LIGA DE ENSINO DO RIO GRANDE DO NORTE

CENTRO UNIVERSITARIO DO RIO GRANDE DO NORTE – UNI-RN

ESPECIALIZAÇÃO EM DESENVOLVIMENTO DE SISTEMAS CORPORATIVOS

GIOVANNI DE PAIVA NICOLETTI

IMPLEMENTAÇÃO DE UM SISTEMA CORPORATIVO UTILIZANDO TECNOLOGIAS FULL STACK

NATAL – RN

2019

LIGA DE ENSINO DO RIO GRANDE DO NORTE

CENTRO UNIVERSITARIO DO RIO GRANDE DO NORTE – UNI-RN

ESPECIALIZAÇÃO EM DESENVOLVIMENTO DE SISTEMAS CORPORATIVOS

GIOVANNI DE PAIVA NICOLETTI

IMPLEMENTAÇÃO DE UM SISTEMA CORPORATIVO UTILIZANDO TECNOLOGIAS FULL STACK

Trabalho de Conclusão de Curso apresentado ao Centro Universitário do Rio Grande do Norte, em cumprimento às exigências legais como requisito final para obtenção do título de Especialista em Desenvolvimento de Sistemas Corporativos.

Orientador: Prof. Alexandre Damasceno

NATAL – RN

2019

GIOVANNI DE PAIVA NICOLETTI

IMPLEMENTAÇÃO DE UM SISTEMA CORPORATIVO UTILIZANDO TECNOLOGIAS FULL STACK

Trabalho de Conclusão de Curso apresentado ao Centro Universitário do Rio Grande do Norte, em cumprimento às exigências legais como requisito final para obtenção do título de Especialista em Desenvolvimento de Sistemas Corporativos.

Orientador: Prof. Alexandre Damasceno

Aprovado (a) em 31/05/2019

BANCA EXAMINADORA

Prof. Alexandre Damasceno

Orientador

Prof. Gleydson de Azevedo Ferreira Lima

Membro

AGRADECIMENTOS

Dedico este trabalho primeiramente a Deus, aos meus amados pais, familiares e amigos. Eles foram minha fortaleza e porto seguro nos momentos difíceis. Portanto, todos têm meu reconhecimento e gratidão nessa minha conquista pessoal.

SUMARIO

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

A área da Tecnologia da informação (TI) evolui cada vez mais rápida, são muitas linguagens de programação, muitos frameworks, novas plataformas de desenvolvimento e soluções tecnológicas convergindo para um único objetivo de atender as necessidades de negócio dos clientes.

Ao passo que criamos várias especialidades e carreiras na área de TI, tais como: analista de banco de dados, analista de infraestrutura de TI, desenvolvedor de aplicações mobile, desenvolvedor front-end, desenvolvedor front-end, eis que surge um novo conceito de profissional de TI, chamado desenvolvedor Full Stack. Esse desenvolvedor é capaz de levantar uma aplicação web completa.

O desenvolvedor Full Stack é um profissional capaz de trabalhar com toda a stack (pilha) de desenvolvimento de uma aplicação, entendendo desde o banco de dados, e sendo capaz de trabalhar tanto no front-end, quanto no back-end. Geralmente ele tem uma área na qual ele possui um maior domínio, e as demais, ele tem um conhecimento no qual ele consegue se virar, sem que isso seja sua principal especialidade. Vamos descrever um exemplo de um desenvolvedor Full Stack: profissional com amplo domínio no front-end, muito fluente em HTML, CSS e frameworks Javascript para criação de SPA's (Single Page Applications), como Vue ou Angular, por exemplo, e com conhecimento de tecnologias de desenvolvimento back-end, como Java, C# e NodeJS, por exemplo, não sendo um exímio desenvolvedor dessas linguagens, mas capaz de criar alguns serviços e corrigir alguns bugs nesta parte. Capaz de trabalhar com banco de dados, criando tabelas e manipulando registros, mesmo que não sendo um especialista em banco de dados, e que não saiba projetar e gerenciar um banco altamente escalável, mas que sabe se virar ali. Da mesma forma, conhece um pouco de operações, conseguindo configurar um servidor e publicar a aplicação no ar. Além disso, também consegue levantar as necessidades do cliente e propor soluções, entendendo as regras de negócio em questão.

O presente trabalho visa mostrar o desenvolvimento de uma aplicação SPA, voltada para área da saúde, que usa as tecnologias C# (back-end), Angular 2 (front-end) e banco de dados SQL Server 2016 Express.

Palavras-Chaves: TI, Full Stack, C#, Angular, Front-end, Back-end.

ABSTRACT

The area of information technology (IT) evolves ever faster, many programming languages, many frameworks, new development platforms and technological solutions converge to a single goal to meet the business needs of customers.

While we have created a number of IT specialties and careers, such as: database analyst, IT infrastructure analyst, mobile application developer, front-end developer, front-end developer, here comes a new concept of IT professional, called Full Stack developer who is a developer able to raise a complete web application.

The Full Stack developer is a professional able to work with the entire stack of an application's development, understanding from the database, and being able to work on both the front-end and the back-end. Generally, he has an area in which he has a bigger domain, and the others he has a knowledge in which he can turn around without that being his main specialty. Let's describe an example of a Full Stack developer: professional with broad domain on the front end, very fluent in HTML, CSS and Javascript frameworks for creating SPAs (Single Page Applications), such as Vue or Angular, for example, and with knowledge of technologies of back-end development such as Java, C # and NodeJS, for example, not being an expert developer of these languages but able to create some services and fix some bugs in this part. Able to work with databases, creating tables and manipulating records, even if you are not a database expert, and who can not design and manage a highly scalable database, but who knows how to turn it around. In the same way knows a little of operations, being able to configure a server and publish the application in the air. In addition, it can also raise customer needs and propose solutions by understanding the business rules in question.

The present work aims to show the development of a SPA application, focused on healthcare, using the technologies C # (back-end), Angular 2 (front-end) and SQL Server 2016 Express database.

Keywords: TI, Full Stack, C #, Angular, Front-end, Back-end.

1 INTRODUÇÃO

A ideia desse projeto veio da necessidade de criar uma aplicação, inicialmente para Procuradoria Geral do Estado do Rio Grande do Norte – PGE/RN, que nos possibilitasse o controle de pedidos dos medicamentos mais onerosos, o entendimento dos custos para o estado, qual público mais demanda solicitações, faixa etária e doenças com maior incidência da nossa população.

A solução foi desenvolver uma aplicação Web, utilizando várias tecnologias de mercado, com uma arquitetura robusta e alinhada aos mais diversos conceitos vistos na pós-graduação de sistemas corporativos web.

A solução não só é capaz de atender as necessidades da PGE/RN, como também nos proporcionou a oportunidade de criarmos uma solução que usa conceitos atuais, relativos ao desenvolvimento de sistemas web.

A arquitetura do sistema utiliza conceitos do desenvolvimento full stack, contendo as seguintes pilhas: stack de banco de dados, stack de back-end e stack de Front-end.

A comunicação entre as camadas de Back-end e Front-end é feita por meio de Web Services REST, transportando dados via JSON.

Olhando a arquitetura de forma Top-Down, temos um conjunto de tecnologias distintas para cada uma das “stacks” (pilhas) a seguir:

Na stack de Front-end (client side), fazemos uso das tecnologias: HTML 5, CSS 3, Bootstrap 4, Angular 7.

Na stack de Back-end (server side), fazemos uso das tecnologias: C#, ASP .NET Web Api, Entity Framework. O BD (Banco de dados) escolhido foi o MSSQL Server e está disponível para ser acessado apenas via back-end. O back-end controla a autenticação dos usuários do sistema, via webservice SOAP, acessando a base de usuários do Active Directory.

Dessa forma, o sistema Jvris Saúde, será de grande importância para área de controle de medicamentos fornecidos pelo estado a pacientes com doenças crônicas e/ou graves, e que não dispõe de recursos financeiros suficientes para arcar com o tratamento.

Esperamos que, de posse desse sistema, a PGE/RN possa reduzir as despesas do estado, criando maior controle com relação aos medicamentos fornecidos e mapeando locais com maior incidência de pacientes crônicos.

2 CONTEXTUALIZAÇÃO E PROBLEMA

2.1 OBJETIVOS

2.1.1 Objetivo Geral

Montar uma aplicação web capaz de integrar varias tecnologias atuais de mercado, explorando conceitos de microserviços, interoperabilidade, webservices REST, troca de mensagem entre tecnologias distintas, transporte de dados em JSON, desenvolvimento back-end, desenvolvimento front-end e desenvolvimento Full Stack.

Para exemplificar o conceito full stack de forma prática, mostraremos partes de uma aplicação web que foi desenvolvida para Procuradoria Geral do Estado – PGE, no intuito de controlar e acompanhar os pedidos de medicamentos pagos pelo estado para a população com doenças graves e/ou crônicas. Assim, mostraremos como a arquitetura do sistema foi construída, mostrando, detalhando e exemplificando cada uma das seguintes tecnologias: ASP .NET Web API, Angular 7, Web Services REST e linguagem de programação C#.

2.1.2 Objetivo Especifico

Mostrar a possibilidade de desenvolver uma aplicação web, usando diversas tecnologias de fabricantes distintos e altamente integradas. Será possível também, explorar conceitos e padrões arquiteturais de desenvolvimento. Iremos esclarecer e fazer uso de conceitos como: padrão MVC, microserviços, Web Services REST, troca de informações usando JSON, linguagens usadas no desenvolvimento back-end, no front-end, e os demais conceitos necessários para que um desenvolvedor seja considerado Full Stack.

A proposta desse trabalho é abordar todas as tecnologias envolvidas no projeto desenvolvido, tais como:

- Banco de dados em SQL Server 2016 Express;
- Desenvolvimento Back-end em ASP .NET Web API;
- Padrão arquitetural MVC;
- Conceitos de Integração com Web Service REST;
- Utilização dos verbos HTTP (GET, POST, PUT, DELETE);
- Transporte de dados via JSON;
- Desenvolvimento Front-end em Angular 7.

Iremos detalhar alguns requisitos do sistema, tais como: autenticar usuários, incluir, alterar, localizar ou excluir um medicamento qualquer. Com isso, o objetivo principal não é especificar todos os requisitos funcionais e não funcionais do sistema, mas sim, mostrar como é feito operações básicas de CRUD (Create, Recorvery, Update, Delete), com foco nas tecnologias que escolhemos usar nesse projeto.

Desse modo, iremos explorar cada um dos requisitos solicitados pelo usuário e detalhar desde a construção da tela do usuário no Angular, passando pela API REST do ASP .Net e persistindo a informação no banco de dados.

3 METODOLOGIA

3.1 Construindo uma Aplicação Web

Com o advento da World Wide Web, ou simplesmente Web, páginas e aplicativos começaram a crescer de maneira acelerada e de forma exponencial. No inicio, websites eram desenvolvidos de maneira ad-hoc, sem uma metodologia ou processo para apoiar seus criadores. No entanto, à medida que eles cresceram em complexidade e tornaram-se verdadeiras Aplicações na Web, tornou-se necessário aplicar métodos disciplinados de Engenharia de Software, adaptados para essa nova plataforma de implementação.

Segundo PRESSMAN et. al (2011), ao considerarmos os problemas de projeto a ser solucionados quando um WebApp é construída, vale a pena considerar categorias de padrão concentrando-nos em duas dimensões: o foco no projeto e seu nível de granularidade. *O foco do projeto* identifica qual aspecto do modelo de projeto é relevante (por exemplo, arquitetura das informações, navegação, interação). *A granularidade* identifica o nível de abstração que está sendo considerado (por exemplo, o padrão se aplica a toda WebApp ou a uma única pagina web, a um subsistema ou um componente WebApp individual?).

