Design Document: Version 1

Table of Contents

1. Introduction

The project aims to create client management software for legal firms, optimizing communication and efficiency for improved client satisfaction and operational transparency.

2. Business Plan

LawLink Client Software is software made for small to medium-sized enterprises, in the field of law. Although it is tailored specifically to the field it could easily be adjusted to the needed areas of your company.

After polishing the software it is going to come out as a ready product to sell to the target audience around the beginning of August. It is going to be sold on a monthly subscription per user. That means that a company can buy a pack of 10 accounts for example which they are going to pay each month. it is going to have bronze, silver, and platinum packages like this : (5 users, 10 users, 20 users). For pricing, we are going to make use of the decoy effect by having 3 packages, with the second and the third one with close prices which in the eyes of the buyers is going to look like a deal.

The **monetization** will happen by leveraging my skills in Facebook & Instagram advertising, trying to get as much reach as possible for the least amount of money. Also having contacts in many fields will be of great help.

This software will be in constant development, implementing new features *(wanted by our users)*, this way we keep good customer relationships. From experience with such software, I’ve seen that the customer relationship is not the best and we want to focus so that we differ from them, so they keep coming back.

3. Ethical and Cultural Aspects

So basically the idea of this software is pretty simple and straightforward. It is to make life easier for big companies. Having software to manage all your stuff improves the work quality, efficiency, and time. This specific software is tailored for law firms but it is simple be remade and adjusted for any type of business.

The Privacy & Personal Data of the users are being taken into account by keeping all the data in databases, protected from SQL injections, sensitive data, like passwords, is being encoded by advanced algorithms and no other user than the logged-in or the responsible for has the right to access any of the users' data.

Transparency is the key to any relationship, so we try to keep full transparency with our customers on how and for what their data is being used.

This project is fully Environmentally sustainable since it does not need anything else than a computer to use the application, and we all have at least one electronic device.

In the future, some more functionalities are going to be added such as language translations since that software will be used worldwide and not everybody knows English perfectly. My intentions for now are that I am going to use the google Translate plugin. The problem is that Google Translate is not the best so I might have to consider creating different versions of the texts in the top 10 used languages and display them when somebody translates to some of the available languages.

4. Architecture Constraints and Design Decisions

Why Spring Boot, React, and MySQL?

Spring Boot:

- Simplifies coding process.
- Vast ecosystem and community support.
- Convention-over-configuration approach.

React:

- Fast, responsive, and component-based.
- Dynamic user interfaces.
- Extensive ecosystem with tools and libraries.

MySQL:

- Reliable and scalable.
- Robust features for managing structured data.
- Support for transactions and integrity constraints.

5. SOLID Principles

In my project, the implementation of SOLID principles is fundamental to ensuring a robust and maintainable software architecture.

Single Responsibility Principle :

Backend Layers:

- **Business Layer**: Responsible for encapsulating the business logic of the application, ensuring that it remains focused on core functionality.
- **Controller Layer**: Handles incoming requests, routing them to appropriate business logic, and preparing responses.
- **Domain Layer**: Defines the core domain models and entities, maintaining the integrity and consistency of the data.
- **Persistence Layer**: Manages the storage and retrieval of data, ensuring efficient data access and management operations.

Open/Closed Principle:

Dependency Injection:

- Utilizing dependency injection allows for the extension of functionalities without modifying existing code. This design choice promotes code reusability and scalability by decoupling components and facilitating easier testing and maintenance.

Liskov Substitution Principle :

Interface Implementation:

- Interfaces are designed to adhere to the LSP, ensuring that derived classes can be substituted without altering the correctness of the system. This approach enables seamless swapping of components and promotes flexibility in design.

Interface Segregation Principle:

Tailored Interfaces:

- Interfaces are tailored to specific client needs, adhering to the ISP to prevent clients from depending on unnecessary functionalities. Each interface exposes only the methods required by its clients, promoting modularity and reducing dependencies.

