Marca Ponto API: Desafio de Software Engineering

Laís Van Vossen

1 Introdução

Este artigo apresenta o processo de resolução do desafio de Software Engineering. O desafio consiste em criar uma solução para um problema de marcação de ponto em que milhares de pessoas mandam requisições em um curto período de tempo para marcar os horários de trabalho. As requisições para o sistema são de que deve atender dispositivos IoT, Mobile e Web, deve ser uma solução backend e deve integrar com um sistema legado cujo tempo de resposta é em torno de 10 segundos.

Levando em consideração os pontos levantados acima, a solução considerada foi a de fazer uma API implementada em .NET 6 para lidar com as múltiplas requisições simultâneas, visto que o modelo de API atende todos os dispositivos necessários, é backend e conseguirá se comunicar com o sistema legado. Para o ambiente de desenvolvimento, foi utilizado o Visual Studio 2022 da Microsoft.

2 Desenvolvimento

O processo de desenvolvimento foi separado em etapas. A primeira é a de criar os endpoints necessários para atender a marcação de ponto e para a comunicação com a API legado. O segundo é a implementação e pesquisa de técnicas para tornar o sistema apto a atender as milhares de requisições simultâneas conforme solicitado. A terceira etapa é a de subir o sistema para um cloud provider e a última etapa é a de realizar testes.

2.1 Criação de Endpoints

Foram criados os endpoints para lidar com requisições e suas características estão descritas na Tabela 1. Além disso, foi adicionado a classe *Requisicao*, para lidar com as informações enviadas para a API, sendo elas o *includedAt*, do tipo *DateTime*, que é a data e hora que ocorreu a marcação de ponto, *employeeId*, do tipo *int*, que é o código do funcionário que fez a marcação de ponto e o *employerId*, do tipo *int*, que é o código da empresa em que o funcionário trabalha.

2.2 Aprimorando os Endpoints

Para garantir que a API possa atender as múltiplas requisições simultâneas, foram utilizados métodos assíncronos nos pontos que representavam um gargalo no código. Esses pontos envolvem

Tabela 1: Mapeamento de Endpoints

Tipo de Requisição	**Nome da Rota**	**O que faz**	**Privacidade**
POST	Marcar	Recebe os dados de data e hora e códigos de funcionário e empresa	Pública
POST	MarcarApiLegado	Envia a requisição para a API legado e retorna se conseguiu salvar	Privada

a comunicação com a API legado, então o método de envio das informações, o *PostAsJsonAsync* que envia um objeto da classe requisição e o método *ReadAsStringAsync* que lê o retorno da API legado utilizam a programação assíncrona.

Além disso, para garantir que o sistema legado não será sobrecarregado de requisições simultâneas e que vai conseguir atender todas elas, foi implementado um sistema de semáforo em torno do ponto onde a API legado é chamada, garantindo assim que um número limitado de threads terá acesso a aquele ponto ao mesmo tempo e evitando que ocorram erros. [2]

2.3 Executando em um Cloud Provider

Para executar o sistema em nuvem, o provedor escolhido foi o Azure da Microsoft, graças a sua fácil integração com a linguagem de programação escolhida e o ambiente de desenvolvimento, além de ampla documentação acessível [4]. Para fazer a marcação do ponto, a API responde no link `https://marcapontoapi.azurewebsites.net/BatePonto/Marcar?IncludedAt=<Data>&EmployeeId=<NumeroFuncionario>&EmployerId=<NumeroEmpresa>`, onde os valores <Data>, <NumeroFuncionario> e <NumeroEmpresa> podem ser substituídos pela data e hora em que o ponto foi marcado, pelo código do funcionário e pelo código da empresa, respectivamente.

2.4 Testes Automatizados

Os testes feitos nessa sessão foram feitos chamando o sistema executando na nuvem. O primeiro tipo de teste realizado foi feito dentro do programa Postman, que é um serviço de consumo de APIs e que possui ferramentas para executar múltiplas requisições de forma não paralela e verificar o resultado obtido [5]. Nesse primeiro teste, o endpoint Marcar foi chamado 100 vezes e retornou o código HTTP 200 em todas elas, como demonstrado na Figura 1.