Quando o assunto é desenvolvimento de Aplicação Web - WebApp, foi possível observar uma verdadeira revolução tecnológica. Depois da criação do Node.js, em 2009, viu-se a possibilidade de usar JavaScript como back-end e não apenas para criar interações entre o usuário e o navegador. Com isso, o JavaScript vem ganhando cada vez mais força no cenário de desenvolvimento de aplicações, sendo uma linguagem de programação de propósito geral, multi paradigmas e multi plataforma. Desse modo, é natural haver várias implementações e ferramentas de variados objetivos. JavaScript está renovando e inovando a maneira como criamos aplicações Web complexas com SPA (Single Page Aplication), aplicações híbridas com Cordova, nativas com NativeScript, Weex e ReactNative, ou no ambiente do servidor com Node.js.

Em meados de 2016, começa a surgir frameworks mais modernos focados em produtividade, com ideias de arquitetura usando o padrão MVC, sites no qual é mostrado todo o conteúdo de uma aplicação em uma única página - SPA (Single Page Application). Abaixo, citamos alguns dos melhores frameworks JavaScript MVC que surgiram no mercado, nessa época.

- **Kendo**: Um ótimo framework para desenvolvimento de aplicativos móveis e web, composto de três widgets: UI Web (tudo que você precisa para construir um website moderno), UI Mobile (oferece a capacidade de construir aplicativos web móveis que podem ser confundidos com aplicativos nativos) e UI DataVis (permite que os desenvolvedores implementem uma bela visualização voltada ao usuário de dados e relatórios).
- **Sencha Touch**: Uma biblioteca JavaScript de interface de usuário criada especificamente para aplicativos da Web para dispositivos móveis e que permite aos desenvolvedores criar experiências de usuário que parecem aplicativos nativos.
- **JQuery Mobile**: Uma estrutura JavaScript móvel de plataforma cruzada, o jQuery Mobile foi projetado para aprimorar e simplificar o desenvolvimento de aplicativos da Web, integrando HTML5, jQuery, CSS e a interface do usuário do jQuery em uma única estrutura única. É robusto e satisfatoriamente bem organizado.
- **AngularJs**: Considerado por muitos como o grande pai das estruturas JavaScript, devido ao fato de ter sido desenvolvido pelo Google. Há uma grande comunidade de desenvolvedores por trás do AngularJS e se destaca particularmente em sobrecarregar o código HTML graças à sua capacidade de auxiliar na construção de interfaces dinâmicas de usuário.
- **Ember**: Originalmente lançado em 2011, o Ember recebe dicas do Angular quando se trata de criar elementos dinâmicos voltados ao usuário para aplicativos da web. Como o framework do Google, o Ember pode atualizar o View quando o Model mudar e vice-versa, mantendo a mecânica dos apps em perfeita sincronia.
- **React**: Facebook e o Instagram contam com o framework React JavaScript devido à sua capacidade de construir aplicativos grandes que são dinâmicos e podem ser amplamente dimensionados. O React possui uma técnica muito eficiente para renderizar interfaces de usuário complexas e está rapidamente se tornando a estrutura JavaScript que mais cresce na era moderna.

Em síntese, qualquer desenvolvedor que se preze saberá o valor dos frameworks na programação JavaScript. Inúmeras horas podem ser perdidas reescrevendo o código que já existe, o famoso: *“reinventar a roda”*. Desse

modo, os frameworks MVC tornam o trabalho que antes era difícil, em uma implementação fácil e com bem menos esforço.

Atualmente, no que tange o desenvolvimento de aplicações web, existe uma divisão de conceitos e tecnologias a serem estudadas e implementadas do lado do front-end e do back-end.

Front-end e back-end são termos usados para caracterizar interfaces de programas e serviços relativos ao usuário dessas interfaces. A aplicação front-end é aquela que interage diretamente com o usuário, atuando como primeiro contato que ele tem com o programa. Já uma aplicação back-end trabalha indiretamente no suporte dos serviços de front-end, normalmente se comunicando com a fonte desses serviços, o server.

Antes de entrar no mundo da programação, é ideal entender alguns conceitos que estão por trás do universo back-end. Para isso, vamos listar os cinco principais conceitos que todo desenvolvedor deve conhecer sobre back-end para desenvolvimento web.

- **Servidor**: Um servidor é um software ou computador, com sistema de computação centralizada que fornece serviços a uma rede de computadores, também conhecidos por clientes. Existem diversos tipos de servidores (DNS, Proxy, FTP…) e quando se trabalha com back-end é importante conhecer a maioria deles. Um desenvolvedor back-end deve ser especializado em linguagens server-side como o ASP.NET, Java, Python e Ruby.
- **Banco de dados**: Funciona como uma biblioteca repleta de informações. Toda vez que você realiza uma ação ou procura algo em um site, o banco de dados é responsável por aceitar sua busca, encontrar o dado e apresentar no website. No back-end, o banco de dados é acessado pelos usuários indiretamente a partir de uma aplicação externa. Um desenvolvedor back-end precisa ser capaz de trabalhar com a maioria das databases existentes, como MSSQL Server, MongoDB, Postgresql e NoSQL.
- **API**: Application Programming Interface, ou Aplicações de Interface de Programação nada mais é que uma interface que permite que dois sistemas, ainda que criados com tecnologias diferentes, se comuniquem através de uma linguagem comum. Elas conectam softwares, aplicações, banco de dados e serviços, substituindo a necessidade de uma programação mais complexa.
- **Escalabilidade**: A tecnologia continua mudando, os modelos de negócio mudam e o sistema de back-end precisa ser construído de

forma a se adaptar facilmente à essas mudanças. É para isso que serve a escalabilidade, um dos conceitos mais importantes no back-end. É preciso pensar em quão flexível é a estrutura de uma aplicação para acomodar novos códigos, mais tráfego e dados.

- **Segurança**: A segurança é um dos principais problemas na maioria dos sistemas e muitos deles são vulneráveis a ataques simples. O desenvolvedor back-end precisa seguir práticas consistentes de segurança, sempre se perguntando: os formulários de login e outras autenticações são processados por meio de um HTTPS reforçado? Os dados estão sendo criptografados sempre que são armazenados?

Diante da profundidade de conhecimento que envolve tanto as tecnologias de back-end, quanto às de front-end, o mercado acaba criando a necessidade de um novo perfil de profissional, o desenvolvedor Full Stack.

3.2 Desenvolvimento Full Stack

O termo full stack vem sendo bastante utilizado nos últimos anos como uma forma de definir um tipo de desenvolvedor. O próprio nome já dá uma ideia do que este desenvolvedor faz, pois a palavra full significa completo e stack pilha. Ou seja, trata-se de um profissional capaz de trabalhar com toda a pilha de desenvolvimento de uma aplicação. Normalmente os projetos de software, principalmente web, são divididos em partes como front-end (client side) e back-end (server side). Enquanto há desenvolvedores que se especializam e atuam em apenas uma das pontas, existem também aqueles que são responsáveis por "tudo". Ou seja, tanto programam o back-end usando uma linguagem como C#, Java ou PHP (além do banco de dados), quanto constroem e mantêm o front-end com HTML, CSS, JavaScript (além dos frameworks). A esse profissional que atua nas duas "frentes" é dado o nome de Full Stack.

A princípio pode parecer um pouco desesperador ter que conhecer tudo de todas as etapas do desenvolvimento de um projeto, mas, na verdade, um desenvolvedor full stack não precisa ser um expert em todas as áreas, pois isso seria praticamente impossível.

Geralmente ele tem uma área na qual ele possui um maior domínio, e as demais, ele tem um conhecimento no qual ele consegue se virar, sem que isso seja sua principal especialidade. Vamos descrever um exemplo de um desenvolvedor full stack: profissional com amplo domínio no front-end, muito fluente em HTML, CSS e frameworks Javascript para criação de SPA's (Single Page Applications), como Angular, por exemplo, e com conhecimento de tecnologias de desenvolvimento back-end, como Java, C# e NodeJS, por exemplo, não sendo um exímio desenvolvedor dessas linguagens, mas capaz de criar alguns serviços e corrigir alguns bugs nesta parte. Capaz de trabalhar com banco de dados, criando tabelas e manipulando registros, não sendo um especialista em banco de dados que saiba projetar e gerenciar um banco altamente escalável, mas que sabe se virar ali. Da mesma forma conhece um pouco de operações, conseguindo configurar um servidor e publicar a aplicação no ar. Além disso, também consegue levantar as necessidades do cliente e propor soluções, entendendo as regras de negócio em questão.

Resumindo, caso um desenvolvedor consiga levantar as necessidades do projeto, desenvolver o sistema e colocá-lo no ar, ele pode se considerar um desenvolvedor full stack. Ele não precisa dominar todas as áreas, mas precisa saber se virar em todas elas e capaz de aprofundar-se em determinado tema caso seja necessário.

No dia a dia, um desenvolvedor full stack é responsável por, além de desenvolver as funcionalidades dos sistemas, levantar os requisitos junto ao cliente. Então, também é importante que ele consiga entender as regras de negócio na qual o projeto está inserido, identificar as necessidades e demandas. Uma vez definido o que deve ser feito, ele irá começar o desenvolvimento da solução. Algumas empresas possuem equipes que são especializadas em determinada área, porém isso não impede que o desenvolvedor full stack tenha conhecimento do que ocorre ali e que consiga participar do processo e trabalhar em conjunto. Já em outras empresas, ele é o responsável por desenvolver o projeto do início ao fim, e de realizar as futuras manutenções e evoluções no mesmo projeto. Sempre que houver alguma dúvida muito específica ou um problema complexo, nada impede o desenvolvedor de recorrer a alguém mais experiente naquele assunto.

As tarefas do dia a dia de um desenvolvedor full stack dependem muito do projeto que ele está trabalhando. Caso o projeto esteja com muitas tarefas atrasadas no back-end, por exemplo, é normal que ele passe mais tempo trabalhando ali.

Claro que a rotina de cada um é diferente. Muitos desenvolvedores full stack também passam grande parte do dia dando manutenção em um sistema já existente, corrigindo bugs ou fazendo melhorias. Um problema que ele resolve frequentemente pode ser uma interação ruim da interface com o usuário, pode ser um erro de lógica em uma regra de negócio, pode ser uma lentidão excessiva em uma consulta no banco de dados ou pode ser um grande consumo de memória no servidor, por exemplo.

O desenvolvedor full stack tem como uma das principais características, a de ser um profissional que consiga aprender sozinho e de forma rápida, pois normalmente os projetos são bem corridos e precisam de agilidade, e nem sempre existe alguém na equipe com conhecimento no assunto e com tempo disponível para ensinar, por isso ser autodidata é um diferencial. Esta busca constante por conhecimento é essencial para que o profissional não fique ultrapassado em pouco tempo e, para que as soluções que ele irá criar gerem, de fato, valor para a empresa.

Este tipo de profissional está sendo muito requisitado pelo mercado, pois muitas empresas gostam de contratar profissionais que conseguem enxergar o projeto como um todo, sendo capaz de participar em todas as áreas. Algumas empresas optam por contratar desenvolvedores especialistas de cada área, outras optam por contratar desenvolvedores full stack, mas há outras que gostam de mesclar os dois perfis.

3.3 Back-end: nos bastidores da aplicação

Como já foi citado anteriormente, o desenvolvimento de aplicações web é dividido em duas partes principais: o back-end e o front-end. No front-end está tudo que acontece entre o usuário e a máquina, é um campo tangível no qual se encontram os botões, imagens, texto, logos, animações, etc. O back-end, por sua vez, é invisível ao usuário, porém é o motor da aplicação e o que de fato a faz rodar.

No back-end também está toda infraestrutura, como servidores, armazenamento e rede. Apesar da complexidade da infraestrutura, a computação em nuvem (Cloud Computing) tem facilitado o trabalho no back-end. Hoje em dia, por exemplo, é possível colocar no ar uma máquina virtual para sua aplicação em apenas alguns minutos. Este tipo de serviço é conhecido como ‘infraestrutura como serviço’ ou IaaS (Infrastructure as a Service).

