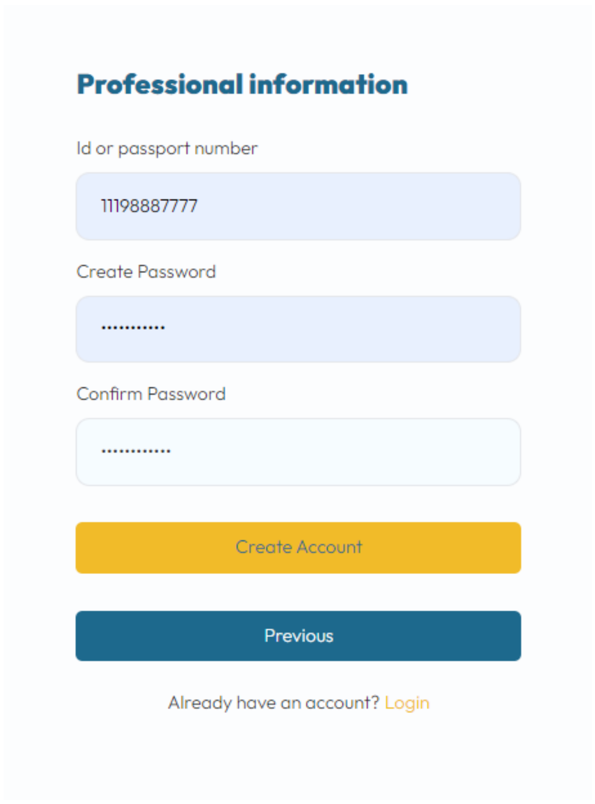
State/province of licence

lublin

Continue

Previous

Already have an account? Login


Professional information

Id or passport number

11198887777

Create Password

•••••••••••

Confirm Password

••••••••••••

Create Account

Previous

Already have an account? Login

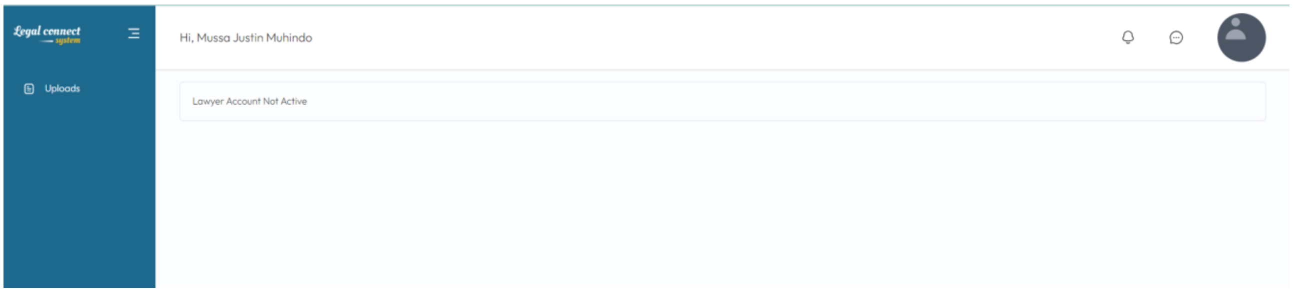
The provided screenshot above illustrates a three-step online registration process for lawyers on a legal services platform. The first image depicts the initial step where personal details such as first name, last name, email, and phone number are entered. The second image shows filled-

out fields with sample data, indicating progress. The third image represents an advanced stage in the process, where professional information specific to the legal profession—such as law firm name, bar license number, and the state or province of license—is required. Navigation options include 'Continue' to proceed, 'Previous' to go back, and 'Login' for existing account holders, illustrating a user-friendly, step-by-step registration for legal professionals.
The screenshots above depict the final step of a lawyer's online registration process on a legal services platform, where the professional is required to input their ID or passport number and set a password. A "Create Account" button is highlighted, which will likely submit the information for account creation. Adjacent to the form is an email confirmation from the legal platform, indicating that the newly created account will be active after being checked and approved by the platform's management team within a specified timeframe. This stage ensures the security and verification of the professional's credentials before account activation.

