Trabalho Prático II – G07

Integração de Sistemas de Informação

Alunos:

Diogo Santos Parente, a23525
Gabriel Jablonski Torres de Melo, a21958
Juciano Gomes Farias Junior , a21875

Docente:

Óscar Ribeiro

Versão	Data
1	2024/01/07

Escola Superior de Tecnologia

Licenciatura em Engenharia de Sistemas Informáticos - PL

Barcelos, 07 de janeiro de 2024

Sumário

1. Introdução 3

2 Objetivos 4

3 Metodologia 5

3.1 Metodologias Ágeis 5

3.2 Docker 5

3.3 Entity Framework 5

4 Solução Desenvolvida 7

5 Regras de Negócio 10

5.1 Tickets 10

5.2 Reservas 10

5.3 Avisos 10

5.4 Regas 11

5.5 Visitas 11

6 Implementação na nuvem (Azure) 12

6.1 NGINX em VM Azure 12

6.2 Acesso remoto a VM 13

6.3 Configuração do Docker 14

7 Teste e documentação 15

8 Conclusão 20

1. Introdução

A contextualização deste relatório é fundamental para compreendermos o que será documentado e qual o seu propósito. Este relatório documentará o desenvolvimento de uma *web api* de gestão de condomínios, que foi concebida como uma solução abrangente para facilitar a gestão eficiente e a comunicação dentro de comunidades residenciais. Neste contexto, a aplicação visa simplificar processos, melhorar a transparência e a qualidade de vida dos moradores, bem como otimizar a administração de condomínios.

A motivação para a criação desta aplicação de gestão de condomínios baseia-se na crescente necessidade de melhorar a organização, a eficiência e a comunicação nos condomínios. Frequentemente, a gestão de condomínios envolve uma série de tarefas administrativas, comunicação fragmentada e potenciais desafios de segurança. Esta aplicação foi concebida para abordar essas questões, tornando a gestão mais simples, ágil e segura.

Neste documento, serão esclarecidos a metodologia de trabalho e a proposta de sistema a ser desenvolvida.

2 Objetivos

O objetivo deste projeto é desenvolver uma Web API robusta e escalável, destinada especificamente à gestão de condomínios. Essa API será projetada para lidar com um volume sob medida de transações e consultas de dados, mantendo um desempenho consistente mesmo sob crescente demanda pré definida. A escalabilidade é um aspecto central, permitindo que a aplicação se adapte facilmente a um aumento no número de usuários ou a um crescimento na complexidade das operações de gestão. Este projeto visa proporcionar uma ferramenta integral para a administração de condomínios, simplificando processos como comunicação entre moradores e administração, gerenciamento financeiro e operacional, e oferecendo uma plataforma intuitiva e acessível para todos os envolvidos.

Este projeto também visa consolidar o aprendizado em sala sobre as tecnologias C# , .NET, base de dados SQL etc. Outrossim, o projeto deverá ser publicado em um serviço de nuvem, ser devidamente testado e documentado, como também integrado a um cliente.

3 Metodologia

3.1 Metodologias Ágeis

A adoção de metodologias ágeis para o desenvolvimento desta aplicação não é apenas uma escolha, mas uma estratégia fundamental para alcançar eficiência e adaptabilidade no processo de criação. As metodologias ágeis, centradas em práticas como Scrum e Kanban, permitiram que a equipa trabalhasse de maneira iterativa e incremental. Isso significou dividir o projeto em ciclos de trabalho menores.

3.2 Docker

O emprego de contêineres Docker no projeto representa um passo significativo em direção à eficiência operacional e consistência técnica. Docker, uma plataforma de contêinerização líder de mercado, é utilizado especificamente para encapsular a base de dados PostgreSQL. Ao utilizar contêineres Docker, asseguramos um ambiente isolado e padronizado para a base de dados, o que elimina os problemas comuns relacionados a 'funciona no meu computador'. Isso significa que a base de dados pode ser executada de maneira uniforme em diferentes ambientes de desenvolvimento, teste e produção, garantindo uma implantação suave e sem contratempos.

3.3 Entity Framework

O Entity Framework, um framework ORM (Object-Relational Mapping) robusto, é dentro da linguagem C#. Isso abstrai a complexidade das consultas SQL e permite que a uma peça central na interação entre nossa aplicação e a base de dados PostgreSQL. Através do Entity Framework, transformamos as tabelas, colunas e relações do banco de dados em objetos e propriedades equipe de desenvolvimento manipule dados de forma mais intuitiva e alinhada com os princípios da programação orientada a objetos

