



Gabriel Zuany Duarte Vargas

Lorenzo Rizzi Fiorot

Rafael Ribeiro Carvalho

Lawyer Center

Vitória, ES

2024

1 Introdução

Este documento apresenta o LawyerCenter. Este, tem como propósito principal conectar clientes que necessitam de serviços jurídicos com advogados que oferecem esses serviços. Funciona de maneira semelhante a um site de freelancer, onde os clientes podem buscar por profissionais específicos com base em localidade e área de atuação e advogados podem verificar os clientes interessados em seu serviço e que desejam entrar em contato.

Além desta introdução, este documento está organizado da seguinte forma: a Seção 2 apresenta os objetivos propostos para o trabalho, o que buscamos alcançar; a Seção 3 apresenta a metodologia utilizada nesta atividade, junto com o por que dessa escolha; por fim, a Seção 4 apresenta os resultados obtidos do trabalho.

2 Objetivos

Seu propósito principal é resolver a questão frequente e desafiadora que muitas pessoas enfrentam ao buscar assistência legal: a dificuldade em encontrar um advogado especializado no tipo específico de problema jurídico que estão enfrentando.

Essa abordagem resolverá uma série de problemas para os usuários. Primeiramente, simplificará drasticamente o processo de encontrar um advogado. Em vez de pesquisar em várias fontes ou depender de recomendações limitadas, os usuários terão acesso a uma lista de profissionais qualificados que estão prontos e disponíveis para ajudar com sua situação específica.

Além disso, o sistema tem o potencial de promover a transparência e a acessibilidade no acesso à justiça. Ao facilitar o contato direto entre os usuários e os advogados especializados, remove-se uma barreira significativa que muitas vezes impede as pessoas de buscar assistência legal quando necessário. Isso pode ser especialmente benéfico para aqueles que podem não ter conhecimento prévio sobre como encontrar um advogado ou podem se sentir intimidados pelo processo.

Em suma, o Lawyer Center tem como objetivo tornar o acesso à assistência jurídica mais eficiente, conveniente e acessível para os usuários, ao mesmo tempo em que simplifica o processo de encontrar um advogado especializado em suas necessidades específicas

3 Metodologia

A metodologia utilizada para a implementação do projeto foi criar uma arquitetura de modo que o frontend não precisasse acessar diretamente os dados armazenados e que houvesse consistência entre as informações registradas. Desse modo, foi implementado uma API para interfacear o banco de dados e garantir o controle de acesso em um servidor remoto.

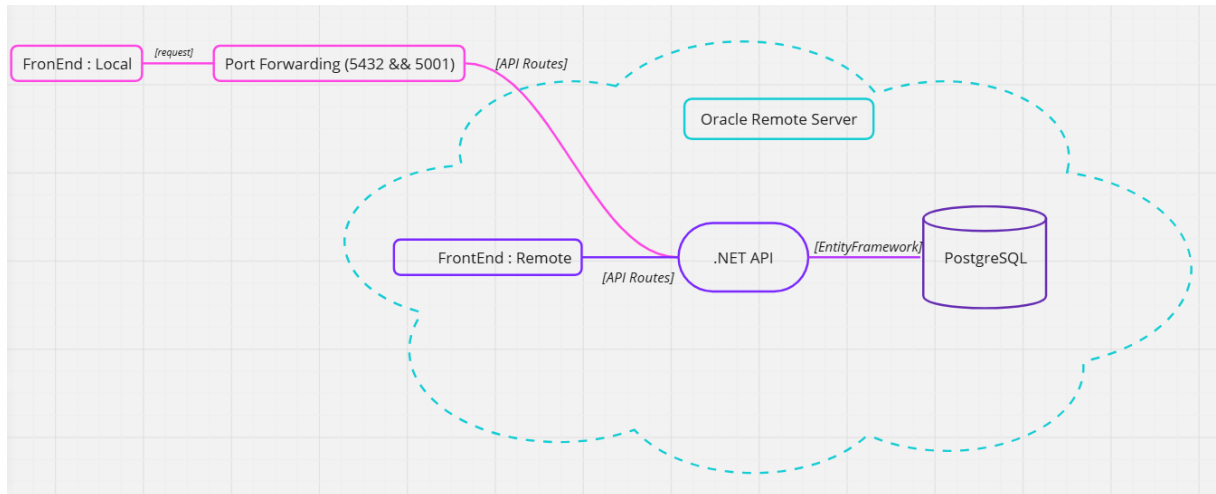


Figura 1 – Comunicação entre Frontend e Backend

Para acessar o painel frontend, há duas opções possíveis:

1. Execução local com o comando *npm start*.
2. Acesso remoto com redirecionamento de porta.

Nas seções subsequentes há a explicação detalhada acerca das etapas que compõem o frontend e o backend.

3.1 Frontend

Nessa etapa, utilizamos como tecnologia principal o React. React é uma biblioteca JavaScript amplamente utilizada para a criação de interfaces de usuário, permitindo a construção de componentes reutilizáveis e uma gestão eficaz do estado da aplicação. Nesta documentação, detalharemos as funcionalidades principais, a estrutura do projeto e as práticas recomendadas adotadas para garantir um código limpo e manutenível.

3.1.1 Estrutura do React

O diretório *src* é onde o código-fonte do React reside. Ele contém os principais arquivos e pastas do projeto, dentre elas:

1. Components: Contém os componentes reutilizáveis do aplicativo. Cada componente organizado em seu próprio arquivo, como o botão e a caixa para escrita do usuário usados amplamente pelas páginas do site.

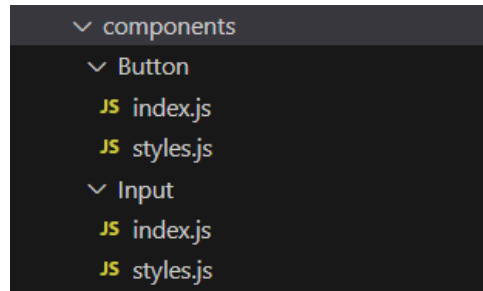


Figura 2 – Estrutura dos componentes

2. Contexts: Local onde os contextos são definidos.
3. Hooks: Armazena os custom hooks criados.
4. Img: Armazena imagens, ícones ou outros recursos visuais.
5. Styles: Contém o arquivo de estilo global (CSS, SCSS, styled-components) para componentes e páginas.
6. Pages: Cada página do seu aplicativo deve ter sua própria pasta aqui.

