Workshop Para Empreendedores - 34ª Edição

A Linguagem JavaScript e seu Ecossistema

Jean Carlos Borba Guimarães da Costa, Maria Aparecida Reis Franca dos Santos
Universidade de Uberaba
guimaraes.jean95@gmail.com, maria.franca@uniube.br

Resumo

O presente artigo visa facilitar o entendimento da linguagem JavaScript, com foco nas bibliotecas e *frameworks*[1] utilizados em desenvolvimento *Front-End*.

Durante o seu desenvolvimento, foram abordados assuntos como a concepção da linguagem, bem como a motivação de sua criação. A evolução do JavaScript foi explorada, e também a padronização da linguagem ao ser submetida ao ECMA, passando ainda pelas principais aplicações no desenvolvimento *web* moderno. Ainda, foram listadas algumas das bibliotecas e *frameworks* JavaScript que são - ou foram - amplamente utilizados, porém mantendo o foco a dois principais atuais - Vue.js e Angular.

No decorrer deste artigo são apresentados conceitos aplicados à essas tecnologias, e paralelos foram traçados entre as mesmas, visando apresentar o ponto de vista do desenvolvedor ao utilizar tais ferramentas.

Por fim, foram desenvolvidas duas aplicações idênticas, utilizando-se das ferramentas citadas, como forma de exemplificar o conteúdo exposto no artigo.

Palavras-chave: ***1.*** *Desenvolvimento Web.* ***2.*** *JavaScript.* ***3.*** *Front-End.* ***4.*** *frameworks e bibliotecas.*

1 Introdução

Em 4 de Dezembro de 1995, Brendan Eich lançava a primeira versão do JavaScript, na época chamada de Mocha. Inicialmente concebida como parte integrante dos navegadores *web*, a linguagem foi projetada para executar *scripts* no *client-side*, ou seja, interagir diretamente com o usuário, sem passar por um servidor. Após mudar de nome novamente, desta vez para LiveScript, foi submetida ao *European Computer Manufacturers Association*, ou simplesmente ECMA. Por razões de patente, optou-se por nomear oficialmente a linguagem como ECMAScript. Apesar disso, o nome antigo já havia se popularizado. Dessa forma, é completamente normal encontrar ambos os nomes ao procurar informações sobre essa tecnologia. Atualmente na versão 9 (ou 2018), o ECMAScript passou a ter *releases*[2] anuais de uma nova versão. Todavia, grande parte dos navegadores suportam completamente a linguagem apenas até a versão 5, com exceção dos *browsers* modernos. Neste caso, existem diversas opções para se contornar a situação de falta de compatibilidade, como por exemplo os *polyfills*, ou utilizar-se do *Babel* durante o desenvolvimento. Essas técnicas e ferramentas *transpilam*[3] o

[1] Conjuntos de ferramentas criadas para facilitar ou acelerar o desenvolvimento em determinada linguagem

[2] Lançamento de uma nova versão, seja versão de desenvolvedor, beta e/ou estável.

[3] Em JavaScript, "*transpilar*" um código significa reescrevê-lo na sintaxe das versões anteriores da linguagem, entendida por navegadores antigos.

código das versões mais atuais para soluções equivalentes às suportadas em navegadores antigos.

Apesar de ter sido concebida inicialmente como uma forma de lidar com a interação do usuário com o navegador, o JavaScript foi, provavelmente, a linguagem mais explorada e modificada para atender a situações específicas. Por exemplo, temos o Node.js, um *fork*[4] do motor de renderização V8, criado pelo Google e utilizado até os dias de hoje no navegador *web* Google Chrome. Outro exemplo são ferramentas como CoffeeScript e TypeScript, sendo que a última se anuncia como "JavaScript que escala" (TYPESCRIPTLANG, 2019), uma referência à dificuldade que se tem ao utilizar JavaScript puro para desenvolver aplicações de grande porte. Além disso, a linguagem evoluiu tornando-se extremamente versátil, sendo possível utilizar os três principais paradigmas de programação: procedural, orientado a objetos e funcional.