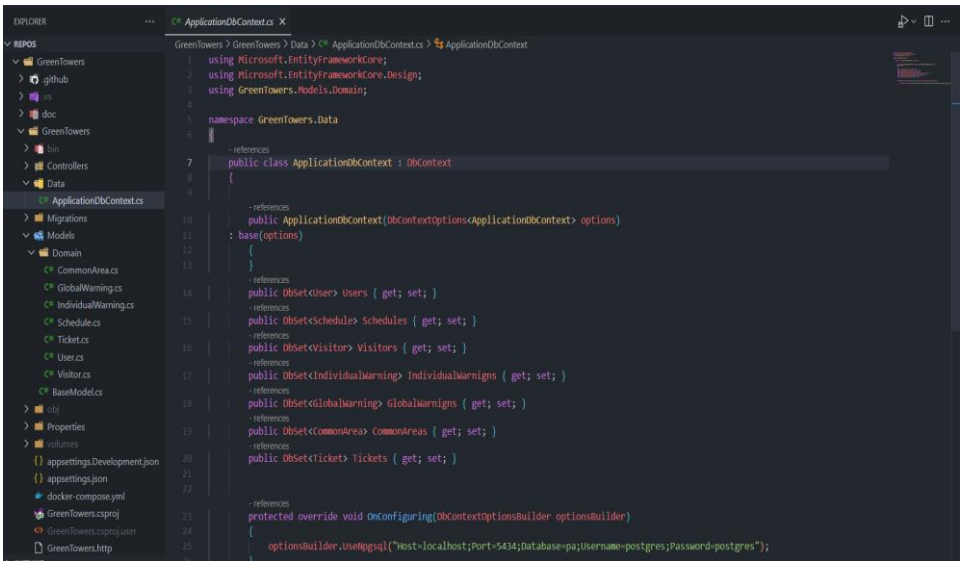

Imagem – Uso do entity framework para mapear as entidades de domínio para a base de dados.

4 Solução Desenvolvida

A construção do backend da aplicação foi realizada utilizando a linguagem C# em conjunto com a versão 8.0 do .NET, uma plataforma de desenvolvimento moderna e versátil. Esta escolha nos permitiu criar uma API robusta e segura, aproveitando as mais recentes práticas de desenvolvimento e características da linguagem C#, como a programação assíncrona. O design do backend foi orientado para suportar cargas de trabalho pre definidas e um número de requisições simultâneas, mantendo um tempo de resposta rápido e uma operação eficiente. A arquitetura da API foi planejada para ser modular e extensível, facilitando futuras atualizações e manutenção, além de promover a reutilização do código. A integração com outras ferramentas e tecnologias, como o Entity Framework para ORM, também foi uma consideração chave, assegurando uma interação harmoniosa e eficiente com a base de dados.

Até o momento de finalização deste relatório, o cliente que consome o serviço desenvolvido não foi completamente implementado. A aplicação de consumo escolhida foi uma mobile android desenvolvida com Kotlin. Utiliza-se, juntamente com bibliotecas como OkHttp ou Retrofit para fazer chamadas de rede à API C#. As requisições são feitas para os endpoints adequados, passando dados no corpo da requisição (para POST, PUT, DELETE) ou através de parâmetros de URL.

Além das tecnologias citadas, a estratégia de segurança, atenticação e autorização utilizada foi a JWT (Javascript Web Token). JWT é um padrão para a transmissão segura de informações entre duas partes como um objeto JSON. Essa informação pode ser verificada e confiável porque é assinada digitalmente. JWTs são compactos e, portanto, podem ser enviados através de uma URL, método POST ou no cabeçalho HTTP (escolhido pela equipa), aumentando a flexibilidade em diferentes contextos de transmissão de dados. Em nossa aplicação, os JWTs desempenham um papel fundamental na autenticação e autorização dos usuários. Quando um usuário se autentica no sistema, um token JWT é gerado e enviado para o usuário. Este token contém várias informações cruciais (claims), como identificador (ID) do usuário e seus papéis (roles) no sistema.

