



FRAMEWORKS BACK END

Leandro da Conceição Cardoso

FRAMEWORKS BACK END

1^a edição

São Paulo
Platos Soluções Educacionais S.A
2021

© 2021 por Platos Soluções Educacionais S.A.

Todos os direitos reservados. Nenhuma parte desta publicação poderá ser reproduzida ou transmitida de qualquer modo ou por qualquer outro meio, eletrônico ou mecânico, incluindo fotocópia, gravação ou qualquer outro tipo de sistema de armazenamento e transmissão de informação, sem prévia autorização, por escrito, da Platos Soluções Educacionais S.A.

Diretor Presidente Platos Soluções Educacionais S.A

Paulo de Tarso Pires de Moraes

Conselho Acadêmico

Carlos Roberto Pagani Junior
 Camila Braga de Oliveira Higa
 Camila Turchetti Bacan Gabiatti
 Giani Vendramel de Oliveira
 Gislaine Denisale Ferreira
 Henrique Salustiano Silva
 Mariana Gerardi Mello
 Nirse Ruscheinsky Breternitz
 Priscila Pereira Silva
 Tayra Carolina Nascimento Aleixo

Coordenador

Henrique Salustiano Silva

Revisor

Arthur Gonçalves Ferreira

Editorial

Alessandra Cristina Fahl
 Beatriz Meloni Montefusco
 Carolina Yaly
 Mariana de Campos Barroso
 Paola Andressa Machado Leal

Dados Internacionais de Catalogação na Publicação (CIP)

Cardoso, Leandro da Conceição
 C268f Frameworks back end / Leandro da Conceição
 Cardoso. – São Paulo: Platos Soluções Educacionais S.A.,
 2021.
 44 p.

ISBN 978-65-89965-87-9

1. Node.js. 2. Larávél. 3. Spring Boot. I. Título.

CDD 005

Evelyn Moraes – CRB-8 SP-010289/0

2021
 Platos Soluções Educacionais S.A
 Alameda Santos, nº 960 – Cerqueira César
 CEP: 01418-002— São Paulo — SP
 Homepage: <https://www.platosedu.com.br/>

FRAMEWORKS BACK END

SUMÁRIO

Introdução à <i>Frameworks Back End</i>	05
Utilização do <i>Framework Express</i>	20
<i>Framework Back End Laravel</i>	37
<i>Framework Back End Spring Boot</i>	53

Introdução à *Frameworks Back End*

Autoria: Leandro C. Cardoso

Leitura crítica: Arthur Gonçalves Ferreira



Objetivos

- Conhecer *frameworks back end* apropriados para o desenvolvimento de projeto.
- Saber selecionar *frameworks back end* conforme as características de cada projeto.
- Estudar os *frameworks back end* para criação de diversas aplicações consumidoras.



1. Conceituando *Frameworks Back End*

Para os profissionais de desenvolvimento *Web full stack* é importante o conhecimento sobre os *frameworks*, que são conjuntos de códigos de uma linguagem de programação específica que auxiliam o desenvolvimento de projetos como de web, softwares, games, aplicativos, entre outros. As vantagens de utilização dos *frameworks* são o baixo nível de programação e o maior controle da programação do projeto que será desenvolvido. Existem diversos *frameworks* em várias linguagens, como Java, que funciona de maneira unificada em diferentes plataformas (AMBLER; CLOUD; HAWKES, 2015). Além disso, temos a plataforma *mobile* que pode ser visualizada nos navegadores utilizando a tecnologia HTML5 e executados em diversos sistemas operacionais como:

- iOS.
- Android.
- Windows.
- Mac.
- Linux.

A utilização de *frameworks* resulta na redução de custos e no aumento da produtividade. Nesse sentido, além do conhecimento das plataformas, os profissionais de desenvolvimento *Web full stack* devem levar em consideração outras características. Smutny (2012, p. 654, tradução nossa) as classifica como aspectos nativos e híbridos:

Nativos: são aplicativos móveis rápidos e confiáveis, mas estão ligados a uma plataforma móvel. Isso significa que o desenvolvedor deve duplicá-los usando a linguagem de programação adequada, caso queira disponibilizá-los em outra plataforma móvel. Por exemplo, caso um aplicativo

desenvolvido nativamente para Android seja disponibilizado para iOS, deverá ser desenvolvido utilizando a linguagem Objective-C.

Híbridos: são aplicativos desenvolvidos com a utilização de frameworks que se comprometem em garantir a compatibilidade entre plataformas de dispositivos móveis diferentes, permitindo o acesso ao hardware (câmera, GPS e NFC). Por exemplo, um aplicativo móvel desenvolvido com o PhoneGap para iOS pode ser reutilizado para a plataforma Android.

Os *frameworks* trazem outros recursos, como grupos de bibliotecas, que possibilitam aos criadores executar alterações de operações de grande volume com uma maior agilidade e em um curto espaço de tempo. Eles facilitam, também, a reescrita de códigos: o desenvolvedor se preocupa apenas com a validação de campos e com a conexão com bancos de dados.

Os *frameworks* disponíveis no mercado possuem documentação para facilitar a sua codificação, tornar mais fácil a manipulação e executar as alterações necessárias para adaptar ao projeto de desenvolvimento *Web full stack*. Os profissionais com conhecimentos avançados de desenvolvimento de *back end*, experiência do usuário e arquitetura da informação são qualificados para construir o próprio *framework*.

1.1 Selecionando *Frameworks Back End*

Há vários *frameworks* prontos, mas, caso seja constado que nenhum *framework* atenda ao objetivo do projeto de desenvolvimento *Web full stack* ou quando for necessário o controle total sobre o código e os *frameworks* disponíveis não proporcionem esse recurso; ou, ainda, caso seja preciso que algo efetivamente bem menos complicado ou com quantidade menor de objetos na programação do que os *frameworks* encontrados possuem, indica-se o desenvolvimento de um *framework* personalizado para atender as necessidades.

Para desenvolver a codificação de *frameworks*, a organização é importante, lembrando que o objetivo principal de um *framework* é de que possa ser utilizado em outros projetos. Ou seja, ele precisa ser de fácil entendimento para outros profissionais utilizarem e modificarem, dessa forma, os componentes e as classes da programação devem ter a melhor organização possível e padronização. Para entender melhor, Pressman e Maxim (2016, p. 45) definem classe como:

[...] uma descrição que abstrai um conjunto de funções com características similares, uma descrição das propriedades ou estados possíveis de um conjunto de funções, bem como os comportamentos ou ações aplicáveis a estas mesmas funções.

Na programação de um *framework*, a padronização pode ser apresentada de várias maneiras, por exemplo, a padronização de classe de botão pode ocorrer de diferentes modos: *.button*, *.botao*, *.btn* e *.bt*. Depois de definido o padrão é importante não esquecer de manter a padronização nas suas variações. Observe o exemplo no qual foi padronizado caracteres *bt* para definir um botão e, em seguida, suas variações: *.bt-salvar*, *.bt-enviar*, *bt-pesquisar*, *bt-cancelar* etc. Com essa organização e padronização, a redundância no prefixo é criada de maneira natural, conforme o exemplo: `Salvar`. Com base nesses fundamentos, a codificação do *framework* torna mais fluida, acompanhando a mesma linha para as outras diferenciações que vierem a aparecer e são aspectos importantes que facilitam no momento da programação de uma interatividade, principalmente quando não existir codificações diferenciais específicas do componente de botão ou se é uma variação genérica, como apresentado a seguir:

```
/* classe full específica */  
<a href="#" title="salvar" class="bt bt-salvar bt-full">Salvar</a>  
/* classe full generica */
```

[Salvar](# "salvar")

Nesse exemplo, o que diferencia a classe específica da classe genérica é que na específica foi necessária a indicação do botão identificado em destaque em negrito, como **bt**; enquanto na genérica é uma alteração do *framework*, no qual não foi necessário identificar esse código. Para facilitar a organização é recomendável contextualizar para outras classes os seus devidos prefixos, conforme o quadro a seguir.

Quadro 1 – Função e indicações de classes para codificação de *framework*

Função	Indicação de classes
Botão	= .button, .botao, .btn, .bt
Tabela	= .table, .tabela, .tbl, .tb
Listas	= .list, .lista, .group
Widgets	= .widgets, .wid
Títulos	= .title, .tit, .tt, .header, .h

Fonte: elaborado pelo autor.

Para o desenvolvimento de um *framework*, é importante ressaltar a organização das nomenclaturas e o seu emprego correto no código, não se pode esquecer que o nome utilizado no componente deve vir como prefixo das suas diversificações. Se acontecer de uma das variações for usada para outros componentes, ela pode ser usada sem prefixo como “clear, full, right, left, error”. É importante mapear a nomenclatura e as possibilidades com a equipe de desenvolvimento e não sozinho, a fim de minimizar as possibilidades de erros.

1.2 Documentação da codificação de *framework*

Recomenda-se que toda codificação e diretrizes de um *framework* estejam disponíveis em um único local, pois, uma das definições de *framework* é a de que ele possa ser reutilizado. No desenvolvimento já deve ser pensado que os elementos possam ser reutilizados, alinhados

e harmonizados em outros locais, e não somente no local que foi pré-desenhado, apenas para uma função específica. Logo, é importante lembrar que se os códigos foram criados para resolverem somente a solução de um projeto, assim, não pode ser considerado um *framework* e sim códigos personalizados para aquele sistema específico.

Para disponibilizar os códigos, caso queira compartilhar para o uso livre, é aconselhável criar uma página web para apresentar todos os elementos padronizados e a documentação necessária para que possa ser reutilizável. Nesta página, indica-se ter uma apresentação contendo a lista completa dos componentes padronizados, pois, é preciso identificar os componentes novos que foram inseridos, as melhorias ou as soluções de *bugs* e, consequentemente, mantê-la atualizada.

Caso tenha um *framework*, recomenda-se o uso da versão para HTML5, projetos responsivos, disponibilizar em uma página dedicada somente para essa opção, por exemplo, essa página pode ter um nome: padrao-mobile.html ou algo parecido com essa nomenclatura. Esse procedimento mostra que o *framework* foi construído de forma profissional, aumentando sua credibilidade. Nessa página também pode ser apresentado exemplos de adaptação da largura do dispositivo pretendido pelo profissional que for utilizar o *framework*.

A codificação de um *framework* pode ser construída da maneira totalmente personalizada, utilizando nomes no idioma em português, abreviações e padronização que o desenvolvedor achar interessante. Ainda, é indicada a utilização do idioma em inglês, além de facilitar a lógica da programação, o *framework* pode ser utilizado no mundo inteiro, o que também irá facilitar a sua compreensão. Mas além das nomenclaturas existem alguns itens comuns que praticamente a maioria dos frameworks disponibiliza, sendo importante que, no desenvolvimento, tenham esses componentes: *grid*, tipografia, botões e formulários.

É evidente que cada projeto tem as suas particularidades e a definição é da equipe que desenvolve, que no caso de projetos responsivos precisam ser flexíveis, sobre o componente *grid*. Além disso, deve-se levar em consideração se o layout terá *float* (será flutuante), se as colunas terão um tamanho fixo em pixels ou serão flexíveis; se serão ajustáveis conforme do tamanho da tela que é exibido ou as colunas serão responsivas de acordo com o dispositivo que será visualizado. Normalmente, essas definições são aplicadas no atributo CSS *box-sizing*, que altera o *display* do *box-model*, passando a considerar o *padding* e *border* na hora de aparecer na largura ou altura final, o *box-model* convencional não os considera na largura e altura, somando no resultado os valores, por exemplo, 600px de largura acaba se tornando $600\text{px} + 2\text{px de borda} + 10\text{px de padding} = 612\text{px total}$.