Essa evolução possibilitou a criação de bibliotecas como jQuery, Prototype.js, underscore.js e React, que prometem tornar o desenvolvimento de uma aplicação muito mais produtivo e solucionar diversos problemas de compatibilidade entre navegadores e versões. Ao mesmo tempo, grandes empresas, como Google e Facebook, lançaram-se no mercado com os *frameworks* Angular e React, respectivamente. Com o respaldo de gigantes da tecnologia por trás de seu desenvolvimento, essas ferramentas se tornaram extremamente populares e amplamente utilizadas, por seguirem os padrões de desenvolvimento mais atuais e fazer uso de conceitos considerados cruciais durante o desenvolvimento de uma *Single-Page Application*[5]. Ao mesmo tempo, surge o Vue.js, com a premissa de ser muito flexível, podendo ser utilizado como biblioteca ou *framework*, e solucionar problemas existentes nas outras ferramentas. Essa propaganda, rapidamente, conquistou a comunidade de desenvolvedores de tal forma que uma campanha de *crowdfunding* foi criada e alcançada com sucesso para que o seu criador, Evan You, se dedicasse, exclusivamente, ao aprimoramento do *framework*.

Em paralelo à escrita deste artigo, foram elaboradas duas aplicações idênticas, uma delas desenvolvida em Angular, a outra em Vue.js, para que, ao final, fosse feito um estudo de caso, analisando os principais pontos que são objetos de discussão da comunidade de desenvolvedores, como facilidade de escrita de código, manutenibilidade e escalabilidade. Tais análises foram feitas do ponto de vista do desenvolvedor, utilizando-se, primariamente, de diversas comparações de código, visando mostrar os pontos positivos e negativos de cada tecnologia.

Ao mesmo tempo, uma *API*[6] única, desenvolvida em Node.js e Express.js, comunicando-se com um banco de dados não-relacional, neste caso, MongoDB, cuidará do tratamento e persistência de dados, comunicando-se com as duas aplicações, de forma individual ou simultaneamente.

[4] Réplica de um código existente em um servidor Git sobre a qual se desenvolve uma nova versão do projeto atual, sem comprometer o projeto original.

[5] As SPAs são aplicações *web* que não renderizam toda a página novamente, mas apenas os módulos necessários, apresentando comportamento parecido ao de um aplicativo para *smartphones*.

[6] As *Application Programming Interfaces* são definições de conjuntos de rotinas, protocolos e ferramentas preparadas para atender à criação de uma aplicação.

2 Materiais e Métodos

As metodologias utilizadas para a elaboração deste artigo podem ser divididas em teóricas, quanto aos métodos de pesquisa, fichamento de artigos e amplo material disponível em livros e na *Internet*, e práticas, quanto ao desenvolvimento das aplicações, utilizando-se das tecnologias escolhidas.

Para a primeira etapa, propôs-se aprofundar em conceitos e boas práticas das bibliotecas e *frameworks* JavaScript, trazendo à tona as definições de modularização, de componentização, de injeção de dependência, entre outros, presentes tanto no *Back-End* como no *Front-End*. Tais conceitos se fazem necessários para o desenvolvimento proposto no início do artigo.

Na segunda etapa, a abordagem escolhida foi a criação de uma aplicação voltada à organização pessoal, com o intuito de exemplificar os conceitos apresentados.

Para o *Back-End*, foi utilizado o Node.js, uma *engine*[7] JavaScript baseada no V8, criada pelo Google, e atualmente utilizada no Google Chrome, seu navegador. Para auxiliar a criação da *API*, foi escolhida a biblioteca Express.js, pela facilidade de sua utilização, além do amplo material disponível. A escolha visa facilitar o desenvolvimento e otimizar o tempo de escrita de código do *Back-End*, disponibilizando mais tempo para a criação das aplicações de *Front-End*, foco deste artigo.