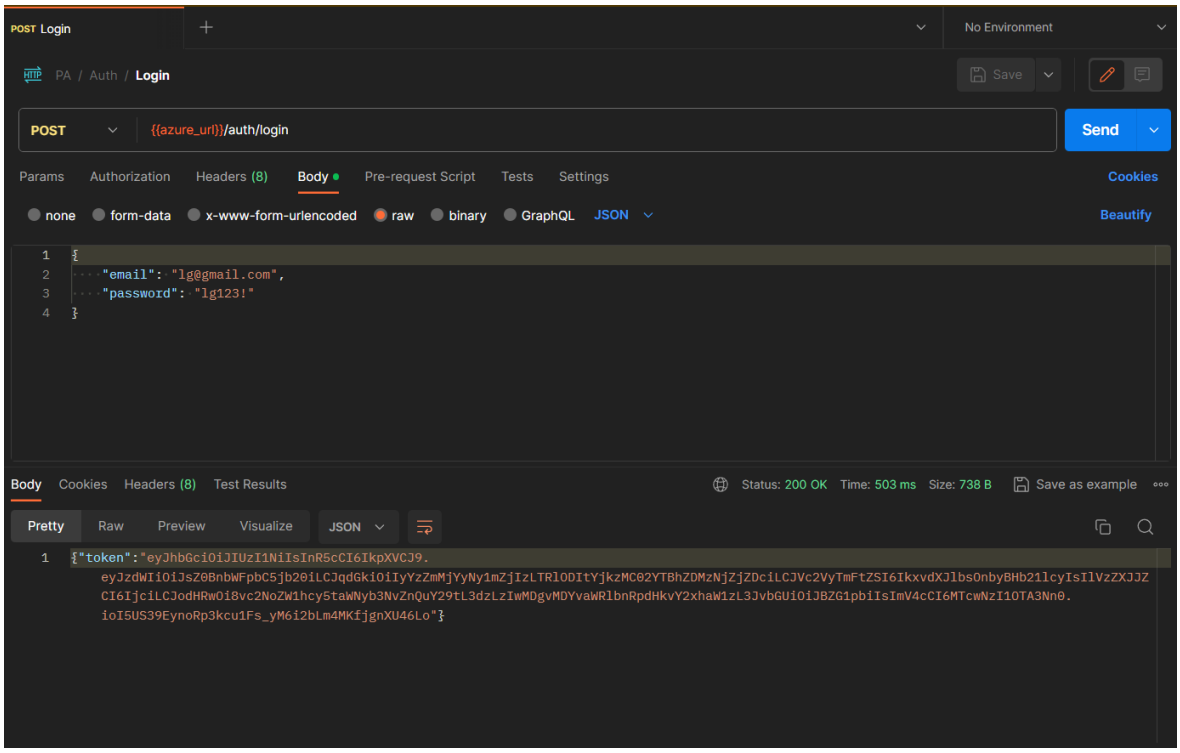

Imagem 1 – Login do a estratégia JWT.

Toda a API foi documentada e testada por meio do Postman. Ele proporciona uma plataforma sólida para documentação e testes, contribuindo significativamente para a qualidade, confiabilidade e sucesso do projeto.

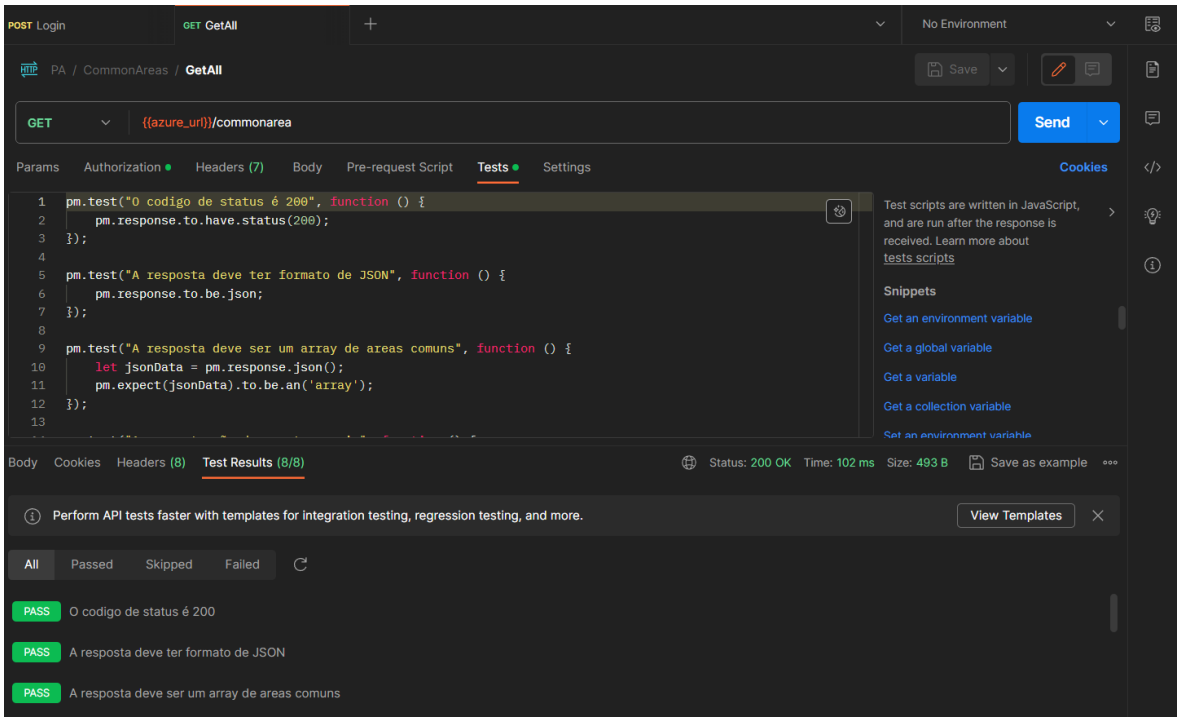

Image 2 – Exemplo de teste de endpoint pelo Postman

5 Regras de Negócio

5.1 Tickets

Na gestão de um condomínio, enfrentar manutenções, imprevistos e mudanças é comum. Para facilitar a comunicação dessas ocorrências com o administrador, implementamos a funcionalidade de tickets. Esta ferramenta permite que moradores relatem problemas ou necessidades específicas, detalhando a natureza e a urgência do assunto. Ao submeter um ticket, o usuário pode acompanhar o status da sua solicitação e receber uma resposta direta do gestor do condomínio. Essa interação é projetada para ser intuitiva e eficiente, assegurando que os problemas dos moradores sejam atendidos prontamente e de maneira organizada.

