Ministério da Educação – MEC
Secretaria de Educação Superior – SESu
Programa de Educação Tutorial – PET

Universidade Federal Fluminense – UFF
Escola de Engenharia – TCE
Grupo PET do Curso de Eng. de Telecomunicações – PET-Tele

Tutoriais PET-Tele

Estudos e experimentos relativos ao uso de APIs Web para o desenvolvimento Web

(Versão: A2021M07D29)

Autor: Gabriel Bueno dos Santos Doria Oliveira
Lucca Sabbatini Reid Rodrigues
Tutor: Alexandre Santos de la Vega

Niterói – RJ
Julho / 2021

Sumário

Capítulo 1

Introdução

O Programa de Educação Tutorial (PET) [Pro], do Ministério da Educação (MEC), exige que os grupos PET desenvolvam atividades que contemplem, de forma indissociável, itens de Pesquisa, de Ensino e de Extensão. Além disso, os grupos devem estimular uma evolução positiva dos seus integrantes, dos demais alunos do seu curso de graduação, do próprio curso e da sua instituição.

Nesse sentido, o PET-Tele [Gru] procura desenvolver atividades e/ou atender a demandas que cumpram tais exigências.

A seguir, são apresentados as motivações, o objetivo e os resultados esperados, para o trabalho em questão.

1.1 Motivações

O grupo possui experiência em temas relacionados ao desenvolvimento de aplicações Web. Além disso, é de interesse comum o avanço contínuo nos tópicos relacionados a esse tema. Sendo assim, a principal motivação para o material presente neste documento surgiu do interesse em estudar e desenvolver projetos relacionados ao desenvolvimento de uma Web API sem fins lucrativos.

O grupo também possui experiência na realização de cursos relacionados aos diversos temas voltados à computação. Dessa forma, o grupo também tem, como motivação, o desenvolvimento de conteúdo educacional prático sobre o desenvolvimento de aplicações Web com base na implementação de uma API Web, bem com a sua disponibilização para toda a comunidade, de forma gratuita. Este documento pode servir como base teórica para a realização de tal curso.

1.2 Objetivo

O projeto em questão visa esclarecer o leitor a respeito da implementação e desenvolvimento de uma Web API, com base nos conceitos sólidos sobre arquitetura e engenharia de *software*, padrões de projetos e nos impactos ocasionados pela não estruturação de uma Web API.

1.3 Resultados esperados

Ao final da leitura deste documento, é esperado que o leitor tenha uma introdução sobre o desenvolvimento de aplicações Web com a utilização de uma Web API e algumas fontes de conteúdo para que seja possível maior aprofundamento nos assuntos aqui abordados.

Capítulo 2

O desenvolvimento de aplicações Web sem a implementação de uma API Web

É comum encontrar em livros, tutoriais e em cursos *on-line*, o "passo a passo" para o desenvolvimento de algum material didático, projeto ou conceito, com uma abordagem que expõe somente o "como fazer", sem mostrar o porquê de tal projeto ser desenvolvido de determinada maneira. Nesse sentido, é possível que o consumidor desse conteúdo não entenda ao certo o que o levou à utilização de uma determinada tecnologia, conceito ou arquitetura de *software*.

Sendo assim, antes de entrar em profundidade a respeito dos conceitos teóricos e práticos sobre APIs Web, o presente capítulo elucidará, através da apresentação dos impasses no cenário do desenvolvimento Web sem a implementação de uma API Web, os pontos pelos quais aplicações modernas implementam APIs em suas arquiteturas.

2.1 Um projeto único

Quando um projeto não contempla a implementação de uma API Web em sua própria arquitetura, é possível notar um alto grau de acoplamento entre o código reponsável pelo lado do servidor (*back-end*) e o código responsável pelo lado do cliente (*front-end*).

Essa abordagem, chamada de "abordagem tradicional" em alguns materiais, era conveniente há alguns anos, quando o poder de processamento no lado do cliente era consideravelmente inferior quando comparado com o cenário atual. Dessa forma, além de ser responsável pelas suas tarefas comuns, o servidor ficava também responsável por formatar e manipular dados de responsabilidade do cliente. Por exemplo, uma página simples em HTML (*Hyper Text Markup Language*). Nesse cenário, o lado do cliente já recebia essa página com a formatação e os dados prontos.