Na etapa de *Front-end* optou-se por utilizar as tecnologias Vue.js e Angular, criando duas aplicações, idênticas, possibilitando traçar paralelos sobre a manutenibilidade, escalabilidade e aplicar na prática os conceitos citados anteriormente. O React, apesar de ser bastante famoso no mercado de desenvolvimento *web*, será tratado apenas teoricamente, por ter uma abordagem bastante diferente dos dois *frameworks* escolhidos.

A persistência de dados foi feita utilizando o banco de dados não relacional MongoDB, por sua capacidade de guardar as informações em formato JSON (*JavaScript Object Notation*, ou Notação de Objeto JavaScript, em tradução livre). Esta formatação de dados retira a necessidade de um *middleware*[8] para converter a informação requisitada no banco de dados para um formato aceito pelo Node.js e, consequentemente, pelo *Front-End*.

3 Resultados

APIs criadas utilizando Node.js utilizam-se, primariamente, da vantagem de as requisições serem processadas pelo *Event Loop*. A principal diferença deste método para outras linguagens como Java, C# e PHP, é que, nas linguagens tradicionais, a cada nova requisição, uma *thread*[9] é criada. Cada nova *thread* exige recursos computacionais, como processamento e memória *RAM*, o que pode se tornar um problema, já que este recurso é finito. O *Event Loop*, por sua vez, trata cada requisição como um evento, que será processado em fila (“*First In, First Out*”, ou “O Primeiro a Entrar é o Primeiro a Sair”, em tradução livre). Apesar disso, o modelo de execução do Node.js trata requisições concorrentes através de entradas e saídas não-bloqueantes, ou

[7] Também chamada de *Motor de Renderização*. Esta é a parte que cuida da interpretação do HTML e CSS em conteúdo visual, bem como da execução do JavaScript.

[8] Trecho de código normalmente utilizado para fazer a comunicação entre duas pontas muito distintas de um *software*, como, por exemplo, converter para JSON as informações de banco de dados relacional.

[9] Execução de um programa ou trecho de código de forma singular, encadeada ou simultânea.

seja, elas são assíncronas e não bloqueiam a *thread* única sobre a qual o *Event Loop* opera. Posto isso, espera-se da API criada como complemento para este artigo um tempo de resposta altamente eficaz, dada a simplicidade das requisições, embora sejam múltiplas.

Ainda no *Back-End*, a escolha de um banco de dados não-relacional para esse tipo de aplicação foi estratégica. O MongoDB armazena e retorna os dados em formato JSON, além de trafegar um volume de dados muito menor. Sendo assim, espera-se que o tempo de resposta entre Node.js e MongoDB sejam mínimos, melhorando a performance da aplicação e, consequentemente, a experiência do usuário.

Para a etapa de *Front-End*, alguns tópicos já são amplamente discutidos na comunidade de desenvolvedores. O primeiro é escalabilidade. Observa-se que, ao comparar as duas tecnologias com o React, existe um certo consenso ao afirmar-se que Angular e Vue.js são mais preparadas para aplicações de grande escala. Isso porque ambas contam com compilação do resultado final, otimização do tamanho de arquivos e *assets*[10] utilizados, além de contar com um grande ecossistema oficial. No caso do React, cria-se a necessidade de utilizar bibliotecas de terceiros, não oficiais, o que pode levar a problemas de compatibilidade e término repentino de suporte ao código externo. A arquitetura de *software* em uma aplicação React deve ser muito bem preparada e pensada para depender o mínimo possível de bibliotecas de terceiros e, caso dependa, que se analise o suporte destas.

[10] Todo e qualquer conteúdo extra apresentado no resultado final de uma aplicação. Ex: imagens, vídeos, arquivos de áudio, arquivos SVG, entre outros.