5.2 Reservas

Para gerenciar o uso de espaços comuns em condomínios, desenvolvemos uma funcionalidade de reservas. Através dela, os moradores podem agendar um local comum para eventos pessoais, garantindo a disponibilidade no dia desejado. Essa funcionalidade inclui uma regra de uso, onde cada morador pode fazer uma reserva em um determinado espaço apenas a cada dois dias. Isso promove um uso justo e equitativo dos espaços comuns, evitando conflitos e sobrecarga de reservas.

5.3 Avisos

Os avisos são uma parte vital da comunicação dentro do condomínio. Implementamos um sistema onde o gestor pode enviar avisos tanto de forma global, para todos os moradores, quanto individualmente, para moradores específicos. Os moradores, utilizando seus tokens, podem acessar uma lista de todos os avisos relevantes associados ao seu ID. Isso facilita a disseminação de informações importantes e assegura que os moradores estejam sempre informados sobre os acontecimentos no condomínio.

5.4 Regas

A funcionalidade de regas, ainda em desenvolvimento até a produção deste relatório, está projetada para dar autorização ao recepcionista para controlar os sistemas de irrigação do condomínio. Essa função permitirá ligar e desligar as regas conforme necessário, promovendo um uso eficiente da água e a manutenção adequada das áreas verdes. A integração dessa funcionalidade na plataforma representa nosso compromisso com a sustentabilidade e a gestão eficiente dos recursos.

5.5 Visitas

O registro de visitantes é essencial para a segurança do condomínio. A funcionalidade de cadastro de visitas facilita para o recepcionista registrar todas as entradas de não moradores. Isso inclui o armazenamento de informações relevantes, como nome do visitante, data e hora da visita, e o apartamento ou morador visitado. Esse registro detalhado auxilia na segurança e na gestão de visitantes, garantindo tranquilidade para os moradores e eficiência na administração do condomínio.

6 Implementação na nuvem (Azure)

Foi estabelecida uma infraestrutura na nuvem usando o Azure para hospedar a aplicação web. Utilizando uma VM (Máquina Virtual) com Ubuntu 20.04, configurou-se o NGINX como um servidor proxy reverso para encaminhar as requisições para a aplicação .NET que está rodando em Kestrel, o servidor web interno do ASP.NET Core. Vamos detalhar um pouco mais sobre cada aspecto da configuração e as práticas empregadas:

6.1 NGINX em VM Azure

A configuração do NGINX na VM Azure foi estabelecida para gerenciar o tráfego da aplicação. O arquivo de configuração do NGINX mostra que ele está configurado para ouvir na porta 80, que é a porta padrão para tráfego HTTP. O proxy pass é configurado para redirecionar requisições para a aplicação .NET Core rodando na porta 5000. Além disso, a configuração inclui cabeçalhos para aprimorar o processamento de WebSocket (que não foi utilizado) e a implementação de CORS (Cross-Origin Resource Sharing), permitindo que a API seja acessada de domínios diferentes do domínio de hospedagem da aplicação. Isso é essencial para aplicações modernas que muitas vezes precisam interagir com outras aplicações ou serviços hospedados em diferentes locais.

```
server {
    listen 80;  # Porta em que o Nginx escuta

    server_name example.com;  # Substitua por seu domínio ou endereço IP

    location / {
        proxy_pass http://localhost:5000;  # Encaminha as requisições para a aplicação em localhost na porta 5202
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;

        # Adicione os seguintes headers para habilitar CORS
        if ($request_method = 'OPTIONS') {
            add_header 'Access-Control-Allow-Origin' '*';  # Permitir de qualquer origem
            add_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS';  # Métodos HTTP permitidos
            add_header 'Access-Control-Allow-Headers' 'DNT,User-Agent,X-Requested-With,If-Modified-Since,Cache-Control,Content-Type,Range';
            add_header 'Access-Control-Max-Age' 1728000;  # Tempo que o navegador deve armazenar o resultado do preflight request
            add_header 'Content-Type' 'text/plain; charset=utf-8';
            add_header 'Content-Length' 0;
            return 204;  # Retorna resposta para o preflight request sem conteúdo
        }
        add_header 'Access-Control-Allow-Origin' '*';
        add_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS';
        add_header 'Access-Control-Allow-Headers' 'DNT,User-Agent,X-Requested-With,If-Modified-Since,Cache-Control,Content-Type,Range';
    }
```

Imagem – Configuraçãod o NGINX

6.2 Acesso remoto a VM

Para configurar a VM foi-se criada uma chave SSH para acesso por meio do software remina. A chave SSH, que é uma forma de autenticação segura, permite que acesse e gerencie as VM sem expor senhas ou outros métodos de autenticação mais vulneráveis.