Com o desenvolvimento das áreas de Arquitetura e Engenharia de *Software* nos últimos anos, foi possível enxergar o quanto esse grau de acoplamento pode prejudicar a disponibilidade de um *software* em ambiente de produção. Com os códigos do *back-end* e *front-end* em um repositório único, tornam-se recorrentes as situações em que todo um sistema é retirado do ar por conta de um erro em somente uma dessas partes. É claro que uma parte depende da outra, mas isso não significa que um erro em uma das partes deva ser responsável pela indisponibilidade da outra, e vice-versa. Nesse sentido, torna-se importante a separação dessas duas partes e de suas respectivas responsabilidades, de modo que a comunicação entre elas não seja perdida e a fim de evitar a indisponibilidade total de um sistema.

Além disso, outro ponto importante a se destacar é a dificuldade no reaproveitamento de serviços. Por exemplo, se uma aplicação Web possui um módulo responsável por envio de men-

sagens de *e-mail* em massa, acoplado aos módulos responsáveis pelas interfaces gráficas dessa aplicação, não é possível reaproveitar esse módulo para reproduzir o mesmo comportamento e funcionalidade em uma interface *mobile*. Assim, esse módulo precisa ser replicado em um outro projeto. Nesse caso, no projeto da aplicação *mobile*, o que gera um retrabalho.

2.2 Acoplamento

Como dito anteriormente, o acoplamento é um dos impasses presentes no desenvolvimento de aplicações Web e *mobile*. E um dos pontos mais importantes aqui é que, dentro do tema tratado neste capítulo, o acoplamento é o que desencadeia todos os outros impasses a serem discutidos, com destaques para:

- Sobrecarga.
- Baixa disponibilidade.
- Manutenibilidade comprometida.

No livro *Clean Architecture* (Arquitetura Limpa) [Mar17], de Robert Cecil Martin, mais conhecido como "Uncle Bob", são abordados alguns princípios de arquitetura de *software* que, juntos, compõem o acrônimo S.O.L.I.D.:

- S: *Single Responsibility Principle* (Princípio de Responsabilidade Única).
- O: *Open/Closed Principle* (Princípio do Aberto/Fechado).
- L: *Liskov Substitution Principle* (Princípio de Substituição de Liskov).
- I: *Interface Segregation Principle* (Princípio da Segregação de Interface).
- D: *Dependency Inversion Principle* (Princípio da Inversão de Dependências).

O Princípio de Responsabilidade Única, que prevê que um determinado módulo seja responsável apenas por um "personagem" e por seus respectivos motivos de mudança/refatoração, retrata muito bem a importância de manter o lado do servidor e o lado do cliente desacoplados. Esse princípio traz uma importante base de entendimento sobre a importância de separar as responsabilidades e desacoplar serviços dentro de uma aplicação Web, e é tomado como guia ao desenvolver uma API eficiente.

2.3 Sobrecarga

Como dito anteriormente, o acoplamento desencadeia diversos outros impasses. Entre eles, a sobrecarga.

Em uma arquitetura de microsserviços, um sistema é modularizado a fim de gerar serviços independentes, o que permite separar as responsabilidades de cada módulo ou componente. Os módulos e componentes comunicam-se entre si através de APIs. Esse cenário soluciona bem o impasse em questão.

Com a modularização de uma aplicação Web, a começar pelo desmembramento entre o lado do servidor e o lado do cliente, é possível evitar tanto a **sobrecarga de responsabilidades**, quanto a **sobrecarga de recursos**.

No que concerne ao código, a sobrecarga de responsabilidades é um ponto negativo na arquitetura de um *software*. Quando uma classe possui muitas responsabilidades, uma simples alteração/refatoração ou inclusão de um método pode causar problemas em cadeia, que não eram previstos nos cadernos de testes do *software*. Esse excesso de tarefas em uma determinada classe compromete a manutenibilidade do sistema, já que dificulta o rastreamento de exceções e também a implementação de novas funcionalidades.

Quanto à sobrecarga de recursos, o cenário está ligado ao ambiente de produção. Um dos desafios no desenvolvimento de sistemas Web é a avaliação dos recursos necessários para colocar uma aplicação em funcionamento. Nesse momento, questões como o **gerenciamento de memória e processamento do servidor de hospedagem**, a **computação em nuvem** e os **ataques DoS (*Denial of Service* ou ataque de Negação de Serviços) ou DDoS (*Distributed Denial of Service* ou ataque Distribuído de Negação de Serviços)**, são discutidas pela equipe de desenvolvimento do *software*, com o objetivo de alcançar a maior disponibilidade e, em casos de necessidades, o menor *downtime* possível.