No componente de tipografia, o profissional de desenvolvimento *Web full stack* precisa pensar em todos os elementos de textos do *framework*, sendo esses os links, os títulos, as listas, os parágrafos etc. Nesse contexto, é indicado efetuar simulações com as combinações possíveis, por exemplo, título grande, parágrafo de duas linhas e outras combinações com título pequeno, parágrafo com três linhas etc., para tentar prever como serão resolvidas essas situações.

A padronização dos grupos de botões também deve ser considerada, normalmente, um *framework* dispõe de dois tipos de botões: o genérico e o primário. O botão primário pode ser identificado na programação, como *.bt-primary*, é o que executa a função principal de uma página, o botão da ação final quando é encerrado um procedimento. Já o botão genérico pode ser utilizado nas funções diversas que não seja da finalização, como cancelar, salvar, avançar, upload etc.

Em relação aos padrões, geralmente, se apresentam os atributos como: *.small* para versão menores e, quando é necessário ocupar uma largura, o atributo *.full*. A utilização de ícones, desenvolver padronização como .

bt-icon e ter os valores como *.bt-disabled*, (desabilitado) *.bt-loading*, (carregando) também são importantes no grupo de botões.

Nos formulários é preciso pensar em como o elemento que compõe os dados podem ser preenchidos, a maneira que podem ser dispostas com a lista de *checkbox*, *radio*, *select*, sendo assim, é importante realizar testes e verificar como eles podem se comportar em navegadores diferentes, no caso, nos projetos desenvolvidos em HTML5. O atributo *box-sizing*, utilizado no *grid*, pode servir para controlar a largura do formulário de forma segura, com o objetivo de que os elementos não fiquem muito longe dos dados inseridos, não ocupando uma linha inteira. Nos formulários, também, deve-se considerar as mensagens de erro e as informações positivas, como as que informam que os dados foram enviados com sucesso, precisam ser padronizados.

1.3 Framework para aplicativos

Com objetivo de manter o padrão, algumas empresas disponibilizam bibliotecas e *frameworks* para desktop e *mobile*, para as grandes empresas desenvolvedoras é interessante que outras empresas sigam o seu padrão, pois, os usuários assimilam mais fácil os aplicativos disponíveis nos seus sistemas. O *Twitter Bootstrap*, *Google Material Library* e *Microsoft UI Fabric* são exemplos de *frameworks* e bibliotecas que estão com a documentação disponível a para serem utilizados nos aplicativos, mas também existem outras bibliotecas. A *React UWP (Fluent Design)*, *Materialize (Material Design)*, *Material UI (Material Design)*, *Ratchet (Apple Guideliness iOS)*, e *Semantic UI* (inspirado na Apple e *Material Design*) são exemplos de outros *frameworks* e bibliotecas disponíveis. Existem inúmeras vantagens de utilizar um sistema, biblioteca pronta ou *frameworks*, como o de facilitar o desenvolvimento e manter a satisfação do usuário por meio de um visual que ele já está acostumado.

Os sistemas de design facilitam a padronização dos projetos, pois é a etapa que é elaborada a documentação das descrições, das informações e de todos os elementos necessário para o desenvolvimento de um projeto, seja coletando dados com informações on-line ou físicas. Com todas as informações é possível descrever as padronizações de todos os elementos de um projeto, ajudando a viabilizar a aplicação desses padrões em diferentes projetos de design, seja para o uso gráfico ou digital, e manter a identidade visual nas diferentes aplicações. Por exemplo, desde o cartão de visitas de uma empresa, passando por todo o material gráfico de papelaria como timbre, envelopes; os materiais promocionais como canetas, chaveiros, camisetas ou brindes personalizados podem ter a mesma identidade visual; mas também, com as devidas adaptações mantenham a identidade visual dos projetos de design digital que a empresa dispõe, como site ou aplicativos; o sistema de design facilita padronizar todos esses elementos.

Na prática, os elementos que são padronizados são bem similares que os profissionais de design gráfico estão acostumados a trabalhar no manual de identidade visual, são padronizados os elementos como grades, formas, tipografias, iconografia, cores, símbolos, etc. Um exemplo prático de padrão em projeto de design, são as interfaces dos softwares pertencentes no pacote Office da Microsoft, são todos softwares da mesma linha e que seguem padronização da interface, embora tenham funções específicas e, em alguns casos, totalmente distintas. Para que aconteça essa uniformidade de padrão da identidade visual é importante que seja desenvolvido sistema de design com objetivo de manter as referências para todos os softwares do pacote já existentes e, também, podem ser aplicados nos novos softwares que podem ser lançados. Além de facilitar o trabalho dos desenvolvedores dos softwares da área da arquitetura da informação e do design, o sistema de design ajuda nas questões de usabilidade para os usuários, facilitando a identificar recursos padrões do mesmo pacote de softwares, como nos softwares do pacote Office praticamente todos têm

a função de salvar e abrir e eles seguem um padrão de formatação dos ícones, cores, localização, ou seja, mantém a identidade visual.

As lojas de aplicativos disponibilizam os seus sistemas de design que são compartilhados para os desenvolvedores, com objetivo que sejam desenvolvidos aplicativos que mantenham a harmonia e as experiências coesas e confortáveis na navegação nos seus sistemas operacionais. Desse modo, em alguns casos, seguir as diretrizes desses sistemas operacionais é uma obrigatoriedade para que o aplicativo possa ser disponível na sua loja, principalmente os destinados para os smartphones da Apple, com sistema operacional iOS. Outras padronizações são consideradas apenas recomendação, que no caso se não for seguido não influencia na aprovação do aplicativo na loja virtual, de certa forma, o desenvolvimento *Web full stack* acaba recebendo uma influência conforme o sistema operacional que irá rodar, algumas empresas têm os seus próprios sistemas de design que recebem os seguintes nomes:

- Apple HIG.
- Google Material Design.
- Microsoft Fluent Design.
- IBM Carbon Design System.
- Salesforce Lightning.
- SAP Fiori.
- Canonical Vanilla Design.

Mantendo o sistema de design conforme o sistema operacional é possível identificar as diferenças visuais do aplicativo que é instalado em um smartphone da Apple, com sistema operacional iOS, é diferente

quando ele está instalado no sistema operacional Android da Google. Com objetivo de manter o padrão algumas empresas disponibilizam bibliotecas e *frameworks* para desktop e *mobile*, para as grandes empresas desenvolvedoras é interessante que outras empresas sigam o seu padrão, pois, os usuários assimilam mais fácil os aplicativos disponíveis nos seus sistemas.

Para manter uniformidade existem algumas diretrizes que se tornaram obrigatórias para que um aplicativo possa ser inserido nas lojas, como a Google atualizou suas políticas que todos os ícones dos aplicativos terão formato unificado. A ideia é de que nas telas dos smartphones do seu sistema operacional Android, os ícones estejam unificados que não tenham formatos diferentes, adotaram como padrão a forma conhecida como *squircle*, ou seja, quadrado, mas com cantos arredondados. Caso os aplicativos não tenham esse formato, a Google não permite que sejam efetuados uploads de apps que não se encaixem nesse padrão. O objetivo é que seja criado um sistema de design que seja unificado não somente para o sistema operacional Android, mas também no ChromeOS. A ideia para que os diversos sistemas operacionais da Google mantenham o sistema de identidade, no qual também se inclui o Android TV e Android Auto.

Seguindo à risca as diretrizes, como a padronização das *guidelines*, propostas pelo Google disponíveis por meio do seu sistema de design chamado de Material Design, que tem como objetivo de padronizar não apenas aspectos visuais, mas também de interação, pode ser levantando a hipótese que os aplicativos terem visual semelhante. Por exemplo, principalmente caso for seguido os padrões visuais como de espaçamento e tipografia, no momento de exibição de listas se todos usarem o *floating action Button* disposto no canto direito e o mesmo tipo de botão para as ações secundárias disponíveis no *header*.

A utilização da padronização e se for analisado especificamente em relação aos *guidelines*, existem desvantagens e vantagens de utilizar as

que são sugeridas pelo sistema operacional, que no caso os principais são o iOS da Apple e o Android da Google. Em que um dos aspectos positivos é a questão da familiaridade, pelo fato que o usuário já está acostumado em utilizar os aplicativos nativos daquele sistema operacional e quando é instalado um aplicativo de terceiro, caso siga o mesmo padrão fica muito fácil o seu entendimento. Esses aspectos estão relacionados na satisfação com o usuário e de usabilidade, principalmente aos padrões de interatividade que tornarão a operação do aplicativo extremamente intuitiva, pois será similar a dos outros produtos da Google. Mas a desvantagem é que pode ser criada uma confusão visual aos usuários, pelo fato da similaridade entre os aplicativos, sendo difícil a identificação de qual aplicativo o usuário está utilizando, principalmente no momento de abrir as janelas para alternar de um aplicativo para o outro. Um aplicativo que possua identidade visual única, de imediato, o usuário consegue identificar qual aplicativo está minimizando uma confusão visual que possa acontecer, mas também deve tomar cuidado para não desenvolver interfaces muito contrastantes que não possua visual agradável, apenas chamativo.

Dessa forma, é importante que se tenha equilíbrio na questão da utilização à risca das diretrizes das lojas virtuais, como a do *Material Design* da Google, em que para estudantes e profissionais iniciantes temos a opção de utilização a *bootstrap* copiando e colando elementos para montar uma interface. Nos primeiros projetos, e principalmente em relação para correr menos ricos de cometer erros, em relação à usabilidade do aplicativo, procure seguir as *guidelines* do Material Design o máximo que for possível, isso pode tornar mais consistente a usabilidade do aplicativo.

Para o desenvolvimento de *frameworks* para aplicativos mobile disponibilizados na plataforma da Apple é importante o conhecimento da linguagem de programação Objective-C, mas também pode ser empregada a linguagem Swift. Essa é uma linguagem de código aberto intuitiva, mas que não deixa de ser consistente e pode ser

utilizada no desenvolvimento de aplicativos para todos os dispositivos da Apple: Iphone, Apple TV, Mac e Apple Watch. A sua codificação é escrita de maneira eficiente e rápida, com respostas em tempo real; ela pode, ainda, ser agregada, de forma simples, a uma codificação da linguagem de programação Objective-C existente, permitindo que os desenvolvedores de codificação escrevam códigos mais confiáveis e mais seguros, otimizando o tempo e, consequentemente, oferecendo uma experiência melhor na utilização do aplicativo.

Essa linguagem de programação se torna mais popular a cada dia, sendo agregada por vários desenvolvedores aos seus aplicativos. Alguns são desenvolvidos totalmente em Swift, como o LinkedIn, o Eventbrite, o Sky Guide, o Lyft e o Kickstarter. Os aplicativos escritos em Swift apresentam desempenhos bem consideráveis em comparação aos de outras linguagens de programação, deixando o resultado mais dinâmico. No algoritmo comum de busca, por exemplo, o resultado é mais rápido com o uso de Swift: é até 2,6x mais rápido do que com o uso da linguagem de programação Objective-C; e até 8,4x mais rápido do que com o uso da linguagem de programação Python 2.7. É importante salientar que a linguagem de programação Swift é de código aberto, portanto, está disponível para desenvolvedores, para educadores e para alunos e utiliza a licença Apache 2.0.