Quando se trata da aplicação dos conceitos modernos de desenvolvimento *web*, as tecnologias se equiparam. Todas são voltadas para a criação de SPAs, conforme apresentado anteriormente o seu funcionamento. Além disso, todas fazem uso do conceito de injeção de dependência. Cada uma à sua maneira exige que o código seja organizado em módulos, serviços, diretrizes, *pipes*, controladores, e que sejam importados - ou injetados - apenas no momento da utilização. Este é um dos principais conceitos por trás da otimização de performance, visto que não é necessário carregar todo o código de uma aplicação previamente à sua renderização, bastando que o *DOM*[11] seja construído no momento em que é solicitado, como por exemplo, quando um determinado elemento HTML responde a um evento de clique.

Outro ponto no qual se equiparam é a aplicação do conceito de reatividade. A definição curta de reatividade é a resposta imediata da camada de controle após a interação do usuário com a camada de visualização, podendo ou não se comunicar com a camada de modelo, onde são buscados os dados. Esse tipo de organização recebe o nome de MVC, ou *Model, View and Controller* (“Modelo, Visualização e Controlador”, em tradução livre). Neste ponto, todas as tecnologias trabalham com resposta imediata.

Em relação à facilidade de escrita de código e da manutenibilidade, espera-se que se sobressaiam React e Vue.js. Ambas foram pensadas e desenvolvidas para contrapor a quantidade de configurações, importações e

[11] O *Document Object Model* (“Modelo de Documento por Objetos”, em tradução livre) é a forma como o motor de renderização de um navegador organiza os elementos HTML da página exibida, e sobre eles aplica os efeitos do CSS e a dinamicidade do JavaScript.

dependências que uma aplicação Angular possui. Ainda sobre escrita de código, aplicações Angular exigem que se escreva o código em TypeScript, o que cria a necessidade de o desenvolvedor aprender uma nova linguagem, ainda que esta seja uma extensão do JavaScript. Em Vue.js também é possível utilizar TypeScript, porém, é opcional, e uma rápida pesquisa mostra que desenvolvedores não utilizam essa abordagem.

4 Discussão

Durante o desenvolvimento do projeto, uma complexidade maior era esperada com o Angular. Isso se confirmou pois, ao utilizar o *framework*, foi necessário fazer uso do TypeScript, uma extensão da linguagem JavaScript. Apesar de ser um *superset* da linguagem, o TypeScript tem suas peculiaridades, que vão desde a maneira como se importa módulos em outros arquivos, à orientação a objetos real, que complementa a tentativa de implementação existente no ECMA. Além disso, a quantidade de injeções de módulos, serviços e diretivas faz com que seja muito fácil esquecer de realizá-la, o que gasta um tempo considerável na correção.

Outro ponto interessante de se notar é a quantidade de arquivos por componente. O componente “list-item” (o qual define cada *task* na aplicação final) possui quatro arquivos, sendo eles um .html, onde fica o *template* base, no qual se aplicam as diretivas; um arquivo .css, que pode ser substituído por um pré-processador, onde aplicam-se os estilos referentes àquele componente; e dois arquivos .ts, sendo o primeiro aquele em que se localiza a lógica, a dinamicidade da aplicação, e o outro, onde se implementam os testes unitários.

Já a aplicação em Vue.js, seguindo a convenção do *framework*, diz que cada componente é composto de um arquivo .vue que, após a compilação, se traduz em .html, .css e .js, os quais o *browser* entende.

Um arquivo .vue é composto por três *tags* HTML, sendo a primeira a *tag* <template>, onde fica localizado o *layout* do componente; a segunda sendo a *tag* <style>, que pode conter ou não um atributo para *“scopar”*[12] o estilo CSS aplicado; e a terceira, a *tag* <script>, onde fica a lógica desenvolvida em JavaScript.

