

APONTA: UM SISTEMA GENÉRICO DE AGENDAMENTOS DE CLÍNICAS ODONTOLÓGICAS

APONTA - GENERAL BOOKING SYSTEM FOR DENTAL CLINICS

Gabriel José Sillva¹ 

Márcia Cristina Dadalto Pascutti² 

Resumo: Com a crescente utilização da tecnologia nas empresas do Brasil, é tendência a implementação de automação de tarefas manuais ou sistemas que permitem controle dos seus processos. Entretanto, a nova revolução da tecnologia não está afetando microempreendedores, que ainda utilizam formas manuais para concretizar suas formas de trabalho, devido ao encarecimento em forjar automatizações e sistemas com funcionalidades para o dia a dia na empresa. Um deles é o meio da Saúde, na qual utiliza agendas físicas para fazer o agendamento de pacientes, um processo comum para controlar os horários reservados. Por isso, foi proposto implementar um serviço provido por uma API, que sirva de forma genérica e barata para várias clínicas odontológicas de todo o país, para que essas empresas comecem a utilizar a tecnologia para promover mais qualidade e eficiência em seus produtos e serviços. Esse artigo transmite o conhecimento fundamental sobre os conceitos e funcionamento das aplicações backends, sendo as APIs, e como elas se comunicam utilizando as rotas, para trafegar as requisições e suas respostas. Além disso, é desenvolvida uma aplicação de uma API utilizando tecnologias recentes como TypeScript, PostgreSQL e Containers Docker. É explicado seu funcionamento, benefícios e seus principais aspectos que fazem esse software genérico ser promissor e otimizar que é o processo manual das agendas físicas das clínicas odontológicas do Brasil.

Palavras-chave: Agendamentos. TypeScript. PostgreSQL. Sistema backend. API.

Abstract: With the increasing use of technology in companies in Brazil, there is a tendency to implement automation of manual tasks or systems that allow control of their processes. However, the new technology revolution is not affecting micro-entrepreneurs, who still use manual methods to carry out their ways of working, due to the increase in the cost of forging automation and systems with functionalities for the company's day-to-day activities. One of them is the Healthcare sector, in which physical diaries are used to schedule patients, a common process for controlling reserved times. Therefore, it was proposed to implement a service provided by an API, which serves in a generic and cheap way for several dental clinics across the country, so that these companies can begin to use technology to promote greater quality and efficiency in their products and services. This article conveys fundamental knowledge about the concepts and functioning of backend applications, such as APIs, and how they communicate using routes, to transport requests and their responses. Furthermore, an API application is developed using recent technologies such as TypeScript, PostgreSQL and Docker

¹ Tecnólogo em Análise e Desenvolvimento de Sistemas, Campus Umuarama, Instituto Federal do Paraná, contato.gabrieljosesilva@gmail.com.

² Mestre em Informática, Instituto Federal do Paraná – Campus Umuarama, marcia.pascutti@ifpr.edu.br.

Containers. Its operation, benefits and main aspects that make this generic software promising and optimizing the manual process of physical diaries in dental clinics in Brazil are explained.

Keywords: Appointments, TypeScript, PostgreSQL, backend system, API.

1 INTRODUÇÃO

A evolução e aplicação da tecnologia da informação têm repercussões em várias áreas, uma vez que seu principal propósito é aprimorar processos mediante soluções tecnológicas. Essa tendência pode ser percebida em uma pesquisa realizada em maio de 2023 no Brasil, onde foi registrada a marca de 464 milhões de dispositivos digitais totais no país, ou seja, 2,2 itens por indivíduo, sendo entre eles, computadores, tablets ou smartphones (FGVCIA, 2023).

Com a grande utilização desses aparelhos é comum, em várias empresas, o uso de sistemas para controle de estoque, venda, compra, gerenciamento financeiro ou pessoal. O principal aspecto que faz com que esses sistemas se tornem atraentes são seus grandes pontos positivos, como, por exemplo, facilitar processos manuais, promover mais qualidade no trabalho dos colaboradores, segurança com os dados dos clientes, rápido processamento e histórico dos dados armazenados.