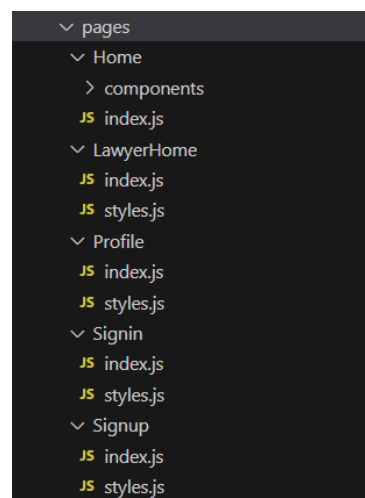


Figura 3 – Estrutura das paginas

7. Routes: É onde as definições de rotas do aplicativo são organizadas. As rotas determinam quais componentes ou páginas serão exibidos com base na URL atual.

```
15
16 const RoutesApp = () => {
17   return (
18     <BrowserRouter>
19       <Fragment>
20         <Routes>
21           <Route exact path="/home/:clientId" element={<Home/>} />
22           <Route path="/" element={<Signin />} />
23           <Route exact path="/signup" element={<Signup />} />
24           <Route exact path="/profile/:lawyerId" element={<Profile />} />
25           <Route exact path="/lawyerHome/:lawyerId" element={<LawyerHome />} />
26           <Route path="*" element={<Signin />} />
27         </Routes>
28       </Fragment>
29     </BrowserRouter>
30   );
31 };
32
```

Figura 4 – Estrutura das rotas

3.1.2 Instalação

Para instalar e executar o frontend do projeto, siga os passos abaixo:

1. Pré-requisitos: Certifique-se de que o Node.js está instalado em seu sistema, pois ele é necessário para gerenciar as dependências do projeto através do Node Package Manager (npm).
2. Clonar o repositório: Clone o [repositório](#) do projeto para o seu computador.
3. Instalar dependências: Abra o terminal na raiz do diretório do projeto e execute o comando `npm install`. Este comando instalará todas as dependências necessárias listadas no arquivo `package.json`.
4. Iniciar o servidor de desenvolvimento: Após a instalação das dependências, execute o comando `npm start` para iniciar o servidor de desenvolvimento. O site estará disponível em seu navegador no endereço padrão, geralmente `http://localhost:3000`.

3.1.3 Estilos

Para a estilização do projeto, utilizamos a biblioteca *styled-components*, que permite escrever estilos CSS em arquivos JavaScript. Cada tela do projeto possui um arquivo `index.js` para a lógica do componente e um arquivo `styles.js` dedicado à estilização. A cor padrão utilizada em todo o projeto é um tom de cinza escuro `#1C1C1C`, garantindo uma identidade visual consistente. Os *styled-components* proporciona uma forma modular e reutilizável de aplicar estilos, facilitando a manutenção e expansão do design.

3.1.4 Telas e funcionalidades

- Tela de Login: Responsável pela autenticação dos usuários através dos campos de email, CPF e senha. O sistema verifica as credenciais na API e, caso o usuário não esteja cadastrado, uma mensagem de erro é exibida. Mensagens de erro também aparecem se as informações estiverem incorretas ou se algum campo estiver faltando. Após a autenticação, o sistema retorna o tipo de usuário: se for um advogado, ele é redirecionado para a home de advogado; se for um cliente, para a home de cliente.

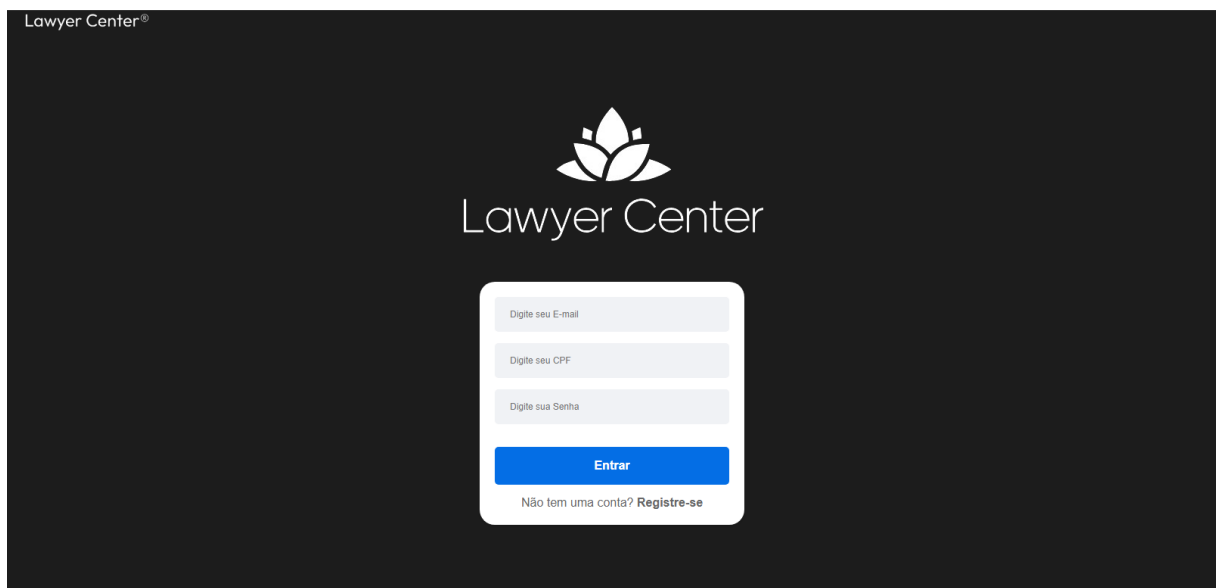
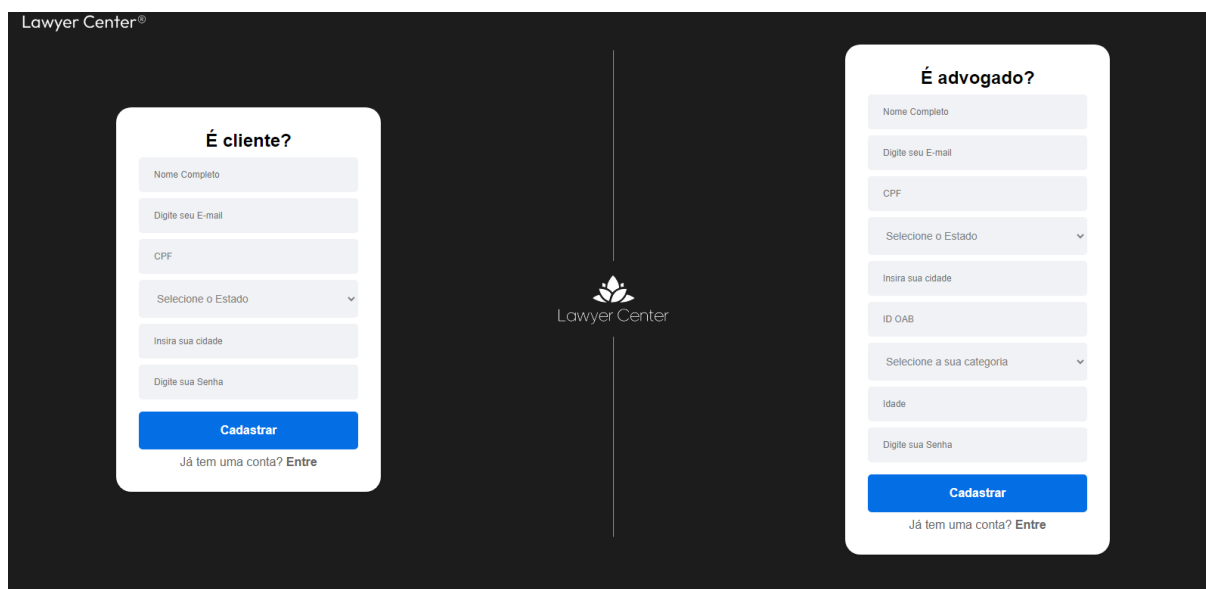