A Apple oferece binários para MacOS e para Linux que podem compilar códigos para iOS, para MacOS, para watchOS, para tvOS e para Linux. Confira um exemplo de como é a sua codificação:

```
struct Player {  
  
    var name: String  
  
    var highScore: Int = 0  
  
    var history: [Int] = []
```

```
init(_ name: String) {  
    self.name = name  
}  
}  
  
var player = Player("Gustavo")
```

A sua codificação, a declaração de novos tipos com sintaxe moderna e direta, fornecendo valores padrão para propriedades de instância e define inicializadores personalizados, como muitos recursos que tornam o código mais expressivo:

1. Genéricos que são robustos e simples de usar.
2. Extensões de protocolo que tornam a escrita codificação genérico mais simples.
3. Funções de primeira classe e sintaxe de encerramento leve.
4. Iteração veloz e essencial em um intervalo ou coleção.
5. Tuplas e múltiplos valores de retorno.
6. Estruturas que sustentam métodos, extensões e protocolos.
7. Enums podem dispor cargas essenciais e ser compatível com padrões de suporte.
8. Padrões de programação funcionais (filtro e mapa).
9. Tratamento de erros nativo usando *try/catch/throw*.

Enfim, a linguagem Swift é interativa; se baseia em uma sintaxe que, apesar de ser concisa, é bastante expressiva; e inclui recursos modernos importantes para os desenvolvedores.

Neste tema, você pode evidenciar a importância do papel do profissional de desenvolvimento *Web full stack*, mesmo com todas as diretrizes

dos *frameworks*, em que é necessário utilizar da criatividade para se diferenciar dos outros projetos. Dessa maneira, fica evidente a importância dos profissionais de desenvolvimento *Web full stack* ter a visão geral sobre *frameworks back end*, conhecendo e saber selecionar para seus projetos.

Referências

AMBLER, T.; CLOUD, N.; HAWKES, R. A. **JavaScript Frameworks for Modern Web Dev.** Nova Iorque: Apress, 2015.

SMUTNY, P. Mobile development tools and cross-platform solutions. In: CARPATHIAN CONTROL CONFERENCE (ICCC), 13., 2012, [S.I.]. **Anais** [...]. [S.I.: s.n.], 2012. p. 653-656.

PRESSMAN, R.; MAXIM, B. **Engenharia de software**. 8. ed. McGraw: Hill Brasil, 2016.

Utilização do Framework Express

Autoria: Leandro C. Cardoso

Leitura crítica: Arthur Gonçalves Ferreira



Objetivos

- Apresentar o *Express* um *Framework Back End* com JavaScript e Node.js.
- Facilitar o entendimento para utilização do *middleware*.
- Compreender a codificação *Express*.



1. Express: um *Framework Back End* com JavaScript e Node.js

Para os profissionais e estudantes que trabalham com desenvolvimento *Web full stack* é importante o conhecimento do *Express*, pois se trata de um *Framework Back End* com JavaScript e Node.js. Para facilitar o entendimento do *Express* é importante conceituar JavaScript e Node.js. JS, ou JavaScript, que é uma linguagem de programação que tem seu uso mais comum para *scripts* dinâmicos. *Scripts* são códigos do lado do cliente em páginas nas páginas de internet, ou seja, web, no qual também pode ser usado no lado do servidor, por meio de um interpretador, esse interpretador a sua terminologia em inglês é *runtime*.

É sempre comum a confundir JavaScript com a linguagem de programação Java, mesmo com as palavras JavaScript e Java sendo registradas pela empresa Oracle em vários países, como nos Estados Unidos, por esses motivos JavaScript e Java são marcas comerciais. Ainda assim, trata-se de duas linguagens de programação que possuem diferenças significativas em relação a semântica, em sintaxe e casos de uso, o uso do JavaScript é geralmente no browser, ou seja, no navegador. O JavaScript é considerado como uma das linguagens mais usadas em todo o planeta, principalmente devido ao crescimento da linguagem e melhorias de desempenho implementadas das APIs disponíveis nos navegadores.

Node.js, ou simplesmente Node, é considerado um ambiente em tempo que executa código aberto ou *open-source* e multiplataforma, permitindo para os desenvolvedores a possibilidade de criar diversos tipos de ferramentas e aplicativos (*backend*) em JavaScript. O Node.js, ou Node Java Script, é usado externamente de um browser, ou seja, ele “roda” diretamente no servidor ou também no computador, mas omitindo o ambiente APIs JavaScript específicas do browser. Nesse sentido, é importante salientar que inclui também suporte para APIs para

Windows, Linux, MacOs, incluindo arquivos e bibliotecas de sistemas HTTP arquivos, verificando para o uso de criação de um servidor web, o Node.js possui de diversos benefícios.

A boa performance é uma das vantagens do o Node.js, deveio a ser planejado justamente para otimização da taxa de transferência e para escalabilidade em aplicações na internet, sendo uma solução para diversos problemas (BROWN, 2019). Por exemplo, no desenvolvimento de aplicações em tempo real da web, outro benefício é que o código escrito em JavaScript é considerado simples e antigo, assim profissionais mais experientes já possuem familiaridade com ele. Assim, não será dispensando o tempo para que tenha que analisar as alterações de código entre servidor web e navegador, pois não se faz necessário mudança na linguagem. Outra vantagem é que várias outras linguagens convertem e compilam em JavaScript, dessa forma, essa é uma maneira de também utilizar essas linguagens. como:

- LiveScript.
- Scala.
- TypeScript.
- CoffeeScript.
- ClosureScript.
- Entre outras.

O NPM, em português é conhecida como Gerenciador de Pacotes do Node.js, proporciona que se tenha acesso a diversos pacotes, com a vantagem de que podem ser utilizados mais do que uma vez. O NPM do Node.js dispõe de uma variada coleção de dependência, nos quais podem ser utilizados para propor automatização de grande parte da cadeia de ferramentas de compilação. Outro benefício do

NPM é que ele tem portabilidade e versões para uma variedade de sistemas operacionais, nos quais se pode destacar: OS X, Microsoft Windows, Linux, NonStop, Solaris, WebOS, FreeBSD e OpenBSD. Além disso, Segundo Cordeiro (2016), ele dispõe de suporte para diversos provedores de hospedagem na web e, em várias situações, até fornecem a infraestrutura e documentação própria para hospedar sites desenvolvidos em Node.js. O Node.js pode ser utilizado para a criação de um simples servidor web, por meio do uso do pacote denominado como Node HTTP.

Veja o exemplo de criação de um servidor web no qual escuta qualquer tipo de requisição HTTP na URL `http://127.0.0.1:8000/` — no momento que uma aquisição é recebida, o *script* pode responder com a *string* de texto *Frameworks back end*. Para executar esse teste é importante a instalação do Node pelo terminal, em que o sistema operacional utiliza o prompt da linha de comando, então, é importante criar um diretório para salvar o programa. Sugestão de nome de diretório: nodejs-teste. Agora, basta entrar no diretório utilizando os seguintes comandos no prompt do windows:

```
cd nodejs-teste
```

Em seguida, utilizando um editor de texto, crie um arquivo denominado como teste.js e cole o seguinte código:

```
// Carrega o modulo HTTP do Node
var http = require("http");
// Cria um servidor HTTP e uma escuta de requisições para a porta 8000
http.createServer(function(request, response) {
    // Configura o cabeçalho da resposta com um status HTTP e um Tipo de Conteúdo
    response.writeHead(200, {"Content-Type": "text/plain"});
    response.end("Frameworks back end");
});
```

```
response.writeHead(200, {'Content-Type': 'text/plain'});  
  
// Enviar o corpo da resposta "Frameworks back end "  
  
response.end('Frameworks back end \n');  
  
}).listen(8000, '127.0.0.1');  
  
// Imprime no console a URL de acesso ao servidor  
  
console.log('Servidor executando em http://127.0.0.1:8000/');
```

Executando esses procedimentos, salve o arquivo no diretório criado anteriormente, ou seja, a pasta que recebe o nome de nodejs, depois vá ao prompt do windows e digite o seguinte comando: teste.js. Por fim, basta abrir o navegador e digitar http://localhost:8000. O texto que irá ser visualizado no canto superior esquerdo será ***Frameworks back end***.

O *Express* nada mais é do que o *framework* Node.js mais conhecido e a biblioteca subjacente para diversos outros *frameworks* do Node.js, que dispõe de várias soluções para gerenciar as diversas requisições de verbos HTTP nas diversas URLs. Ele permite a integração *view engines* possibilitando a inserção de dados nos modelos, isto é, de templates, define os parâmetros rotineiros da aplicação web, por exemplo, a porta a ser utilizada para conectar e a localização dos modelos nos quais serão utilizados para renderizar a resposta. O *Express* também permite a adição de novos processos de requisição pelo *middleware* em quaisquer pontos da “fila” de requisições, sendo considerado minimalista, mas é possível que os desenvolvedores criem pacotes específicos de middleware. Esses pacotes, por sua vez, podem resolver determinados problemas que podem aparecer no decorrer do desenvolvimento de uma aplicação, além de existir bibliotecas para utilizar em diversas situações, como:

- Login de usuários.

- Cookies.
- Parâmetros de URL.
- Sessões.

Ademais, pode-se utilizar-los no cabeçalho de segurança, nos dados em requisições POST, entre outros, dispõe de relação de pacotes de *middleware* mantidos pela própria equipe *Express* no site: <http://expressjs.com/en/resources/middleware.html> (Acesso em: 27 set. 2021). Nesta URL também se encontra lista de pacotes nos quais também foram criados por terceiros.

1.1 Utilizando *middleware*

Middleware é muito utilizado em aplicativos *Express*, com objetivo de que as tarefas possam dispor arquivos estáticos no que diz respeito a compreensão de respostas HTTP e ao tratamento de erros. No momento no qual as atribuições de rota finalizam o ciclo de solicitação-resposta HTTP, é necessário retornar alguma resposta ao cliente HTTP, assim as funções de *middleware* geralmente executam alguma operação na solicitação ou resposta. Em seguida, ligam para a próxima etapa na “pilha”, no qual pode ser, por exemplo, uma rota manipuladora ou mais um *middleware*, sendo que a ordenação no qual o *middleware* é chamado dependerá do desenvolvedor do aplicativo.

É importante salientar que o *middleware* permite a execução de quaisquer operações, código, alterações no objeto tanto de resposta quanto de solicitação, como o encerramento do ciclo de solicitação-resposta. Caso o ciclo não seja encerrado, deve-se chamar o `next()` e que seja possível passar o controle para a próxima etapa de *middleware*, caso não execute esses procedimentos, a solicitação ficará pendurada. Normalmente, os aplicativos utilizam *middleware* de terceiros, pois simplificam tarefas comuns de desenvolvimento web, como trabalhar

com autenticação de usuários, cookies, acessar dados POST e JSON, sessões, log, entre outros.