Por isso, muitas empresas do Brasil têm investido nesse recurso, isso porque chegam a gastar até 9% do seu lucro em recursos tecnológicos (FGVCIA, 2023). Esses gastos se concentram em planos de softwares ou serviços bem conhecidos, como Pacote Office, Google, SASP, *Cloud* entre outros. Tudo isso porque esse gasto retorna como lucro em longo prazo.

Sendo assim, após uma observação feita no município de Altônia - PR, é perceptível que o uso de sistemas não é comum, devido às regras de negócios de cada empresa serem muito particulares ao próprio funcionamento. Então, como solução para isso, foi elaborado um projeto de desenvolvimento de um software backend, para criação de um sistema genérico que supra as principais regras de negócios dessas empresas do município.

Assim, para se desenvolver esse sistema genérico foi utilizada a linguagem de programação TypeScript, com uma criação de uma Interface de Programação de Aplicação, com banco de dados PostgreSQL, com o ambiente local, gerenciado e construído e orquestrado com imagens pelo Docker.

Este artigo tem como objetivo descrever o processo de análise de requisitos, desenvolvimento e implementação desse projeto. Nele será exposto

o funcionamento em geral de aplicações *backends*, dividido em tópicos desde seus conceitos fundamentais, tecnologias utilizadas, e as percepções do software depois de terminado.

2 DESENVOLVIMENTO

Em primeiro lugar, antes de iniciar o desenvolvimento foi preciso analisar e identificar os principais aspectos que geram uma aplicação *backend* genérica. Com isso, foi possível identificar três entidades comuns em qualquer regra de negócio, usuário do sistema, cliente e agendamento. Com isso, foi possível escolher as tecnologias que deveriam ser utilizadas para o desenvolvimento deste projeto.

2.1 Aplicação

2.1.1 Arquitetura da Aplicação

Para criação da aplicação, foi determinado que seria uma Interface de Programação de Aplicação (API), em modelo de *APIs REST*. Esse modelo se baseia em um contrato, no qual ficam estipuladas formas de aplicações se comunicarem umas com as outras. Essa comunicação se baseia por protocolos de *Hypertext Transfer Protocol* (HTTP), o qual é a base da comunicação da web. Os principais protocolos de comunicação em um API são *GET*, *POST*, *PUT*, *PATCH* e *DELETE* (WEBBER; PARASTATIDIS; ROBINSON, 2010).

Nessa comunicação, é comum ter uma conversa recíproca, com uma requisição seguida de uma resposta. Na aplicação proposta, serão adotados os termos *Request* e *Response* para esse tipo de diálogo. Em uma *Request* é preciso definir uma *Uniform Resource Locator* (URL), a qual irá contar com seu tipo de método HTTP. Ademais, as *Requests* podem conter um cabeçalho, chamado *Header*, no qual dispõe de informação do remetente. Por fim, sendo um elemento opcional, existem os corpos das requisições, também chamados de *Body*, nos quais se trata de algum conteúdo a ser enviado junto das

requests. Nessa aplicação esse *Body* tem que ser enviado em formato *JavaScript Object Notation* (JSON) que é um formato amplamente utilizado para representar e estruturar os dados enviados em uma *Request*. Na figura 1 é possível identificar um exemplo de *Request*.

Figura 1 - Exemplo de Request para uma API






Metódo HTTP	URL	VERSÃO
GET	www.url.example.com	HTTP/1.1
Header	HOST: 127.0.0.1: 1234 Accept: text/css,*/*;q=0.1 Accept-Language: en-GB, en;q=0.5 Accept-Encoding: gzip, deflate, br User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv: 102.0) Gecko/20100101 Firefox/102.0 Connection: keep-alive	
Body	{ }	

Fonte: Autor (2023)

Sendo assim, após um recebimento de um requisição é esperada a *Response*, que por sua vez transmite para a *Request* se as solicitações enviadas foram concretizadas com sucesso ou não. Por isso, é adotado um sistema de numeração que indica o status da *Response*, esse indicativo é chamado de *Status Code* (MOZILLA, 2023). Na figura 2 podem ser vistos os *Status Codes* e seus significados.

