# **Take-Home Assignment: Legal SaaS Customer & Matter Management**

## **Objective**

Your task is to build a **simple backend API and UI** for managing **customers** and their associated **matters**. The project should follow **RESTful API principles** and include a basic **frontend** to interact with the system.

*If you are applying for a BE-focused position, spend more time on the backend; if a FE-focused position, spend more time on the front end.*

### **What You'll Build**

* A **backend API** with authentication and customer/matter management.
* A **simple frontend UI** to list, create, and view customers and matters.

## **Technical Requirements**

### **Backend**

* Use C#/.Net 8, Ruby on Rails, or a Node.js framework
* Database: **PostgreSQL** (use an ORM like Prisma, Sequelize, EF, AR, or Knex, etc)
* Authentication: **Cookie** or **JWT-based authentication**
* Routes should follow **RESTful best practices** (see API routes below)
* Implement **basic error handling** (invalid input, missing records, etc.)
* Write at least **one database migration** to initialize the schema

### **Frontend**

* Use **React with TailwindCSS** (or Next.js if preferred)
* Build a minimal UI with:
  + A login form
  + A list of **customers** (clicking a customer should show their matters)
  + A form to create **new customers**
  + A form to create **new matters** under a customer

### **API Routes to Implement**

#### **Authentication**

* POST /api/auth/signup → Create a new user (email, password, firm name)
* POST /api/auth/login → Login and receive JWT
* GET /api/auth/me → Return authenticated user info (JWT protected)

#### **Customers**

* GET /api/customers → Retrieve a list of customers
* POST /api/customers → Create a new customer (name, phone)
* GET /api/customers/{customer\_id} → Retrieve details of a customer
* PUT /api/customers/{customer\_id} → Update a customer
* DELETE /api/customers/{customer\_id} → Delete a customer

#### **Matters**

* GET /api/customers/{customer\_id}/matters → Retrieve matters for a customer
* POST /api/customers/{customer\_id}/matters → Create a matter
* GET /api/customers/{customer\_id}/matters/{matter\_id} → Retrieve matter details

## **Requirements for Submission**

1. **GitHub Repository** with:
   * A README.md file explaining how to run the project
   * API documentation (can be simple markdown or Postman collection)
   * .env.example file for environment variables
2. **Deliverables:**
   * Backend API with working endpoints
   * Basic UI with customer and matter listing + creation
   * Authentication implemented
   * PostgreSQL database setup with migrations
   * Any **bonus features** (if time permits)

## **Evaluation Criteria**

| **Category** | **Criteria** |
| --- | --- |
| **Code Quality** | Is the code modular, readable, and well-structured? |
| **RESTful API Design** | Are the API routes well-structured and follow REST principles? |
| **Database Schema** | Is the PostgreSQL schema properly normalized? |
| **Authentication & Security** | Are passwords hashed? Does JWT authentication work properly? Or does the cookie use proper security (given its development mode, would it in production?) |
| **Error Handling** | Are errors (e.g., invalid input, missing records) handled gracefully? |
| **UI/UX Considerations** | Does the UI provide a clean and intuitive experience? |
| **Documentation** | Is there a clear README with setup instructions? |
| **Bonus (Not Required)** | Extra features like customer/matter search, better UI, testing, or role-based access |

## **Time Expectation**

* This task is designed to be **completed within 3-6 hours**.
* We **don’t expect a fully polished product**—focus on writing clean, structured, and working code.
* If you run out of time, **leave a README note explaining what’s missing and how you'd improve it**.

## **Bonus Ideas (Optional)**

If you have extra time, feel free to **enhance** your submission:

* **Search & Filtering** for customers and matters
* **Unit Tests** (Jest or Mocha for backend)
* **Docker Support** (Dockerfile + docker-compose)
* **Role-Based Access** (e.g., admin vs. standard users)
* **Improved UI/UX** (Better styling, real-time updates)

## **Submission Instructions**

1. **Upload your code to a GitHub repository** (public or private, but share access).
2. Include setup instructions in your README.md.
3. Email us with:
   * The **GitHub repo link**
   * A short note on **what you completed** and **what you'd improve with more time**.

## **What Happens Next?**

* We'll **review your submission** and evaluate it based on the criteria above.
* If you pass, we'll invite you for a **live code review + system design discussion**.

## **Final Thoughts**

This assignment is **not about perfection**—it’s about seeing **how you think, structure code, and solve problems**. If anything is unclear, **feel free to ask questions!** 🚀