Dependency Inversion Principle:

Inversion of Control (IoC):

- Implementing IoC containers facilitates dependency inversion, decoupling high-level modules from low-level implementations. Controllers depend on abstractions (interfaces) rather than concrete implementations, promoting flexibility and testability.

Backend Classes and Interfaces:

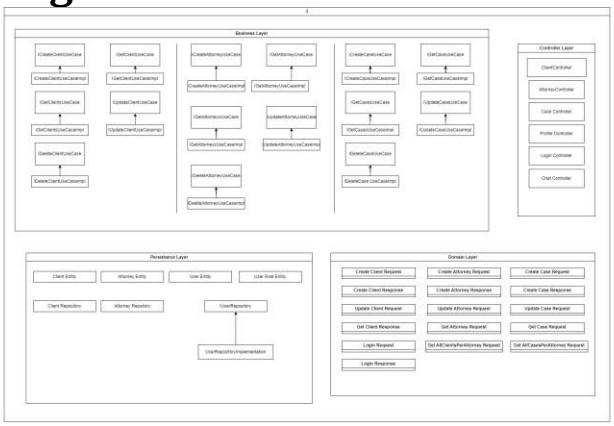
- Each class and function in the backend is meticulously designed to adhere to the SRP, with a clear focus on a single responsibility. Additionally, the use of interfaces ensures loose coupling between components, allowing for easier maintenance and testing.

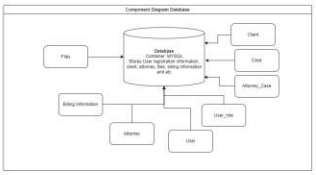
Frontend Layering:

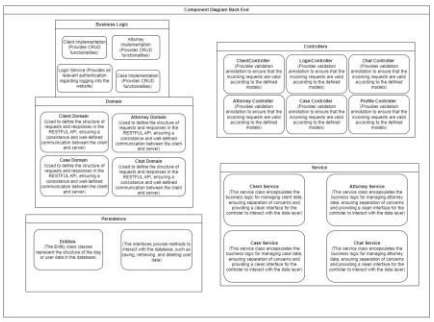
Components, Pages, and APIs:

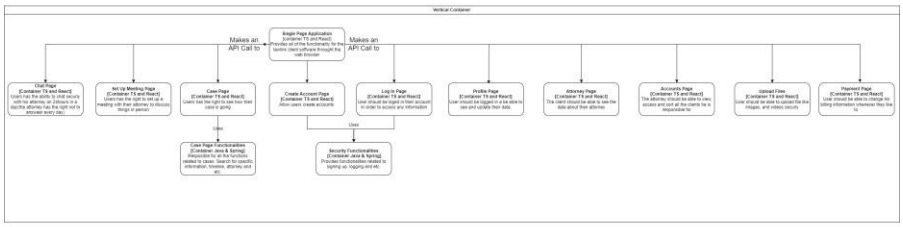
- The frontend architecture is organized into layers for components, pages, and APIs. For example, a user page consists of three components - single account, list of accounts, and account creation. This structured approach facilitates easier development, maintenance, and scalability of the frontend."