Figura 2 - Status Codes

HTTP STATUS CODES

 100-199 INFORMATION	 200-299 SUCCESS	 300-399 REDIRECTS	 400-499 CLIENTS ERRORS	 500-599 SERVER ERRORS
---	---	---	--	---

Fonte: Autor (2023)

2.1.2 TypeScript

Para o desenvolvimento foi utilizado o *TypeScript* (TS) como linguagem de programação. A linguagem TS é derivada da linguagem *Javascript* (JS), sendo sua principal diferença é que é adicionada digitação estática. Ou seja, essa tecnologia permite que o *Javascript*, uma linguagem de tipagem fraca, se torne fortemente tipada em tempo de desenvolvimento. Ademais, o TS possui suas particularidades, porque todo código escrito no seu formato de extensão “.ts” em tempo de execução é compilado em JS. Isso porque, sendo o JS uma linguagem de tipagem fraca, faz que erros fatais possam acontecer em tempo de execução. Entretanto, se o código é escrito em TS, durante o processo de desenvolvimento é indicado, por meio dos editores de códigos, os erros de tipagem, os quais sinalizam que se o código for compilado haverá conflitos naquele determinado trecho do código (TypeScript, 2023). Nas figuras 3 e 4 é exemplificado um código que demonstra as linguagens e problema de tipagem do JS.

Figura 3 - Bloco de código escrito em JavaScript

<pre>1 const x = 1; 2 const y = '2'; 3 4 5 const somar = (x, y) => { 6 return x + y; 7 }; 8 9 console.log(somar(x, y)); 10</pre>	<div>Entrada do programa</div> <div>Saída do programa</div> <div>12</div> <div>[Execution complete with exit code 0]</div>
---	--

Fonte: Autor (2023)

É perceptível que a variável “y” se trata de um *String*, a qual a partir do momento onde é executada em uma função de cálculo, o JS atribui uma concatenação das variáveis em vez da soma. Sendo assim, a função “somar” não faz sentido em casos em que uma das entradas seja uma entrada de *String*. Todavia o mesmo código escrito em TS, salvo a função somar corrigindo o problema da concatenação de *Strings*, será exibido na figura 4.

Figura 4 - Bloco de código escrito em TypeScript

<pre>1 const x = 1; 2 const y = '2'; 3 4 5 const somar = (x: number, 6 y: number) => { 7 return x + y; 8 }; 9 10 console.log(somar(x, y));</pre>	<div>Saída do programa</div> <div>main.ts(9,22): error TS2345: Argument of type 'string' is not assignable to parameter of type 'number'.</div> <div>[Compilation failed with exit code 2]</div>
---	--

Fonte: Autor (2023)

Sendo assim, a função “somar” passa a declarar que seus parâmetros “x” e “y” devem ser do tipo *number*, somente números. Essa simples alteração, tem o efeito de que o mau uso da função não seja compilado o código para JS. Além disso, o tratamento de erro traz clareza dos elementos que o causam, que, no exemplo, indica que o parâmetro do tipo *String* não confere com o parâmetro do tipo *number*. Tais mensagens de erros em tempo de

desenvolvimento ajudam identificar os problemas do código escrito, o que facilitará o desenvolvimento e integridade do código bom.

2.1.2 PostgreSQL

O PostgreSQL é um Sistema de Gerenciamento de Banco de Dados (SGBD), sendo um banco de dados relacional e *open source*. Ele foi criado na Universidade da Califórnia na cidade de Berkeley, pelo departamento de Ciências da Computação (POSTGRESQL, 2023). Sua escolha na aplicação se deve pelo amplo uso no mercado de tecnologia e pelas suas funções de execução dos *Structured Query Language* (SQL).