A sobrecarga de recursos em um servidor de hospedagem afeta diretamente a disponibilidade de um sistema. No caso do desenvolvimento de aplicações Web sem a implementação de uma API Web, o *back-end* e o *front-end* são comumente hospedados no mesmo servidor, com a utilização dos mesmos recursos. Nesse caso, dependendo da dimensão do *software*, os recursos do servidor de hospedagem podem ser sobrecarregados e, consequentemente, o sistema pode ficar indisponível.

A Figura 2.1 demonstra graficamente os impasses abordados.

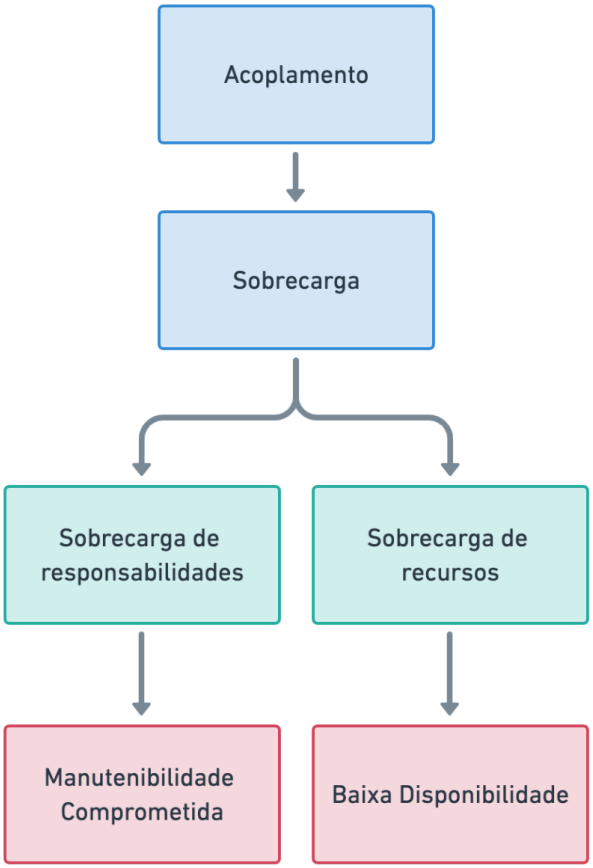

Figura 2.1: Diagrama sobre acoplamento e sobrecarga.

2.4 Por que implementar uma Web API?

Com o objetivo de manter encapsuladas e desacopladas as regras de negócios e os casos de uso de uma aplicação em relação às interfaces gráficas que permitirão a interação dos usuários com tais casos de uso, bem como a padronização e organização dos dados a serem manipulados, a implementação de uma API Web na arquitetura do sistema torna-se extremamente importante, visto que, nesse sentido, é essencial mitigar os problemas relacionados ao acoplamento, sobrecarga, manutenibilidade e disponibilidade.

A organização na arquitetura de um projeto não só está relacionada a fatores técnicos, mas também mercadológicos.

Quando falamos de indisponibilidade, levamos em conta o tempo em que uma aplicação fica fora do ar. Um sistema fora do ar em ambiente de produção significa que a empresa responsável pelo sistema precisa realocar verba para solucionar o problema ou deixar de ganhar dinheiro, caso o *software* seja diretamente voltado ao o consumidor final.

Quando falamos de manutenibilidade, levamos em conta a facilidade de implementar novas funcionalidades ou refatorar funcionalidades já existentes em um sistema com uma arquitetura bem definida. Um projeto com elevada dificuldade na manutenção também gera gastos exponenciais para a empresa que mantém a aplicação. O gráfico da Figura 2.2, por exemplo, mostra a relação entre a produtividade da equipe de desenvolvimento e a implementação de novas versões, diante de um *software* com manutenibilidade comprometida. Esse é um estudo de caso de uma empresa de grande porte que, segundo Robert Cecil Martin, disponibilizou os dados para análise, mas que prefere manter o anonimato.