Figura 5 – Tela de login

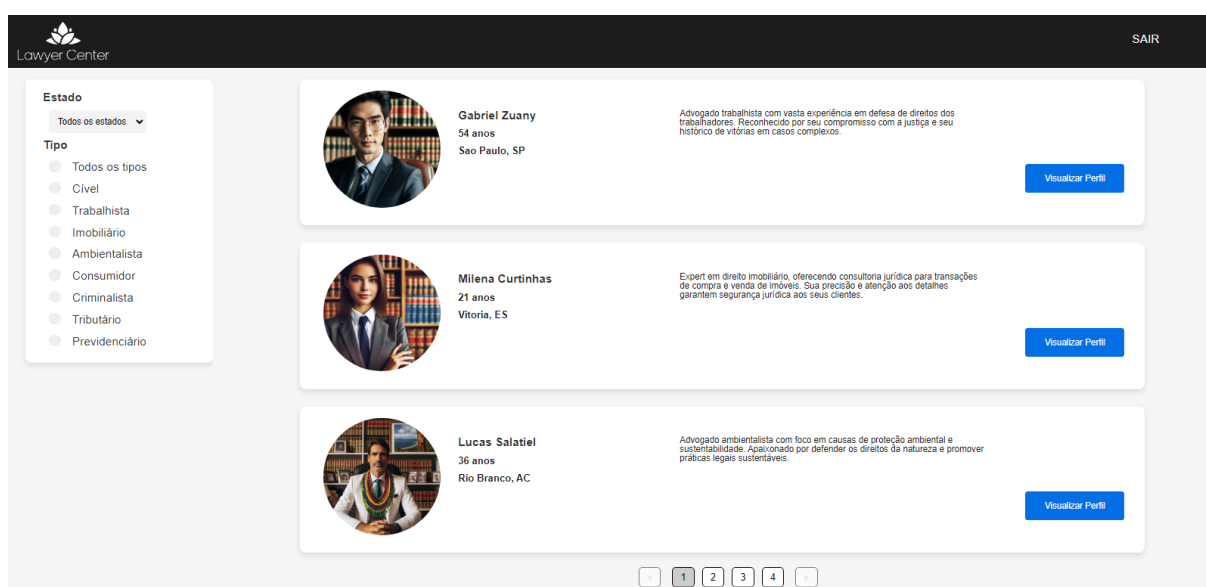
- Tela de Cadastro: Os usuários podem se registrar como advogados ou clientes, cada um com campos específicos a serem preenchidos. Caso todas as informações não sejam preenchidas corretamente, um aviso será exibido, solicitando o preenchimento completo dos campos necessários.



A imagem mostra a interface de cadastro do Lawyer Center, dividida em duas seções: "É cliente?" e "É advogado?". Ambas as seções possuem campos para Nome Completo, E-mail, CPF, Estado (selecionado via dropdown), Cidade, ID OAB (para advogados), Categoria (selecionada via dropdown), Idade e Senha. Cada seção termina com um botão azul "Cadastrar" e um link "Já tem uma conta? Entre". O logo do Lawyer Center está centralizado entre as duas seções.

Figura 6 – Tela de cadastro

- Home do Cliente: Uma página onde são listados todos os advogados cadastrados. Há filtros disponíveis para selecionar o estado do advogado e sua especialidade. Cada card de advogado possui um botão "Visualizar Perfil", que redireciona o usuário para a página de perfil do advogado.



A imagem mostra a interface de home do cliente do Lawyer Center. No topo, há o logo do Lawyer Center e o link "SAIR". À esquerda, há um menu de filtros com "Estado" (dropdown "Todos os estados") e "Tipo" (lista de especialidades: Todos os tipos, Cível, Trabalhista, Imobiliário, Ambientalista, Consumidor, Criminalista, Tributário, Previdenciário). À direita, há uma lista de cards de advogados. Cada card contém uma foto circular, o nome, idade, estado, uma breve descrição e um botão "Visualizar Perfil".

Nome	Idade	Estado	Descrição
Gabriel Zuany	54 anos	Sao Paulo, SP	Advogado trabalhista com vasta experiência em defesa de direitos dos trabalhadores. Reconhecido por seu compromisso com a justiça e seu histórico de vitórias em casos complexos.
Milena Curtinhas	21 anos	Vitoria, ES	Expert em direito imobiliário, oferecendo consultoria jurídica para transações de compra e venda de imóveis. Sua precisão e atenção aos detalhes garantem segurança jurídica aos seus clientes.
Lucas Salatiel	36 anos	Rio Branco, AC	Advogado ambientalista com foco em causas de proteção ambiental e sustentabilidade. Apoiado por defender os direitos da natureza e promover práticas legais sustentáveis.

Figura 7 – Tela de home cliente

- Perfil do Advogado: Exibe as principais informações sobre o advogado, incluindo sua descrição. Há também um botão para fazer contato com o advogado, permitindo que os clientes entrem em comunicação diretamente via email.