2.2 Fluxo de aplicação

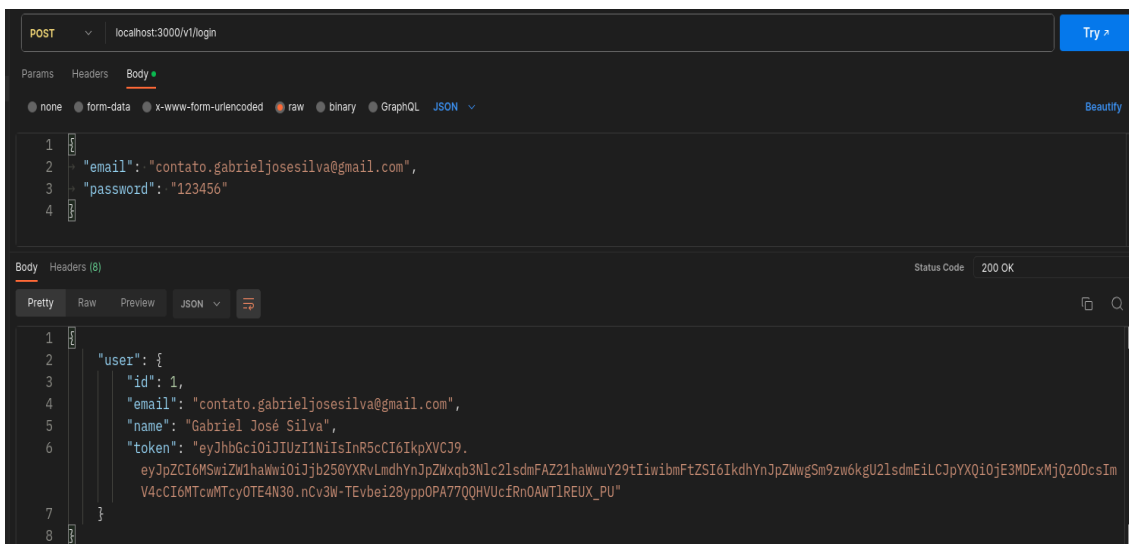
2.2.1 Postman

O Postman é um software utilizado para fazer *Request* e visualizar as *Response*, sendo que sua função principal é consumir as APIs. Além disso, a mesma suporta fluxo de teste para cada *Request*, compartilhamento do espaço com outros desenvolvedores e declaração de variáveis de ambiente no qual facilita o desenvolvimento de APIs (POSTMAN, 2023).

2.2.2 Rotas

Toda API tem seu fluxo de operação mapeado por rotas e métodos *HTTPs*. Esse mapeamento pode ser chamado de rota e é responsável por gerir o fluxo da aplicação, levando cada solicitação de *Request* ao seu devido objetivo. Cada rota tem um domínio e a porta de onde está hospedado e um nome, na qual é dado geralmente pela entidade ou serviço geral daquela rota. A figura 5 mostra a rota para o login da API.

Figura 5 - Exemplo da rota para realização de login



Fonte: Autor (2023)

A rota tem o nome de “login”, método *POST* e sua única função é realizar login. Essa rota demonstra as principais características das rotas, que é fazer somente uma coisa. Ou seja, cada rota deve ter sua própria e única responsabilidade, fazendo com que o funcionamento de cada rota não fique complexo para o desenvolvimento e nem para manutenção do código.