Imagem – Chave SSH para acesso remoto

6.3 Configuração do Docker

Adicionalmente, a utilização do Docker para orquestrar a base de dados PostgreSQL serviu de isolamento de ambiente e facilitou a implantação e portabilidade entre diferentes ambientes de desenvolvimento e produção. Isso, junto com a opção de usar o comando "docker-compose up -d" para rodar os serviços em segundo plano, facilitou a publicação e desenvolvimento. Finalmente, o uso do comando dotnet publish -c Release -o out indica que a aplicação estará compilada na pasta /out, otimizando-a para desempenho e assegurando que todos os artefatos necessários sejam compilados para o diretório de saída especificado. Este conjunto de práticas demonstra uma aplicação eficiente das tecnologias de nuvem, contêineres e práticas de desenvolvimento modernas para criar uma solução de hospedagem escalável, segura e confiável para a sua aplicação da disciplina.

Imagem – Compilação da aplicação

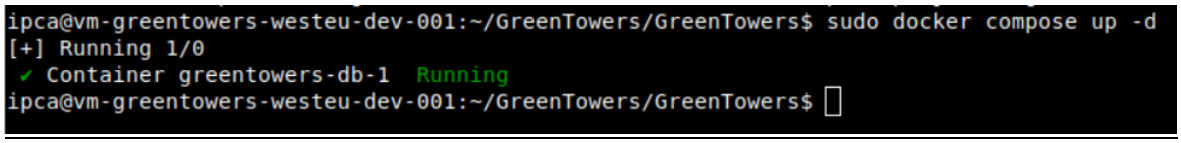

Imagem – Comando para rodar os serviços configurados

Imagem – Por fim, reload do servidor nginx

7 Teste e documentação

Para a documentação em ambiente de desenvolvimento da API, foi utilizado o Swagger com suas principais funcionalidas, como: listagem de schemas, formado das chamadas http, campos obrigatórios, formato da respostas e da requisição, autenticação e proteção dos endpoints de teste etc.