No back-end, destacam-se linguagens como Pyhton, Ruby on Rails, PHP, Java, C# e ainda conhecimento de webservers como Apache e Nginx, modelagem de dados, linguagem SQL e estruturas de aplicação. Os especialistas ainda precisam se preocupar com desempenho, modelagem e construção de APIs em REST, além de subirem instâncias para a nuvem e configurarem todo o ambiente para que ele fique estável e escalável.

O desenvolvimento de aplicações usando Web Services REST e os conceitos de microserviços viraram uma tendência de mercado. Dentre as principais motivações, temos a enorme quantidade de tecnologias que precisam ser integradas, a descentralização do código e a quebra da dependência.

Também é possível desenvolver um melhor redimensionamento da equipe de desenvolvedores, uma vez que com microserviços podemos dividir a equipe por funcionalidades ou por contexto, como na **Figura 1**.

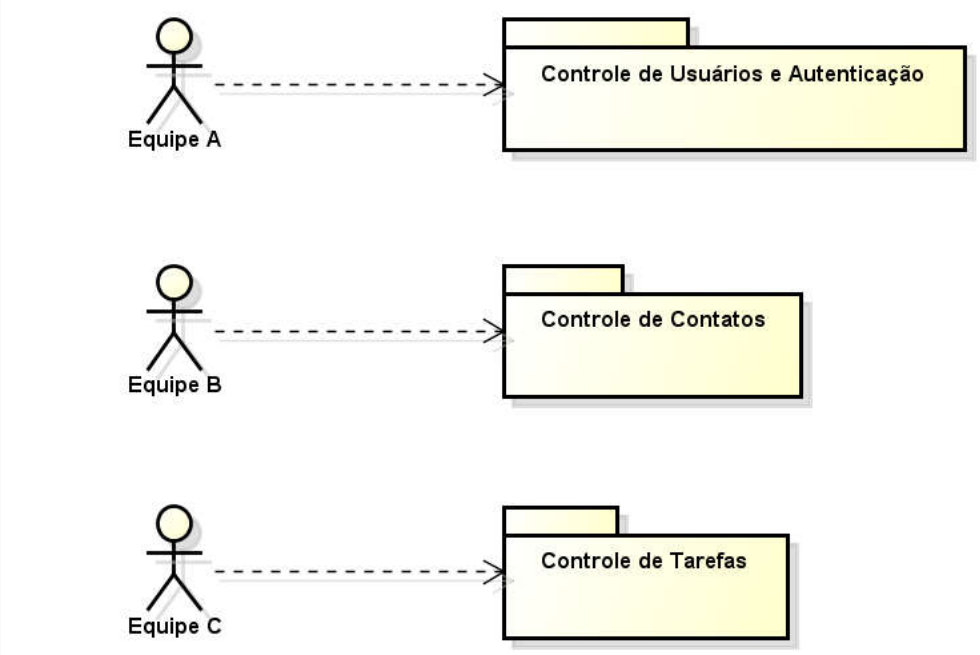


Figura 1. Divisão da Equipe por Funcionalidade

Normalmente, de acordo com o Scrum, uma Equipe Ágil precisa ter entre 06 a 10 membros, logo, se nosso time de desenvolvimento tiver 20 profissionais, podemos dividir a equipe em dois e com isso trabalhar em duas sprints com funcionalidades distintas para um mesmo produto, que ao final da iteração é entregue funcionando e testável através de suas interfaces gráficas, por exemplo.

Dessa forma, é possível percebermos que se formos capazes de dividir nossos projetos em módulos distintos da forma correta, criando microserviços que facilitem a integração entre os módulos, maior será nossa capacidade de contratarmos especialistas para cada uma das tecnologias do nosso projeto.

Segundo Cesar Fernandes, Engenheiro de Computação, com MBA em Gestão de Projetos, “Ao desenvolver aplicações apoiadas nesse serviço, às empresas podem encontrar benefícios como economia de até 80% nos custos de desenvolvimento de back-end e redução de até 50% no esforço de desenvolvimento de um aplicativo. Além disso, a ferramenta evita que o desenvolvedor perca tempo para aprender a subir máquinas, codificando APIs e ainda oferece escala. Com essas vantagens, a empresa pode focar nas melhorias de seus negócios ao mesmo tempo em que cria aplicações que atendam às expectativas dos seus clientes, garantindo a eles uma boa experiência.”.

Na proposta do serviço back-end desse projeto de TCC, foi implementado um Framework para **desenvolvimento de web services RESTful** que funciona sobre o .NET Framework.

O .NET Framework é uma grande plataforma de desenvolvimento de aplicações mantida pela Microsoft e que dispõe de diversas ferramentas e classes que tornam possível a criação de aplicações para: Desktop, Mobile, Web, inclusive para internet das coisas.

O ASP .NET é um sub framework especifico para desenvolvimento de aplicações web. Em cima desse framework, podemos dizendo ainda, que existem diversos outros frameworks específicos para cada tipo de aplicação que desejamos desenvolver, tais como: **ASP NET (MVC)** padrão atual para desenvolvimento de Websites, Blogs entre outros, **ASP NET (Web API)** para criação de web services Restful, **ASP NET (Web Forms)** versão mais antiga para criação de aplicações web equivalente ao MVC, **ASP NET (SignalR)** para criação de aplicações em tempo real.

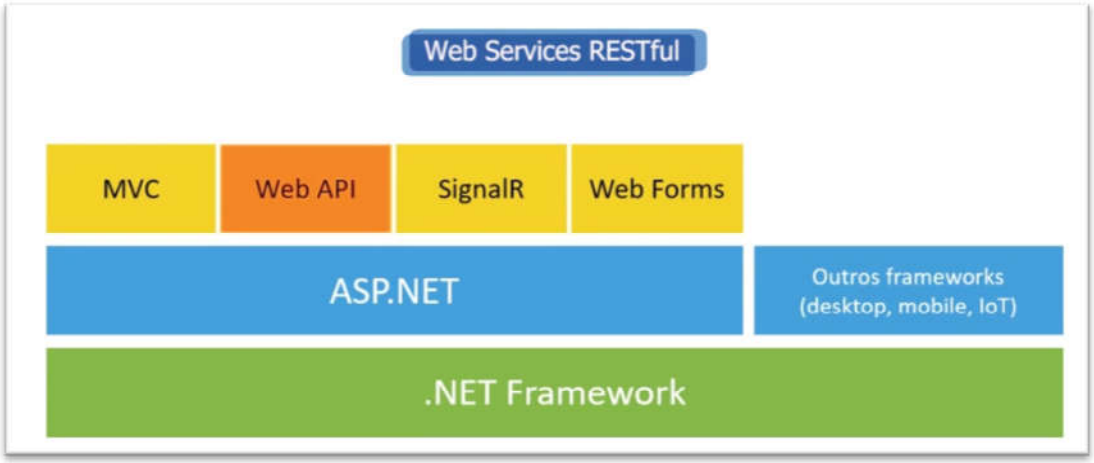


Figura 2 – Arquitetura do framework .Net

Na **Figura 2**, é possível identificar em qual parte da Arquitetura do dotNet framework fica situada a API de comunicação "Web API": responsável pela criação de serviços de comunicação via RESTFUL.

3.4 Framework Web API para criar WebServices RESTful

Sistemas sem uma API externa vivem isolados em silos de dados, algo cada vez mais raro, já que a necessidade de se integrar com outros sistemas é crescente. Por isso, ao desenvolver uma nova aplicação, pensa-se em como será sua integração com outras no futuro.

> "Existem diversas opções de integração para sistemas distribuídos, cada uma apresentando níveis de acoplamento e coesão distintos, centralizando ou distribuindo o controle sobre os processos que ocorrem, abordagens focadas em sistemas homogêneos ou nas quais os clientes serão aplicações desconhecidas de terceiros."[...]
>
> "Entre as varias possibilidades, tem ganhado destaque as que facilitam atingir requisitos não funcionais sem um trabalho extra, além de possibilitar uma evolução dos serviços evitando a quebra de clientes. É nesta linha de pensamento que a Web (com HTTP e hipermídia) como plataforma e a abordagem REST se sobressaem. Infraestrutura já existente, como caches HTTP, proxies, balanceadores e ferramentas de monitorações passam a funcionar como um middleware mais transparente e simples." (Silveira, et al., 2012, p. 193).

Web Services REST são serviços baseados no protocolo HTTP, que permitem diversas aplicações clientes de diferentes tipos de plataformas e linguagens consumir funcionalidades e dados a partir de um ponto central, ou seja, o Web Service. Esses dados normalmente estarão armazenados em um banco de dados qualquer.

Uma vez que ele se baseia no protocolo HTTP, permite que diversos tipos de aplicações se comuniquem com ele apenas utilizando requisições pelo protocolo HTTP, que é o protocolo base para o funcionamento das aplicações web.

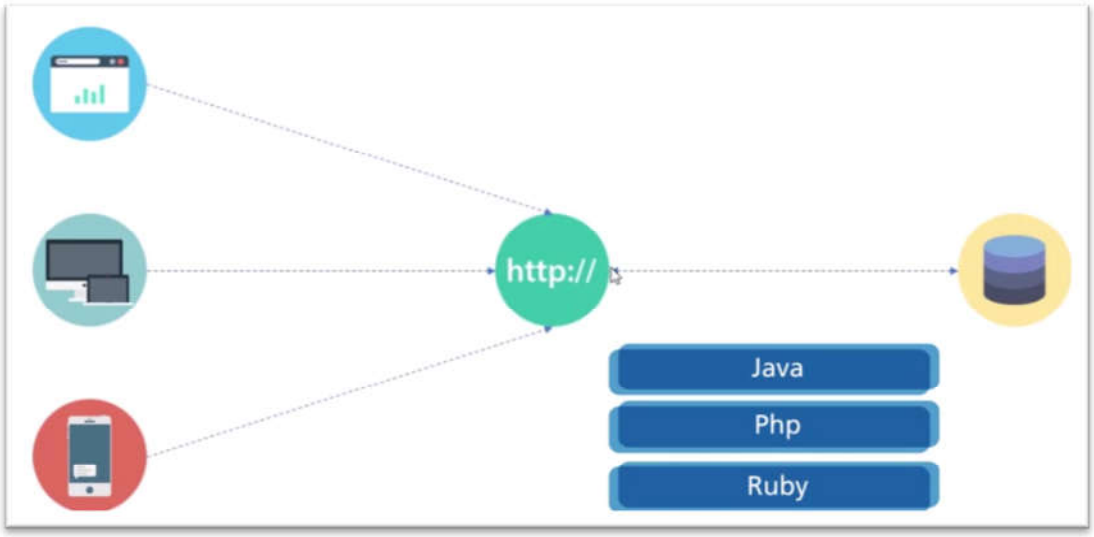


Figura 3 – Funcionamento de um Web Services RESTful / WebAPI

Como ilustra a **figura 3**, as aplicações clientes que irão consumir o meu serviço podem ser desenvolvidas, inclusive, em diferentes linguagens.

REST não é apenas para os tipos de dados JSON ou XML, pois é possível trafegar informações em outros formatos tais como: HTML, XHTML, PDF (binários), Imagens ou Textos comum.

Características do ASP NET Web API

- Baseia-se no padrão MVC (Model - View – Controller), provendo dados no formato JSON / XML;
- Possui um eficiente sistema de Roteamento e URLs amigáveis, com possibilidade de envio de parâmetros por aplicações clientes;
- Possui recursos de autenticação e autorização;
- Acesso a banco de dados;
- Manipulação de arquivos;
- Criptografia.

3.5 Utilização do protocolo HTTP

Hypertext Transfer Protocol (HTTP) é o método utilizado para enviar e receber informações na web. A versão mais utilizada atualmente é a 1.1, definida pela especificação RFC 2616. O protocolo pertence à camada de aplicação da suíte de protocolos TCP/IP. É um dos mais utilizados na internet, permitindo a interação dos clientes com os servidores web. (Pereira, 2012, p. 173).