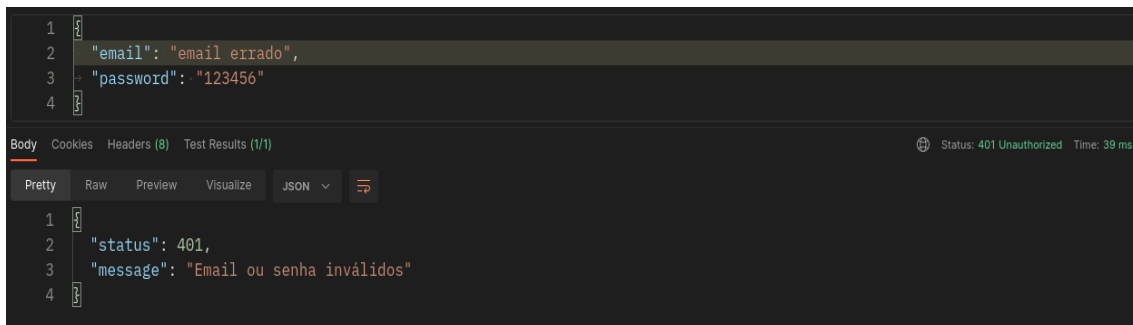
2.2.3 Validação e exceções

Quando ocorre uma *Request*, um conteúdo ou resultado é solicitado para a aplicação. Porém nem sempre uma solicitação pode ser atendida. Um dos motivos de uma *Request* não ser aceita é por não satisfazer a entrega de dados necessários ou inválidos. Por isso, é comum que cada rota adote verificações e validações, esses interceptadores são chamados de *middlewares*. Esses *middlewares* interrompem o fluxo e realizam as verificações necessárias para que os próximos passos da rota aconteçam com uma maior probabilidade de êxito.

Para realização dos *middlewares* é utilizada uma biblioteca externa chamada Zod, que é a pioneira quando se trata a respeito de validação na linguagem TS. Seu funcionamento se dá a partir de esquemas, nos quais são feitas comparações dos conteúdos recebidos com o esquema (ZOD, 2023).

Caso no processo de validação conste uma invalidez, ele devolverá uma mensagem de erro, na qual conterá o status e uma mensagem, que explica o motivo da devolução. Na figura 6 é mostrado um exemplo de erro devolvido.

Figura 6 - Exemplo de mensagem de erro após validação

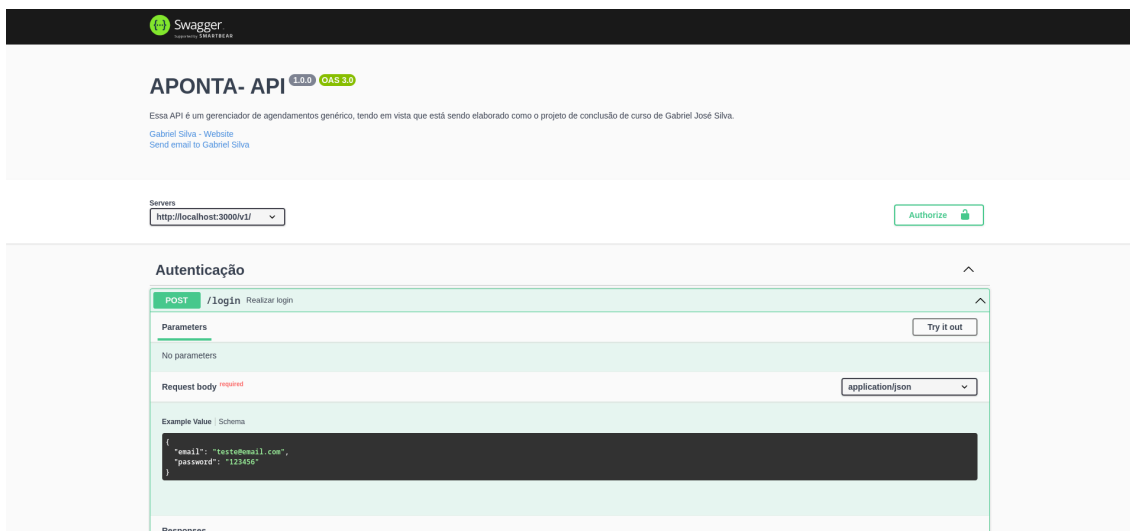


Fonte: Autor (2023)

2.2.4 Swagger UI

Para um mapeamento público da API é disponibilizada uma rota que é uma interface para a documentação da API. Essa rota é gerada pelo Swagger UI, que consiste em um serviço utilizado por uma biblioteca externa, sendo que seu principal uso é documentar a API de maneira dinâmica, facilitando a sua exibição para os desenvolvedores, quanto ao público que irá consumir a API (SWAGGER, 2023). Na figura 7 é mostrada a interface Swagger da API.