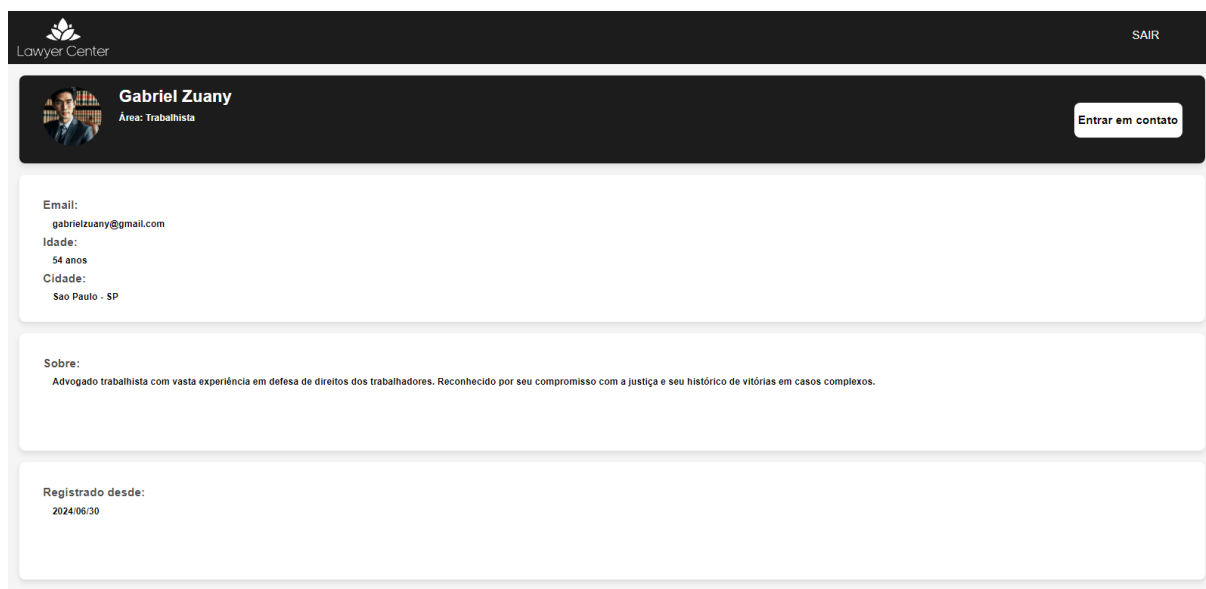


Figura 8 – Tela de perfil de advogado

- Home do Advogado: Quando um advogado faz login, ele é redirecionado para a sua página de perfil, onde pode modificar suas informações, verificar suas mensagens e novas solicitações de contato. Esta tela oferece um ponto centralizado para que os advogados gerenciem suas atividades e interações com os clientes.

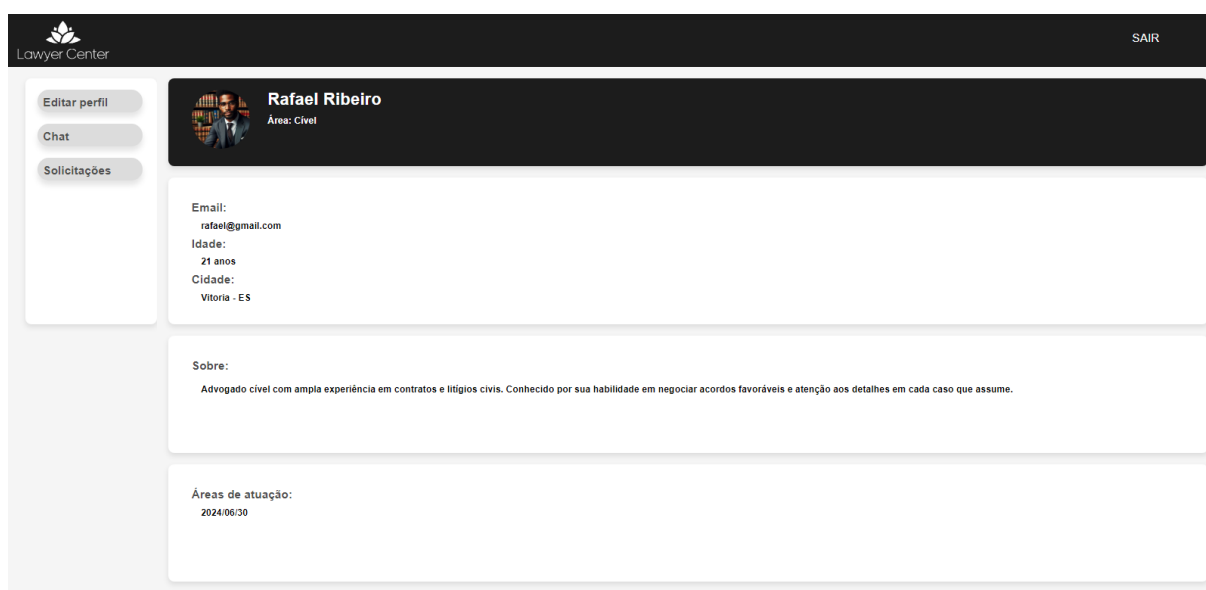


Figura 9 – Tela de home advogado

3.2 Backend

Nessa seção, encontra-se a arquitetura explicada do backend, a qual compreende: servidor remoto, containeres, banco de dados, API, criptografia e serviços de infra para armazenamento de arquivos.

3.2.1 Oracle Remote Server

Para a implementação do acesso remoto aos dados, foi erguido um servidor remoto na Oracle, no qual estão em execução o banco de dados, a API e o frontend em React.