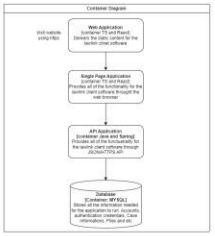
6. C4 Diagrams

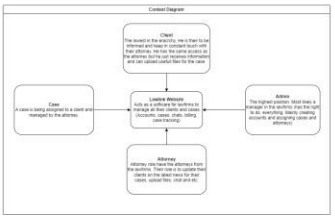




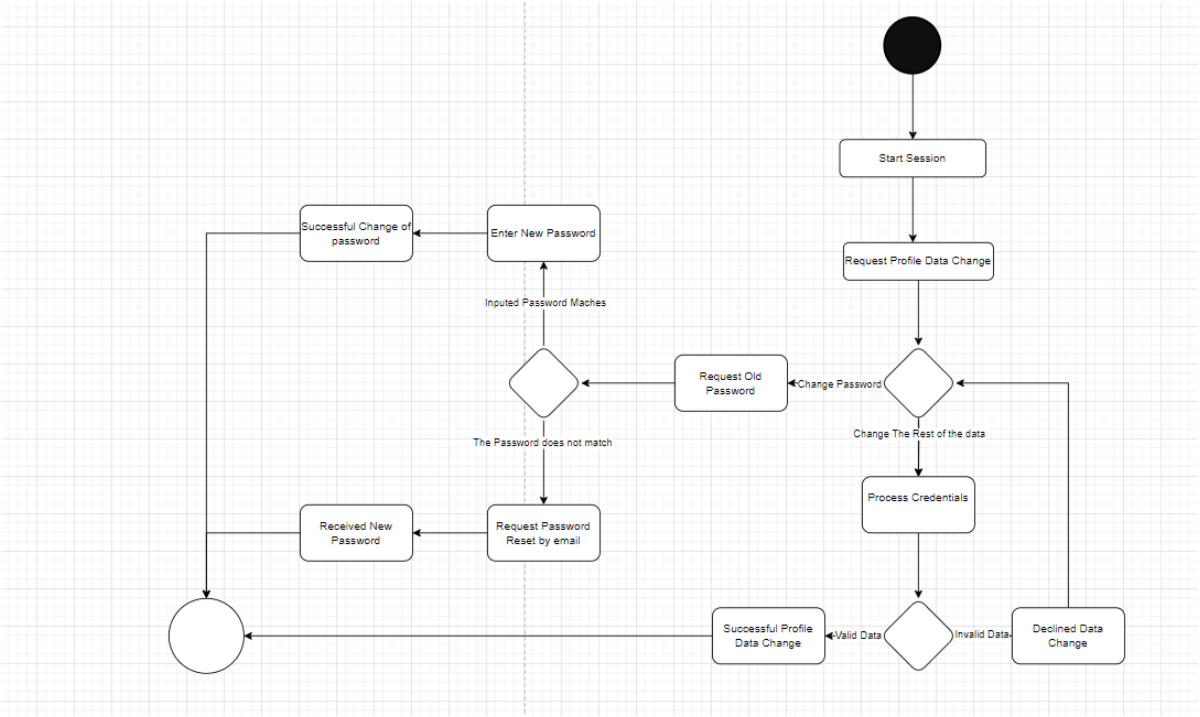


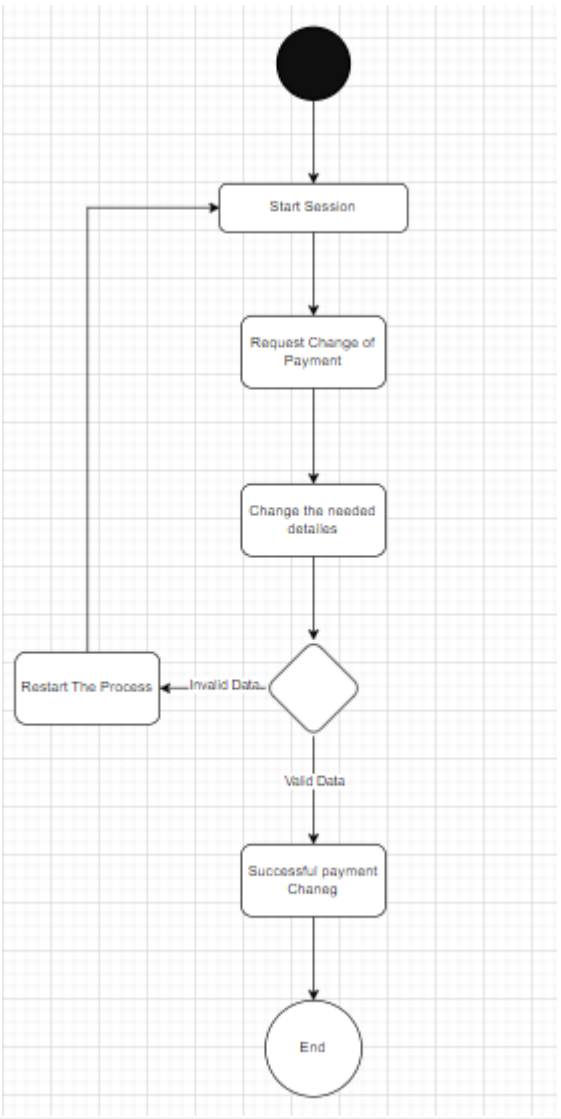


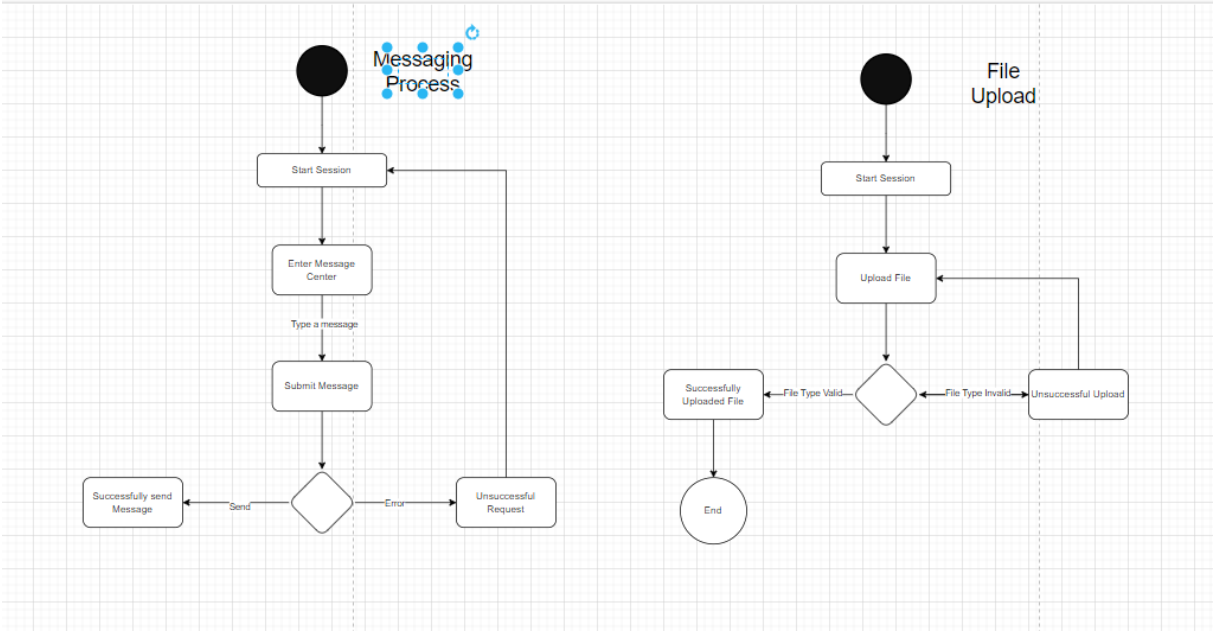


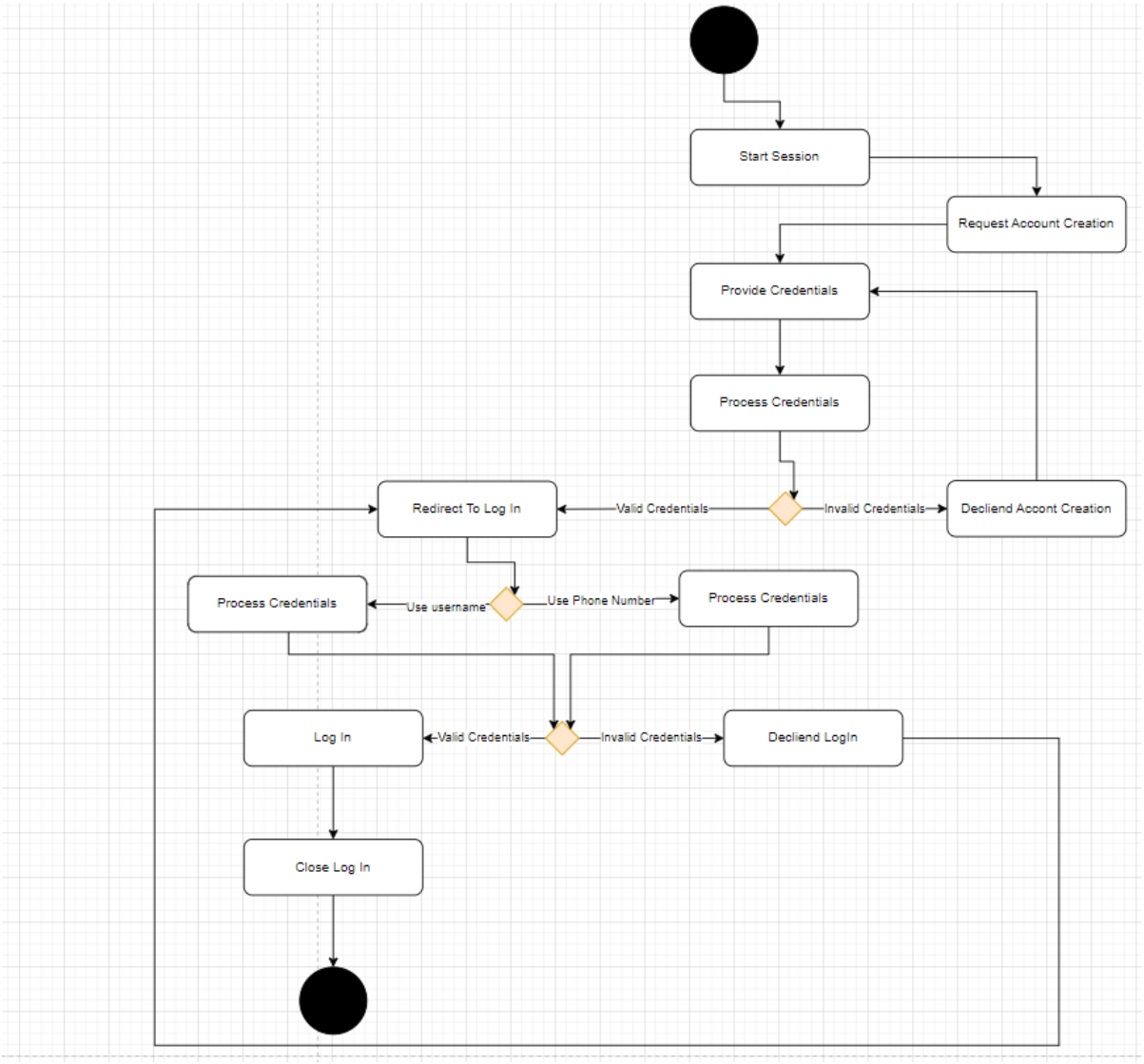


7. Activity Diagrams

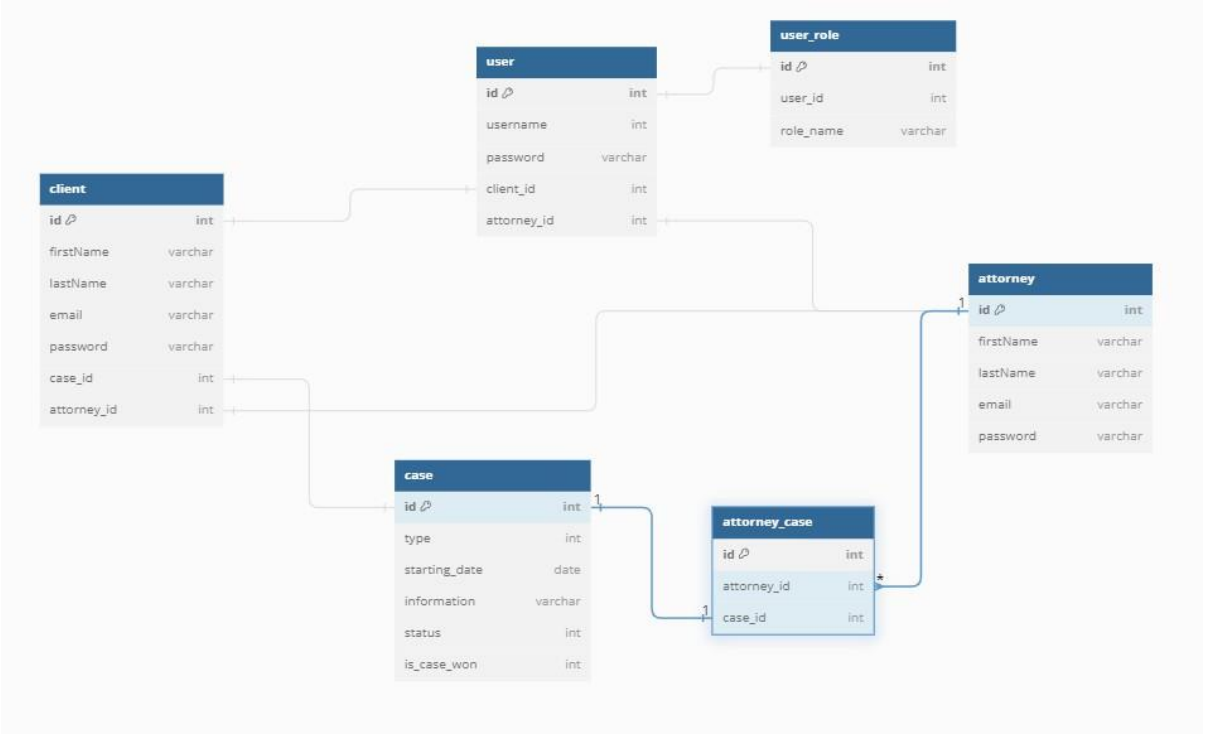



Start Session
Request Change of Payment
Change the needed detailes
Restart The Process
Invalid Data
Valid Data
Successful payment Chaneg
End


Messaging Process
Start Session
Enter Message Center