Figura 7 - Interface do Swagger da API



Fonte: Autor (2023)

3 APLICAÇÃO FINALIZADA

Após a conclusão da aplicação, a mesma se encontrava pronta para ser utilizada em qualquer clínica odontológica. Sendo utilizada, ela poderia impactar o negócio das seguintes maneiras:

3.1 Segurança

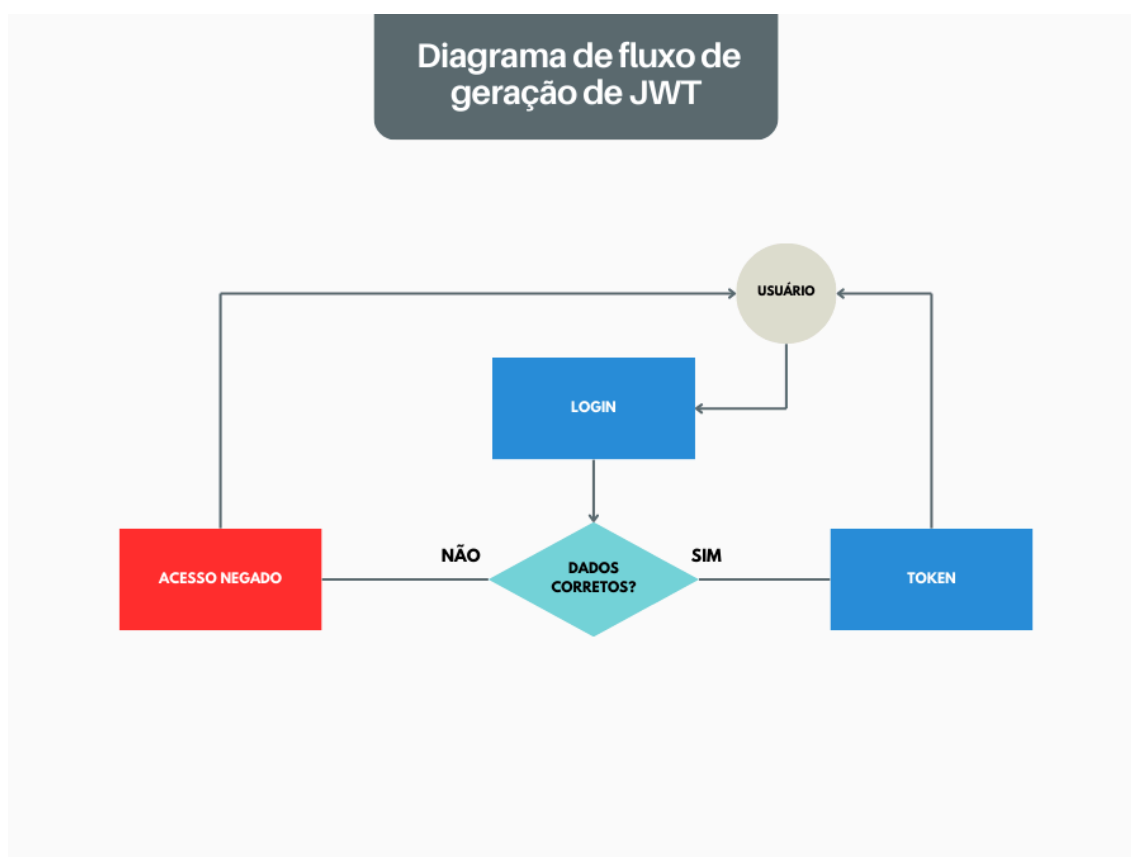
3.1.1 Credenciais de requisições

Primeiramente, a segurança da aplicação consiste em que qualquer requisição para a API, deve apresentar uma credencial no seu cabeçalho. Com isso, uma única solicitação que não precisa dessa credencial é a de *login*, que é por meio dela que é realizada autenticação do usuário e entregue a credencial, além de informações sobre o mesmo usuário. Portanto fazendo que o tráfego das informações fique seguro a de qualquer invasor que não tenha credenciais válidas.