Essa organização do Vue.js se dá porque o nome da tecnologia é um trocadilho em inglês com a palavra *view*, que remete à camada de visualização, ou seja, com a camada de *layout*. Por ser desenvolvido todo dentro de arquivos com *tags* HTML, o nome dá sentido à forma em que os componentes se organizam.

É bastante notória a vantagem do Vue.js em relação à facilidade de escrita de código, pois uma vez que se importa num único lugar um componente, é possível utilizá-lo por toda a aplicação. Por outro lado, o Angular depende de importações nos próprios componentes e nos chamados módulos, que agrupam determinados componentes.

Essa “burocracia” criada em torno do Angular tem um ponto positivo no quesito escalabilidade. Por ter regras bastante definidas para utilização dos componentes, é impossível utilizar um componente em locais onde não era suposto utilizá-lo, uma vez que é necessário injetar seu código em cada parte em que seja necessário chamá-lo.

[12] Um estilo escopado se aplica somente ao componente onde o código CSS foi escrito, não se aplicando aos demais no restante da aplicação, mesmo que estes possuam o mesmo id ou classe.

O Vue.js, por sua vez, é bem mais flexível neste quesito. Por sua premissa de poder ser utilizado também como biblioteca, basta referenciar a instância de um módulo armazenada numa variável em qualquer ponto da aplicação, e os dados estão disponíveis. Para aplicações de grande porte, com inúmeras rotas, lógicas e regras de negócio, um arquiteto de *Front-End* é indispensável, pois pode definir regras rígidas sobre cada maneira de se utilizar o *framework*.

Em questão de velocidade, as duas se equiparam. O resultado final da compilação de ambos são arquivos HTML, CSS e JS, que podem ser *minificados*[13] com *plug-ins* específicos. Se forem bem feitos, o resultado de saída é idêntico, variando apenas em padrões pré-definidos de nomenclatura pós-compilação. No caso do projeto apresentado, não houve qualquer divergência nesse aspecto em relação às duas aplicações.

Com relação às ferramentas disponíveis por tecnologia, ambos empatam em questão de diretivas e modificadores de dados, que podem ser serviços, *pipes*, entre outros. Sobre as diretivas, as duas possuem as mesmas funcionalidades, variando apenas a forma como é inserida no DOM (com v- ou *ng), como por exemplo a diretiva condicional *if*, que no Vue.js se chama v-if e no Angular, *ngIf. No mais, cada *framework* possui as mesmas funcionalidades, porém definiu nomes diferentes para cada uma, bastando ler a documentação para descobrir qual precisa utilizar.

A manutenção é o tópico mais relativo dentre os escolhidos para análise, pois vai além de apenas escolher uma tecnologia. Um código bem organizado, bem escrito, orientado a domínio, que segue as convenções de desenvolvimento *Front-End* é facilmente modificado por qualquer programador, mesmo que seja escrito em uma linguagem relativamente desconhecida. Para citar como exemplo, o *kernel* do sistema operacional Linux foi desenvolvido durante anos por uma única pessoa, que convocou desenvolvedores para ajudá-lo posteriormente, saiu da liderança do projeto, este foi disponibilizado de forma aberta para sugestões da comunidade via *pull request*, e por fim, Linus Torvalds, seu criador, reassumiu a direção do projeto. Em todo esse tempo, e com todas as mudanças realizadas, o projeto se mantém estável e é base de diversas distribuições, batendo de frente com o BSD e o próprio Windows. Tendo isso em mente, basta que se siga as boas práticas de programação para que a manutenibilidade seja preservada.

5 Conclusão

Após a criação de duas aplicações diferentes, pôde-se observar a velocidade com que o universo do desenvolvimento *Front-End* caminha. Cada um dos *frameworks* estudados está em uma versão bastante avançada, o que prova que não é exclusividade de grandes empresas o desenvolvimento de novas tecnologias.