Imagem – Autenticação swagger

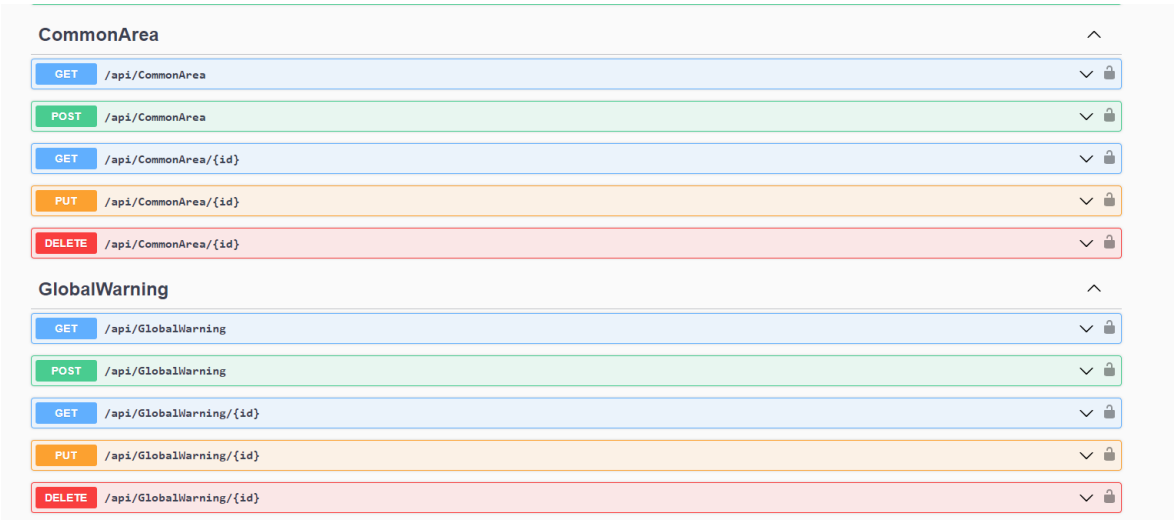

Imagem- Endpoints para avisos globais e áreas comuns

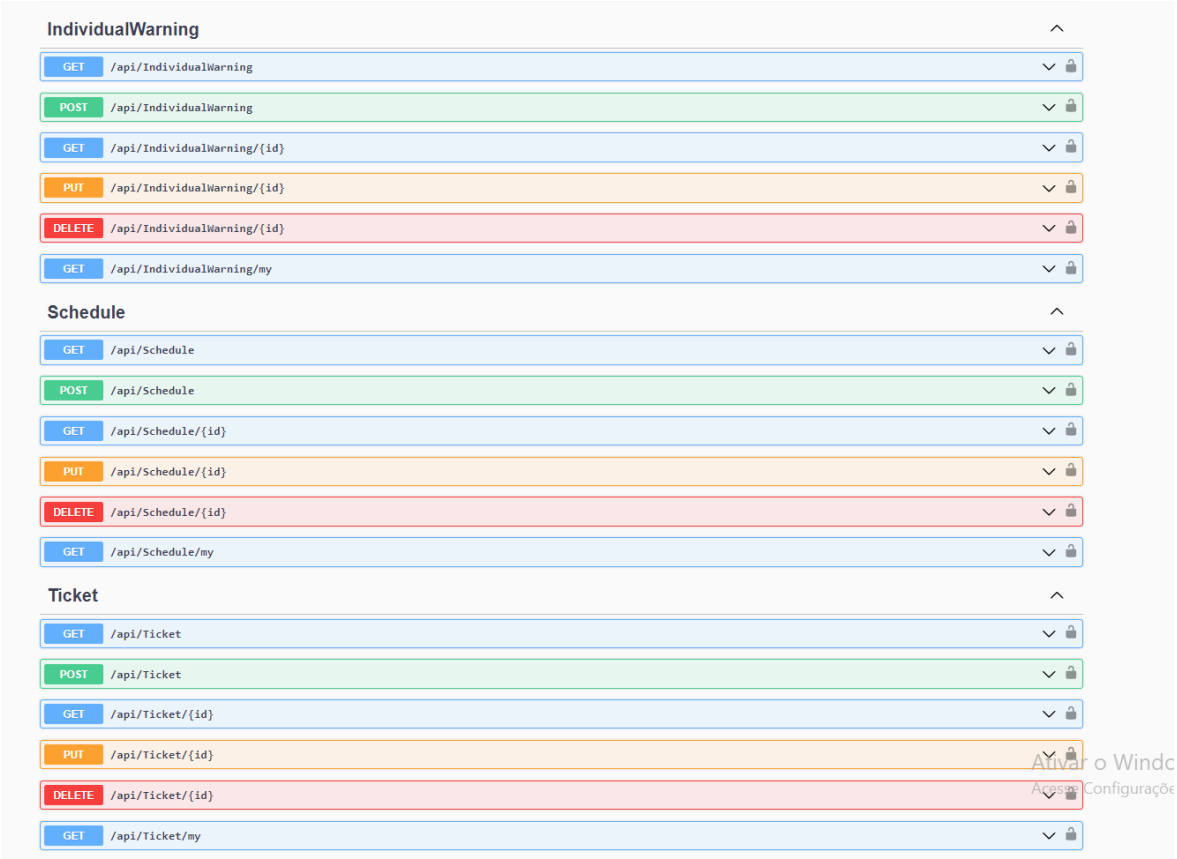

Imagem – Endpoints para agendamento, avisos e tickets

Imagem – Endpoints para usuários, visitantes e alguns dos schemas utilizados

Para o ambiente de produção e teste dos endpoints implementados, aplicou-se testes por meio do postman e usou-se suas principais funcionalidades de documentação. Abaixo, pode-se visualizar alguns testes automatizados em cada um dos verbos HTTP utilizados por toda a API.