6.3.2 Login

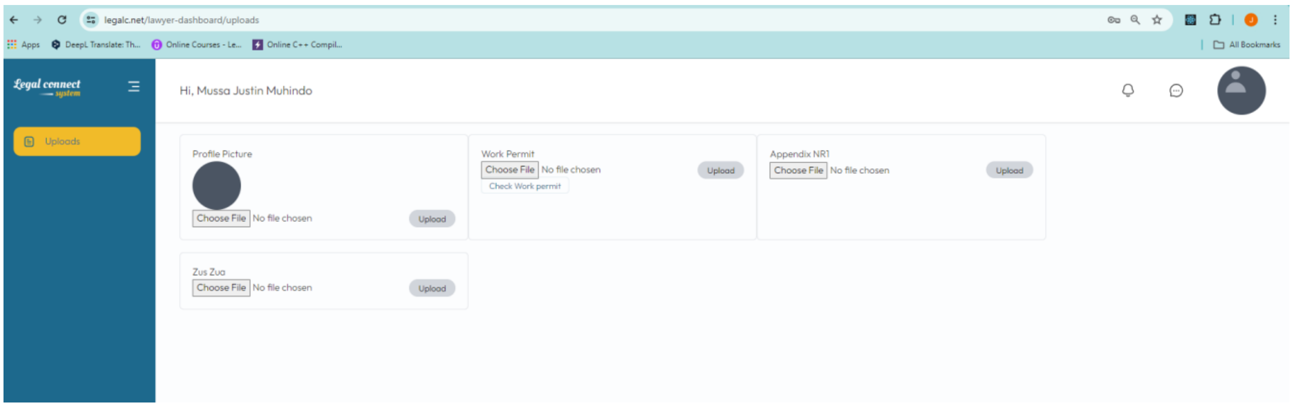
The screenshot displays a login screen with the header "Welcome Back." It prompts for an email and password, providing options for users who have forgotten their password or need to sign up for a new account. A 'Login' button is available for users to proceed after entering their credentials.

6.3.3 Account not activated.



Hi, Mussa Justin Muhindo

Lawyer Account Not Active



Hi, Mussa Justin Muhindo

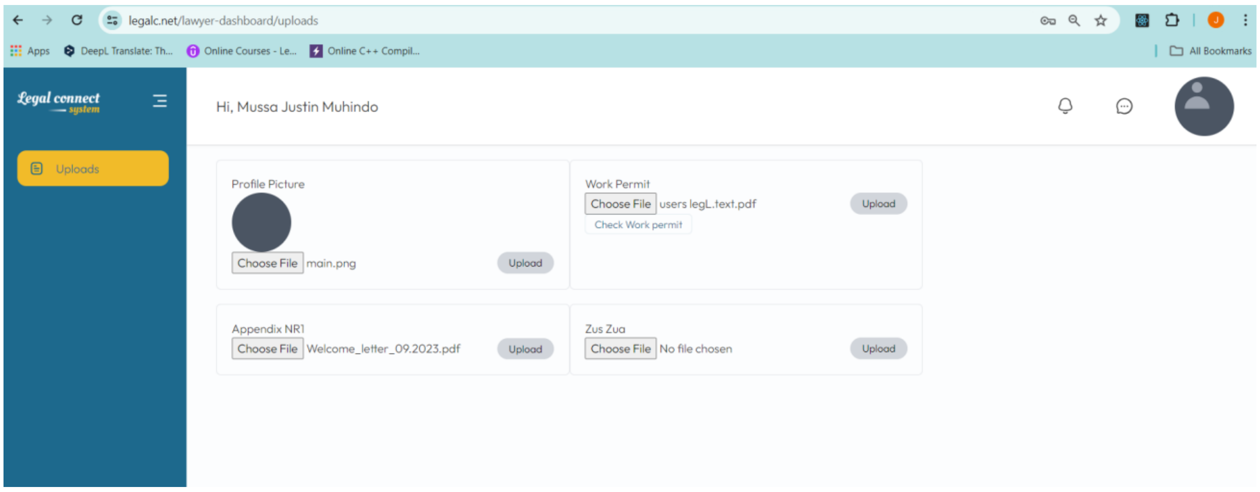
Profile Picture
Choose File No file chosen Upload