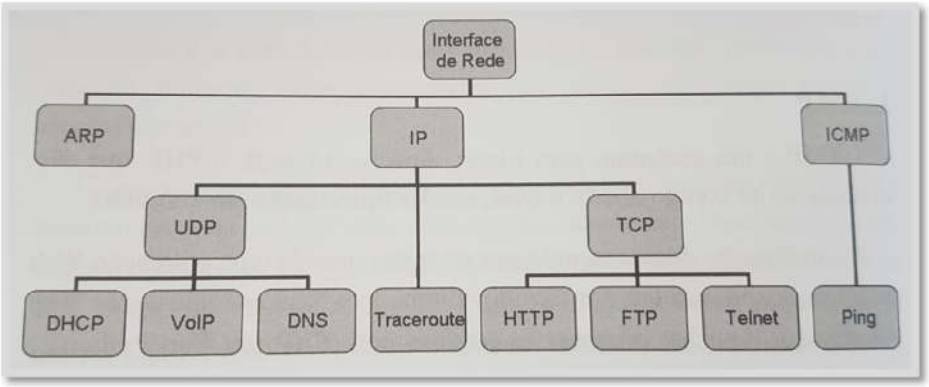


Figura 4 – Principais protocolos do TCP/IP

Como podemos observar na **figura 4**, na arquitetura TCP/IP, o protocolo HTTP fica abaixo – sob o guarda-chuva – do protocolo TCP, que é um protocolo confiável de entrega.

O **protocolo TCP** *(Transmission Control Protocol)* é usado para um transporte fim a fim **confiável** entre a origem e o destino. Para isso é estabelecida uma conexão entre os dois pontos, usando técnicas para assegurar que os pacotes sejam entregues de maneira ordenada, integra e confiável. (Pereira, 2012, p. 171).

Hoje, o protocolo padrão para troca de informações é o HTTP. A porta 80 é bem-vinda pelos firewalls, e, ao mesmo tempo, existem diversas ferramentas, clientes, servidores e intermediários que suportam o protocolo. Muitas dessas ferramentas trazem soluções, por exemplo, para problemas de desempenho e escalabilidade no uso do HTTP. Além disso, é possível desenvolver sistemas Web nas mais diversas linguagens de programação.

> Adotando a característica do protocolo HTTP de que toda URI identifica algo, alguns sistemas utilizam diversas URIs e um único método, em geral *POST*. Em um próximo passo de adoção do protocolo, diversos métodos são utilizados para facilitar a identificação da tarefa sendo executada, como *GET*, para obter informações, *POST*, para criar, *PUT*, para alterar, *DELETE*, para remover, etc. Isso permite que o sistema como um todo se beneficie das vantagens de cada método onde o mesmo for adequado, como ao cachear resultados de uma query executada através de uma requisição *GET*.
>
> Com a popularização de outros formatos que fazem ou não uso de *schemas*, como o JSON, diversos serviços expostos na Web passaram a visar à simplicidade, expressividade, ou, ainda, aumentar a compatibilidade entre versões diferentes dos serviços. (Silveira, et al., 2012).

Com o advento das tecnológicas baseadas em REST, desenvolvedor de aplicações web passou a ter uma necessidade maior de conhecer mais a respeito do protocolo HTTP.

3.6 Como funciona uma arquitetura baseada em REST

Quando começamos a desenvolver – ou consumir - nossos primeiros serviços RESTful, a primeira coisa que precisamos entender é o papel dos verbos HTTP dentro do contexto REST.

A ideia geral é a seguinte: seu serviço vai prover uma *URL* base e os verbos HTTP vão indicar qual ação está sendo requisitada pelo consumidor do serviço.

Por exemplo, considerando a URL www.dominio.com/rest/notas/, se enviarmos para ela uma requisição HTTP utilizando o verbo GET, provavelmente obteremos como resultado uma listagem de registros (notas, nesse caso). Por outro lado, se utilizarmos o verbo POST, provavelmente, estaremos tentando adicionar um novo registro cujos dados serão enviados no corpo da requisição.

Da mesma forma, a URL www.dominio.com/rest/notas/1, por exemplo, poderia ser usada para diferentes finalidades, dependendo do verbo enviado na requisição. No caso do GET, essa URL provavelmente deveria nos retornar o registro de ID 1 (nesse caso, a nota de ID = 1). Já o verbo DELETE indicaria que desejamos remover esse registro.

Repare que a URL se mantém – o verbo indica o que estamos fazendo de fato. Por exemplo, não precisamos disponibilizar no serviço uma URL como /notas/listar ou /notas/remover/1.

Em linhas gerais, as aplicações usam os Verbos HTTP em frameworks de CRUD da seguinte forma: GET (Buscas), POST (Inserções), PUT (Atualizações), DELETE (Remoções) e PATCH: Usado para editar o recurso sem a necessidade de enviar todos os atributos – o consumidor envia apenas aquilo que de fato foi alterado (mais o ID como parâmetro, para que o serviço saiba o que vai ser alterado).

3.7 Padrões de resposta do serviço

A documentação indica que o serviço pode retornar o resultado em diversos formatos – JSON, XML, texto plano, etc. Contudo, atualmente o formato mais adotado tem sido o JSON, por seu formato leve, legível e sua fácil interpretação por diversas tecnologias.

Além disso, o protocolo HTTP dispõe de diversos códigos (ou status) que devem ser incluídos na resposta, indicando o resultado do processamento.

Os códigos iniciados em "2" indicam que a operação foi bem sucedida. Nessa categoria temos, por exemplo, **código 200 (OK)**, iniciando que o método foi executado com sucesso; **201 (Created)** quando um novo recurso foi criado

no servidor; e **204 (No Content)** quando a requisição foi bem sucedida, mas o servidor não precisa retornar nenhum conteúdo para o cliente.

Já os códigos iniciados em "4" indicam algum erro que provavelmente partiu do cliente. Por exemplo, o código **400 (Bad Request)** indica que a requisição não pôde ser compreendida pelo servidor, enquanto o **404 (Not Found)** indica que o recurso não foi localizado.

Há, ainda, os códigos que indicam erro do lado do servidor. Nesse caso, eles iniciam com "5", como **o 500 (Internal Server Error)**, que indica que ocorreu um erro internamente no servidor que o impediu de processar e responder adequadamente a requisição.

Existe uma grande quantidade de status, divididos em diversas categorias. Abaixo, ilustraremos numa tabela os códigos de status mais comuns.

Status	Descrição Detalhada
200 OK	A requisição foi bem sucedida.
301 Moved Permanently	O recurso foi movido permanentemente para outra URI.
302 Found	O recurso foi movido temporariamente para outra URI.
304 Not Modified	O recurso não foi alterado.
401 Unauthorized	A URI especificada exige autenticação do cliente. O cliente pode tentar fazer novas requisições.
403 Forbidden	O servidor entende a requisição, mas se recusa em atendê-la. O cliente não deve tentar fazer uma nova requisição.
404 Not Found	O servidor não encontrou URI correspondente.
405 Method Not Allowed	O método especificado na requisição não é válido na URI. A resposta deve incluir um cabeçalho Allow com uma lista dos métodos aceitos.
410 Gone	O recurso solicitado está indisponível, mas seu endereço atual não é conhecido.
500 Internal Server Error	O servidor não foi capaz de concluir a requisição devido a um erro inesperado.
502 Bad Gateway	O servidor, enquanto agindo como proxy ou gateway, recebeu uma resposta inválida do servidor upstream a que fez uma requisição.
503 Service Unavailable	O servidor não é capaz de processar a requisição pois está temporariamente indisponível.

Tabela 1 – Lista de status do protocolo HTTP

Apesar da **tabela 1** listar os status mais utilizados na maioria dos projetos, a lista completa de status do protocolo HTTP pode ser facilmente encontrada na especificação RFC 2616, que é um documento que especifica um protocolo de rastreamento de padrões da Internet para comunidade, e solicita discussão e sugestões para melhorias.

3.8 Front-end: Construindo Aplicações com Angular

O Angular é uma plataforma que facilita a criação de aplicativos com a web. Angular combina modelos declarativos, injeção de dependência, ferramentas de ponta a ponta e práticas recomendadas integradas para resolver desafios de desenvolvimento. O Angular permite que desenvolvedores criem aplicativos que estejam na Web, em dispositivos móveis ou na área de trabalho.

É preciso ressaltar que o framework do Angular sofreu uma grande mudança desde sua versão de lançamento com o AngularJs até o Angular 2. A forma de programar e arquitetura do framework mudaram completamente. Não existe mais a necessidade de usar $scope do AngularJs e passamos a criar componentes, módulos, templates, serviços e diretivas de uma maneira completamente diferente. Outra novidade foi a inclusão da linguagem TypeScript.

> "O TypeScript pode ser considerado uma linguagem de programação que contém as seguintes particularidades:
>
> - Compatível com a nova especificação ECMAScript 2015 (ES6)
> - Linguagem tipada
> - Implementa Annotations
> - Implementa Generics
>
> Podemos afirmar que o TypeScript torna a linguagem JavaScript semelhante às linguagens derivadas do c, como php, c#, java, etc.
>
> [...] Após escrever o código TypeScript com alguma funcionalidade, é preciso converter este código para JavaScript puro. (Schmitz & Georgii, 2016, p. 11).

Em termos de Arquitetura, o Angular é uma plataforma e estrutura para criar aplicativos clientes em HTML e TypeScript. Angular é escrito em TypeScript. Ele implementa a funcionalidade principal e opcional como um conjunto de bibliotecas TypeScript que você importa para seus aplicativos.

Na documentação oficial do framework temos uma imagem que exibe tudo aquilo que você deve conhecer sobre o Angular 2. A figura a seguir mostra a arquitetura do Angular e como as partes básicas estão relacionadas.

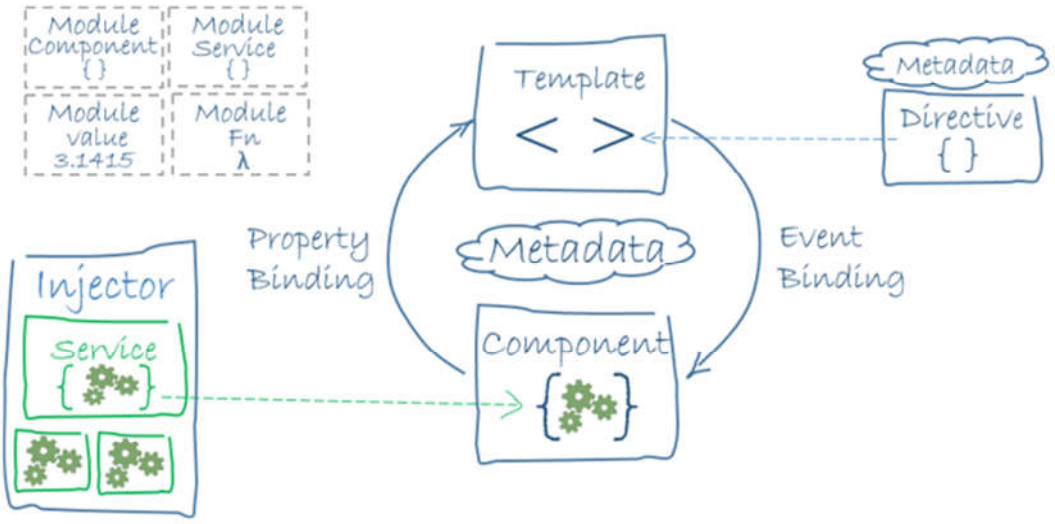


Figura 5 – Arquitetura do Angular 7

De forma geral, no angular, a partir da versão 2, para construirmos uma aplicação é preciso entender cada um dos conceitos de Modulo, Componente, Template, Metadata, Serviço, Diretiva e Injeção de dependência e entender como esses conceitos estão relacionados entre si.