3.1.2 JWT

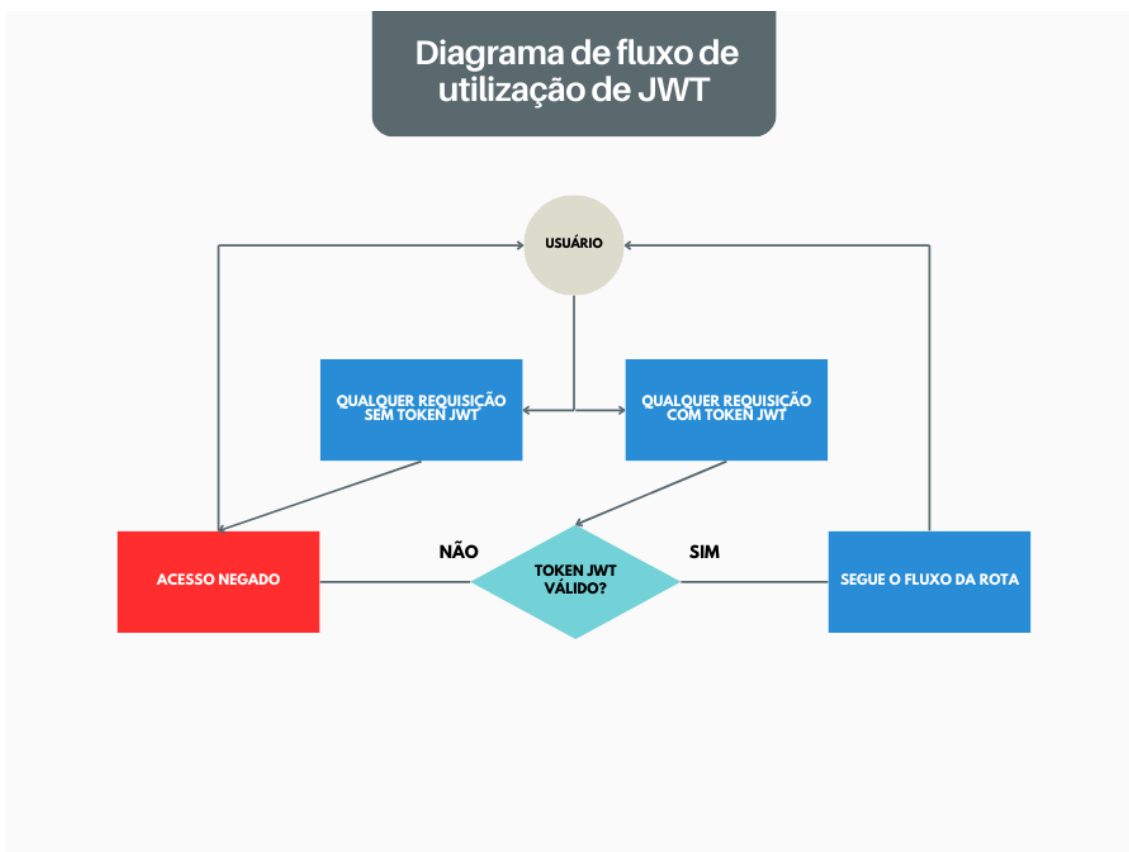
As credenciais necessitam que sejam trafegadas de forma criptografada para que nenhum dado contido nela seja acessado. Por causa disso, as credenciais se tornaram JWT (JSON Web Tokens), que são uma forma de comprimir e criptografar informações de credenciais de sistemas. De forma a explicar o seu funcionamento, a aplicação faz autenticação se os dados do *login* estão corretos, e após verificados, caso estejam corretos ela cria um *token* que é um conjunto de caracteres sem padrão lógicos criptografado, que é gerado, comprimido e assinalando com uma chave secreta e um prazo de validade. Fazendo assim que somente a própria aplicação que contém a chave secreta consiga extrair os dados e validar se o token expirou (JWT, 2024). Na figura 8 e 9 é possível notar o fluxo da geração de JWT e como é funcionalidade é utilizada por essa API

Figura 8 - Diagrama de fluxo de geração de JWT



Fonte: Autor (2024)

Figura 9 - Diagrama de fluxo de utilização de JWT



Fonte: Autor (2024)

3.2 Agilidade

3.2.1 Velocidade entre requisições e respostas

O principal objetivo da API era ser rápida, para facilitar o trabalho dos funcionários em respeito a manipulação de agendas físicas. Com isso o sistema foi construído com diversos códigos e tecnologias que oferecem rapidez, provendo a aplicação uma latência mínima chegando a 0,003 milissegundos até no máximo de 2 segundos a depender do tipo de requisição. Sendo um ótimo ganho de tempo em tarefas que geraram esforços manuais e organizacionais.