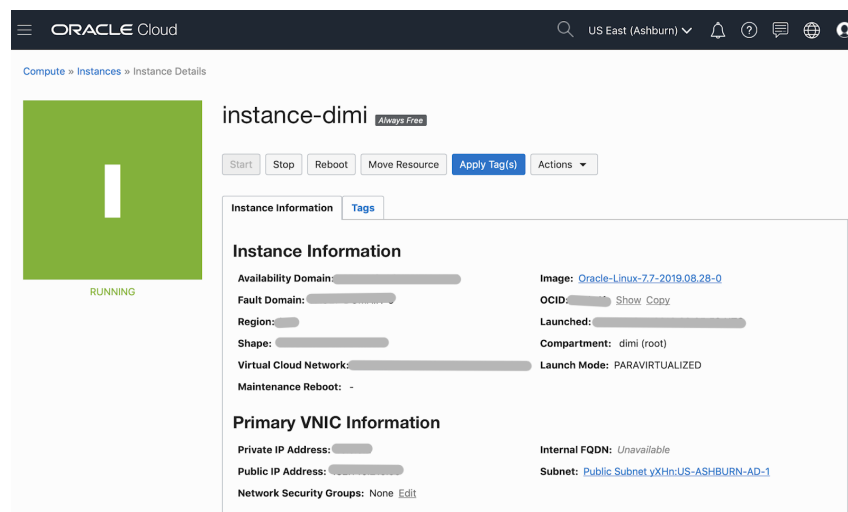


Figura 10 – Oracle Server

O acesso ao CLI do servidor se dá por meio do tunelamento por *SSH (Secure Shell)* com par de *chaves RSA* protegidas por *secure passphrase*.

```
zuzu@zuany:~/dev/auth$ sudo ssh -i ssh_rsa_lolo ubuntu@
Enter passphrase for key 'ssh_rsa_lolo':
Welcome to Ubuntu 22.04.4 LTS (GNU/Linux 6.5.0-1023-oracle x86_64)

* Documentation:  https://help.ubuntu.com
* Management:    https://landscape.canonical.com
* Support:        https://ubuntu.com/pro

System information as of Sun Jun 30 02:26:53 UTC 2024

System load:                0.0
Usage of /:                  38.4% of 44.96GB
Memory usage:                41%
Swap usage:                  1%
Processes:                   151
Users logged in:              0
IPv4 address for br-20c8c3335c85: 172.18.0.1
IPv4 address for br-7cb8d8c5672f: 172.22.0.1
IPv4 address for br-8b6eec1f059c: 172.20.0.1
IPv4 address for br-8fc9d90dacf5: 172.21.0.1
IPv4 address for br-b390ca85504b: 172.19.0.1
IPv4 address for docker0:      172.17.0.1
IPv4 address for ens3:         10.0.0.8

* Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
  just raised the bar for easy, resilient and secure K8s cluster deployment.

https://ubuntu.com/engage/secure-kubernetes-at-the-edge

Expanded Security Maintenance for Applications is not enabled.

24 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

*** System restart required ***
Last login: Sat Jun 29 05:56:00 2024 from 
ubuntu@instance-20240329-2102:~$ cat .ssh/
authorized_keys  known_hosts
ubuntu@instance-20240329-2102:~$ cat .ssh/authorized_keys
ssh-rsa AAAA
```

Figura 11 – SSH Access

3.2.2 Docker

No servidor, está em execução o *Docker*, no qual há containeres executando o banco de dados (*PostgreSQL*), a *API* e o website em *React*.

```
ubuntu@instance-20240329-2102:~$ sudo docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
fc612c6b5718	lc_api_img	"dotnet API.dll"	51 minutes ago	Up 51 minutes		lc_api_container
a2ede4e32cc7	postgres:14	"docker-entrypoint.s..."	3 weeks ago	Up 3 weeks	0.0.0.0:5432→5432/tcp, :::5432→5432/tcp	LC_pgsql