3.9 Arquitetura do Sistema

Neste tópico comentaremos a arquitetura da aplicação JVRIS Saúde e como ela foi desenvolvida.

Criamos uma arquitetura hibrida, capaz de abordar varias tecnologias de diferentes fabricantes, desse modo, o projeto foi construído em cima duma arquitetura que utiliza o framework Angular 7 no front-end e ASP .NET Web API no bastidores (back-end).

As aplicações SPA – Single Page Applications – são a vedete do momento. Muitas empresas e desenvolvedores estão migrando suas aplicações, tornando-as aplicações SPAs, usando frameworks como o Angular

ou React com o intuito de melhorar cada vez mais a experiência final do usuário.

O dot Net, assim como Java, possui um framework capaz de prover um middleware que faça a busca da solicitação do usuário, vai ao modelo, que por sua vez, busca o dado no banco de dados, retorna com ele para a camada de controle, serializa-o em formato JSON e disponibiliza o conteúdo no WebService. A partir daí, o programador front-end pode pegar os dados via protocolo REST, pega o conteúdo em formato JSON, deserializa e exibe na interface que o usuário desejar, seja ela uma interface WEB, Mobile ou até mesmo Desktop.

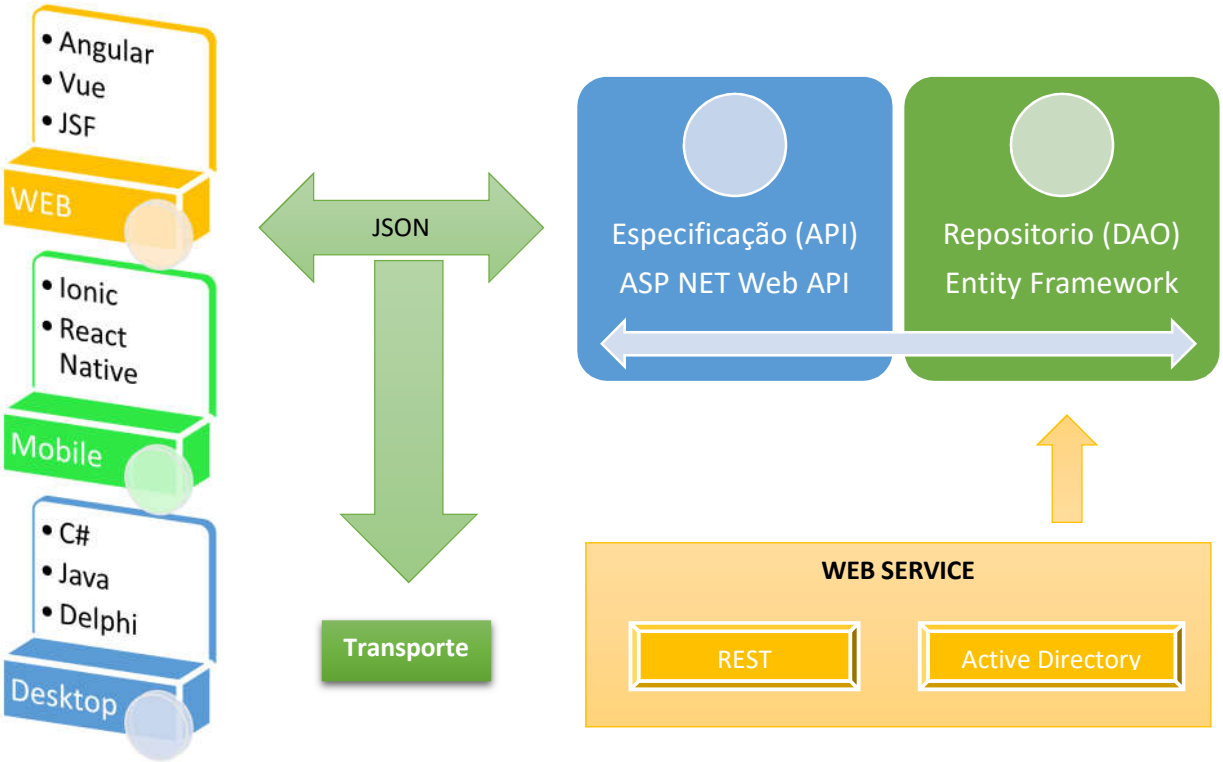


Figura 6 – Arquitetura do projeto proposto

A **figura 6**, que ilustra a arquitetura do sistema que acabamos de observar, refere-se ao projeto que foi desenvolvido para a Procuradoria Geral do Estado - PGE.

Uma vez explicado a arquitetura inicial, vamos explorar alguns requisitos do sistema, fazendo um paralelo entre o código-fonte, interface do usuário e arquitetura do sistema.

Requisito 1: Um servidor da PGE deseja entrar no sistema do JVRIS Saúde com o seu usuário e senha.

Assim que o servidor informar seu usuário e senha, o sistema vai ao WebService, faz uma busca na base de usuários do Active Directory (AD), envia um objeto de confirmação para o back-end (ASP NET Web API), o objeto de confirmação ou rejeição é enviado para a camada de front-end (Angular 7), o angular trata o objeto de retorno (a mensagem) e, em seguida, pode ir por dois caminhos distintos: ou redireciona o usuário para o menu principal do sistema, ou exibe uma mensagem de usuário ou senha inválidos e mantém o sistema na tela de Autenticação.

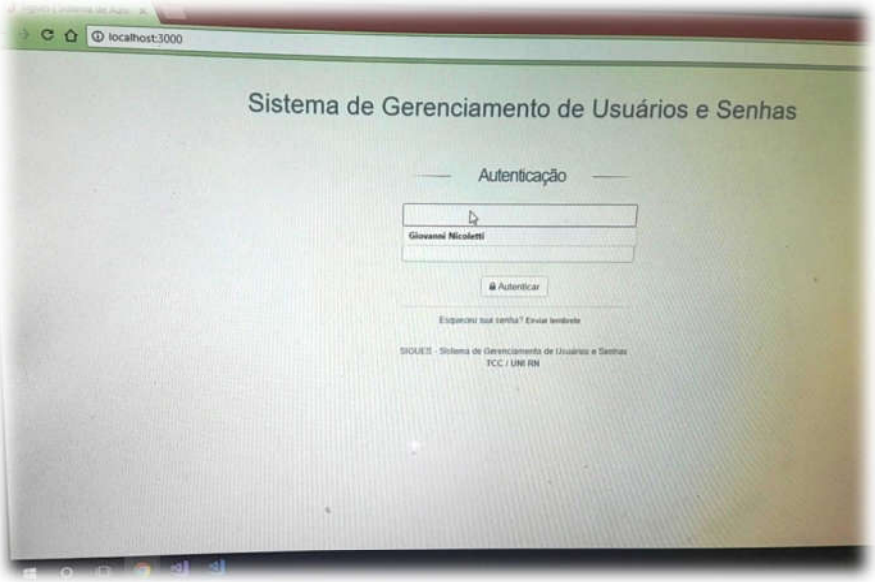


Figura 7 – Tela de Autenticação do sistema

No trecho de código abaixo escrito em C#, o método Autenticar do back-end está recebendo um objeto “token”, e vai consultar no WebService do Active Directory se o usuário e senha decodificados são validos. Nesse caso, há três tipos de retornos possíveis: 200 – ok e envia o nome do usuário de volta na mensagem, 403 - forbirden (acesso não permitido) ou 500 – erro interno no servidor.

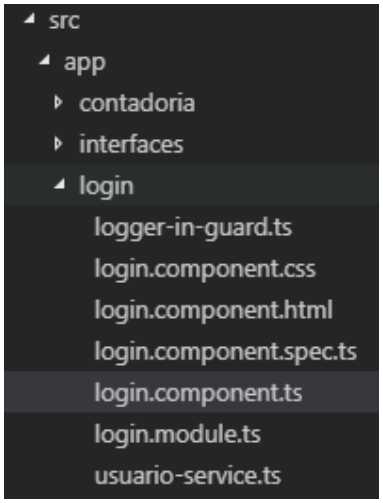
```
[EnableCors(origins: "http://localhost:4200", headers: "*", methods: "*")]
public class AutenticarController : ApiController
{
    [HttpPost]
    [Route("api/Autenticar")]
    public HttpResponseMessage Autenticar([FromBody]object token)
    {
        try
        {
            var vUsr = UsuarioHelper.DecodeUsuario(token);
            var usuario = new Usuario
            {
                nuInstrucao = Instrucao.Validar,
                txUsuario = vUsr.usuario,
                txContraSenha = vUsr.senha,
                Perfil = new Perfil { idSistema = vUsr.idSistema }
            };

            var ws = new WsSiGUeSSoapClient();
            usuario = ws.UsuarioProcessar(usuario);

            if (usuario.isSucessfull)
            {
                usuario.txContraSenha = "";
                return Request.CreateResponse<Usuario>(HttpStatusCode.OK, usuario);
            }
            else
            {
                return Request.CreateErrorResponse(HttpStatusCode.Forbidden,
                usuario.Mensagem);
            }
        }
        catch (Exception ex)
        {
            return Request.CreateErrorResponse(HttpStatusCode.InternalServerError,
                "Ocorreu um erro enquanto o servidor realizava operação.", ex);
        }
    }

}
```

Passando para o front-end, que foi implementada em Angular 7. Ao abrir a pasta de login é possível visualizar a estrutura de arquivos e pastas necessários para fazer funcionar a tela da aplicação.



O modulo de login possui o componente login.component.ts, responsável pela submissão do formulário de autenticação do login e senha do usuário. Nesse mesmo modulo, temos ainda outros arquivos que não serão detalhados, mas que fazem parte do Login, auxiliando desde a montagem do HTML e CSS até o encapsulamento dos métodos mais complexos, módulos e serviços complementares.

O código abaixo, **figura 08**, detalha como a aplicação Angular trata a requisição de login do usuário. De forma sucinta, vai ao serviço REST, invocando o método SendPost e envia o usuário e senha criptografado. O ASP NET Web API decodifica o objeto, invoca o WebService do Active Directory – AD para validar o usuário informado no Angular. A resposta é tratada dentro do método *(subscribe)*, que é assíncrono e por isso, fica aguardando até que a resposta do servidor chegue, para que assim, a aplicação possa seguir o fluxo da autenticação.

```
onSubmit(form) {
  this.codeResposta = '';
  this.errorResposta = '';

  let _usuario: Usuario = new Usuario(this.usuario, this.senha);

  let _usuarioJson = JSON.stringify(_usuario);      giovanni nicoletti, 2 years ago • #314 implement
  if (_usuarioJson) {
    _usuarioJson = this._convertBase64.encode(_usuarioJson);

    this._httpService.SendPost<RespostaModel>('Usuario', _usuarioJson)
    .subscribe (
      (res) => { // 200 OK - Resposta padrão para pedidos HTTP bem-sucedidos.
                 this.codeResposta = res;
                 this.errorResposta = '';
                 this._user.autentica(_usuario); },
      (error) => { // 403 Forbidden - O pedido foi válido, mas o servidor está recusando a ação.
                   this.codeResposta = '403';
                   this.errorResposta = error;
                   this._user.autentica(null); },
      () => {  } )
  } else  {
    // 401 Unauthorized - Não autorizado. Usado quando a autenticação é necessária e falhou ou
    // ainda não foi fornecida
    this.codeResposta = '403';
    this.errorResposta = '401 Unauthorized - autenticação é necessária e ainda não foi fornecida.';
    this._router.navigate(['']);
  }
}
```

Figura 8 – Trecho de código em Angular, método onSubmit()

Detalhando o serviço RESTFUL

O REST significa *Representational State Transfer*. Em português, **Transferência de Estado Representacional**. Trata-se de uma abstração da arquitetura da Web. Resumidamente, o REST consiste em princípios/regras/constraints que, quando seguidas, permitem a criação de um projeto com interfaces bem definidas. Desta forma, permitindo, por exemplo, que aplicações se comuniquem.