Além de estar disponível, a lista de pacotes de *middleware* desenvolvidos pela própria equipe *Express*, nos quais inclui alguns pacotes de terceiros, outros pacotes *Express* podem ser acessados no próprio gerenciador de pacotes do NPM. Para utilização de *middleware* de terceiros é necessário, primeiramente, a instalação em seu aplicativo usando NPM. Veja o exemplo de instalação do logger morgan HTTP:

```
$ npm install morgan
```

Em seguida, é possível chamar `use()` no objeto do aplicativo *Express* para adicionar o *middleware* à pilha:

```
var express = require('express');
```

```
var logger = require('morgan');
```

```
var app = express();
```

```
app.use(logger('dev'));
```

```
...
```

Nesses sentidos, é preciso ter cuidado, pois as funções de roteamento e o *middleware* são chamadas na ordem nos quais são declaradas e, dependendo do *middleware*, a ordem tem a sua importância, principalmente nos casos em que o *middleware* de sessão tem dependência do *middleware* de cookies. Nessa situação, o manipulador de cookies deve ser adicionado primeiro, assim, o *middleware* é chamado primeiro antes da definição de rotas ou seus manipuladores de rotas não terão acesso às funcionalidades que foram adicionadas pelo *middleware*. É possível escrever as próprias funções de *middleware*, provavelmente terá que ser feito apenas para a criação de código de

manipulação de erro, em que a distinção entre retorno de chamada de manipulador de rotas e uma função de *middleware*. As funções de *middleware* possuem um terceiro argumento next, nos quais as funções de *middleware* devem chamar se não completam o ciclo de solicitação, mas no momento em que a função de *middleware* é chamada, ela contém a próxima função que deve ser chamado.

Ademais, é possível a adição de uma função de *middleware* na cadeia de processamento com app.add() ou app.use(), mas isso dependerá do objetivo, se é de aplicar o *middleware* no GET, POST etc., ou seja, com um verbo HTTP específico ou a todas as respostas. Em ambos os casos, deve ser especificadas as rotas, mesmo que a rota seja opcional ao chamar app.use(), veja o exemplo a seguir apresentando, como pode ser a adicionado da função *middleware* usando os dois métodos sem rota e com rota:

```
var express = require('express');

var app = express();

// Exemplo de função middleware

var a_middleware_function = function(req, res, next) {

    // ... Executar determinada operação

    next(); // next() Chamar a próxima função de rotas ou middleware

}

// Função adicionada com use() para todas rotas e requisições

app.use(a_middleware_function);

// Função adicionada com use() para uma rota determinada
```

```
app.use('/somerroute', a_middleware_function);

// função middleware adicionado para uma rota e requisição determinada

app.get('/', a_middleware_function);

app.listen(3000);
```

Analizando o exemplo anterior, fica fácil de identificar que foi executada a declaração de função de *middleware*, mas de maneira separada. Ainda observando o código, nota-se que foi executada a configuração com objetivo de obter retorno de chamada, em que na função anterior do operador de rotas, ela foi declarada a função de retorno de chamada, dessa maneira, no JavaScript, os dois métodos são válidos.

1.2 A codificação *Express*

Para entender melhor, um site padrão que se baseia em dados ou um aplicativo da Web precisa aguardar pedidos HTTP do browser, ou seja, do navegador web, ou de outro cliente, e no momento em que é recebido o pedido, o aplicativo faz a descrição das ações que devem ser executadas. Com referência as diretrizes de URL e dados nos quais possui associações que inclui informações POST ou GET. E conforme que necessário, a função irá executar a escrita ou a leitura de dados em específicos BD (banco de dados), ou até mesmo a execução de outras tarefas que é preciso para solucionar a solicitação. Depois desses procedimentos, o aplicativo terá o retorno com uma resposta ao browser, cujo resultado será a criação de maneira dinâmica a apresentação de uma página HTML para o browser, no qual serão inseridos e exibidos os dados que foram recuperados em espaços reservados em um template, ou seja, no modelo HTML.

O *Express* oferece metodologia que pode executar a especificação de qual tarefa será chamada, no momento que a requisição é solicitada

HTTP (SET, POST, GET, entre outras) e de rotas como também de métodos para determinar o mecanismo de modelo (*view*) utilizado, no qual o modelo arquivos estão localizados e qual modelo será utilizado com objetivo de renderizar uma resposta, da mesma forma é possível usar o *middleware Express* para exercer a função de adicionar suporte para:

- Sessões.
- Usuários.
- Cookies.

Assim, resultam nos parâmetros tanto de GET quanto de POST, entre outros, e que permite usar quaisquer procedimentos de BD, mas logicamente os que possuem suporte de Node.js. É importante compreender que o *Express* não executa qualquer definição relacionada ao comportamento relativo ao BD, a seguir mostramos um exemplo de seção que facilita o entendimento de determinadas semelhanças em relação aos códigos Node, com o *Express*.

Para evitar erros é preciso se atentar, caso já tenha o Node.js e o *Express* já instalados, para salvar o código a seguir em um arquivo denominado como app.js, e executá-lo diretamente em um prompt, apenas digitando o comando node app.js.

```
var express = require('express');
```

```
var app = express();
```

```
app.get('/', function(req, res) {
```

```
    res.send('Frameworks back end');
```

```
});
```

```
app.listen(4000, function() {  
  console.log('App de Exemplo escutando na porta 4000!');  
});
```

Observe que a primeira e a segunda linha require() têm a função de importar o módulo *Express*, e criar uma aplicação *Express*, sendo que esse objeto que geralmente nomeado de *app* possui métodos de rotear requisições HTTP, como:

- Alteração das diretrizes.
- Configurações de *middleware*.
- Registro de *engines* de templates (motores de modelo).
- Renderização de *views* HTML.

É importante considerar que todos esses parâmetros têm o objetivo de controle do comportamento do aplicativo. Para melhor exemplificar, ao usar o ambiente no momento em que as definições de rota são sensíveis, os caracteres maiúsculas e minúsculas, entre outros. E continuando analisando o código, observe as três linhas que têm início app.get na parte do meio código, note que sua função é para a definição da rota e no momento em que o método app.get() tem ação de determinar uma função com objetivo de retorno de chamada. Essa chamada será invocada toda vez que existir a solicitação HTTP GET, mais especificamente com um caminho ('/'), que tem relação com a raiz do site. Observe que a tarefa de retorno de chamada tem a função de execução de requisição e um objeto para resposta, mas com argumentos e simplesmente chama send() para executar a função de retorno a *string Frameworks back end*. Já o bloco final tem a função de iniciar o servidor na porta '4000', e executar a impressão de um comentário de log no console, e com o servidor em execução, permite

acessar o localhost:4000 no browser, para executar a verificação da resposta no qual é retornado.

1.3 Importação e criação de módulos

Para entender melhor, um módulo nada mais é do que um arquivo ou biblioteca de JavaScript no qual é possível a importação para outro código pelo uso da função `require()` do Node.js, o *express* já é um modulo. Além disso, é o banco de dados e as bibliotecas de *middleware* que são utilizados nos aplicativos *Express*, veja o exemplo de uma codificação, em que apresentamos como é importando um módulo pelo nome. A primeira ação é de invocar a função `require()`. Depois é preciso especificar o nome do módulo como uma *string* (`<express>`), chamando o objeto retornado para o desenvolvimento de um aplicativo *Express*, depois de executado esses procedimentos, é possível ter o acesso às propriedades e funções do objeto da aplicação.

```
var express = require('express');
```

```
var app = express();
```

Além disso, é possível a criação de módulos personalizados com objetivo de importar da mesma forma, então, é importante criar módulos próprios por permitir que a organização do código seja do desenvolvedor em peças gerenciáveis, como em um aplicativo monolítico, ou seja, de arquivo único, no qual pode ser difícil o entendimento a sua manipulação, em que a utilização de módulos facilita o gerenciamento do *namespace*. Pelo fato de manter apenas as variáveis que foram exportadas explicitamente, nos quais são importadas no momento da utilização de um modulo e para tornar os objetos à disposição externamente, ou seja, forma do módulo. É necessário somente a atribuição ao *exports*, veja o exemplo a seguir do módulo `square.js`, no qual é um arquivo que exporta os métodos `area()` e `perimeter()`:

```
exports.area = function(width) { return width * width;};
```

```
exports.perimeter = function(width) { return 4 * width;};
```

Desse modo, é possível executar a importação do módulo utilizando require(), posteriormente conectar aos métodos que foram exportados, como o seguinte exemplo:

```
var square = require('./square'); // Chamamos o arquivo utilizando o require()
```

```
console.log('The area of a square with a width of 4 is ' + square.area(4));
```

É importante salientar que é permitida a especificação de um nome ou caminho absoluto para o módulo, como foram executados anteriormente, caso queira a exportação de um objeto de maneira completa em uma atribuição. Contudo, não apenas a criação de uma propriedade de cada vez, basta atribuir ao module.exports como apresentado no modelo a seguir, em que é possível executar esses procedimentos para que se torne a raiz do objeto exportar outra função ou um construtor.

```
module.exports = {
```

```
    area: function(width) {
```

```
        return width * width;
```

```
    },
```

```
    perimeter: function(width) {
```

```
        return 4 * width;
```

```
    }
```

```
};
```

Outro ponto importante de considerar é que a codificação JavaScript, geralmente, utiliza APIs assíncronas em vez de síncronas para operações, que podem levar determinado tempo para serem concluídas (AMBLER; CLOUD; HAWKES, 2015). Para compreender melhor esse assunto, uma síncrona se caracteriza no qual cada operação precisa ser concluída antes que a próxima operação tenha o seu início, veja no exemplo a seguir de funções de log síncronas e que irá imprimir o texto (um, dois) no console em ordem:

```
console.log('um');
```

```
console.log('dois');
```

Em compensação, uma API com característica assíncrona se caracteriza em que a API terá início com uma operação e o retorno de maneira instantânea, antes mesmo de concluir a operação, e no momento que a operação finalizar, ela utilizará mecanismo determinado para executar operações adicionais. Veja o exemplo a seguir, a codificação irá imprimir “um, dois” mesmo que o método setTimeout() seja chamado primeiro e retornasse instantaneamente, a operação necessita de quatro segundos para terminar.

```
setTimeout(function() {
```

```
    console.log('um');
```

```
}, 4000);
```

```
console.log('dois');
```

Ainda é mais importante o uso de APIs assíncronas não bloqueadoras, quando está utilizando Node.js do que quando está no browser, pelo fato que o Node.js seja um ambiente de execução orientado *single threaded*, ou seja, por evento único. *Single threaded* quer dizer que a totalidade dos pedidos enviados para o servidor a sua execução será no

mesmo tópico, em vez da sua geração ser processos separados, dessa forma, se torna um modelo eficiente e veloz. No que diz respeito aos recursos do servidor, assim, quaisquer uma das suas tarefas chamam métodos síncronos que pode demorar bastante tempo para finalizar quando acontecer essa situação bloqueiam somente a solicitação atual. Mantendo as demais solicitações que serão tratadas por sua aplicação web, há diversas formas de uma API assíncrona fazer a notificação para a aplicação que alguma tarefa foi finalizada, em que a maneira mais comum é executar um registro de uma função de retorno de chamada quando é invocado a API assíncrona, que será chamada de volta quando a operação for concluída, no qual foi usado no exemplo anterior.

Deve-se tomar cuidado para usar *callbacks*, pois, de certa forma, pode “bagunçar” caso possua uma sequência de operações assíncronas dependentes, nos quais devem ser executadas em ordem, pois, essa operação resultará em múltiplo níveis de *callbacks* aninhados. Essa é uma convenção comum para Node.js e Express é utilizar as devoluções de retorno de erro e, nessa convenção, o primeiro valor nas tarefas de retorno de chamada é um valor de erro, enquanto os argumentos posteriores irá conter dados de sucesso.