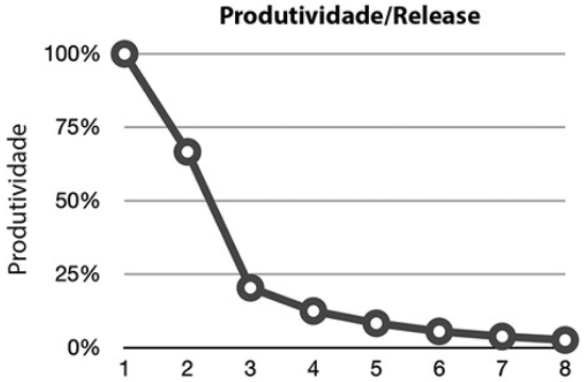

Figura 2.2: Curva da produtividade em relação à implementação de novas versões.

Por sua vez, o gráfico da Figura 2.3, mostra a relação entre o aumento dos gastos da empresa em função do desenvolvimento de novas versões do *software*.

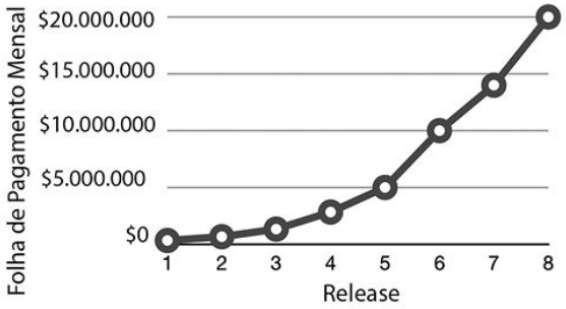

Figura 2.3: Curva da folha de pagamento em relação à implementação de novas versões.

Sendo assim, é possível notar que a dificuldade de manutenção compromete recursos financeiros de um determinado projeto. Com a queda da produtividade e a necessidade de mais pessoas envolvidas no projeto com o intuito de realizar a manutenção necessária, um projeto com um grande potencial pode se tornar financeiramente insustentável.

Capítulo 3

O que é uma API Web?

Uma API (*Application Programing Interface*), como o nome já sugere, é uma interface. Quando é dito "interface", é comum o entendimento desse termo como se fosse uma interface gráfica. Porém, uma API é um conceito totalmente independente de um componente gráfico. Nesse sentido, vale esclarecer que o termo Interface, no nome "Interface de programação de aplicações", faz referência a um protocolo.

Em computação, o termo protocolo é definido como uma convenção que, através de algumas regras e processos, controla e possibilita a conexão, comunicação e transferência de dados entre dois sistemas. Sendo assim, uma API pode ser entendida como um sistema que permite a comunicação entre dois ou mais sistemas, em que essa comunicação é conduzida por um protocolo.

Uma API Web é uma API envolvida em aplicações Web e que utiliza o protocolo HTTP (*Hyper Text Transfer Protocol*) para efetivar a comunicação entre tais sistemas.

Todo e qualquer sistema Web é movido por dados. Através da troca de dados na rede, seja pela interação de um usuário humano ou por uma outra máquina, as aplicações Web ganham vida e passam a resolver diversos problemas. O fato é que tudo se resume à manipulação de dados.

Como estamos falando da Internet, é indiscutível que o volume de dados trafegados é incalculável. Esse fato traz diversos desafios no que diz respeito à organização dos dados e, cada vez mais, soluções são criadas com o intuito de reconhecer e organizar padrões de comunicação entre aplicações na Web, justamente para que seja possível essa integração.

O principal intuito de uma API Web é padronizar e organizar, em uma determinada estrutura, os dados necessários para a comunicação entre interfaces de usuários baseadas nos mesmos casos de uso. Dessa maneira, a partir de uma API Web, é possível a implementação de uma arquitetura que permita o encapsulamento de partes de extrema importância para um sistema, tais como:

- Tratamento de dados.
- Controle de rotas, seus respectivos recursos e requisições.
- Validações.
- Regras de negócio.
- Casos de uso.
- Tratamento de erros.
- Testes unitários e de integração.

Assim, as partes que ditam o pleno funcionamento de um sistema tornam-se totalmente independentes de interfaces gráficas, as quais, por sua vez, utilizam os dados, organizados e padronizados, que são retornados pela API Web. Dessa forma, uma mesma API Web pode alimentar diferentes interfaces gráficas e até mesmo outras APIs Web, como é mostrado na Figura 3.1. Isso permite o funcionamento de diferentes formas de interação com o usuário baseadas nos mesmos casos de uso, sem o retrabalho de implementar diversas soluções diferentes.