Todo REST é um Web Service, mas nem todo Web Service é REST. REST simplifica o uso de Web Services. Web Services REST usam verbos HTTP para indicar ações (Put, Post, Get e Delete). Web Services REST são Stateless (Toda requisição é autossuficiente).

Uma vez entendido o conceito do termo REST, podemos finalizar com o conceito do que seria o termo RESTful. Desse modo, podemos dizer que REST e RESTful representam o mesmo principio. No entanto, sistemas que utilizam os princípios REST são chamados de RESTful. Enquanto que REST é o conjunto de princípios de arquitetura, o RESTful é a capacidade de determinado sistema aplicar os princípios de REST.

Na aplicação front-end, foi criado o serviço chamado: http-restful.service.ts, com um método especifico para tratar todos os verbos HTTP. No caso do Login, fizemos uso do método SendPost. Esse método invoca o serviço HTTP, que é nativo do Angular - por Injeção de dependência - chama o verbo POST e retorna a resposta num padrão JSON.

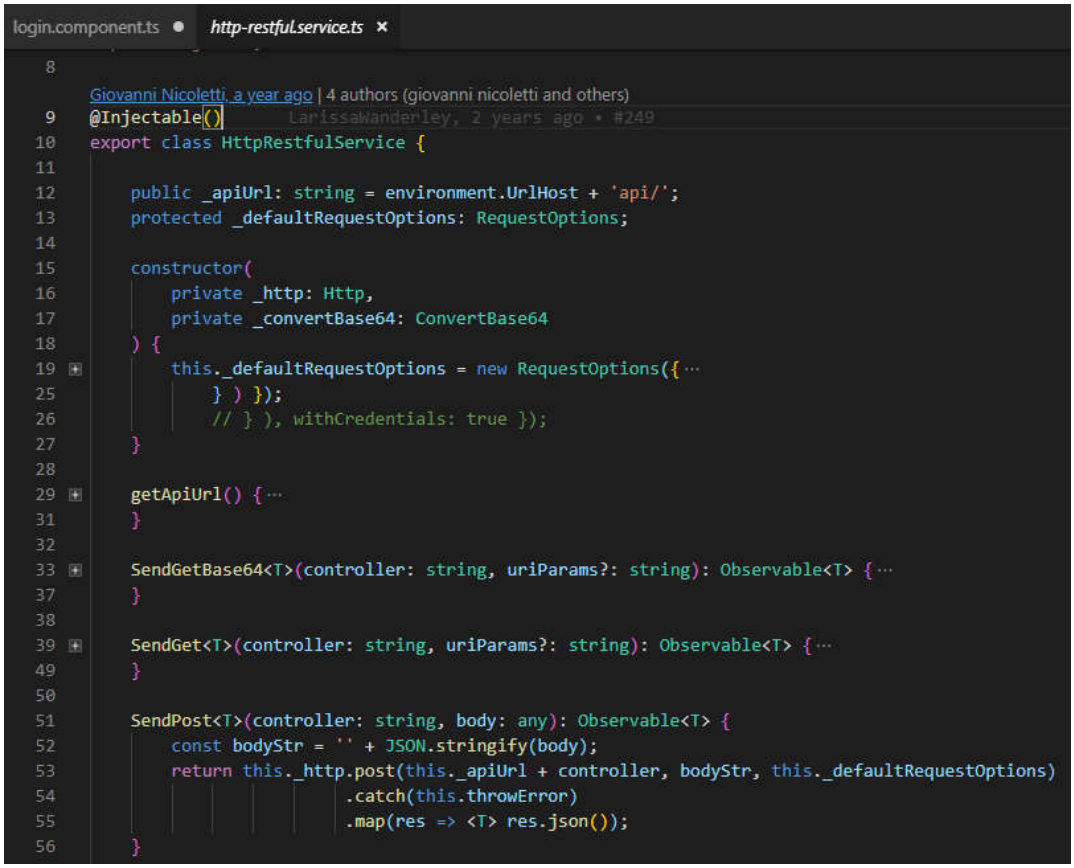


Figura 9 – Trecho de código da API do serviço REST no Angular 2 (Front-end)

Na **figura 9** acima, temos alguns métodos implementados em Angular, dentre eles o SendPost, que mais internamente chama o verbo POST do protocolo HTTP, passando a URL a ser invocada como parâmetro do método. Existem ainda, no mesmo método, mais duas opções que são catch e map: o **catch** faz o tratamento de exceção, Pois, caso algum erro seja gerado que tenhamos a possibilidade de lançar essa exceção para cima. Já o **map** é responsável por retorna a resposta vinda do back-end em formato JSON.

Requisito 2: O servidor deseja consultar um medicamento. Ou seja, o usuário deseja abrir a tela de consulta de medicamentos e filtrar do um medicamento especifico e, se o medicamento existir, o sistema deve apresentar no browser todos os dados do medicamento solicitado em detalhes.

Vamos explicar o requisito 2, usando o modelo bottom-up. Ou seja, desde o ASP .NET WebAPI (back-end), passando pelo Angular (front-end) e mostrando a aplicação Web rodando no browser.

```
using DBBroker.Mapping;
using System.ComponentModel.DataAnnotations;

namespace Jvris.Domain.Module.Saude.Dominio
{
    [DBMappedClass(Table = "saude.tbMedicamentos", PrimaryKey = "idMedicamento")]
    public class Medicamento
    {
        public int Id { get; set; }

        [Required, MaxLength(120)]
        public string txMedicamento { get; set; }

        public Forma Forma { get; set; }

        public Grupo Grupo { get; set; }

    }
}
```

Figura 10 – Classe de Domínio (Medicamento)

Como podemos observar na **figura 10**, no nosso modelo de domínio, a classe "**Medicamento"** tem nome de medicamento, tem uma **Forma Farmacêutica**, que pode ser em cápsula, drágea, comprimido, grânulo, em pó, etc. e pertence a um **Grupo**, tais como: Analgésicos, Anti-inflamatórios, Anti-infecciosos, Antissépticos, Antifúngicos, Antibióticos, etc.

```
public class MedicamentoController : ApiController
{
    [HttpGet]
    [Route("api/Medicamento")]
    public HttpResponseMessage ListarTodos()
    {
        var tb = DBMedicamento.GetAll("txMedicamento");
        return Request.CreateResponse(HttpStatusCode.OK, tb);
    }
```

Figura 11 – Classe Controller de Medicamento

Na **figura 11**, o método ListarTodos() faz exatamente o que está escrito, traz uma lista completa com todos os medicamentos cadastrados no banco de dados.

- A linha **[HttpGet]** define que iremos invocar esse métodos através do protocolo HTTP, usando o verbo GET.
- A linha **[Route("api/Medicamento")]** define que iremos acessar o método através de uma rota, algo semelhante ao endereço: http://localhost:1200/api/medicamento.
- A linha **public HttpResponseMessage ListarTodos()** declara o método ListarTodos como um método publico e retorna uma mensagem no padrão HTTP. Pois, a classe HttpResponseMessage é responsável por esse tipo de implementação.
- A linha **var tb = DBMedicamento.GetAll("txMedicamento");** instancia uma variável tb e recebe o retorno do GetAll, que por sua vez, retorna uma lista de todos os medicamentos ordenados por txMedicamento.
- A linha **return Request.CreateResponse(HttpStatusCode.OK, tb);** lança para fora do back-end o retorno HTTP = 200 e o objeto tb, contendo a lista de medicamentos do sistema.

Temos outras variações de consulta como podemos ver na figura abaixo:

```
[HttpGet]
[Route("api/Medicamento/{id}")]
public HttpResponseMessage ConsultarPor([FromUri]int id)
{
    var tb = DBMedicamento.GetById(id);
    return Request.CreateResponse(HttpStatusCode.OK, tb);
}

[HttpGet]
[Route("api/Medicamento/{medicamento}/{forma}/{grupo}")]
public HttpResponseMessage ConsultarPorGrupo([FromUri]string medicamento,
                                             [FromUri]int forma,
                                             [FromUri]int grupo)
{
    if (medicamento == "null") medicamento = "";
    var tb = DBMedicamento.Pesquisa(medicamento, forma, grupo);
    return Request.CreateResponse(HttpStatusCode.OK, tb);
}
```

Figura 12 – Invocando o método GET na Classe MedicamentoController

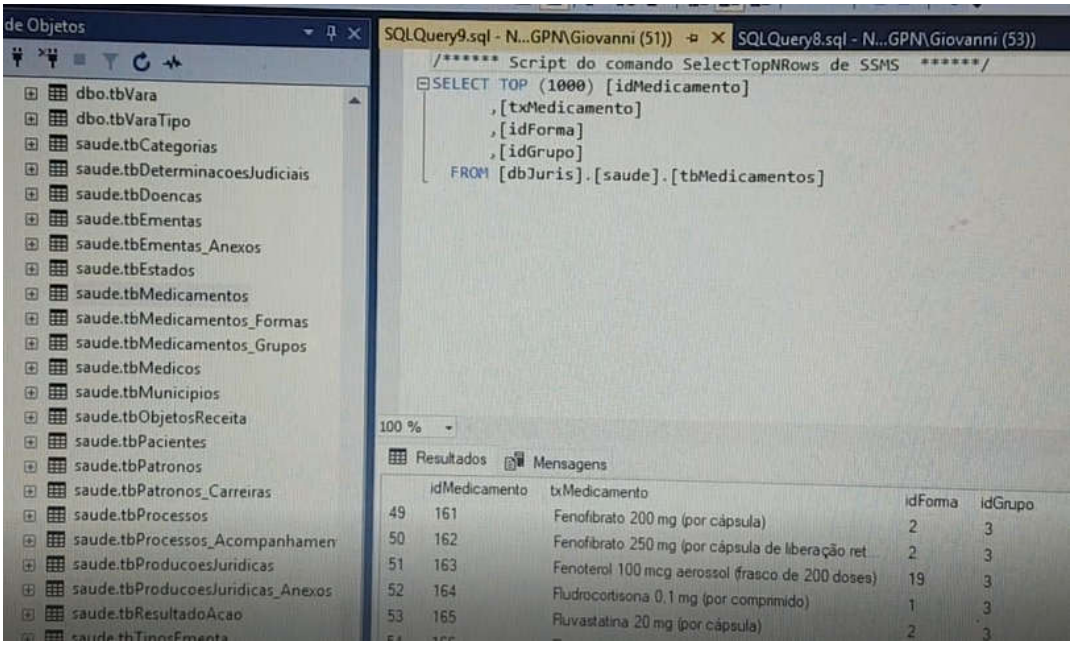


Figura 13 – Tabela criada no BD MSSQL Server

Como podemos observar na **figura 13**, a Classe Medicamentos possui uma tabela tbMedicamentos associada a ela. A tabela Medicamento possui relacionamento de 1:N com Grupo e Forma. Ou seja, Medicamento N:1 Grupo e Medicamento N:1 Forma.