É importante saber criar manipuladores de rotas, nos exemplos anteriores foram definidos uma *callback* função manipuladora de rota para requisição GET HTTP para a raiz do site ('/').

```
app.get('/', function(req, res) {  
  res.send('Frameworks back end');  
});
```

Observe que a função de retorno de chamada necessita uma solicitação e um objeto de resposta como argumento nessa situação, o método apenas chama `send()` na resposta para retornar a string “Frameworks”

"back end". Existem outros métodos de resposta para finalizar o ciclo de solicitação/resposta, chamar res.json() para envio de uma resposta JSON ou res.sendFile() para enviar um arquivo. Além disso, pode ser utilizado quaisquer argumentos de uso pessoal do desenvolvedor nas funções de retorno de chamada e, no momento no qual o retorno de chamada seja invocado, o primeiro argumento sempre será o pedido e o segundo sempre será a resposta. Dessa maneira, recomenda-se nomeá-los de tal forma que facilite a identificação do objeto que está trabalhando no corpo do retorno de chamada.

O Express fornece métodos de definição de manipuladores de rotas para todas as outras requisições HTTP, que são utilizadas especificamente do mesmo modo: connect(), purge(), search(), move(), post(), patch(), proppatch(). Como também unsubscribe(), subscribe(), put(), delete(), notify(), options(), trace(), copy(), lock(), mkcol(), propfind(), unlock(), report(), mkactivity(), checkout(), merge() e m-search(). Existe também método de rotear especial, app.all(), que será chamado em resposta a qualquer método HTTP. Ele é utilizado para carregamento de funções de *middleware*, em um caminho específico para todos os métodos de solicitação. Veja o exemplo a seguir da documentação Express, apresentando um manipulador que será executado para solicitações /secret, independentemente do verbo HTTP usado, desde que seja suportado pelo módulo HTTP.

```
app.all('/secret', function(req, res, next) {  
  
  console.log('Acessando a sessão secreta...');  
  
  next(); // passa o controle para o próximo manipulador  
  
});
```

Analizando a codificação fica fácil de observar que as rotas proporcionam combinar padrões de determinados caracteres em

um URL, além de extrair determinados valores do URL e enviar esses valores como atributos para o manipulador de rotas, como parâmetros do objeto de solicitação passado como parâmetro. Por meio do conteúdo aqui apresentado fica fácil de compreender a importância dos profissionais de Desenvolvimento Web full stack, conhecer o *Express*, um *Framework Back End* com JavaScript e Node.js.

Referências

AMBLER, T.; CLOUD, N.; HAWKES, R. A. **JavaScript Frameworks for Modern Web Dev.** Nova Iorque: Apress, 2015.

BROWN, E. **Web development with node and express:** leveraging the JavaScript stack. Sebastopol: O'Reilly Media, 2019.

CORDEIRO, A. C. **Dyfocus:** Desenvolvimento do Back-End de um Aplicativo Mobile para Smartphone. 2016. Monografia (Graduação em Engenharia de Controle e Automação)–Universidade Federal de Santa Catarina, Florianópolis, 2016.

MORAES, W. B. **Construindo aplicações com NodeJS.** São Paulo: Novatec, 2018.

Framework Back End Laravel

Autoria: Leandro C. Cardoso

Leitura crítica: Arthur Gonçalves Ferreira



Objetivos

- Orientar os conceitos fundamentais do *Framework Back End Laravel*, potencializando o PHP.
- Entender a utilização do *Framework Back End Laravel* nas plataformas maços, Windows e Linux.
- Analisar os recursos disponíveis do *Framework Back End Laravel*.



1. Framework Back End Laravel

Laravel surgiu em 2011, da necessidade de uma ferramenta mais completa para codificação em PHP, sendo um framework de aplicação web com sintaxe expressiva, ele possui estrutura da web que dispõe de uma estrutura e um ponto de partida para a criação de seu aplicativo. Essas características permitem que o desenvolvedor se dedique na criação do que dos mínimos detalhes da codificação, o Laravel oferece recursos considerados robustos como:

- Testes de integração.
- Injeção de dependência completa.
- Agendamento de filas.
- Testes de unidade.
- Agendamento de trabalhos.
- Camada de abstração de banco de dados expressiva.
- Entre outros.

Ele dispõe de diversas ferramentas e estruturas para o desenvolvimento de aplicativos da web, facilitando a criação de aplicativos robustos e moderno utilizando os recursos Laravel, que possui uma extensa biblioteca de documentação. Segundo Gabardo (2017), além de tutoriais em vídeo e guias que facilita o aprendizado das informações básicas para os desenvolvedores experientes, o Laravel dispõe de ferramentas completas que permitem de serem configuradas, ou seja, personalizadas para os diversos projetos. O Laravel é escalonável devido as suas características de escalonamento do PHP, ao suporte embutido para sistemas de cache rápido e distribuído como o Redis, facilitando o escalonamento horizontal.

É recomendado o uso quando for utilizar aplicativos que precisam lidar com centenas de milhões de solicitações por mês, além da facilidade de escalonamento mais robustos, por meio de uso de plataformas como a denominada Laravel Vapor. Essa plataforma proporciona para o desenvolvedor executar o aplicativo em escala, quase ilimitada na mais recente tecnologia sem servidor da *Amazon Web Services* (AWS), plataforma de serviços de computação em nuvem. Outra vantagem do Laravel é combinar os melhores pacotes do ecossistema PHP, com objetivo de proporcionar o *framework* robusto e *friendly*, ou seja, considerado amigável para os desenvolvedores e com diversas opções para execução e criação de projetos.

Os projetos podem ser desenvolvidos no próprio computador do desenvolvedor ou utilizar os recursos como a que recebe o nome de Sail, que se trata de uma solução embutida para executar projetos Laravel usando o Docker. Por sua vez, o Docker é uma série de plataformas e produtos, como serviço que utilizam virtualização de nível de sistema operacional para entregar programas em pacotes chamados contêineres. Os contêineres são isolados uns dos outros e executam agrupamento seus próprios programas, bibliotecas e arquivos de configuração. De maneira geral, o Docker é uma ferramenta para execução de aplicativos e serviços em “contêineres”, que podem ser leves e pequenos, e que não façam interferência na configuração, ou no software que está instalado no computador do profissional que está desenvolvendo. Assim, não existe a preocupação das diretrizes ou configuração de ferramentas de desenvolvimento complexas, como os bancos de dados e os servidores da web no próprio computador do desenvolvedor, ou seja, para utilizar só é necessário instalar a ferramenta Docker Desktop.

O Laravel Sail é uma interface de linha de comando considerada simples, que tem objetivo de interação com a configuração padrão do Docker do Laravel, dessa forma, ele é um bom ponto de partida para construção

de aplicativos Laravel, utilizando Redis, PHP, MySQL e sem precisar conhecimentos avançados do Docker.

1.1 Utilização nas plataformas macOS e Windows

O primeiro procedimento é a instalação do Docker Desktop e, posteriormente, no sistema operacional Mac, basta utilizar um comando de terminal simples para criação de um novo projeto Laravel. Veja o exemplo de criação de um aplicativo Laravel da pasta que recebe o nome “aplicativo-exemplo”, para acessar basta executar os seguintes comandos no terminal do MacOS:

```
curl -s "https://laravel.build/aplicativo-exemplo" | bash
```

As pastas do aplicativo Laravel serão criadas dentro da pasta em que será executado o comando e, depois do desenvolvimento do projeto, é possível navegar até a pasta do aplicativo e começar o Laravel Sail. Ele dispõe de uma simples interface de linha de comandos e interações das diretrizes padrão do Docker do Laravel, bastando somente a execução dos seguintes comandos:

```
cd aplicativo-exemplo
```

```
./vendor/bin/sail up
```

Na primeira vez em que for executado o comando, ele receberá o nome de up comando Sail e serão constituídos os contêineres de aplicativos do Sail no computador local do desenvolvedor. Ainda, é importante se atentar que pode demorar alguns minutos, mas os próximos acessos serão mais velozes. Após os contêineres do Docker do aplicativo serem iniciados, é possível o acesso do aplicativo no browser, ou seja, no navegador web pelo caminho:

```
http://localhost
```

Quanto é utilizado no sistema operacional da Microsoft Windows, também é necessário que o aplicativo Docker Desktop esteja instalado, depois é importante ter a certeza que o Subsistema Windows para Linux 2 (WSL2) está ativado e instalado. Como o Windows Subsystem for Linux (WSL) viabiliza que possa ser executados os executáveis binários do Linux originário no Windows 10. O Subsistema Windows para Linux nada mais é do que um módulo do Windows 10 com objetivo de disponibilização de um ambiente Linux, mas um ambiente que seja compatível no sistema da Microsoft, de maneira que seja possível a execução de programas nativos dos sistemas GNU/Linux dentro do próprio Windows sem precisar utilizar máquinas virtuais ou emuladores, depois de executado esses procedimentos já é possível o início do primeiro projeto.

No Windows basta acessar o terminal, ou seja, o prompt do comando por meio do executável “cmd.exe” e depois é preciso iniciar uma nova sessão no terminal para o sistema operacional WSL2 Linux, e posteriormente é necessário somente utilizar o comando de terminal simples para o desenvolvimento de um projeto novo Laravel. Observe esse exemplo para a criação de um novo projeto Laravel em um diretório no qual será nomeado de “aplicativo-exemplo”, basta executar o comando no prompt do windows:

```
curl -s https://laravel.build/aplicativo-exemplo | bash
```

Quando utilizado o sistema operacional da Apple MacOs, a pasta do aplicativo Laravel será criada dentro da pasta, a partir do qual é executado o comando, depois da criação do projeto é possível navegar nas pastas do aplicativo e iniciar o Laravel Sail pelos seguintes comandos:

```
cd aplicativo-exemplo
```

```
./vendor/bin/sail up
```

Na primeira vez em que é executado o comando `Sail`, os contêineres de aplicativos do `Sail` serão construídos no computador do desenvolvedor, o que pode levar algum tempo, mas nas próximas execuções serão mais rápidas. Após os contêineres do `Docker` do aplicativo forem iniciados, é possível o acesso do aplicativo no browser pelo endereço:

`http://localhost`

Para criar internamento no `WSL2` é preciso executar a alteração dos arquivos do aplicativo `Laravel` que foram desenvolvidos na instalação do `WSL2`, para a execução desses procedimentos é recomendado usar o editor de codificação do `Visual Studio` da `Microsoft`. E a extensão original para desenvolvimento remoto, depois que essas ferramentas estão instaladas, é possível abrir quaisquer projetos `Laravel` executando o código comando da pasta principal, ou seja, diretório raiz do aplicativo utilizando o prompt do `Windows`.

1.2 Recursos do Laravel

Para utilizar no `Linux` com o `Docker` já instalado, basta utilizar um comando de terminal para o desenvolvimento de um novo projeto `Laravel`, veja o exemplo de criação de um novo aplicativo `Laravel` na pasta chamada “`aplicativo-exemplo`”, basta a execução do seguinte comando:

```
curl -s https://laravel.build/aplicativo-exemplo | bash
```

Para navegar até a pasta do aplicativo e iniciar o `Laravel Sail`, execute o mesmo comando do sistema operacional `windows` e `macOs`:

```
cd aplicativo-exemplo
```

```
./vendor/bin/sail up
```

Em seguida, o acesso do aplicativo também será pelo endereço no browser web em:

`http://localhost`

Depois da criação de um novo aplicativo Laravel via Sail é possível utilizar a *with* variável *query string*, que permite escolher quais serviços devem ser configurados no `docker-compose.yml`, arquivo do novo aplicativo, em que os serviços disponíveis que estão inclusos são:

- Pgsql.
- Mysql.
- Redis.
- Mariadb.
- memcached.
- Selenium.
- Mailhog.
- Meilisearch.