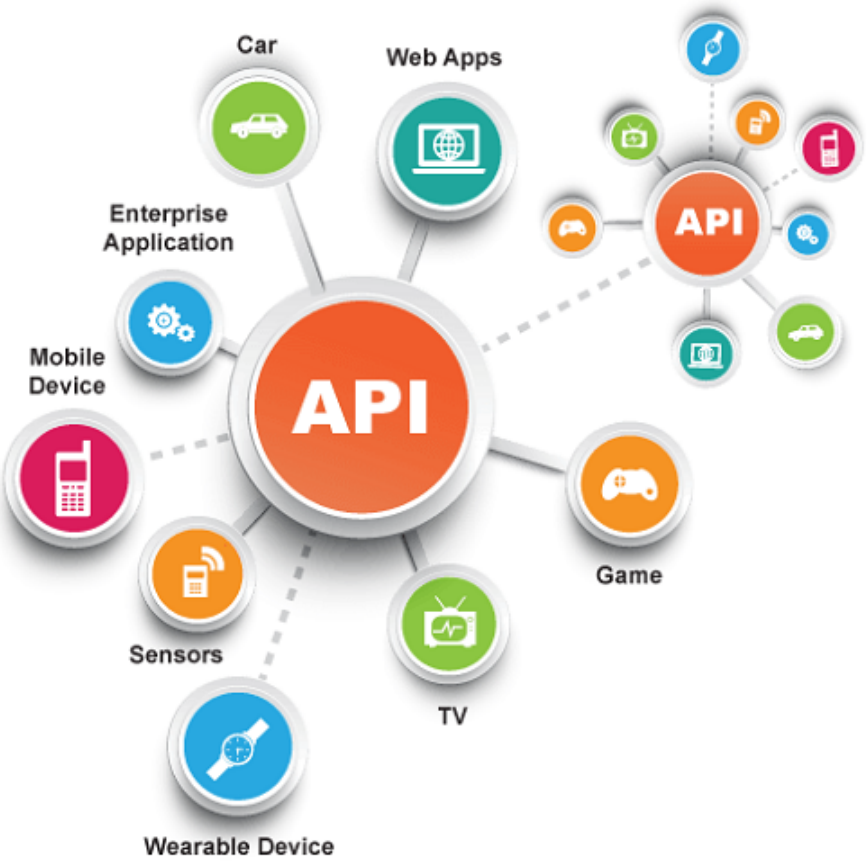

Figura 3.1: Consumo de uma API por diferentes interfaces.

3.1 O fluxo de uma requisição

Uma API Web é uma intermediária entre os dados e as interfaces que os consomem. Dessa forma, a partir de uma requisição, a API Web interage com os dados necessários para retornar uma resposta estruturada à aplicação que a utiliza.

O fluxo de uma requisição HTTP pode iniciar-se tanto manualmente, quando um usuário digita o URL (*Uniform Resource Locator* ou Localizador Uniforme de Recursos) na barra de endereços do navegador, quanto de forma programática, através de outros programas. Quando é digitado, por exemplo, **meudominio.com** na barra de endereços do navegador, ocorre a interpretação desse URL, em que é localizado o endereço IP (*Internet Protocol* ou Protocolo de Rede) associado a esse domínio. Essa interpretação é realizada através de um servidor DNS (*Domain Name System* ou Sistema de Nomes de Domínio).

Em uma consulta de DNS, a primeira parada da requisição é em um resolvedor recursivo de DNS. Esse resolvedor atua como intermediário entre a requisição do cliente e um *nameserver* (servidor de nome) de DNS, direcionando a requisição ao servidor-raiz de DNS, incluindo o nome do domínio, que, no caso de exemplo, é o **meudominio.com**. Em seguida, o servidor-raiz direciona o resolvedor recursivo para um servidor TLD (*Top-Level Domain* ou Domínio de nível superior), com base na extensão (.com, .net, .org, etc.) do domínio recebido. Um servidor de domínio de nível superior mantém informações sobre todos os nomes de domínio que compartilham uma extensão de domínio comum. No caso de exemplo (**meudominio.com**), a extensão de domínio é a **.com**. Nesse caso, o servidor solicitado é o TLD de domínios **.com**, que contém as informações de todos os sites que terminam em **.com**. Em seguida, ao receber a resposta do servidor TLD, o resolvedor recursivo de DNS é redirecionado para um servidor autoritativo, que contém informações específicas do nome de domínio atendido por ele. Por fim, o servidor autoritativo retorna ao resolvedor recursivo o endereço de IP associado ao domínio requisitado, que, por sua vez, retorna ao cliente o IP obtido.