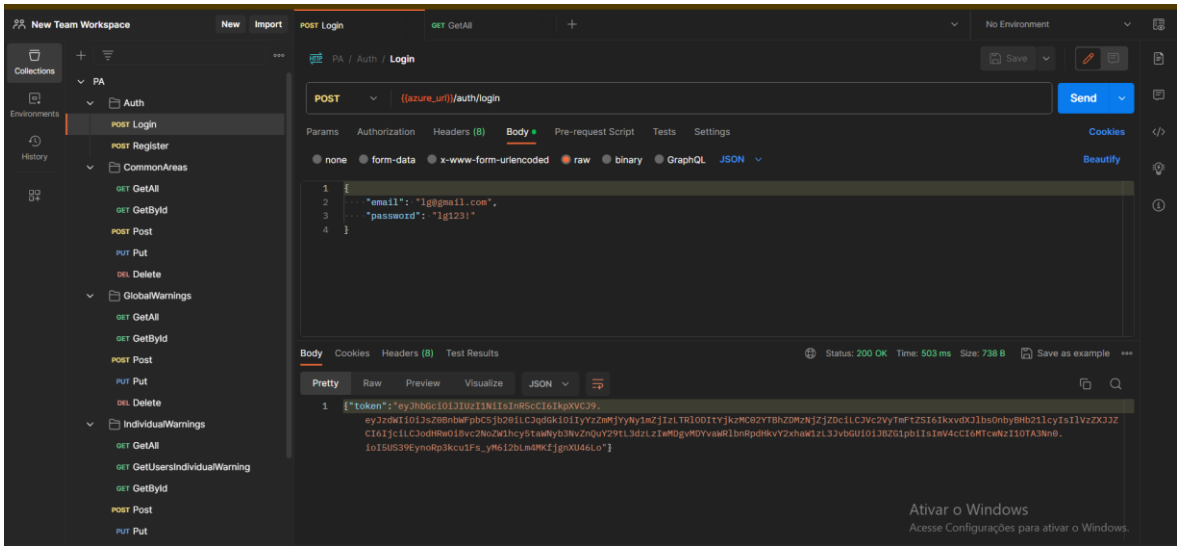

Imagem – Endpoint de login do postman

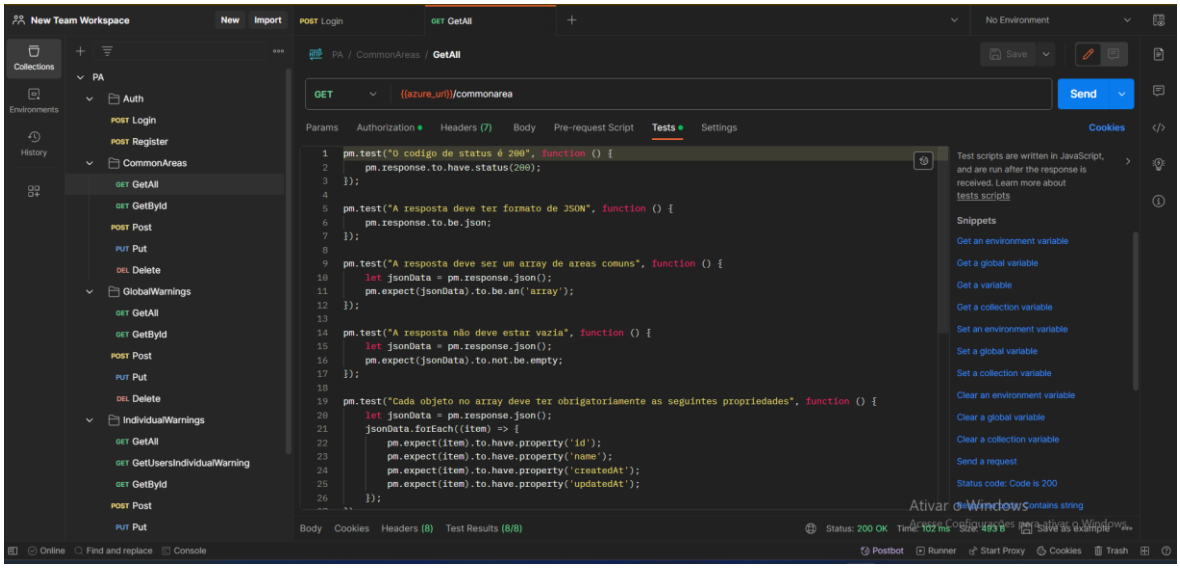

Imagem – Testes automatizados do GET de áreas comum

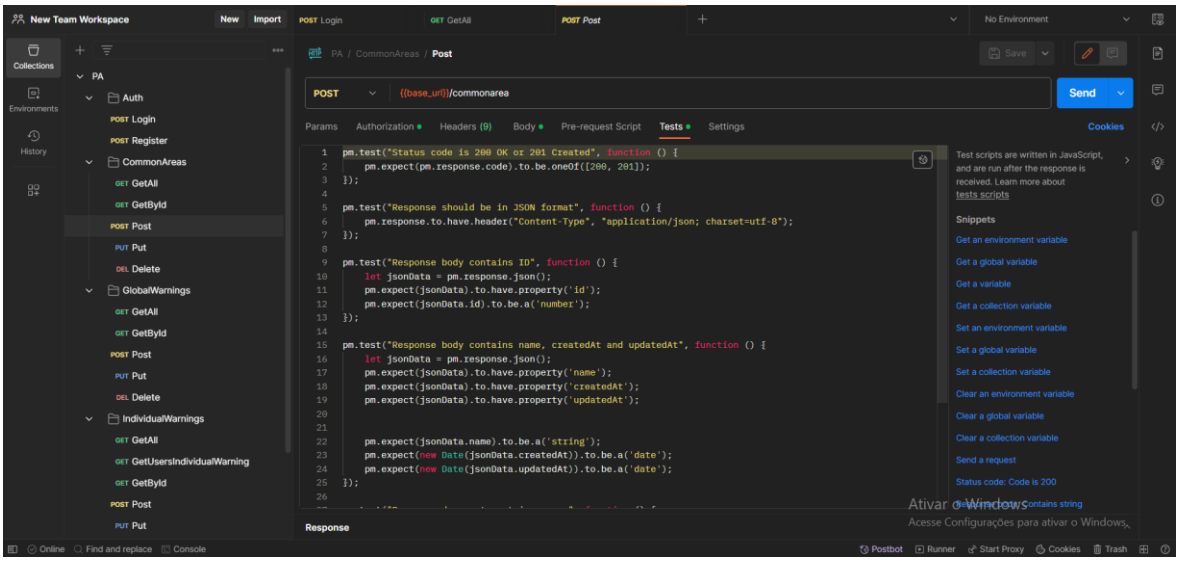

Imagem – Testes do POST

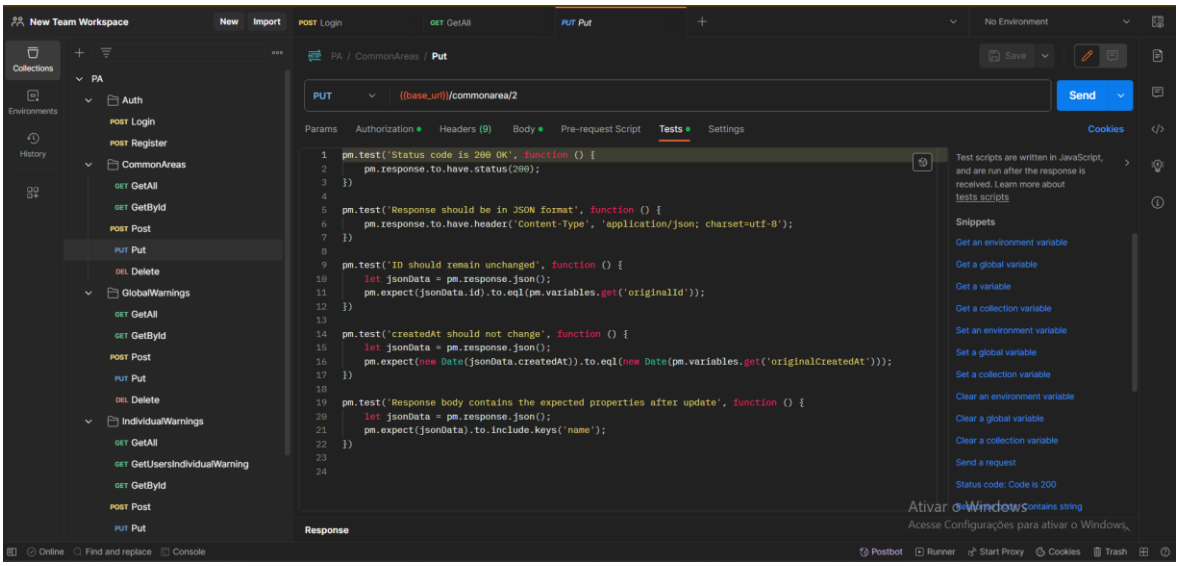

Imagem – Testes de PUT

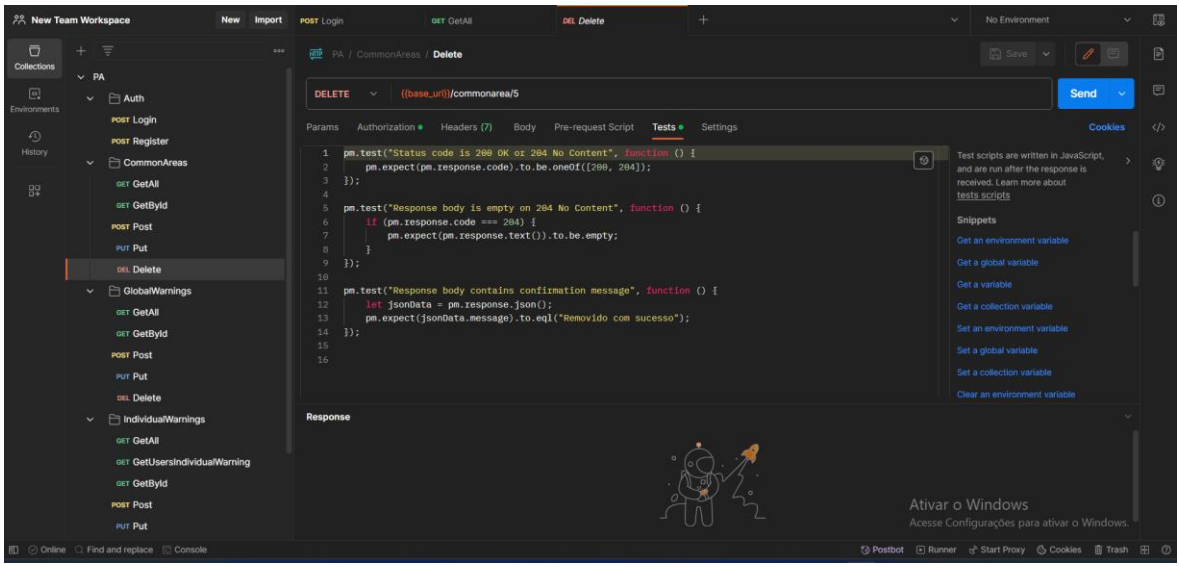

Imagem – Testes de DELETE

8 Conclusão

Na conclusão deste projeto, podemos afirmar que os objetivos estabelecidos foram alcançados com sucesso. A aplicação de gestão de condomínios, através da sua robustez e escalabilidade, poderá proporcionar uma melhoria tangível na eficiência operacional e na comunicação entre moradores e administração. Enfrentamos desafios significativos, especialmente no que se refere à segurança e à integração de diferentes tecnologias, mas estes foram superados com a adoção de práticas de desenvolvimento ágeis e soluções de infraestrutura modernas, como o uso de contêineres Docker e a implementação na nuvem Azure. Olhando para o futuro, vemos um caminho claro para melhorias contínuas e expansões da aplicação, como refatoração e aplicação de uma arquitetura ou filosofica de desenvolvimento (como DDD), a incorporação de novas funcionalidades que respondam às demandas emergentes dos usuários e a otimização do desempenho com base em dados analíticos e feedback dos usuários. Este projeto não só cumpriu suas promessas iniciais mas também estabeleceu uma base sólida para inovação e crescimento contínuos.

9 Referências

https://learn.microsoft.com/en-us/azure/?product=popular

https://learn.microsoft.com/en-us/dotnet/

https://docs.docker.com/get-started/overview/

https://learn.microsoft.com/en-us/dotnet/csharp/

https://nginx.org/en/docs/

https://jwt.io/introduction