Work Permit
Choose File No file chosen Upload
Check Work permit

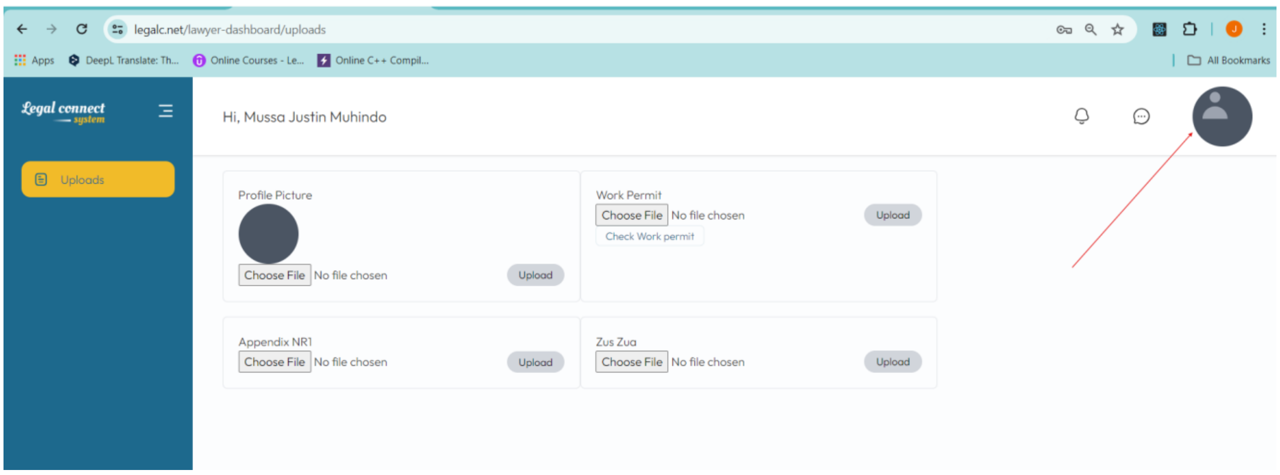
Appendix NR1
Choose File No file chosen Upload

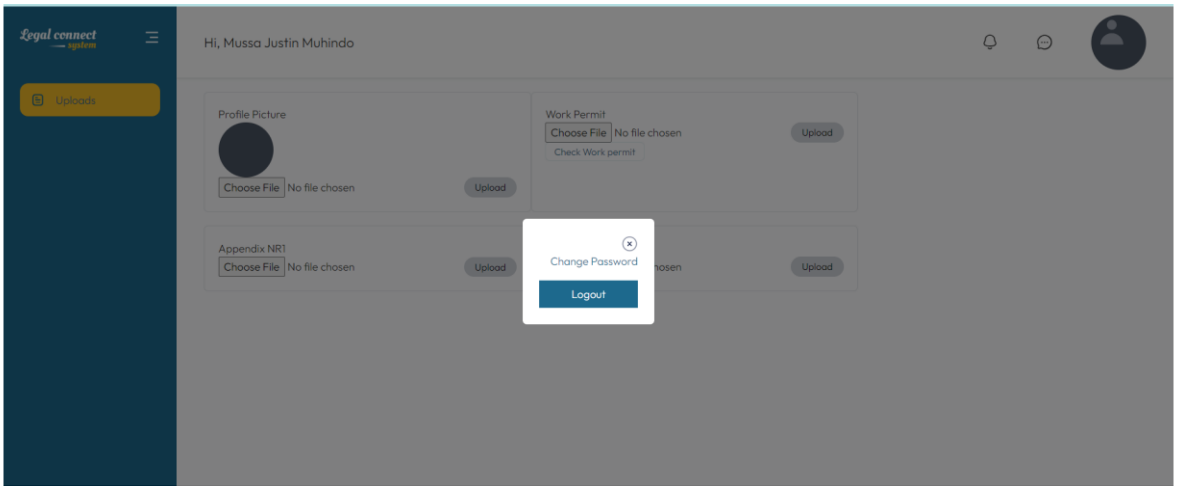
Zus Zua
Choose File No file chosen Upload

The screenshots depict a user interface within the "Legal Connect System," focusing on an uploads section for a user. The first image indicates that the user's lawyer account is not active, suggesting they may need to complete additional steps. The second image shows the detailed uploads page where the user can submit various documents, including a profile picture, work permit, and an appendix NR1, each with an upload button highlighted by red arrows. This interface is presumably designed for lawyers to submit required documentation for account verification or case management purposes.