Veja um exemplo através da linha de comando abaixo:

```
curl -s "https://laravel.build/aplicativo-exemplo?with=mysql,redis" | bash
```

Caso não seja especificado quais serviços serão configurados, uma pilha padrão de mysql, redis, meilisearch, mailhog e selenium serão configurados. Para instalar pelo Composer, primeiramente, é importante que o computador já tenha o PHP e o Composer instalados. Para entender melhor o Composer, trata-se de um gerenciador de dependências para PHP, que é uma ferramenta simples e considerada

confiável, nos quais os desenvolvedores utilizam para o gerenciamento e integração de pacotes ou bibliotecas externas de projetos baseados em PHP. É possível a criação de um novo projeto Laravel utilizando o Composer diretamente, após a criação do aplicativo, pode-se dar início do servidor de desenvolvimento local do Laravel usando o comando Artisan CLI:

```
composer create-project laravel/laravel aplicativo-exemplo
```

```
cd aplicativo-exemplo
```

```
php artisan serve
```

Para facilitar, você pode utilizar o instalador do Laravel ou, se preferir, a instalação do instalador como uma dependência global do Composer, para essa segunda opção basta utilizar os seguintes comandos:

```
composer global require laravel/installer
```

```
laravel new aplicativo-exemplo
```

```
cd aplicativo-exemplo
```

```
php artisan serve
```

É significativo executar a verificação de inserção do diretório que recebe o nome de "bin" do fornecedor da totalidade do sistema do Composer em seu \$PATH. Para que o Laravel executável possa ser localizado pelo sistema é importante tomar o cuidado, pois, essa pasta está disponível em vários locais de acordo com sistema operacional utilizando, porém existem similaridades de alguns locais comuns que incluem:

- Mac OS: \$HOME/.composer/vendor/Bin
- Janelas: %USERPROFILE%\AppData\Roaming\Composer\vendor\bin

- Distribuições GNU / Linux: \$HOME/.config/composer/vendor/Bin ou \$HOME/.composer/vendor/Bin

Para facilitar o instalador do Laravel, também se pode criar um repositório Git para o novo projeto, em que para fazer essa indicação que um repositório Git seja criado basta indicar --git no momento de criar o projeto conforme o exemplo:

```
laravel new aplicativo-exemplo --git
```

A função desse comando é executar a inicialização de um novo repositório Git para o projeto indicado com envio automático do esqueleto base do Laravel, o git assume que foi instalado e configurado corretamente o Git. Além disso, é possível utilizar o -branch, em que para definição do nome inicial do branch siga o exemplo:

```
laravel new aplicativo-exemplo --git --branch="main"
```

Assim, em vez de utilizar a -gits é possível utilizar a -github, para a criação de um repositório Git e, também, a criação de um repositório privado correspondente no GitHub, conforme o exemplo a seguir:

```
laravel new example-app --github
```

Como o repositório criado estará disponível em <https://github.com/<your-account>/my-app.com>, o github deduz que foi instalada corretamente a gh ferramenta CLI está autenticado com o GitHub. Além disso, também é importante dispor do git instalado e configurado corretamente, caso seja preciso, deve-se permitir sinalizadores adicionais compatíveis com a CLI do GitHub, por exemplo:

```
laravel new aplicativo-exemplo --github="--public"
```

É possível também utilizar a -organization para a criação do repositório em uma organização GitHub específica, como:

```
laravel new aplicativo-exemplo —github="—public" —organization="laravel"
```

Todos os arquivos de configuração do *framework* Laravel são armazenados na pasta config, em que cada opção é documentada. Dessa maneira, é importante examinar os arquivos e se familiarizar com as opções disponíveis. O Laravel precisa de poucas configurações que não estejam no padrão do próprio *framework*, mas é importante executar uma revisão do arquivo denominado como config/app.php e sua documentação, pois é o local que está contido diversas opções, como *timezone* e local que deseja alterar conforme as características do projeto de aplicativo que está desenvolvendo.

A configuração tem como base o ambiente, pois, uma vez que vários dos parâmetros das possibilidades de configurações do Laravel são capazes de possuir variações de acordo a sua aplicação e estão em execução no computador local. Além disso, no momento que está sendo executado no servidor web de produção, vários parâmetros de configuração fundamentais serão determinados utilizando o .env, que nada mais é do que um arquivo existente na raiz da aplicação do projeto que está em desenvolvimento. É fundamental que que o arquivo denominado como o .env não possa sofrer comprometimentos em relação ao controle de origem do aplicativo no qual está sendo criado. Então, é importante lembrar que cada desenvolvedor ou servidor faça o uso de seu próprio aplicativo e pode solicitar uma configuração ambiente diferente. Nessa situação, pode ser considerado um risco de segurança, especialmente se acontecer a situação de um hacker ou invasor tiver acesso ao repositório de controle de origem, tendo em vista que quaisquer credenciais consideradas confidenciais podem entrar em exposição.

É importante o conhecimento dos tipos de variáveis de ambiente, pois todas as variáveis em seus .env arquivos são, normalmente, analisadas como *strings*, portanto, alguns valores reservados foram criados para permitir que retorne uma gama mais ampla de tipos da env()função, conforme quadro a seguir:

Quadro 1 – Tipos de variáveis de ambiente

.env Valor	env() Valor
Verdadeiro	(bool) verdadeiro
(verdadeiro)	(bool) verdadeiro
falso	(bool) falso
(falso)	(bool) falso
Vazio	(corda) “
(vazio)	(corda) “
.env Valor	env() Valor
Nulo	(nulo)
(nulo)	(nulo)

Fonte: elaborado pelo autor.

Em relação as configurações de pastas, o Laravel deve sempre ser servido a partir da raiz da “pasta web” configurada para o seu servidor web. Nesse contexto, é importante que não opte por servir a um aplicativo Laravel fora de uma subpasta da “pasta da web”, pois executando esse procedimento existe grande possibilidade de exposição de arquivos confidenciais, que estão no aplicativo que está sendo desenvolvido. Depois da criação do projeto Laravel, é importante se familiarizar com o funcionamento do Laravel, principalmente no que diz respeito as documentações referentes às:

- Fachadas.
- Estruturas de pastas ou diretórios.
- Solicitações de ciclo de vida.
- Configurações e contêiner de serviço.

O interessante é que o Laravel pode servir como um *framework full stack*. Para entender melhor o que é um *framework “full stack”* é aquele que pode ser usado o Laravel para roteamento de requisições, aplicação que está sendo desenvolvida. Além disso, também é possível renderizar o *frontend* do projeto por meio de templates Blade ou utilizando uma tecnologia híbrida de aplicação de página única, como o Inertia.js. Essa é a forma considerada mais comum de usar o *framework* Laravel, para isso, é importante verificar a documentação de roteamento, ORM do Eloquent ou visualizações. Além do aprofundamento dos pacotes Livewire e Inertia.js, pois permitem a utilização do Laravel como uma estrutura full-stack no mesmo momento em que são utilizados todos os benefícios da interface do usuário, fornecidos por aplicativos JavaScript de página única (BROWN, 2019). Nesse sentido, recomenda-se o conhecimento adicional relacionado como compilar o CSS e JavaScript da aplicação que está sendo desenvolvida, utilizando o Laravel Mix.

O Laravel também pode ser utilizado como um *backend* de API para um aplicativo JavaScript de página única ou aplicativo móvel, caso deseje usar o Laravel como um *backend* de API para um aplicativo Next.js (CORDEIRO, 2016). Nessa situação, deve-se utilizar o Laravel para o fornecimento de autenticação e recuperação ou armazenamento de dados da aplicação que está sendo desenvolvida. É importante salientar que para o desenvolvimento de um arquivo novo de aplicativo no Laravel, a primeira ação é configurar o banco de dados e executar as migrações de bancos de dados, de acordo com o exemplo a seguir:

```
curl -s https://laravel.build/aplicativo-exemplo | bash
```

```
cd aplicativo-exemplo
```

```
php artisan migrate
```

Em seguida, para o desenvolvimento de um aplicativo novo Laravel, deve-se executar a instalação do Laravel Breeze utilizando Composer, conforme o exemplo a seguir:

```
composer require laravel/breeze --dev
```

Após o Composer executar a instalação do pacote Laravel Breeze, deve-se executar o `breeze:install`, comando Artisan, em que é esse comando o responsável por publicar as visualizações de autenticação, rotas, controladores e outros recursos para aplicativo que está sendo desenvolvido. O Laravel Breeze publica toda codificação na aplicação, para permitir total controle e visibilidade sobre os recursos e implementações. Depois da instalação do Breeze é possível compilar seus ativos para que o arquivo CSS do aplicativo esteja disponível conforme o exemplo:

```
php artisan breeze:install
```

```
npm install
```

```
npm run dev
```

```
php artisan migrate
```

Assim, é possível navegar o aplicativo `/login` ou `/register` URLs no browser, em que todas as rotas do Breeze são definidas no `routes/auth.php` arquivo, o Laravel Breeze também oferece uma implementação de front-end `Inertia.js` desenvolvida por Vue ou React. Para utilizar uma pilha de inércia, deve-se especificar o `vue` ou `react` como a pilha desejada ao executar o `breeze:install` comando Artisan, conforme o exemplo a seguir:

```
php artisan breeze:install vue
```

```
// Ou...
```

```
php artisan breeze:install react
```

```
npm install
```

```
npm run dev
```

```
php artisan migrate.
```

Caso esteja implantando o aplicativo em um servidor que esteja executando o Nginx, no qual é um servidor HTTP, proxy reverso, proxy de e-mail IMAP/POP3 é considerado leve, pois consome menos memória que o Apache, pelo fato de lidar com requisições Web do tipo *“event-based web server”*; e o Apache é baseado no *“process-based server”*, podendo trabalhar juntos. É importante certificar de que, como a configuração abaixo, o servidor web direcione todas as solicitações para o public/index.php arquivo do aplicativo, mas tome o cuidado para nunca mover o index.php arquivo para a raiz do projeto, pelo fato de servir o aplicativo da raiz do projeto irá expor muitos arquivos de configuração confidenciais para a Internet pública:

```
server {  
  
    listen 80;  
  
    server_name example.com;  
  
    root /srv/example.com/public;  
  
    add_header X-Frame-Options "SAMEORIGIN";  
  
    add_header X-Content-Type-Options "nosniff";  
  
    index index.php;  
  
    charset utf-8;
```

```
location / {  
  
    try_files $uri $uri/ /index.php?$query_string;  
  
}  
  
location = /favicon.ico { access_log off; log_not_found off; }  
  
location = /robots.txt { access_log off; log_not_found off; }  
  
error_page 404 /index.php;  
  
location ~ \.php$ {  
  
    fastcgi_pass unix:/var/run/php/php7.4-fpm.sock;  
  
    fastcgi_param SCRIPT_FILENAME $realpath_root$fastcgi_script_name;  
  
    include fastcgi_params;  
  
}  
  
location ~ /\.well-known.* {  
  
    deny all;  
  
}  
  
}
```

Ao implantar o aplicativo para produção, também, é importante se certificar de executar o config:cache, comando Artisan durante o processo de implantação, por meio dos seguintes comandos:

```
php artisan config:cache
```

Esse comando combinará todos os arquivos de configuração do Laravel em um único arquivo em cache, o que reduz bastante o número de viagens que o *framework* deve fazer ao sistema de arquivos ao carregar seus valores de configuração. Caso seja executado o config:cache, durante o processo de implantação é importante verificar que está apenas chamando a env função de dentro dos arquivos de configuração. Depois que a configuração for armazenada em cache, o .env arquivo não será carregado e todas as chamadas para a env função para .env variáveis serão retornadas null.