Uma vez explicado os bastidores da aplicação back-end, é hora de aprofundarmos na explicação no front-end e mostrar como que o Angular faz para interagir com a API REST do ASP .NET. Para isso, inicialmente precisamos compreender que a tela de medicamentos para ser construída, deve ser composta de quatro arquivos: um HTML, chamado medicamento.component.html, que tem o DOM da pagina web; um arquivo CSS, medicamento.component.css, para definir a apresentação (aparência) da tela; um COMPONENTE, medicamento.componente.ts, que contém a implementação em TypeScript do arquivo medicamento.component.html; e um SERVIÇO, service.ts, que é responsável pela interação com a API do back-end.

```
goListarTodos() {
    try {
        this._service.ListarTodos('Medicamento')
            .subscribe(
            (res) => {
                this.pages.listaTotal = res;
                this.pages.popularLista(res);
            },
            (error) => { this.ComponentMsg.objMensagem.onError(error); },
            () => { });
    } catch (errors) {
        console.log('erro ao listar: ' + errors);
    }
}
```

Figura 14 – Componente, medicamento.componente.ts.

```
7  import { HttpRestfulService } from '../../shared/services/http-restful.service';
8  import { IService } from './../interfaces/service.interface';
9  import { RespostaModel } from '../models/resposta.model';
10
   giovanni nicoletti, 2 years ago | 2 authors (LarissaWanderley and others)
11 @Injectable()
12 export class MedicamentoService implements IService<Medicamento> {
13
14   private headers: Headers = new Headers({ 'Content-Type': 'application/json' });
15   private _url: string;
16
17   constructor(
18     public http: Http,
19     public _restfulService: HttpRestfulService
20   ) { }
21
22   ListarTodos(controller: string): Observable<RespostaModel> {
23     return this._restfulService.SendGet<RespostaModel>(controller);
24   }
```

Figura 15 – Serviço, medicamento.service.ts

Nas **figuras 14 e 15**, podemos observar que o método goListarTodos (**figura 14**) chama o serviço (**figura 15**), invoca o método ListarTodos(), que por sua vez, faz uma chamada HTTP através do método GET para um Webservice REST e retorna a lista de medicamentos para o componente. Caso ocorra algum erro, o Angular captura a exceção vinda do back-end e lança para um componente especifico de mensagens.

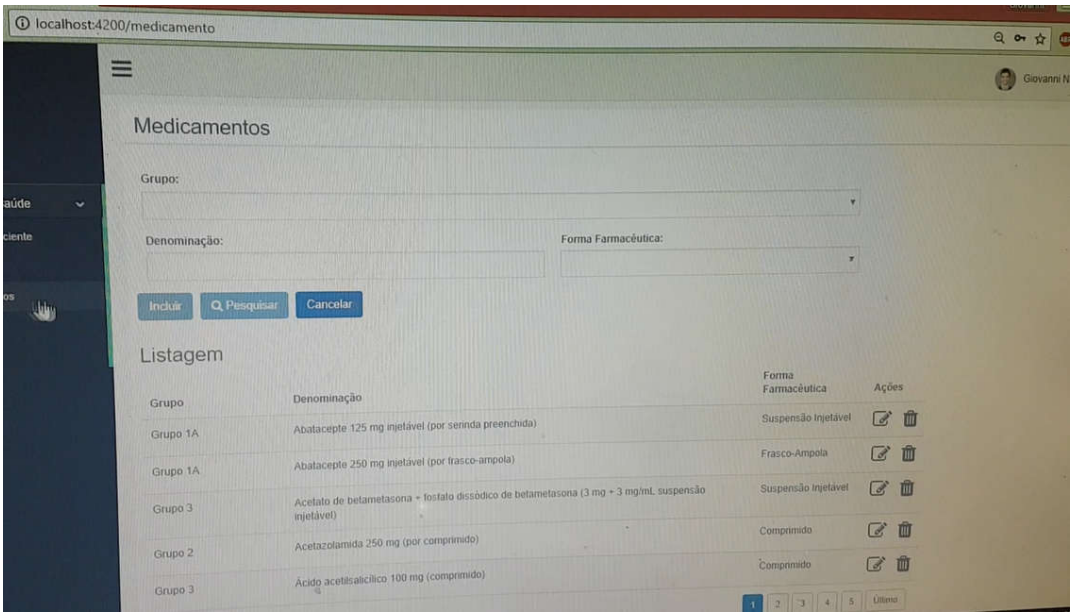


Figura 16 – Tela de Medicamentos

Como podemos observar na **figura 16**, na tela de medicamentos é possível realizar todas as operações de CRUD (Create, Recovery, Update, Delete). Os botões em azul permitem que o usuário possa incluir um novo medicamento, cancelar operações de inclusão e alteração ou pesquisar por um medicamento por qualquer, conforme os filtros dos inputs da parte superior da tela. Abaixo, após os botões, temos: a listagem dos medicamentos com varias colunas que vem do banco de dados e uma coluna de **"Ações"** com os ícones responsáveis por Alterar e Excluir um medicamento em especifico.

Requisito 3: O servidor deseja alterar informações de um medicamento existente. Ou seja, o usuário irá escolher o medicamento, podendo, para isso, fazer uso dos filtros de consulta dispostos na parte superior da tela de medicamentos (ver Figura 16), e clicar no primeiro ícone da coluna de **"ações"**, que é o ícone responsável por alterar informações da linha.

```
onPesquisar() {
    let denominacao;
    let forma;
    let grupo;

    (this.medicamento.txMedicamento == null || this.medicamento.txMedicamento.length == 0) ?
        denominacao = 'null' : denominacao = this.medicamento.txMedicamento;
    forma = (this.medicamento.Forma.Id == null ? 0 : this.medicamento.Forma.Id)
    grupo = (this.medicamento.Grupo.Id == null ? 0 : this.medicamento.Grupo.Id)
    try {
        this._service.Pesquisar('Medicamento', denominacao, forma, grupo)
            .subscribe(
            (res) => {
                this.pages.listaTotal = res;
                this.pages.popularLista(res);
                this.pages.firstPage();
            },
            (error) => { this.ComponentMsg.objMensagem.onError(error); },
            () => { });
    } catch (errors) {
        console.log('erro ao listar: ' + errors);
    }
}

onAlterar() {
    try {
        this._service.Alterar('Medicamento', this.medicamento)
            .subscribe(
            (res) => {
                this.mostrarDados(res);
                // this.goListarTodos();
                this.goLimpar();
                this.ComponentMsg.objMensagem.Sucesso();
            },
            (error) => { this.ComponentMsg.objMensagem.onError(error); },
            () => { });
    } catch (errors) {
        console.log('erro ao alterar: ' + errors);
    }
}
```

Figura 17 – Método para Pesquisar e Alterar Medicamentos

Na **figura 17**, podemos observar a implementação dos dois métodos que juntos, são responsáveis por atender completamente o requisito 3. O método onPesquisar() tem um funcionamento semelhante ao método goListarTodos(),

que cometamos na **figura 14**. Embora os dois métodos invoquem, nos bastidores, o método GET do HTTP, o método onPesquisar() possui a opção de passarmos parâmetros para dentro da função. Com essa diferença, enquanto o método goListarTodos() só consegue trazer todos os registros numa única chamada GET, o método onPesquisar() pode trazer os medicamentos de forma filtrada, ou seja, possibilita trazer pequeno conjunto de dados

O método onAlterar(), por sua vez, é acionado pelo usuário, através do botão salvar da interface web, quando o mesmo tenta salvar informações dos medicamentos. Para concretizar tal operação, o componente do Angular, faz uma chamada ao serviço, que por sua vez chama o método PUT.

> O método *PUT* é o contrario de *GET*; em vez de ler a pagina, ele escreve a página. Esse método possibilita criar uma coleção de paginas Web em um servidor remoto. O corpo da solicitação contém a página. Ela pode ser codificada usando *MIME*, quando as linhas após o *PUT* poderiam incluir cabeçalhos de autenticação, para provar que quem chamou realmente tem permissão para realizar a operação solicitada. (Tanenbaum & Wetherrall, 2011, p. 432).

Aprofundando na implementação do Angular, entraremos agora dentro do serviço RESTful para entender como os métodos onPesquisar e onAlterar() invocam os métodos GET e PUT.

http-restful.service.ts

```
39    SendGet<T>(controller: string, uriParams?: string): Observable<T> {
40        if (uriParams) {
41            return this._http.get(this._apiUrl + controller + '/' + uriParams, this._defaultRequestOptions)
42                             .catch(this.throwError)
43                             .map(res => <T> res.json());
44        } else {
45            return this._http.get(this._apiUrl + controller, this._defaultRequestOptions)
46                             .catch(this.throwError)
47                             .map(res => <T> res.json());
48        }
49    }
50
51    SendPut<T>(controller: string, body: any): Observable<T> {
52        const bodyStr = '' + JSON.stringify(body);
53        return this._http.put(this._apiUrl + controller, bodyStr, this._defaultRequestOptions)
54                         .catch(this.throwError)
55                         .map(res => <T> res.json());
56    }
```

Figura 18 – Serviço HTTP do Angular

O arquivo *http-restful.service.ts* funciona como um framework no Angular com todos os verbos mais utilizados do protocolo HTTP, esse framework faz a intermediação entre a API do ASP .NET e o Angular, sua existência é fundamental para que possamos manipular os dados da nossa aplicação Web.

Na **figura 18**, podemos observar dentro do arquivo http-*restful.service.ts*, os dois métodos responsáveis por consultar e alterar informações no back-end, são eles os métodos: http.get() responsável por realizar operações GET e http.put responsável por realizar operações PUT.

Passando para a camada de back-end, mostraremos na figura a seguir, como a API do ASP .NET implementa o mecanismos de alteração de medicamentos capaz de receber a requisição vinda do Angular, converter o JSON em um objeto que o back-end possa entender, persistir as informações no Banco de Dados e retornar, em JSON, a mensagem de confirmação ou erro para que o Angular possa exibir em tela.

```
[HttpPut]
[Route("api/Medicamento")]
public HttpResponseMessage Alterar([FromBody]Medicamento tb)
{
    try
    {
        tb.Forma = DBForma.GetById(tb.Forma.Id);
        tb.Grupo = DBGrupo.GetById(tb.Grupo.Id);
        DBMedicamento.Save(tb);
        return Request.CreateResponse(HttpStatusCode.OK, tb);

    }
    catch (Exception ex)
    {
        DBAppLog.Save(new AppLog { txMensagem = ex.Message, txStack = ex.StackTrace });
        return Request.CreateErrorResponse(HttpStatusCode.InternalServerError,
            "Erro: Registro não foi alterado", ex);

    }
}
```

Figura 19 – Método para alterar um medicamento no ASP .NET

Requisito 4: O servidor deseja cadastrar um novo medicamento. Ou seja, o usuário deseja abrir a tela de medicamentos, clicar num botão de adicionar medicamento, informar todos os dados e ter a possibilidade de salvar alteração.

Começando pela implementação do Angular, a figura abaixo mostrará o serviço responsável por acionar, de fato, o método POST no back-end.O método é genérico e recebe qualquer controller (tela de cadastro do Angular). O SendPost<T> faz uma chamada POST via HTTP para ser capturada pelo WebService REST e tratado no back-end. O retorno do método POST é convertido em JSON e lançado para o componente do Angular. Quando o componente Angular é atualizado, o HTML que se refere ao componente também passa a ser atualizado e exibido em tela.

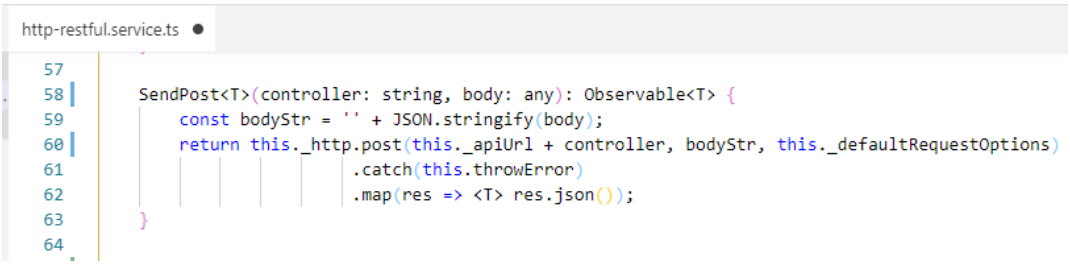


Figura 20 – Método POST do Angular

A figura 20, mostra como o Angular implementa uma requisição usando o protocolo HTTP, e envia os dados via POST para a API do ASP .NET.