The screenshot shows the upload interface of the "Legal Connect System," where the user, has already selected files for upload, including a profile picture ("main.png"), a work permit ("users_leg_toy.pdf"), and an appendix document ("welcome_letter_09.2023.pdf"). Each file is associated with a button to upload the respective document. Red arrows point to these files, indicating the action to be taken or drawing attention to the selected files, likely for verification or completion of the user's profile and credentials on the platform.

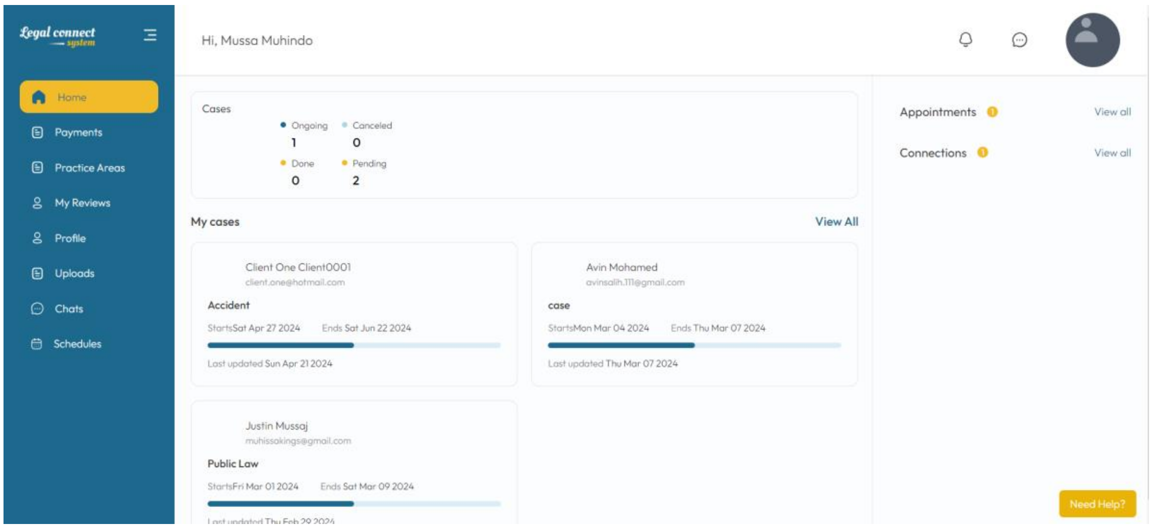




A red arrow points to the user's profile icon in the top right corner, indicating an action taken to reveal a dropdown menu with options to change the password or log out of the system. This suggests the user is accessing account settings or preparing to exit the platform. The user cannot do any action before account being activated.