Sendo assim, de forma resumida, temos os seguintes passos:

- O usuário digita o URL na barra de endereços do navegador.
- O navegador envia uma requisição aos *nameservers* associados ao domínio, passando por todo o processo descrito anteriormente.
- Os *nameservers* enviam uma resposta, contendo o endereço de IP associado ao domínio.
- O navegador solicita os dados necessários ao servidor Web, identificado pelo IP recebido do *nameserver* associado ao domínio.
- O servidor Web envia a resposta, que, no caso das APIs Web, é comumente retornada em um formato de representação chamado JSON (*JavaScript Object Notation* ou Notação de Objeto em JavaScript), o qual será brevemente abordado a seguir.

Ao comprar um domínio, é necessário informar os servidores de DNS a serem utilizados. Uma plataforma bem eficiente para realizar isso é a Cloudflare. Nela, é possível gerenciar todos os *nameservers* associados ao domínio adquirido. Assim, ao adquirir o domínio em alguma plataforma, é necessário apontar esse domínio para os servidores de DNS da Cloudflare (evangeline.ns.cloudflare.com e felicity.ns.cloudflare.com). Feito isso, a Cloudflare realizará todo o processo de resolução de DNS que foi descrito nessa seção.

A Figura 3.2 mostra o processo de resolução de DNS.

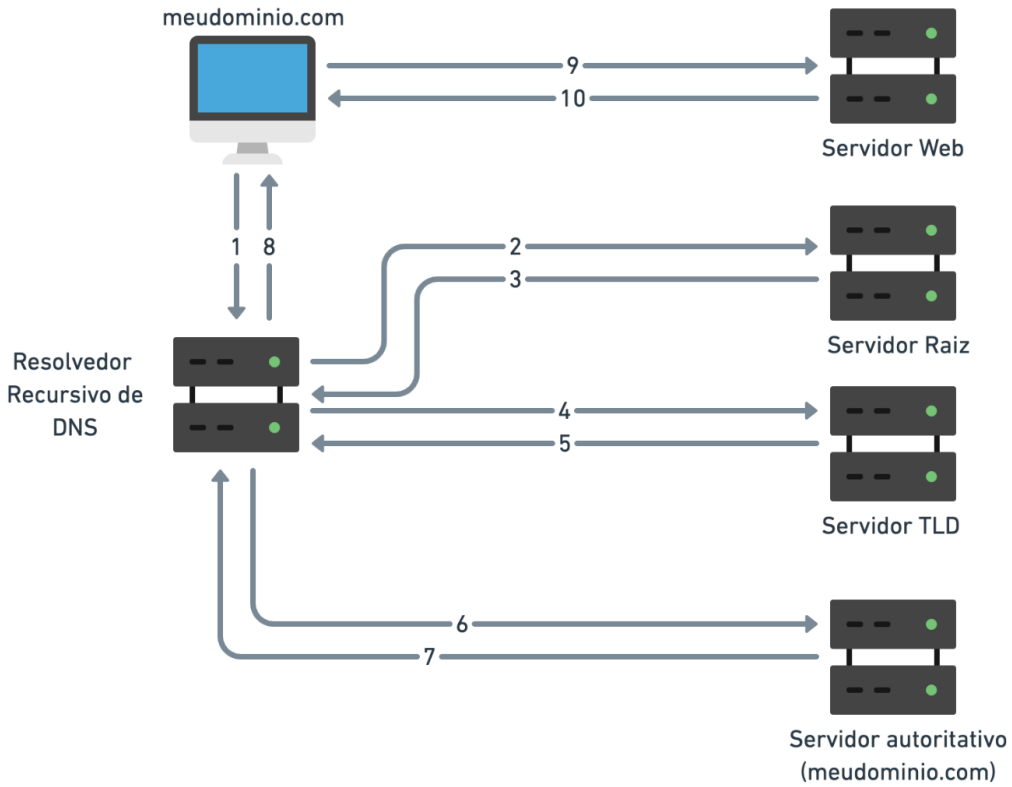

Figura 3.2: Fluxo de resolução de DNS.