> O método POST é usado quando os formulários são enviados. Tanto ele quanto o GET são usados também para Webservices[..].
>
> [...] Assim como GET, ele contém um URL, mas, em vez de simplesmente capturar uma página, ele faz o upload de dados para o servidor (ou seja, o conteúdo do formulário ou parâmetros de RPC). O servidor, então, faz algo com os dados que depende do URL, conceitualmente acrescentando os dados ao objeto. O efeito poderia ser comprar um item, por exemplo, ou chamar um procedimento. Finalmente, o método retorna uma página indicando o resultado. (Tanenbaum & Wetherrall, 2011, p. 432).

medicamento.component.ts

```
120
121     onIncluir() {
122         try {
123             this._service.Incluir('Medicamento', this.medicamento)
124                 .subscribe(
125                 (res) => {
126                     this.mostrarDados(res);
127                     // this.pages.listaTotal.push(this.medicamento);
128                     this.goLimpar();
129                     this.ComponentMsg.objMensagem.Sucesso();
130                 },
131                 (error) => { this.ComponentMsg.objMensagem.onError(error); },
132                 () => { });
133         } catch (errors) {
134             console.log('erro ao inserir: ' + errors);
135         }
136     }
```

Figura 21 – Salvando um medicamento

O método onIncluir(), ilustrado na **figura 21**, é acionado assim que o usuário clicar no botão de salvar da tela de Medicamentos (**figura 16**). O método chama o this._service.Incluir(), esse método vai no serviço base do framework do Angular, ilustrado na **figura 20**, e aciona o método POST do protocolo HTTP.

Estando o Webservice REST rodando perfeitamente, a API do ASP .NET consegue capturar a requisição POST e tratá-la da forma que for mais conveniente.

```
[HttpPost]
[Route("api/Medicamento")]
public HttpResponseMessage Incluir([FromBody]Medicamento tb)
{
    try
    {
        tb.Forma = DBForma.GetById(tb.Forma.Id);
        tb.Grupo = DBGrupo.GetById(tb.Grupo.Id);
        DBMedicamento.Save(tb);
        return Request.CreateResponse(HttpStatusCode.OK, tb);
    }
    catch (Exception ex)
    {
        DBAppLog.Save(new AppLog { txMensagem = ex.Message, txStack = ex.StackTrace });
        return Request.CreateErrorResponse(HttpStatusCode.InternalServerError,
            "Erro: Registro não foi incluido", ex);
    }
}
```

Figura 22 – Adicionar um medicamento no banco de dados

Na **figura 22**, podemos observar que o método Incluir, implementado em C# - na camada de back-end do ASP .NET - recebe um parâmetro do tipo Medicamento. O C# é inteligente o suficiente para converter o JSON que vem do Angular em um objeto do tipo Medicamento. Assim, temos um objeto Medicamento instanciado na variável ***tb***. Com essa variável *"em mãos"* é possível persistir o objeto no banco de dados. O framework de persistência encapsula a complexidade do INSERT no banco de dados de tal forma, que nos basta implementar a seguinte linha de comando: DBMedicamento.Save(**tb**). DBMedicamento é a classe de persistência responsável pelo Mapeamento Objeto-Relacional (MOR) com o banco de dados (BD), o método Save é responsável por serializar no BD um objeto qualquer, no nosso caso, foi serializado no BD o objeto Medicamento.

Requisito 5: O servidor deseja excluir um medicamento. Ou seja, o usuário deseja abrir a tela de consulta de medicamentos, filtrar do um medicamento especifico e ter a possibilidade de excluir o medicamento selecionado.

Excluir um medicamento tem algumas semelhanças com o requisito 3, alterar medicamento, ambos possuem ícones exibidos na coluna de Ações, da tela de listagem de medicamentos.

Uma vez que o usuário deseja realizar a exclusão de um medicamento do sistema, é muito comum que ele realize uma pesquisa usando os filtros / inputs da parte superior da tela de medicamentos, tal como é feito durante o processo de alteração (requisito 3). Assim que o usuário tiver consultado e escolhido o medicamento a ser excluído, basta um clique no ícone da lixeira para que o controller de medicamento seja acionado e o medicamento seja excluído do banco de dados.

medicamento.component.ts

```
156     onExcluir(Medicamento) {
157         // tslint:disable-next-line:prefer-const
158         let isConfirmed = confirm('Deseja realmente excluir ?');
159         if (isConfirmed) {
160             try {
161                 this._service.Excluir('Medicamento', Medicamento.Id)
162                     .subscribe(
163                     (res) => {
164                         // tslint:disable-next-line:prefer-const
165                         let novaLista = this.pages.listaTotal.slice(0);
166                         // tslint:disable-next-line:prefer-const
167                         let indice = novaLista.indexOf(Medicamento);
168                         novaLista.splice(indice, 1);
169                         this.pages.listaTotal = novaLista;
170                         this.goLimpar();
171                         this.ComponentMsg.objMensagem.Sucesso(res);
172                     },
173                     (error) => { this.ComponentMsg.objMensagem.onError(error); },
174                     () => { });
175             } catch (errors) {
176                 console.log('erro ao excluir: ' + errors);
177             }
178         }
179     }
```

Figura 23 – Excluindo Medicamento no Angular

O componente é interligado com a View (HTML) e é responsável por enviar a solicitação de exclusão do medicamento para o Webservice via REST através do verbo DELETE.

DELETE faz exatamente isso: exclui a página ou, pelo menos, indica que o servidor Web concordou em excluir a página. A exemplo de *PUT*, a autenticação e a permissão tem papel fundamental. (Tanenbaum & Wetherrall, 2011, p. 432).

A figura a seguir, ilustrará a implementação do método SendDelete(), que faz o envio da solicitação de exclusão do medicamento via HTTP.

http-restful.service.ts ×

```
65
66      SendDelete<T>(controller: string, id: any): Observable<T> {
67          // tslint:disable-next-line:triple-equals
68          if (typeof([1, 2]) == typeof(id)) {
69              let ids = '';
70              id.forEach((val, i, a) => {
71                  // ids += ((i)?'&':'') + 'ids=' + val;
72                  ids += ((i) ? '&' : '') + val;
73              });
74              return this._http.delete(this._apiUrl + controller + '/' + ids, this._defaultRequestOptions)
75                               .catch(this.throwError)
76                               .map(res => <T> res.json());
77          } else if (typeof('') === typeof(id) || (typeof(1) === typeof(id))) {
78              // let valor = "ids=" + id;
79              const valor = id;
80              return this._http.delete(this._apiUrl + controller + '/' + valor, this._defaultRequestOptions)
81                               .catch(this.throwError)
82                               .map(res => <T> res.json());
83          }
84      }
```

. **Figura 24** – Método DELETE do Angular

A **figura 24** mostra a implementação em mais baixo nível do Angular usando o serviço HTTP para chamar o método DELETE. O método SendDelete é genérico (assim como todos os outros métodos do serviço http-restful.service.ts), e recebe dois parâmetros: uma classe controller e um ID. O controller especifica a classe de domínio e o id determina o item a ser excluído.

Uma vez compreendida a maneira como o Angular envia uma requisição DELETE para o Webservice via REST, é hora de entender como a API do ASP .NET captura a mensagem do Angular e exclui o medicamento do banco de dados.

```
[HttpDelete]
[Route("api/Medicamento/{id}")]
public HttpResponseMessage Excluir([FromUri]int id)
{
    var excluir = DBMedicamento.GetById(id);

    if (excluir.Id > 0) // existe o registro
    {
        try
        {
            DBMedicamento.Delete(id);
            return Request.CreateResponse(HttpStatusCode.OK, "Excluido com sucesso.");

        }
        catch (Exception ex)
        {
            DBAppLog.Save(new AppLog { txMensagem = ex.Message, txStack = ex.StackTrace });
            return Request.CreateErrorResponse(HttpStatusCode.InternalServerError,
                "Erro: Registro não foi excluido.", ex);
        }
    }
    else
    {
        return Request.CreateErrorResponse(HttpStatusCode.NotFound, "Registro não encontrado.");
    }
}
```

Figura 25 – Excluir um medicamento no ASP .NET

A **figura 25** mostra que o método Excluir é acionado por uma chamada de DELETE para o WebService REST. A sintaxe *[HttpDelete]* na primeira linha do código serve para informar que o método Excluir irá escutar requisições de Delete trafegadas pelo protocolo Http. Já a sintaxe *Route* na segunda linha do código serve para que a API fique escutando numa URI especifica. Entrando na implementação do corpo do método, temos uma rotina simples e enxuta de exclusão do um Medicamento. Basicamente, o método Excluir realiza uma consulta pelo **ID** recebido e, se o registro a ser indicado para exclusão - de fato - existir no banco de dados, apague o medicamento cujo **ID** do parâmetro da requisição corresponda ao item. Ao final, é enviada uma mensagem de retorno igual a 200 para o caso de sucesso, e se ocorrer qualquer erro que faça o código entrar na rotina de exceção, o back-end lança o erro 500 com a mensagem detalhada da exceção para a camada de front-end.

4 CONSIDERAÇÕES FINAIS

Acompanhar os avanços tecnológicos não é uma tarefa fácil. O profissional de tecnologia da informação – TI precisa estar buscando novos conhecimentos a cada dia se quiser acompanhar as necessidades do mercado.

De acordo com o Engenheiro de Software Joel Spolsky, criador do software de gerenciamento de projetos Trello, *"pessoas de todo o mundo estão constantemente criando aplicações Web usando .NET, Java e PHP. Nenhuma delas está falhando por causa da escolha da tecnologia"*. Em outras palavras, faz-se necessário que o desenvolvedor de aplicações conheça mais de padrões de projeto e arquitetura de software, é preciso conhecer mais os conceitos por trás das ferramentas.

O projeto desenvolvido teve o objetivo de mostrar a facilidade de integração de diversos componentes de tecnologias distintas, em que todas elas se comunicavam muito bem entre si. Nesse projeto, criamos uma solução em que utilizamos: ASP .NET Web API no back-end, responsável pelas transações via Web Service com Restful, banco de dados em SQL Server e o Angular, um framework MVC, no front-end para exibir os dados no Browser.

Ademais, já que a arquitetura do sistema é toda dividida em módulos, é perfeitamente possível que a implementação do back-end seja feita com Java, a API de integração via restful com JAX-RS, o banco de dados em postgres e, ainda assim, o sistema seria perfeitamente simples de ser integrado com o front-end feito em Angular.

Devido à versatilidade e as particularidades inerentes aos sistemas Web, percebe-se que não existe um framework ideal, capaz de sanar todas as necessidades de um desenvolvedor. Por isso, cada vez mais, desenvolver softwares com varias tecnologias distintas e integradas - se comunicando - vem sendo uma tendência de mercado, cada vez mais comum nos dias de hoje.

5 REFERENCIAS

DEVMEDIA. 2019. **O que é ASP NET Web API?.** Disponível em: <https://www.devmedia.com.br/view/viewaula.php?idcomp=38132>. Acesso em: 01 Maio. 2019.

Documentação Oficial do Angular. **What is Angular?.** Disponível em: <https://angular.io/docs >. Acesso em: 01 Maio. 2019.

FOWLER, Martin. **Micrservices. A definition of this new architectural term.** Disponível em: <https://martinfowler.com/articles/microservices.html>. Acesso em: 18 de Maio de 2019.

FOWLER, M. (2004). ***Refatoração: Aperfeiçoando o projeto de código existente.*** Porto Alegre: Bookman.

PEREIRA, R. B. (2012). ***Java e Web PARA CONCURSOS***. Rio de Janeiro: Ciência Moderna.

PRESSMAM, R. S. (2011). ***Engenharia de Software* (7ª ed.).** Porto Alegre: AMGH Editora Ltda.

SCHMITZ, D., & GEORGII, D. P. (2016). ***Angular 2 na prática.*** Leanpub eBook.

SILVEIRA, P., SILVEIRA, G., LOPES, S., MOREIRA, G., STEPPAT, N., & KUNG, F. (2012). ***Introdução à arquitetura e design de software.*** Rio de Janeiro: Elsevier.

TANENBAUM, A. S., & WETHERRALL, D. (2011). ***Redes de computadores.*** São Paulo: Pearson Prentice Hall.