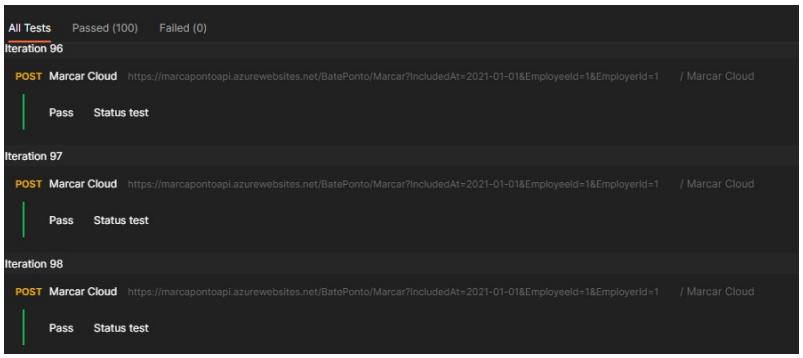

Figura 1: Imagem do resultado dos testes.

O problema dos testes realizados no Postman é que eles não abrangem o cenário inteiro, por fazer apenas requisições lineares, não é possível observar como o sistema se comporta quando múltiplas requisições são feitas ao mesmo tempo, logo é necessário um outro mecanismo de testes.

Para testar o sistema no cenário em que recebe múltiplas requisições simultâneas, foi utilizado um código em Python [1], adaptado para o problema, que cria threads para chamar a API ao mesmo tempo e controla o tempo até cada thread receber uma resposta. Os resultados de um dos testes se encontra na Figura 2. Os parâmetros configurados no código para gerar esse resultado foram a utilização de 5 threads, fazendo 10 requisições para o endereço `https://marcapontoapi.azurewebsites.net/BatePonto/Marcar`.

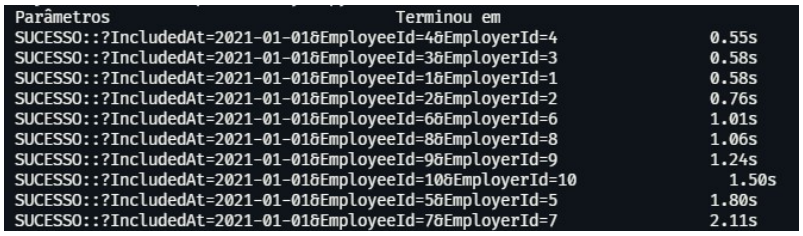

Figura 2: Imagem do resultado dos testes com threads.

3 Considerações Finais

O sistema desenvolvido contou com a utilização de threads, semáforos e programação assíncrona para lidar com a grande quantidade de requisições simultâneas. Como pontos de melhoria fica a verificação dos tipos na classe requisição, já que o sistema pretende atender uma quantidade muito grande de funcionários, deve-se validar se o tipo inteiro para o atributo EmployeeId é o suficiente ou pode ocasionar erros de Overflow [3]. Além disso, é necessário identificar quais seriam os melhores parâmetros de número de threads do semáforo para garantir maior eficiência e rapidez sem sobrecarregar o sistema legado.

Referências

[1] Tyler Burdsall. **How to run asynchronous web requests in parallel with Python 3.5 (without aiohttp)**. Online. Acessado em 26/01/2022, `https://medium.com/hackernoon/how-to-run-asynchronous-web-requests-in-parallel-with-python-3-5-without-aiohttp-264dc0f8546`.

[2] Sai Kumar Koona. **Understanding Semaphore in .NET Core**. Online. Acessado em 27/01/2022, `https://www.c-sharpcorner.com/article/understanding-semaphore-in-net-core/`.

[3] Microsoft. **OverflowException Class**. Online. Acessado em 26/01/2022, `https://docs.microsoft.com/en-us/dotnet/api/system.overflowexception?view=net-6.0`.

[4] Microsoft. **Quickstart: Deploy an ASP.NET web app**. Online. Acessado em 26/01/2022, `https://docs.microsoft.com/en-us/azure/app-service/quickstart-dotnetcore?tabs=net60&pivots=development-environment-vs#launch-the-publish-wizard?utm_source=aspnet-start-page&utm_campaign=vside`.

[5] Postman. **Writing tests**. Online. Acessado em 26/01/2022, `https://learning.postman.com/docs/writing-scripts/test-scripts/`.