3.2 O que são rotas?

Quando a requisição chega ao servidor Web que hospeda a API, através do endereço de IP recebido pelo servidor DNS, ela é redirecionada para a porta que executa o serviço da API Web. Dessa forma, é possível interagir de fato com a API.

Em sua estrutura, uma API Web é definida através de rotas, pelas quais seus clientes tornam-se aptos para solicitar determinados recursos. A título de exemplo, convém analisar o seguinte cenário:

- Um restaurante possui um sistema administrativo, em que é possível incluir novas receitas, editar receitas existentes, excluir receitas e listar receitas.
- Nesse sistema, foi desenvolvida uma API Web, capaz de realizar essas 4 tarefas.
- No painel administrativo, um usuário pretende listar as receitas em sua tela. Sendo assim, ao efetuar essa requisição, a interface gráfica requisita a API Web na rota responsável pelas receitas, que, nesse caso, poderia ser algo similar ao seguinte endereço: **http://exemplorestaurante.com/receitas**.
- Nesse caso, a rota é todo o endereço "**http://exemplorestaurante.com/receitas**" e o trecho "**/receitas**" é denominado recurso da rota.

Além do recurso, uma rota também possui um método em sua sintaxe. Os recursos e métodos, na sintaxe de uma rota, proporcionam organização e semântica nas requisições, permitindo melhores tratamentos de erros, melhores documentações, validações e entendimento a respeito do funcionamento de uma determinada funcionalidade de uma API Web.

Os métodos são estabelecidos pelo protocolo HTTP (*Hyper Text Transfer Protocol* ou Protocolo de Transferência de Hipertexto) e também são conhecidos como verbos HTTP, já que definem ações a serem realizadas numa rota.

Dentre os métodos HTTP, os mais utilizados são:

- GET: Utilizado para solicitar informações de um determinado recurso da API. Requisições utilizando o método GET retornam apenas dados.
- POST: Utilizado para submeter dados a um determinado recurso na API. Requisições que utilizam o método POST possuem um corpo, por onde os dados são submetidos. O corpo da requisição que utiliza esse método normalmente é no formato JSON. Esse método é utilizado, por exemplo, em rotas de cadastro e *login*, onde é necessário submeter o *e-mail* e a senha como campos a serem validados.
- PUT: Utilizado para solicitar uma edição em todas as propriedades de um determinado recurso. Assim como o POST, também possui corpo da requisição, pelo qual os dados a serem editados são submetidos.
- PATCH: Utilizado para solicitar uma edição parcial em um determinado recurso, em que nem todas as propriedades precisam ser modificadas. Também possui corpo da requisição, pelo qual os dados a serem editados são submetidos.
- DELETE: Utilizado para remover dados de um determinado recurso.

3.3 O formato de representação JSON

Uma requisição só termina quando os dados manipulados pela API Web são retornados ao cliente que a requisitou. Como dito anteriormente, as respostas das APIs Web são comumente retornadas em um formato chamado JSON (*JavaScript Object Notation* ou Notação de Objeto em JavaScript).

O JSON tem sua estrutura baseada no formato atributo-valor e se destaca pela facilidade de manipulação do dados. Ele é organizado por objetos e estruturas de dados mais simples, como, por exemplo, vetores. O formato JSON possui alguns tipos básicos, tais como:

- Number: um número que pode ser inteiro ou fracionário, tanto positivo quanto negativo.
- String: uma cadeia de zero ou mais caracteres. São delimitados por aspas duplas (”str”).
- Boolean: um dos valores *true* e *false*, correspondendo aos valores lógicos verdadeiro e falso, respectivamente.
- Array: uma lista ordenada de zero ou mais valores, cada um podendo ser de qualquer tipo. Arrays são delimitados por colchetes ([ ]), dentro dos quais ficam os valores, também conhecidos como elementos, separados por vírgulas. O primeiro elemento é o de índice 0.
- Object: uma coleção não ordenada de pares atributo-valor, onde os atributos (ou nomes ou chaves) são *strings*. Objects são delimitados por chaves ({ }) e usam vírgulas para separar cada par, enquanto que, no par, o atributo e o valor são separados por dois pontos (:).
- null: valor vazio ou nulo. É importante mencionar que **null** tem um significado diferente de 0 (zero).