```
ubuntu@instance-20240329-2102:~$
```

Figura 12 – Docker Containers

O docker está configurado para bloquear qualquer acesso externo ao banco e API sem autenticação (ssh), mesmo que a porta do servidor esteja exposta. Apenas o painel está disponível (via *PortForwarding*).

3.2.3 Banco de Dados

Para a persistência dos dados, foi escolhido o PostgreSQL como banco relacional. O mesmo, roda no container na porta 5432 (padrão) e conta com autenticação e controle de usuário (instância de leitura e root). O modelo ERD - Entity Relational Diagram - a seguir exibe a estrutura do modelo relacional adotado.

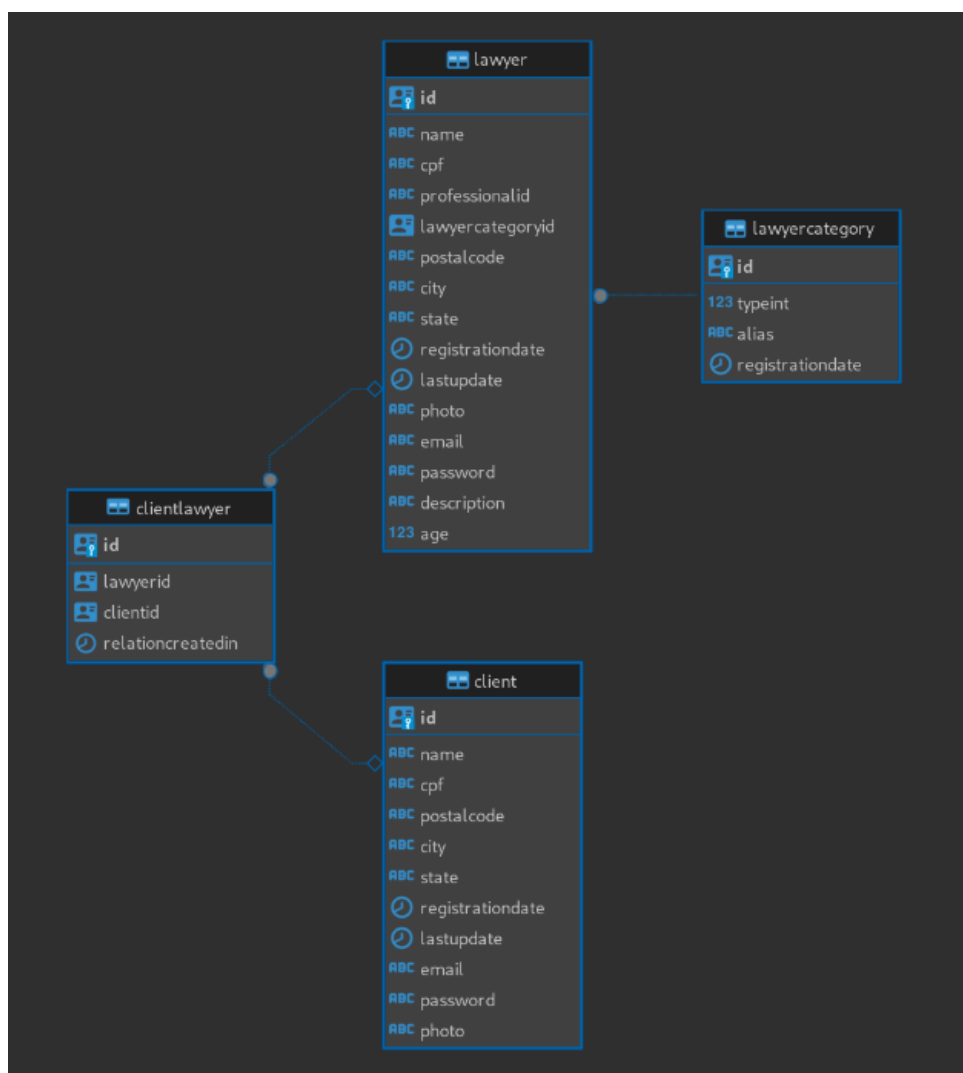


Figura 13 – PgSQL ERD

3.2.4 API

A API foi construída com o objetivo de interfacear o acesso do Frontend (e demais serviços) aos dados do banco. Assim, garantimos o controle do que se é permitido ser acessado ou modificado no banco.

Para a implementação, foi usado o Microsoft .NET 7 (C#), EntityFramework e Criptografia AES.

- **Microsoft .NET 7:** O [.NET 7](#) é um ambiente de execução gerenciado que oferece uma variedade de serviços aos aplicativos em execução. Ele consiste em dois componentes principais: o [CLR \(Common Language Runtime\)](#) e a biblioteca de classes.
- **EntityFramework:** O [.NET Entity Framework](#) é um mapeador de relação de objetos que permite criar uma camada de acesso a dados de modo direto, modelando as classes do programa de acordo com as tabelas no RDMBS.
- **Criptografia AES:** A API implementa o algoritmo de criptografia [AES - Advanced Encryption Standard](#) - para garantir a segurança no armazenamento dos dados sensíveis do usuário (senha, CPF, CEP, ...). O AES é um [algoritmo de chave simétrica](#) que funciona de modo matricial, alterando a posição original dos bits, das colunas e das linhas, com shuffling e transformações lineares.

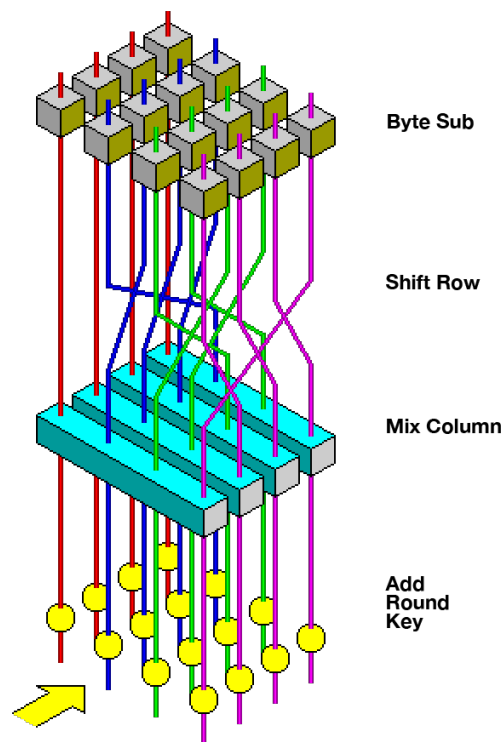


Figura 14 – AES Round

A API está em execução no Docker para facilitar o gerenciamento e isolamento do serviço, facilitando etapas como rebuild e restart.