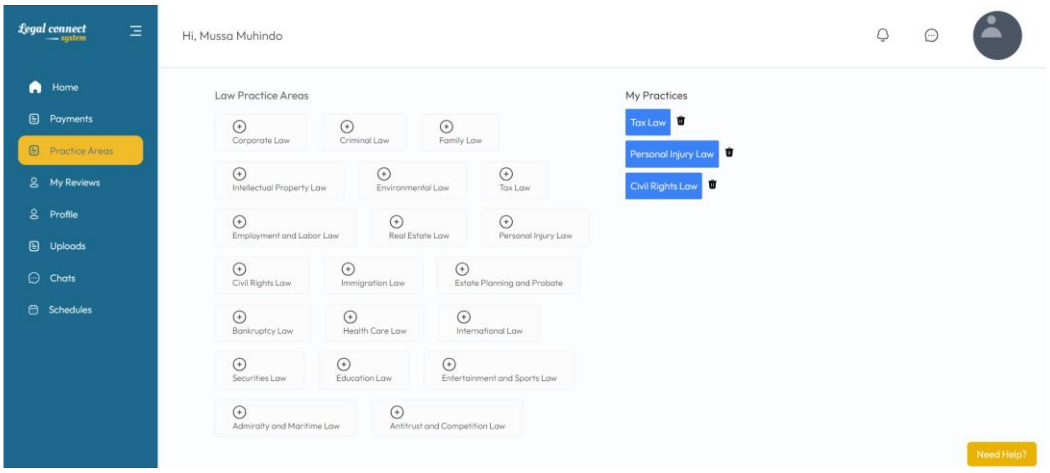
6.3.4 Account activated

6.3.4.1 Home



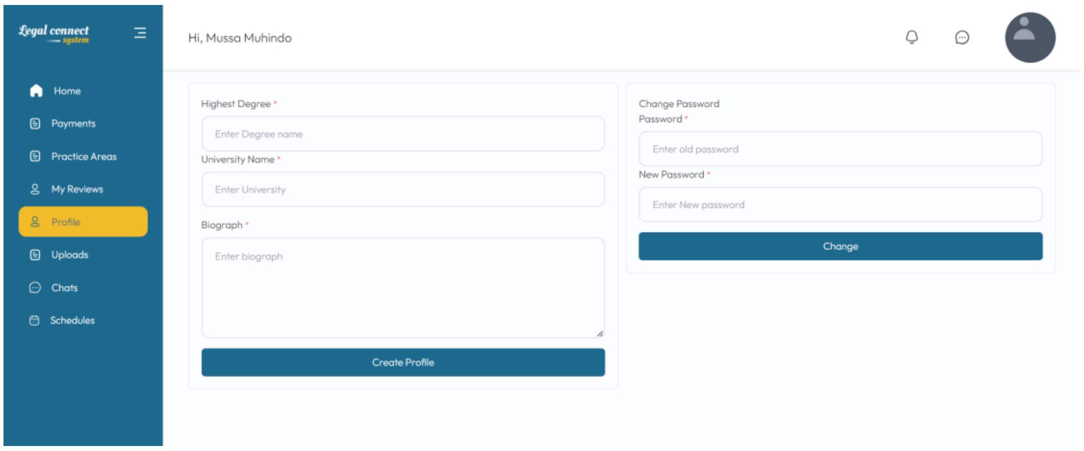
The dashboard summarizes case statuses, with categories like 'Ongoing,' 'Canceled,' 'Done,' and 'Pending,' though all are currently at zero. A sidebar menu includes navigation links such as Home, Payments, Practice Areas, and others, indicating different sections of the platform that the user can access.

6.3.4.2 Practice Areas



The screenshots show the 'Practice Areas' section of user's profile on the "Legal Connect System." The above image displays a variety of legal practice areas that can be selected, with red arrows pointing to specific areas such as Immigration Law, Tax Law, and Personal Injury Law. The lower image reflects the selected practice areas of the lawyer, indicating areas of specialization. This feature allows lawyers to tailor their profile based on their expertise.

6.3.4.3 Lawyer Profile



Profile details of the Lawyer, which will be review to the client side.

6.3.4.4 Accept Connection

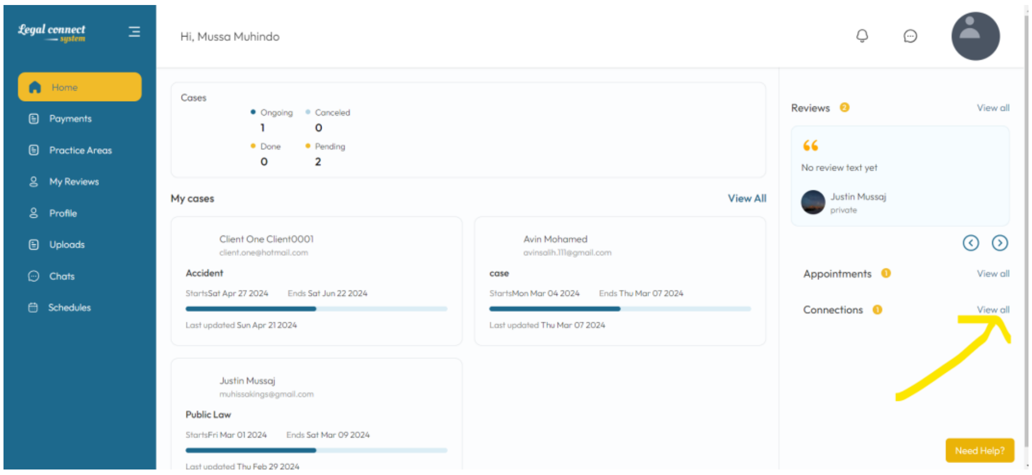
Justin Mussaj
Client
accepted
Thu Feb 29 2024