Arquivos no formato JSON possuem a extensão **.json**. Como exemplo de sua estruturação, convém analisar a Figura 3.3, que contém uma lista de alunos e suas respectivas notas.

```
{"Alunos":[
     { "nome": "Edson Sales Arantes", "notas": [ 8, 9, 5 ]  },
     { "nome": "Luiz Livelli ", "notas": [ 8, 10, 7 ] },
     { "nome": "Caique Caicedo De Plata", "notas": [ 10, 10, 9 ] }
]}
```

Figura 3.3: Exemplo de arquivo JSON, contendo uma lista de alunos e suas respectivas notas.

3.4 Os tipos de API

As APIs podem ser classificadas em alguns tipos diferentes, os quais são brevemente descritos a seguir.

3.4.1 APIs públicas ou abertas

As APIs abertas, também conhecidas como APIs públicas, estão disponíveis de forma gratuita para os usuários.

Existem APIs totalmente públicas, em que não é necessário realizar cadastro ou qualquer tipo de autenticação, como, por exemplo, API de consulta de CEP (Código de Endereçamento Postal). Por outro lado, existem APIs públicas que exigem algum tipo de cadastro, chave de autenticação ou algo similar. Nesse sentido, podemos citar a API de *login* do Facebook. É uma API gratuita, mas que exige um cadastro no painel *Facebook Developers*.

A maioria das APIs públicas possuem uma documentação de fácil acesso, onde é possível entender a forma correta da sua utilização.

3.4.2 APIs privadas ou internas

As APIs internas, também conhecidas como APIs privadas, são ocultadas de usuários externos e expostas apenas para assinantes e/ou membros do detentor ou detentora da API, seja uma pessoa física ou uma organização empresarial. Esse tipo de API possui acesso restrito para os usuários que possuem autorização para utilizá-la, mas costuma disponibilizar um ambiente de testes gratuito com um número limitado de requisições. São comumente utilizadas em projetos não livres e também possuem um módulo de autenticação responsável por validar se um determinado usuário possui permissão de utilização.

3.4.3 APIs de parceiros

As APIs de parceiros são APIs expostas por parceiros de negócios estratégicos. Elas não estão disponíveis publicamente e precisam de direitos específicos para acessá-las. As APIs de parceiros são as intermediárias entre as integrações implementadas entre duas ou mais empresas e utilizadas para realizar a comunicação além dos limites da empresa.

O objetivo das APIs de parceiros é agregar novas funcionalidades a soluções já existentes.

3.4.4 APIs compostas

As APIs compostas combinam vários dados ou APIs de serviços. Podem ser interpretadas como um agregado de funcionalidades modularizadas.

As APIs compostas são úteis, por exemplo, em um padrão de arquitetura de microsserviços, em que é necessária a utilização de informações de vários serviços para executar uma única tarefa.

Referências Bibliográficas

[Bet] BetterProgramming. *Understand the Flow of a HTTP Request*. Disponível em: "https://betterprogramming.pub/understand-the-flow-of-a-http-request-1a268ec193f0". Acesso em: 29/07/2021.

[Clo] Cloudflare. *DNS server types*. Disponível em: "https://www.cloudflare.com/pt-br/learning/dns/dns-server-types". Acesso em: 29/07/2021.

[Doc] MDN Web Docs. Métodos de requisição HTTP. Disponível em: "https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Methods". Acesso em: 29/07/2021.

[Gru] Grupo PET-Tele. *Webpage* do Grupo PET-Tele. Disponível em: "http://www.telecom.uff.br/pet". Acesso em: 29/07/2021.

[KIN] KINSTA. *What Is a Nameserver? Why Are Nameservers Important?* Disponível em: "https://kinsta.com/knowledgebase/what-is-a-nameserver". Acesso em: 29/07/2021.

[Lin] LinkApi. Quais são os tipos de APIs? Disponível em: "https://www.linkapi.solutions/blog/quais-sao-os-tipos-de-apis". Acesso em: 29/07/2021.

[Mar17] Robert C. Martin. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Prentice Hall Press, One Lake Street Upper Saddle River, NJ, United States, 2017.

[Pro] Programa de Educação Tutorial (PET). *Webpage* do Programa PET. Disponível em: "http://portal.mec.gov.br/index.php?option=com_conent&view=article&id=12223&ativo=481&Itemid=480". Acesso em: 29/07/2021.