Foi possível provar a eficácia de um sistema independente como o Vue.js, bem como seu engajamento por parte da comunidade, visto que esta patrocina o projeto e seu criador continuamente para

[13] Um *plug-in* de minificação lê todos os arquivos antes de compilar, os une, remove todos os espaços, indentações e troca todos os nomes de variáveis e funções por outros menores, sempre visando gerar o menor tamanho possível de arquivo final, para reduzir o tráfego de dados.

que ele continue progredindo o *framework* e se torne cada vez mais completo.

Ainda, foi possível observar alguns pontos críticos, como manutenibilidade, escalabilidade, dentre outros, os quais um arquiteto deve definir previamente ao desenvolvimento de qualquer aplicação. Viu-se a superioridade do Angular para criação de aplicações de grande porte, o que não diminui o potencial da tecnologia Vue.js, mas é inversamente proporcional à qualidade técnica do arquiteto.

Em paralelo, alguns pontos foram comparados ao React, que é uma biblioteca com o mesmo intuito de trabalhar com SPA, porém não possui um ecossistema que garanta uma integração perfeita. Além disso, o React está migrando para a vertente do *mobile*, com um investimento cada vez maior no React Native, que une tecnologias que se comunicam nativamente com dispositivos Android e iOS.

Em todo o caso, viu-se que o desenvolvimento de uma aplicação SPA para *web* tem diversas possibilidades, com ferramentas que não se limitam apenas às estudadas, pois o mercado de *Front-End* em geral é extremamente completo e oferece possibilidades para todos os tipos de profissionais.

6 Referências

CECHINEL, Alexandre. **Avaliação do framework Angular e das bibliotecas React e Knockout para o desenvolvimento do Frontend de aplicações Web**. 2017. 77 p. Trabalho de Conclusão de Curso – Universidade Federal de Santa Catarina, Santa Catarina, 2017.

LENON. **Node.js**: o que é, como funciona e quais as vantagens. 2018. Disponível em: https://www.opus-software.com.br/node-js/. Acesso em: 03 jun. 2019.

LUIZ, Andrey. **JavaScript #1**: uma breve história da linguagem. 2016. Disponível em: http://shipit.resultadosdigitais.com.br/blog/javascript-1-uma-breve-historia-da-linguagem/. Acesso em: 15 maio 2019.

MDN WEB DOCS. **JavaScript**. Disponível em: https://developer.mozilla.org/bm/docs/Web/JavaScript. Acesso em: 09 mar. 2019.

MDN WEB DOCS. **New In JavaScript**. Disponível em: https://developer.mozilla.org/en-US/docs/Web/JavaScript/New_in_JavaScript. Acesso em: 12 mar. 2019.

SCHNEIDER, Adolfo Henrique. **Desenvolvimento web com Client Side Rendering: combinando Single Page Application e serviços de backend**. 2016. 49 p. Monografia – Universidade Federal do Rio Grande do Sul, Porto Alegre, 2016.

SIMPSON, K. **You Don't Know JS**: Async & Performance. 1 ed. Newton, Massachusetts, EUA: Editora O'Reilly Media, 2014.

SIMPSON, K. **You Don't Know JS**: ES6 & Beyond. 1 ed. Newton, Massachusetts, EUA: Editora O'Reilly Media, 2014.

SIMPSON, K. **You Don't Know JS**: Scopes & Closures. 1 ed. Newton, Massachusetts, EUA: Editora O'Reilly Media, 2014.

SIMPSON, K. **You Don't Know JS**: this & Object Prototypes. 1 ed. Newton,

Massachusetts, EUA: Editora O'Reilly Media, 2014.

SIMPSON, K. **You Don't Know JS**: Types & Grammar. 1 ed. Newton, Massachusetts, EUA: Editora O'Reilly Media, 2014.

SIMPSON, K. **You Don't Know JS**: Up & Going. 1 ed. Newton, Massachusetts, EUA: Editora O'Reilly Media, 2014.