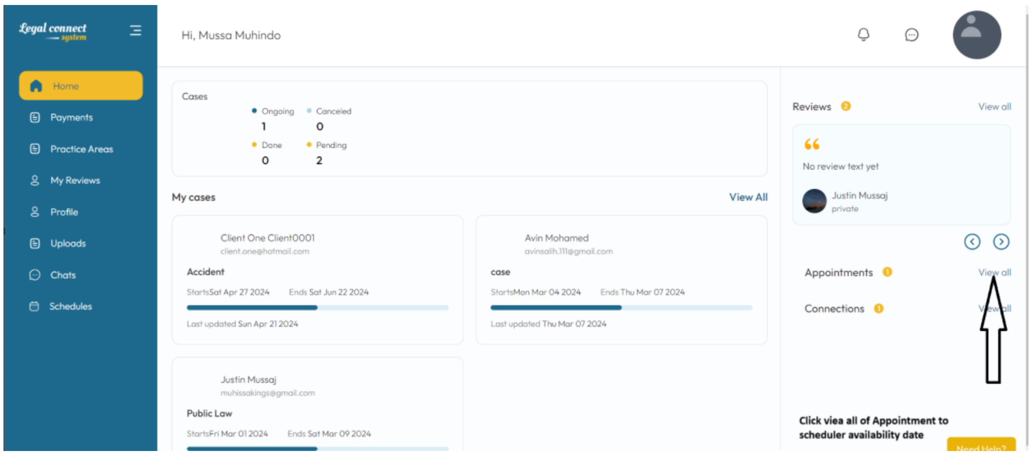
The screenshot shows a user named Souvenir Umutoni logged into the "Legal Connect System."

The user’s dashboard is displaying a summary of cases, with a tally indicating one pending case. Below, details of the pending case are visible, including the client's name (John Doe), the case type (TRC Application), and the case duration with start and end dates. There is also a note on the last update time for this case, providing a quick overview of the user's current legal engagements within the platform.

6.3.4.5 Set Scheduler

User should click to view all of Appointment so that will be able provide available date to the Client. Connections stand to view all requested from users looking to connect to the lawyer.