```

root@instance-20240329-2102:/home/ubuntu/LawyerCenter/API# sudo docker rm lc_api_container && docker build . -t lc_api_img && docker run --network host -
-name lc_api_container -p 5001:5001 -e ASPNETCORE_URLS=http://+:5001 -e ASPNETCORE_ENVIRONMENT=Development lc_api_img
[+] Building 40.4s (12/12) FINISHED                                docker:default
=> [internal] load build definition from Dockerfile                                0.0s
=> => transferring dockerfile: 344B                                             0.0s
=> [internal] load metadata for mcr.microsoft.com/dotnet/sdk:7.0                0.8s
=> [internal] load .dockerignore                                                 0.0s
=> => transferring context: 2B                                                  0.0s
=> [build-env 1/6] FROM mcr.microsoft.com/dotnet/sdk:7.0@sha256:d32bd65cf5843f413e81f5d917057c82da99737cb1637e905a1a4bc2e7ec6c8d 0.0s
=> [internal] load build context                                                0.0s
=> => transferring context: 12.27kB                                             0.0s
=> CACHED [build-env 2/6] WORKDIR /app                                          0.0s
=> CACHED [build-env 3/6] COPY *.csproj ./                                     0.0s
=> CACHED [build-env 4/6] RUN dotnet restore                                    0.0s
=> [build-env 5/6] COPY . ./                                                    0.3s
=> [build-env 6/6] RUN dotnet publish -c Release -o out                       35.4s
=> [final-env 3/3] COPY --from=build-env /app/out .                            0.7s
=> exporting to image                                                            0.6s
=> => exporting layers                                                            0.3s
=> => writing image sha256:7c10db2f4c0d185a2233ebbc296fe517697dfc1f28f81d6d7e40bfd146be0e8a 0.1s
=> => naming to docker.io/library/lc_api_img                                   0.0s
WARNING: Published ports are discarded when using host network mode
info: Microsoft.Hosting.Lifetime[14]
    Now listening on: http://[::]:5001
info: Microsoft.Hosting.Lifetime[0]
    Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
    Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
    Content root path: /app

```

Figura 15 – API Docker Build

Além disso, os [endpoints](#) estão documentados com o [swagger](#). A documentação da API pode ser acessada com `/swagger` no final da URL base (rodando local).

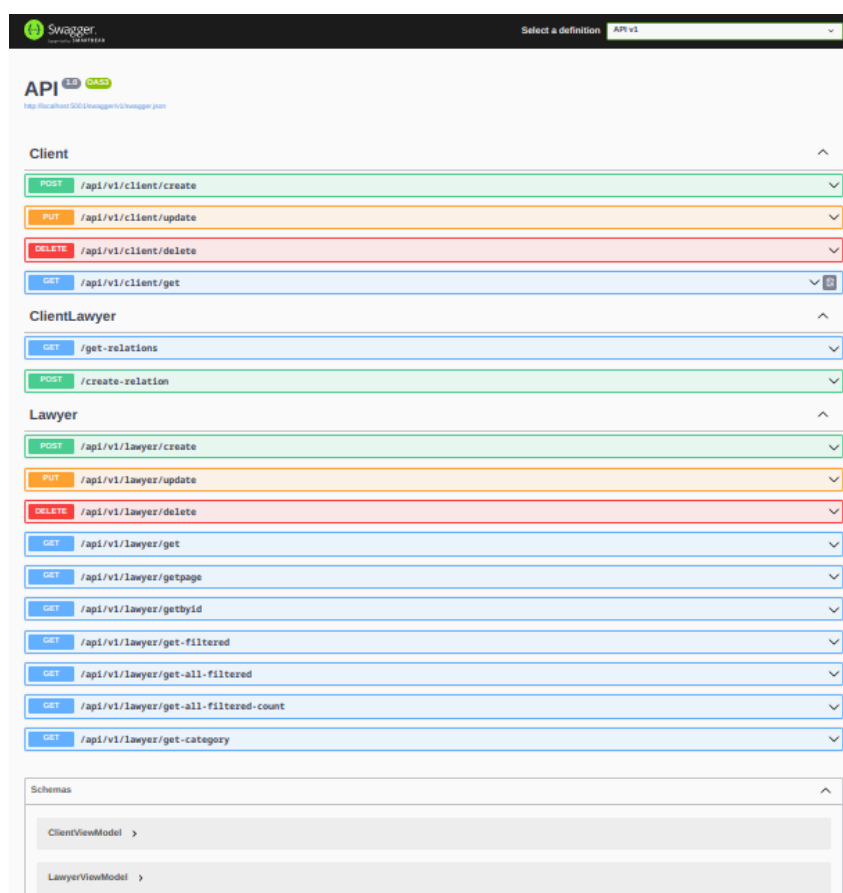


Figura 16 – API Documentation

3.2.5 IBM Cloud Object Storage

Para o armazenamento de arquivos remoto, foi escolhido trabalhar com o [IBM Cloud Object Storage - COS](#). O IBM COS é um serviço de bucket semelhante ao [AWS S3](#), no qual, temos a instância base (*'Cloud Object Storage - LawyerCenter'*), com um bucket interno para o armazenamento das fotos dos usuários. Essas fotos, possuem um link público que é registrado no banco (para ser recuperado pelo Frontend posteriormente).

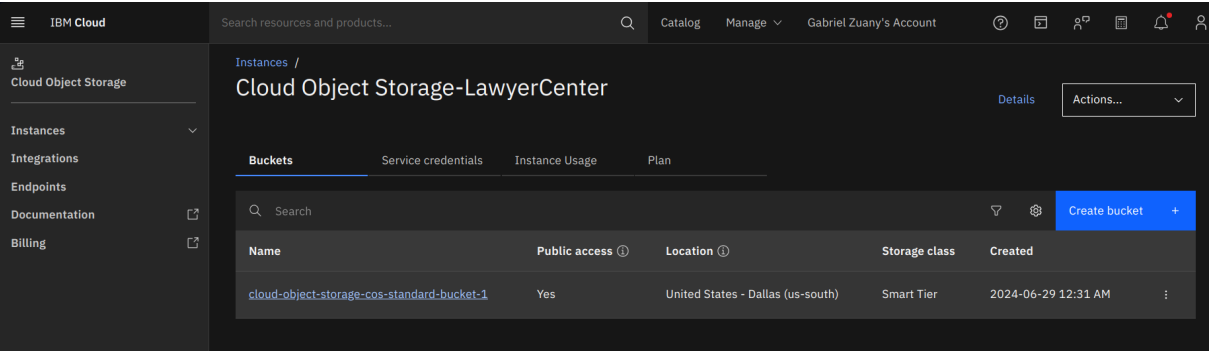


Figura 17 – IBM COS

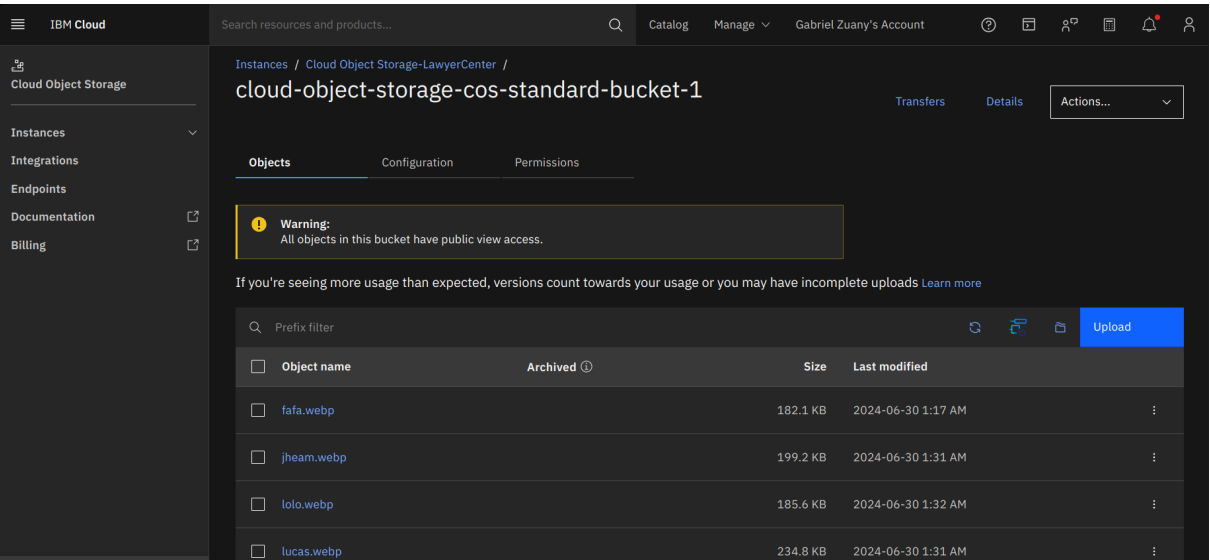


Figura 18 – IBM COS Bucket

3.2.6 Visão geral da arquitetura Backend

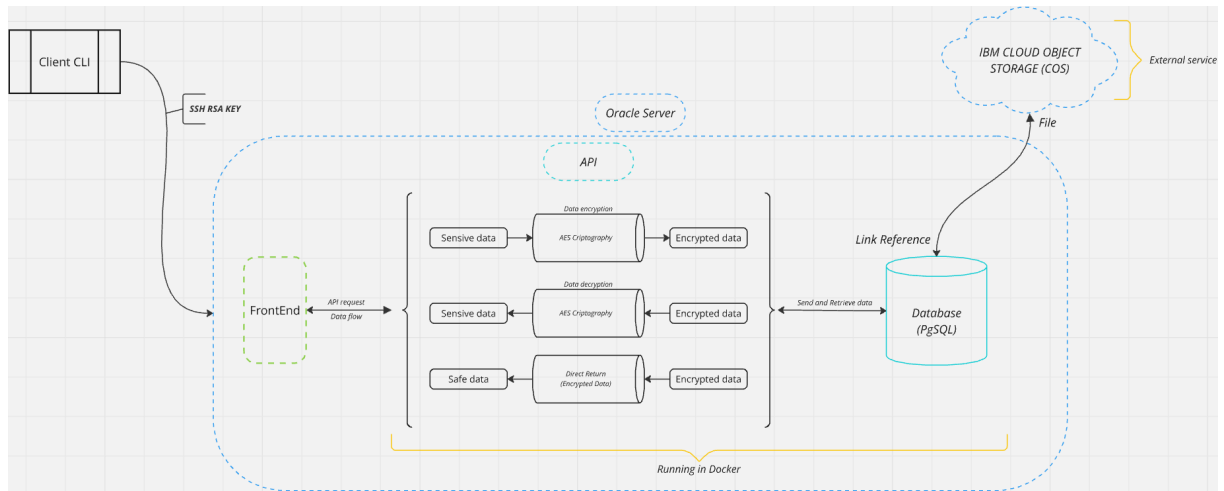


Figura 19 – Arquitetura Backend

4 Testes

4.1 API

Desenvolver testes para a API é uma prática fundamental para garantir a qualidade e robustez do sistema. Para o LawyerCenter, foi escolhido o [xUnit](#) como Framework de testes unitários. Ele é uma estrutura de teste unitário amplamente adotada na comunidade .NET devido à sua simplicidade e poderosa integração com ferramentas de build e CI/CD.

Ao criar testes unitários para as rotas da API, o objetivo principal foi validar o comportamento esperado de cada endpoint em diferentes cenários. Foram utilizados mocks ([FakeItEasy](#)) para isolar algumas dependências externas. Isso não só assegura a funcionalidade correta da API, mas também facilita a detecção precoce de regressões durante o ciclo de desenvolvimento.

Durante a execução da build, esses testes são disparados, proporcionando um feedback imediato sobre a integridade do código. Isso é crucial para manter a estabilidade do sistema e para evitar problemas que possam surgir devido a mudanças inadvertidas.

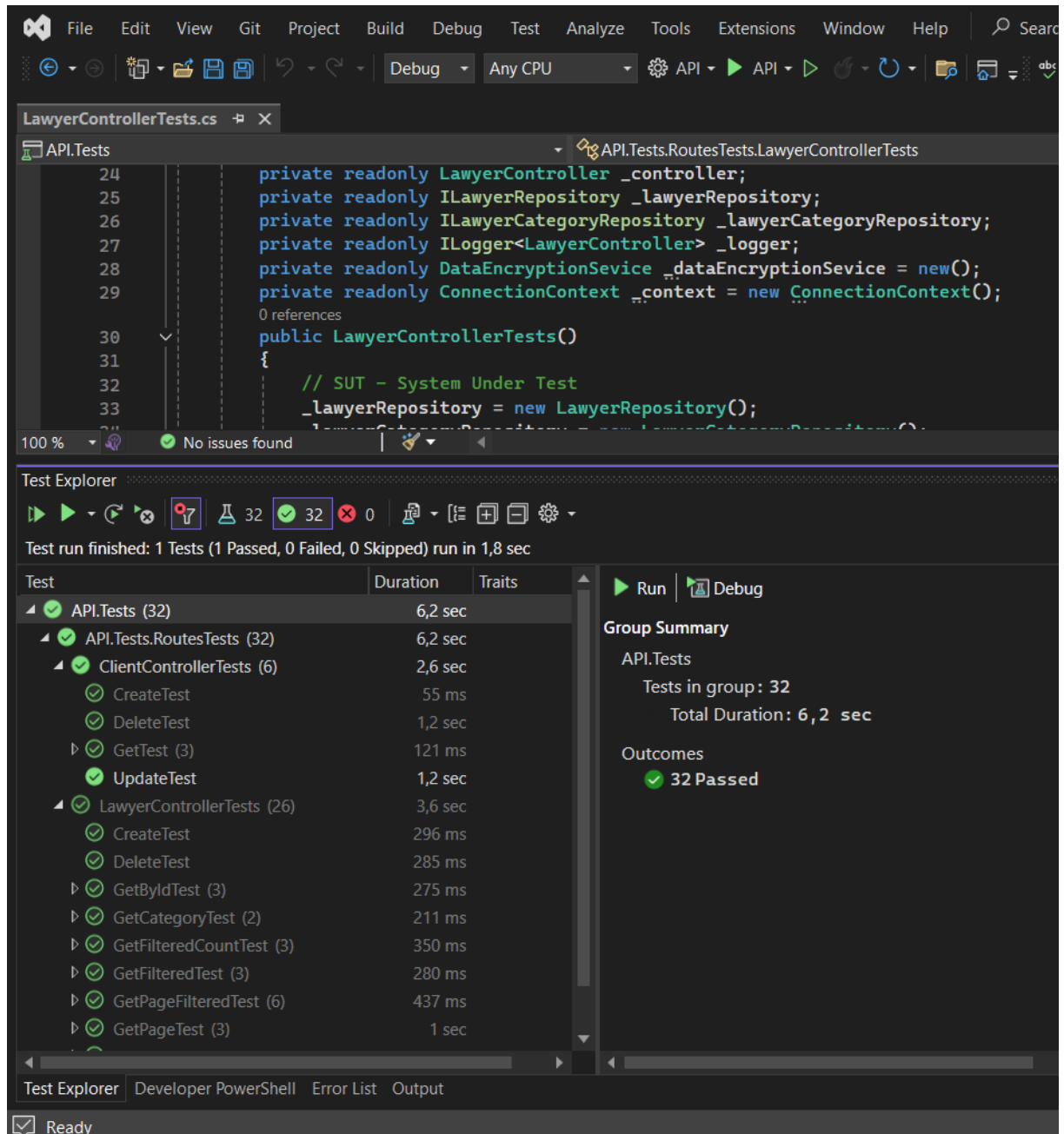


Figura 20 – XUnit

Referências