A opção de depuração do arquivo de configuração config / app.php determina quanta informação sobre um erro é realmente exibida para o usuário, por padrão, essa opção é definida para respeitar o valor da variável de ambiente APP_DEBUG, que é armazenada em seu arquivo .env. Em seu ambiente de produção, esse valor deve ser sempre *false*. Caso a APP_DEBUG variável for definida como *true*, se corre o risco de expor valores de configuração confidenciais aos usuários finais do aplicativo, sendo que para o desenvolvimento *Web full stack* os profissionais que conhecem o *Framework Back End Laravel* são potencializados com o PHP.

Referências

BROWN, E. **Web development with node and express:** leveraging the JavaScript stack. Sebastopol: O'Reilly Media, 2019.

CORDEIRO, A. C. **Dyfocus:** Desenvolvimento do Back- End de um Aplicativo Mobile para Smartphone. 2016. Monografia (Graduação em Engenharia de Controle e Automação)-Universidade Federal de Santa Catarina, Florianópolis, 2016.

GABARDO, A. C. **Laravel para ninjas.** São Paulo: Novatec, 2017.

| **Framework Back End Spring Boot**

Autoria: Leandro C. Cardoso

Leitura crítica: Arthur Gonçalves Ferreira

Objetivos

- Conhecer os fundamentos do Framework Back End Spring Boot.
- Entender o desenvolvimento de um serviço da Web RESTful.
- Saber interpretar o desenvolvimento de um JAR executável.

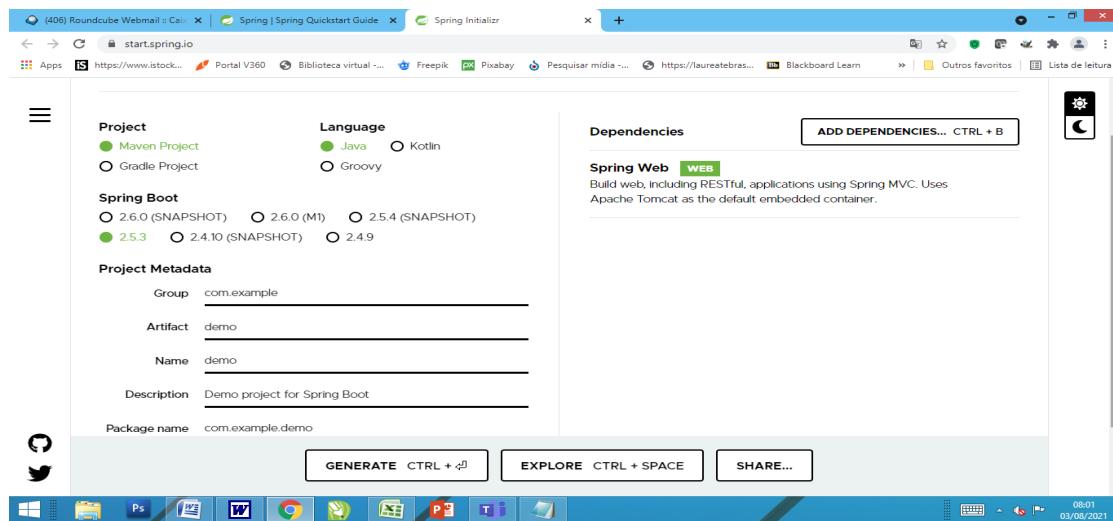


1. Framework Back End Spring Boot

Normalmente quando o Spring é utilizado para usar uma estrutura que possibilita o desacoplamento de uma aplicação, tornando a troca de implementações ou componentes, ou utilizando injeção de dependência. O *Spring* dispõe de plataforma de diversos componentes finalizados para serem utilizados principalmente para projetos web, por exemplo, o controle de acesso utilizando o *Spring* é mais simples e robusto. E eles são fatores de segurança, principalmente no momento de conexão de banco de dados. Segundo Leonard (2020), o *Spring Boot* facilita a criação de aplicativos autônomos baseados em *Spring* de nível de produção, no qual o desenvolvedor apenas se preocupa com a execução.

Para iniciar um novo projeto *Spring Boot* e desenvolver um novo projeto web, uma das maneiras é utilizar a plataforma disponível no start.spring.io. Depois na caixa de diálogo Dependencies (Dependências), clique na opção ADD DEPENDENCIES (adicionar a dependência) ou CTRL + B e escolha a opção *Spring Web*, conforme figura a seguir.

Figura 1 – Desenvolvendo um novo projeto no *Spring Boot*



Fonte: captura de tela Start Spring.

Depois de executados os procedimentos, clique no botão GENERATE (Gerar) ou a tecla de atalho CTRL + ENTER, que irá gerar um arquivo

compactado zip, basta apenas descompactar em um diretório. Os projetos desenvolvidos pela plataforma Start Spring já contêm o *Spring Boot*, com sua estrutura tornando-o pronto para funcionar dentro do aplicativo que está em desenvolvimento. Contudo, é importante salientar que ele não está totalmente pronto, podendo faltar várias codificações e configurações que são essenciais para o funcionamento. O *Spring Boot* é um dos modos mais rápido e fáceis de iniciar projetos *Spring*.

Após de executados os procedimentos iniciais, é recomendado utilizar e abrir o projeto em uma *Integrated Development Environment* (IDE), em português, Ambiente de Desenvolvimento Integrado, no qual é um software que dispõe de ferramentas e recursos de apoio ao desenvolvimento de aplicações. Dessa forma, eles otimizam esse processo, sendo indicado o uso da *Spring Tool Suite* (STS), pelo fato da fácil integração dos recursos. Caso esteja utilizando uma IDE, o próximo passo é localizar o arquivo denominado como *DemoApplication.java*, arquivo localizado no diretório *src/main/java/com/exemple/demofolder*. Em seguida, o próximo procedimento é executar alterações na codificação, adicionando, por exemplo, anotações e métodos como no código a seguir, no qual irá responder tema4 com a *string* de nome “Mundo”:

```
package com.example.demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.bind.annotation.RestController;
```

```
@SpringBootApplication  
  
@RestController  
  
public class DemoApplication {  
  
    public static void main(String[] args) {  
  
        SpringApplication.run(DemoApplication.class, args);  
  
    }  
  
    @GetMapping("/tema4")  
  
    public String tema4(@RequestParam(value = "name", defaultValue =  
"Mundo") String name) {  
  
        return String.format("Tema4 %s!", name);  
  
    }  
  
}
```

Observe que o tema4()método que foi adicionado foi projetado para receber um parâmetro *String* chamado *name* e, posteriormente, executar a combinação desse parâmetro com a palavra “tema4” na codificação. Diante disso, caso seja definido o nome como “beta” na solicitação, a resposta será “tema4 beta”. Já a @RestController anotação tem a função de informar ao *Spring* que essa codificação descreve um endpoint que deve ser disponibilizado na web. O @GetMapping(“/tema4”) informa ao *Spring* para usar tema4()método para responder as solicitações enviadas para o http://localhost:8080/tema4 endereço. Já o @RequestParam informa ao *Spring* para aguardar um namevalor na solicitação, mas se não estiver lá, ele usará a palavra “Mundo” por padrão.

Nesse sentido, é importante executar testes, por exemplo, de execução e construção de aplicação. Para isso, basta abrir uma linha de comando, terminal, como prompt de windows, e navegar até o diretório que foram salvos os arquivos do projeto. Veja a seguir exemplos de construção e execução do aplicativo por meio de envio dos seguintes comandos no MacOS / Linux e no Windows, respectivamente:

```
./mvnw spring-boot:run
```

```
mvnw spring-boot:run
```

Depois, basta abrir o browser e na barra de endereço digitar `http://localhost: 8080/tema4`, embora o *Spring* seja simples, ele também é bastante flexível.

1.1 Desenvolvimento de um serviço da Web RESTful

Java com o *Framework Back End Spring Boot* pode ser utilizado para o desenvolvimento de um *service web RESTful*, a fim de entender melhor REST modo de arquitetura, como diversas restrições que devem ser seguidas no processo de desenvolvimento de um serviço web (AMBLER; CLOUD; HAWKES, 2015). Assim, o RESTful nada mais é do que a API que está conforme com todas as restrições definidas por Roy (linguagem de programação que tem função de compilar para javascript). Veja um exemplo do processo de desenvolvimento e um serviço da web RESTful “tema4, Mundo” com *Spring*, no qual aceitará solicitações HTTP GET no local fictício em `http://localhost:9090/saudacoes`, que responderá uma representação JSON de uma saudação:

```
{"id":1,"content":"tema4, Mundo!"}
```

A saudação pode ser personalizada com um *name* parâmetro opcional na *string* de consulta, como apresentado na lista a seguir:

<http://localhost:9090/saudacoes?name=User>

O namevalor do parâmetro substitui o valor padrão de Mundo e é refletido na resposta, logo:

```
{"id":1,"content":"tema4, User!"}
```

Para iniciar o projeto, uma das opções é utilizar a plataforma disponível no link <https://start.spring.io> (acesso em: 28 set. 2021), no qual extrai todas as dependências de que são necessárias para um aplicativo e executa grande parte das configurações. Depois de acessar a plataforma, escolha Gradle ou Maven e a linguagem de programação Java, executando esse processo o próximo passo é clicar em Dependencies (Dependências) e selecionar *Spring Web*, depois em Generate (Gerar). Agora basta baixar o arquivo compactado .zip, nos quais terão os arquivos de um aplicativo da web configurado com as personalizações, caso o IDE que o desenvolvedor utilizar possua a integração Spring Initializr também pode ser utilizado diretamente no próprio IDE.