3.3 Qualidades de software

3.3.1 Versionamento ágil

A construção da regra de negócio é exposta em blocos de códigos fáceis de serem manipulados. Qualquer implementação ou alteração por meio da solicitação dos clientes da API pode ser provida em prazos menores, trazendo a agilidade e melhorando a expectativa de vida do software. Sendo um software duradouro e escalável.

3.3.1 Containers

A aplicação é provida de dentro de um container, na qual é empacotado e executado não em hardware físico e sim uma máquina virtual. Trazendo escalabilidade para fazer a produção utilizando tanto computadores físicos quanto os novos serviços de computação em nuvem. Para isso é utilizado o Docker um orquestrador de containers, o qual viabiliza a manipulação fácil do container de diversas aplicações (IBM, 2024).

4 CONSIDERAÇÕES FINAIS

O artigo teve objetivo de explicar o funcionamento comum de uma API, e como esse formato foi utilizado para construção do sistema APONTA. Sendo um sistema bastante promissor, o APONTA pode prover experiências agradáveis aos clientes, pois ele cumpre e com excelência sua proposta. Sendo um sistema genérico no qual múltiplas clínicas podem se beneficiar da sua utilização.

4.1 Futuras atualizações

Após a conclusão do projeto é possível perceber que sua arquitetura comprimiu todas as funcionalidades num único projeto, embora podendo ser de fácil manipulação e alterações deve se pensar em evoluir projeto para uma arquitetura de serviços separados em outras novas APIs. Além disso pode ser

adicionar uma regra de negócio flexível para a forma de agendamentos, no qual cada cliente que utilize o APONTA consiga definir optar pelas regras de negócios das quais façam mais sentido para o seu tipo de negócio, fazendo o sistema evoluir de um sistema de agendamento para clínicas odontológicas para qualquer tipo de negócio que tenha processos de trabalho com agendamento de horários marcados.

REFERÊNCIAS

FGV CIA. **USO DE TI NAS EMPRESAS**, 2023. Disponível em: eaesp.fgv.br/sites/eaesp.fgv.br/files/u68/pesti-fgv cia-2023-resumoppt.pdf. Acesso em: 19 nov. 2023.

WEBBER, Jim; PARASTATIDIS, Savas; ROBINSON, Ian. **REST in Practice**: Hypermedia and Systems Architecture, O'Reilly Media, Sebastopol, 2010. E-book. Disponível em: github.com/GunterMueller/Books-3/tree/master. Acesso em: 23 nov. 2023.

MOZILLA. **Códigos de status de respostas HTTP**. 2023. Disponível em: developer.mozilla.org/pt-BR/docs/Web/HTTP/Status. Acesso em: 23 nov. 2023.

TypeScript. **TypeScript Documentation**. 2023. Disponível em: www.typescriptlang.org/docs/. Acesso em: 23 nov. 2023.

POSTGRES SQL. **What Is PostgreSQL?**. 2023. Disponível em: www.postgresql.org/docs/current/intro-what-is.html. Acesso em: 25 nov. 2023.

POSTMAN. **Intro to API Platforms**. 2023. Disponível em: www.postman.com/api-platform/. Acesso em: 27 nov. 2024.

ZOD. **Introduction**. 2023. Disponível em: zod.dev/?id=introduction. Acesso em: 27 nov. 2023.

SWAGGER, **API Documentation**. 2023. Disponível em: swagger.io/solutions/api-documentation/. Acesso em: 27 nov. 2023.

Revista Mundi Engenharia, Tecnologia e Gestão. Paranaguá, PR, v.??, n.??, p. ??-??, ??/??., 20??
DOI: 10.21575/xxxxxxxxxxxxxxxxxxxxxxxxxx

JWT, **Introduction to JSON Web Tokens**. 2024. Disponível em:
jwt.io/introduction. Acesso em: 13 jan. 2024.

IBM, **O que é o Docker?**. 2024. Disponível em:
www.ibm.com/br-pt/topics/docker. Acesso em: 13 jan. 2024.