Type a message
Submit Message
Successfully send Message
Send
Error
Unsuccessful Request
File Upload
Start Session
Upload File
Successfully Uploaded File
File Type Valid
File Type Invalid
Unsuccessful Upload
End


Start Session
Request Account Creation
Provide Credentials
Process Credentials
Valid Credentials
Invalid Credentials
Decliend Accont Creation
Redirect To Log In
Process Credentials
Use username
Use Phone Number
Process Credentials
Log In
Valid Credentials
Invalid Credentials
Decliend LogIn
Close Log In

8. Database Diagram



9. RESTful API Documentation

1. Endpoint: `/accounts`

- **Description**: Endpoint to manage clients.

1.1 `GET` - Retrieve Clients

- **Description**: Retrieves a list of clients.
- **Request**:
  - Method: `GET`
  - URL: `/accounts`

```
"clients": [
        {
            "id": 1,
            "firstName": "John",
            "lastName": "Doe",
            "email": "john.doe@example.com",
            "password": "password123",
            "caseId": null,
            "attorney": {
                "id": 1,
                "firstName": "Alex",
```

- "lastName": "A",
- "email": "agg@s",
- "password": "a",
    - Headers:
        - `Authorization`: Bearer <token>
- **Response**:
    - Status Code: `200 OK`
    - Body:
    - "clients": [
    - {
    - "id": 1,
    - "firstName": "John",
    - "lastName": "Doe",
    - "email": "john.doe@example.com",
    - "password": "password123",
    - "caseId": null,
    - "attorney": {
    - "id": 1,
    - "firstName": "Alex",
    - "lastName": "A",
    - "email": "agg@s",
    - "password": "a",
    -