Após configurar o projeto e o sistema de construção é o momento de iniciar o serviço da web, assim, indica-se iniciar o processo planejando as interações de serviço, como o serviço tratará as GET solicitações de /saudacoes. Como opção com um *name* parâmetro na string de consulta, a GET solicitação deve retornar, por exemplo, uma 200 OK resposta com JSON no corpo que representa uma saudação, veja um exemplo de saída:

```
{
```

```
  "id": 1,  
  "content": "tema4, Mundo!"
```

```
}
```

Observe que O id campo é um identificador exclusivo da saudação e contém a representação textual da saudação, em que para modelar a representação da saudação, basta criar uma classe de representação de recursos (CORDEIRO 2016). Para executar esse procedimento, deve-se fornecer um objeto Java simples e antigo com campos, construtores e acessadores para os dados id e content, como: src/main/java/com/example/restservice/Saudacoes.java, no qual resultará na seguinte listagem:

```
package com.example.restservice;

public class Saudacoes {

    private final long id;

    private final String content;

    public Greeting(long id, String content) {

        this.id = id;

        this.content = content;

    }

    public long getId() {

        return id;

    }

    public String getContent() {

        return content;

    }

}
```

}

Esse aplicativo utiliza a biblioteca JSON para empacotar de maneira automática instâncias do tipo “Saudacoes” em JSON, é importante salientar que na abordagem do *Spring* para o desenvolvimento de serviços da Web RESTful, as solicitações HTTP são tratadas por um controlador. Esses componentes são identificados pela @RestController anotação e o SaudacoesController apresentado na listagem a seguir: src/main/java/com/example/restservice/SaudacoesController.java. Ela lida com GETs solicitações /saudações, retornando uma nova instância da “Saudacoes” classe, como apresentado a seguir:

```
package com.example.restservice;

import java.util.concurrent.atomic.AtomicLong;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.bind.annotation.RestController;

@RestController

public class SaudacoesController {

    private static final String template = "Tema4, %s!";

    private final AtomicLong counter = new AtomicLong();

    @GetMapping("/saudacoes")

        public Saudacoes saudacoes(@RequestParam(value = "name", defaultValue =
"World") String name) {
```

```
        return new Saudacoes(counter.incrementAndGet(), String.  
format(template, name));  
  
    }  
  
}
```

Observando fica fácil de identificar que esse controlador é conciso e simples, mas executa várias funções, como você poderá ver nos exemplos a seguir, para o melhor entendimento o @GetMapping anotação garante que as solicitações HTTP GET para /saudacoes sejam mapeadas para o saudacoes()método. Existem anotações complementares para outros verbos HTTP, como @PostMapping para POST) e, também, uma @RequestMapping anotação da qual todos derivam e podem servir como um sinônimo, por exemplo, @RequestMapping(method=GET)). Já o @RequestParam vincula o valor do parâmetro da *string* de consulta *name* ao *name* parâmetro do saudacoes()método, caso o *name* parâmetro estiver ausente na solicitação, o defaultValue de Mundo será usado.

A implementação do corpo do método cria e retorna um novo "Saudacoes" objeto com atributos id e com content base no próximo valor de counter e formata o dado *name*, usando a saudação padrão. É importante observar a diferença entre um controlador MVC tradicional e o controlador de serviço da web RESTful, esse é o modo como o corpo de resposta HTTP é criado. Em vez de depender de uma tecnologia de visualização para realizar a renderização do lado do servidor dos dados de saudação para HTML. Esse controlador de serviço da web RESTful preenche e retorna um Saudacoes objeto, os dados do objeto serão gravados diretamente na resposta HTTP como JSON.

Em relação ao código que utiliza a @RestController anotação *Spring*, que marca a classe como um controlador no qual cada método retorna um objeto de domínio em vez de uma visualização. Essa é uma

forma abreviada de incluir ambos @Controller e @ResponseBody e o `Saudacoes` objeto deve ser convertido em JSON. Devido ao suporte ao conversor de mensagem HTTP do *Spring*, o desenvolvedor não precisa executar essa conversão de maneira manual. Como Jackson 2 está no caminho de classe, o `Spring MappingJackson2HttpMessageConverter` será de forma automática escolhido para converter o `Saudacoes`, já o `@SpringBootApplication` é uma anotação de conveniência que adiciona todos os seguintes, como:

1. **@Configuration**: marca a classe como uma fonte de definições de *bean* para o contexto do aplicativo.
2. **@EnableAutoConfiguration**: informa ao *Spring Boot* para começar a adicionar *beans* com base nas configurações de *classpath*, outros *beans* e várias configurações de propriedade. Por exemplo, caso o `spring-webmvc` estiver no caminho de classe, essa anotação sinaliza o aplicativo como um aplicativo da web e ativa comportamentos-chave, como configurar um `DispatcherServlet`.
3. **@ComponentScan**: informa ao *Spring* para procurar outros componentes, configurações e serviços no `com/example` pacote, permitindo que encontre os controladores.

O `main()` método usa o método `Spring Boot SpringApplication.run()` para lançar um aplicativo, é importante observar que não havia uma única linha de XML e, também, não há nenhum `web.xml` arquivo, pois se trata de aplicativo da web 100% Java, no qual não é necessário configurar nenhum encanamento ou infraestrutura.

1.2 Desenvolvimento de um JAR executável

Para os profissionais que trabalham com Desenvolvimento *Web full stack* é importante o conhecimento de Java com o *Framework Back End Spring Boot*, no qual se pode executar os aplicativos a partir da linha de comando, por exemplo, com Gradle ou Maven. Mas também permite o

desenvolvimento de um único arquivo JAR executável, no qual contém todas as dependências, classes e recursos que são preciso para “rodar” a aplicação. Uma das vantagens de utilizar um JAR executável é simplificar o envio, a versão e a implantação do serviço como um aplicativo em todo o ciclo de vida de desenvolvimento, em diferentes ambientes e assim por diante. Caso o desenvolvedor utilize o Gradle, a execução do aplicativo pode ser por meio do ./gradlew bootRun. Outra possibilidade é o desenvolvimento do arquivo JAR usando ./gradlew build e, posteriormente, fazer a execução do arquivo JAR pelos comandos:

```
java -jar build / libs / gs-rest-service-0.1.0.jar
```

Caso o desenvolvedor utilize o Maven, é possível a execução do aplicativo utilizando ./mvnw spring-boot:run. Outra possibilidade é o desenvolvimento do arquivo JAR com ./mvnw clean package e, posteriormente, fazer a execução do arquivo JAR pelos seguintes comandos:

```
java -jar target / gs-rest-service-0.1.0.jar
```

As etapas exemplificadas desenvolvem um JAR executável, mas também podem ser desenvolvido um arquivo WAR clássico, assim a saída de registro é exibida e o serviço deve estar instalado e funcionando em alguns segundos. Em seguida, é importante executar teste para verificar se o serviço está funcionando, visite o endereço fictício http://localhost:9090/saudacoes, no qual deve resultar no seguinte código:

```
{"id":1,"content":"tema4, Mundo!"}
```

É importante indicar um *name* parâmetro de string de consulta visitando <http://localhost:9090/saudacoes?name=User>, veja que como o valor do *content* atributo muda de tema4, Mundo! para tema4, User!, como é apresentada na próxima lista:

```
{"id":1,"content":"tema4, Mundo!"}
```

Além disso, é possível a personalização da saudação com um *name* parâmetro opcional na *string* de consulta, como o seguinte exemplo:

`http://localhost:9090/saudacoes?name=User`

O *name* valor do parâmetro tem a função de substituir o valor padrão de Mundo e é refletido na resposta, como é apresentado na lista a seguir:

```
{"id":1,"content":"tema4, User!"}
```

Além disso, há a possibilidade de inicializar o projeto manualmente em vez de utilizar os links apresentados anteriormente, para isso, é necessário executar os seguintes procedimentos. Primeiramente, acesse o link: <https://start.spring.io> (acesso em: 28 set. 2021), para extrair todas as dependências necessárias para um aplicativo. Depois, você deve escolher Gradle ou Maven e, preferencialmente, a linguagem de programação Java. Executando esses procedimentos basta clicar em Dependencies (Dependências) e selecionar a opção denominada como *Spring Web* e, por último, clicar em Generate (Gerar) ou CTRL + ENTER. A próxima etapa é baixar o arquivo zip, nos quais possuem várias informações fundamentais de um aplicativo da web configurado conforme a personalização.

Caso o desenvolvedor possua IDE com integração com Spring Initializr, esse processo pode ser concluído a partir do próprio programa IDE, depois é importante a criação de uma classe de representação de recursos. Para finalizar, depois da configuração do projeto e do sistema de construção, bastar criar o serviço da web. Para isso, é importante começar o processo pensando nas interações de serviço, no qual o serviço irá tratar as GET solicitações de /ontente, de modo opcional com um *name* parâmetro na *string* de consulta. A GET solicitação deve fazer um retorno, por exemplo, de uma 200 OK resposta com JSON no corpo que representa uma saudação, conforme o exemplo a seguir de saída:

```
{  
    "id": 1,  
  
    "ontente": "tema4, Mundo!"  
}
```

Analisando a codificação, o id campo é um identificador exclusivo da saudação e content nada mais do que a representação textual da saudação e, para modelar a representação da saudação, é necessário criar uma classe de representação de recursos. Os procedimentos para executar essas ações, primeiramente, deve-se fornecer um objeto Java simples e antigo com campos, construtores e acessadores para os dados id e content, como src/main/java/com/example/restservice/saudacoes.java, no qual apresenta a seguinte listagem:

```
package com.example.restservice;  
  
public class Saudacoes {  
  
    private final long id;  
  
    private final String content;  
  
    public Saudacoes(long id, String content) {  
  
        this.id = id;  
  
        this.content = content;  
  
    }  
  
    public long getId() {  
  
        return id;  
    }
```

```
}

public String getContent() {

    return content;

}

}
```

Ao observar, é possível identificar que esse aplicativo usa a biblioteca JSON para empacotar de maneira automática as instâncias do tipo *Saudacoes* em JSON. Além disso, é possível converter um aplicativo *Spring Boot* JAR em um WAR por meio de dois plug-ins:

`spring-boot-gradle-plugin`

`spring-boot-maven-plugin`

Os dois são similares, fornecendo a capacidade de executar aplicativos *Spring Boot* a partir da linha de comando, bem como agrupar JARs executáveis, vários desenvolvedores desejam gerar arquivos WAR para serem implantados dentro de contêineres. Em tanto o `spring-boot-gradle-plugin` quanto o `spring-boot-maven-plugin` tem a funcionalidade de geração de WAR, basta apenas reconfigurar o projeto para produzir um arquivo WAR e declarar as dependências do contêiner integrado como “fornecidas”. Isso garante que as dependências do contêiner integrado relevantes não sejam incluídas no arquivo WAR.

É importante salientar que o *Spring Boot* fornece suporte ao *Spring Boot* no Gradle, o que permite que o desenvolvedor empacote arquivos executáveis JAR ou WAR, execute aplicativos *Spring Boot* e use o gerenciamento de dependências fornecido por `spring-boot-dependencies`. Para começar a usar o plugin, ele precisa ser aplicado ao

projeto, o plug-in é publicado no portal de plug-ins do Gradle e pode ser aplicado usando o plugins bloco, no Groovy:

```
plugins {  
    id 'org.springframework.boot' version '2.5.3'  
}
```

No Kotlin:

```
plugins {  
    id("org.springframework.boot") version "2.5.3"  
}
```

O que diferencia é que, no Groovy, o código inicia sem um parêntese e apenas uma aspa; já o Kotlin inicia com parênteses e duas aspas, há pequenas diferenças, mas que caso a codificação esteja errada o projeto ficará inviabilizado. Considerando tudo que foi exposto sobre o Java com o *Framework Back End Spring Boot*, essa é uma ótima ferramenta para os profissionais que *Desenvolvimento Web full stack*. Dessa maneira, é muito importante o conhecimento dos seus principais recursos.

Referências

AMBLER, T.; CLOUD, N.; HAWKES, R. A. **JavaScript Frameworks for Modern Web Dev.** Nova Iorque: Apress, 2015.

CORDEIRO, A. C. **Dyfocus:** Desenvolvimento do Back- End de um Aplicativo Mobile para Smartphone. 2016. Monografia (Graduação em Engenharia de Controle e Automação)–Universidade Federal de Santa Catarina, Florianópolis, 2016.

LEONARD, A. **Spring Boot Persistence Best Practices.** Nova Iorque: Apress, 2020.

STAR SPRING. Disponível em: start.spring.io. Acesso em: 28 set. 2021.



BONS ESTUDOS!