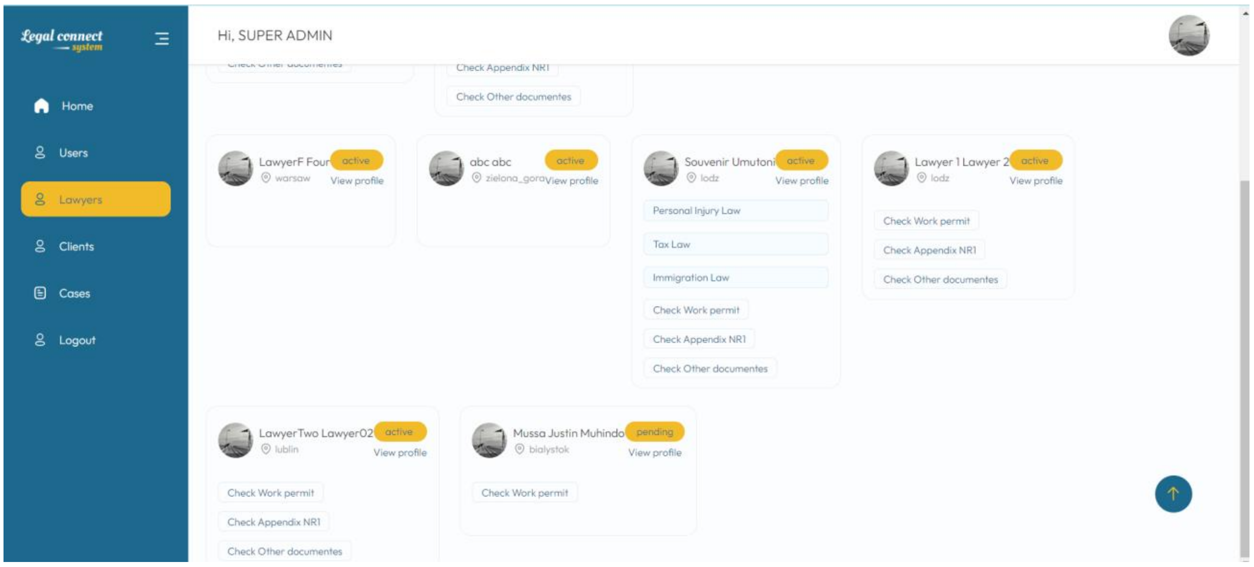
6.4 System Administrator

In software systems, administrative accounts are typically pre-configured during the initial system setup and are not subject to the standard user registration process. These accounts, created for system administrators, come with higher privileges and are essential for managing and maintaining the system. As such, administrators can directly log in to perform their duties without undergoing the registration steps required for regular users.

6.4.1 Login

The administrator will provide username and password as well as of client and Lawyer.

6.4.1.1 Home Sections



The screenshot shows a dashboard view of the "Legal Connect System" from the perspective of a 'SUPER ADMIN'. The sidebar highlights several administrative sections including Home, Users, Lawyers, Payments, Cases, and Logout, with an arrow pointing to 'Home'. The main content displays ongoing cases with details like the lawyer's name, case type, client name, and category, alongside a list of the latest transactions, indicating payment status. There is also a notification for 'Lawyer Requests' with an option to 'Verify', suggesting an action required by the admin to authenticate lawyer profiles or information.

6.4.1.2 Lawyers Sections

'Lawyers' section of a dashboard from the perspective of a 'SUPER ADMIN' on the "Legal Connect System." It features a list of lawyer profiles with varying statuses like 'active' and 'pending.' An action quired from the admin, such as reviewing documents and approving the profile. The interface allows the admin to 'View profile' and check specific documents like 'Work permit' and 'Appendix NR1' for each listed lawyer.

6.4.1.3 Lawyer's Status

The profile is currently marked as 'PENDING' and features a "Change Status" button, indicated by a red arrow, which the admin can use to update the account's status after reviewing the lawyer's information and availability. This interface streamlines the management of lawyer profiles and their approval process on the platform.

7 Bibliography

7.1 Books

1. Wieruch, R.W., The Road to React, Free Computer Books https://freecomputerbooks.com/The-Road-to-Learn-React.html, 2018.
2. Banks, A.B. and Porcello, P.E., Learning React, O'Reilly Media, Inc. https://www.oreilly.com/library/view/learning-react/9781491954614/, 2017.
3. Roldán, C.S., React 17 – Design Patterns and Best Practices, O'Reilly Media., https://www.oreilly.com/library/view/learning-react/9781491954614/ , 2021.

7.2 Articles in professional journals

1. Bafna, Sumangla A., Review on Study and Usage of MERN Stack for Web Development, International Journal for Research in Applied Science and Engineering Technology 10, no. 2, 2022.

7.3 Internet knowledge sources

1. ReactJS Documentation, https://reactjs.org/docs/getting-started.html,
2. Node.js Documentation, https://nodejs.org/en/docs/,
3. Express.js Documentation, https://expressjs.com/en/starter/installing.html,
4. Sequelize Documentation, https://sequelize.org/master/manual/getting-started.html,
5. W3Schools SQL Tutorial, https://www.w3schools.com/sql/
6. Figma Documentation, https://help.figma.com/hc/en-us/articles/14552901442839-Overview-Introduction-to-design-systems

8 Resources

1. React JS, https://react.dev/,
2. Node.js, https://nodejs.org/en,
3. SQL.js, https://github.com/sql-js/sql.js,
4. Pictures, https://unsplash.com/photos/brown-wooden-smoking-pipe-on-white-surface-6sl88x150Xs
5. For design Figma, https://www.figma.com/file/Amjh3WkQNfh8VnSrzzdjtd/Legal-connect-system?node-id=0%3A1&mode=dev