1.2 POST - Create Resource

- **Description**: Creates a new resource.
- **Request**:
    - Method: `POST`
    - URL: `/accounts`
    - Headers:
        - `Content-Type`: `application/json`
        - `Authorization`: Bearer <token>
    - Body:
    - {
    - "firstName": "",
    - "lastName": "",
    - "email": "",
    - "password": "",
    - "caseId": 0,
    - "attorneyId": 0,
    - "role": ""
    - }
    -
- **Response**:
    - Status Code: `201 Created`
    - Body:
    - {
    - "id": "2",
    - "firstName": "Stanislav",
    - "lastName": "Nikolov",
    - "email": "stasnikolov03@gmail.com",
    - "password": "stas",
    - "caseId": 0,
    - "attorneyId": 2,
    - "role": "CLIENT"

- }
-

1.3 `GET` - Retrieve Resource by ID

- **Description**: Retrieves a single resource by its ID.
- **Request**:
  - Method: `GET`
  - URL: `/accounts/{id}`
  - Headers:
    - `Authorization`: Bearer
- **Response**:
  - Status Code: `200 OK`
  - Body:
  -
  - {
  - "id": 15,
  - "firstName": "Alexa",
  - "lastName": "A",
  - "email": "adgg@s",
  - "password": "a",
  - "caseId": null,
  - }

1.4 `PUT` - Update Resource

- **Description**: Updates an existing resource.
- **Request**:
  - Method: `PUT`
  - URL: `/accounts/{id}`
  - Headers:
    - `Content-Type`: application/json
    - `Authorization`: Bearer <token>
  - Body:
  - jsonCopy code
  - {
  - "id": 1,
  - "firstName": "UpdateStas",
  - "lastName": "Updated",
  - "email": "john.doe@stas.com",
  - "password": "password123",
  - "caseId": null,
  - "attorney": 2
  - }
  -
- **Response**:
  - Status Code: `200 OK`
  - Body:
  - {
  - "id": 1,
  - "firstName": "UpdateStas",
  - "lastName": "Updated",
  - "email": "john.doe@stas.com",
  - "password": "password123",
  - "caseId": null,

```
            "attorney": 2
    }
```

1.5 `DELETE` - Delete Resource

- **Description**: Deletes a resource by its ID.
- **Request**:
  - Method: `DELETE`
  - URL: `/accounts{id}`
  - Headers:
    - `Authorization`: Bearer <token>
- **Response**:
  - Status Code: `204 No Content`

2. Endpoint: `/login`

- **Description**: Endpoint to manage clients.

1.1 `POST` - Login

```
"clients": [
        {
            "id": 1,
            "firstName": "John",
            "lastName": "Doe",
            "email": "john.doe@example.com",
            "password": "password123",
            "caseId": null,
            "attorney": {
                "id": 1,
                "firstName": "Alex",
                "lastName": "A",
                "email": "agg@s",
                "password": "a",
```

- **Description**: lLog in the application
- **Request**:
  - Method: `POST`
  - URL: `/login`
  - Headers:
    - `Content-Type`: `application/json`
    - `Authorization`: Bearer <token>
  - Body:
  - ```
    {
        "username": "adgg@s@lalwink.com",
         "password": "a"
    }
    ```
- **Response**:
  - Status Code: `201 Created`
  - Body:
    ```

```
{
    "accessToken":
"eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZGdnQHNAbGFsd2luay5jb20iLCJpYXQiOjE3MTc3N
zUyNjYsImV4cCI6MTcxNzc3NzA2Niwicm9sZXMiOlsiQVRUT1JORVkiXX0.xo1XquFQS06_oIJB
3YH3YfdyfGHaLmT_gBF9NF6_Wu8",
    "userId": 15,
    "userRoles": "ATTORNEY"
}
```