UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
UNIDADE ACADÊMICA DE SISTEMAS E COMPUTAÇÃO
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

ARTHUR SAMPAIO PERICO CORREIA

JAVASCRIPT ACADEMY – UM ARCABOUÇO PARA LIÇÕES EM JAVASCRIPT

CAMPINA GRANDE - PB

2019

ARTHUR SAMPAIO PERICO CORREIA

JAVASCRIPT ACADEMY – UM ARCABOUÇO PARA LIÇÕES EM JAVASCRIPT

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Professor Dr. Dalton Dario Serey Guerrero.

CAMPINA GRANDE - PB

2019

C824j Correia, Arthur Sampaio Perico.
JavaScript Academy – um arcabouço para lições em Javascript. / Arthur Sampaio Perico Correia. – 2019.

10 f.

Orientador: Prof. Dr. Dalton Dario Serey Guerrero.
Trabalho de Conclusão de Curso - Artigo (Curso de Bacharelado em Ciência da Computação) - Universidade Federal de Campina Grande; Centro de Engenharia Elétrica e Informática.

1. Ensino de javascript. 2. Engenharia de Software. 3. Desenvolvimento de web. 4. Javascript. I. Guerrero, Dalton Dario Serey. II. Título.

CDU:004(045)

Elaboração da Ficha Catalográfica:

Johnny Rodrigues Barbosa
Bibliotecário-Documentalista
CRB-15/626

ARTHUR SAMPAIO PERICO CORREIA

JAVASCRIPT ACADEMY – UM ARCABOUÇO PARA LIÇÕES EM JAVASCRIPT

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA:

Professor Dr. Dalton Dario Serey Guerrero
Orientador – UASC/CEEI/UFCG

Professor Dr. João Arthur Brunet Monteiro
Examinador – UASC/CEEI/UFCG

Professor Dr. Tiago Lima Massoni
Examinador – UASC/CEEI/UFCG

Trabalho aprovado em: 02 de julho 2019.

CAMPINA GRANDE - PB

JavaScript Academy - Um arcabouço para lições em Javascript

Arthur Sampaio Perico Correia
arthur.sampaio.correia@ccc.ufcg.edu.br
Universidade Federal de Campina Grande
Campina Grande, Paraíba

RESUMO

Muitos são os agentes que ensinam funcionalidades e conceitos da linguagem de programação Javascript, mas poucos são os ambientes pedagógicos que permitem suas práticas. Isto torna a experiência de ensino incompleta, além de passar uma falsa sensação de conhecimento para quem aprende. Profissionais com uma formação incompleta tende a introduzir *bad smells* em códigos de produção, causando muitas vezes, consequências negativas. Nesta perspectiva, proporcionar um ambiente que permite a prática do ensino é, a médio e longo prazo, uma ótima alternativa para melhorar a qualidade dos códigos em geral e consequentemente dos produtos. Diante disto, foi projetado e desenvolvido o *JavaScript Academy*. Um sistema web em que é possível criar, compartilhar e corrigir de maneira autônoma listas de exercícios bem como obter métricas das respostas. Testes de satisfação mostraram que os usuários testados se sentiram satisfeitos e que utilizariam o sistema para ensinar ou aprender Javascript. Os mesmos testes também evidenciaram pontos a serem melhorados, como as mensagens de erro e feedback. Por fim, os mesmos testes também apontaram que permitir a manipulação de HTML seria um avanço.

Palavras chaves: ensino de javascript, engenharia de software, desenvolvimento web.

ACM Reference Format:
Arthur Sampaio Perico Correia. 2019. JavaScript Academy - Um arcabouço para lições em Javascript. In *Trabalho de Conclusão de Curso do curso de Computação, Junho, 2019, Campina Grande, PB.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUÇÃO

Os desenvolvedores de sistemas web são os pioneiros da era da Internet. Tanto os sites que navegamos quanto as figurinhas que compartilhamos no Whatsapp, passando pelos vídeos no Youtube, só são possíveis graças aos profissionais que projetam, implementam e constróem tais aplicações.

O uso e penetração da Internet no mundo está crescendo rapidamente. Segundo Julia Murphy e Max Roser em sua pesquisa intitulada "Internet"[8], entre 2010 e 2016, o crescimento da quantidade mundial de usuários cresceu 71%, enquanto que no mesmo período a população mundial cresceu 7%. Assim, é evidente que uma das consequências deste aumento vertiginoso é a exigência da oferta de mais conteúdos. Logo, a procura por profissionais de desenvolvimento web cresce de modo análogo.

Esta afirmação pode ser observada nos dados do Escritório de Estatística do Trabalho norte-americano, entre 2016 e 2026, a previsão de crescimento do número de postos de trabalho em desenvolvimento web será de 15% (contra os 7% da média das ocupações) e com o dobro do salário anual, da média dos postos de trabalho [9]. Isso tudo torna o ambiento fértil para o pleno emprego.

Porém, construir uma aplicação web não é trivial. É necessário, além de apenas resolver o problema proposto, que a solução contenha uma mistura de conceitos e valores específicos que vão de design visual, acessibilidade, internacionalização até a otimização de mecanismos de busca. Tais necessidades elevam a complexidade do projeto em relação a outro projeto computacional comum.

Javascript é uma linguagem poderosa que tem ganhado notoriedade pelo seu uso constante em aplicações web. Hoje, a linguagem é a que possui mais repositórios ativos no Github[6]. Javascript é fracamente e dinamicamente tipada e possui funções como cidadãs de primeira ordem. É uma linguagem orientada a objetos livre de classe que utiliza herança prototipal ao invés de herança clássica. Isto é, objetos em Javascript herdam propriedades de outros objetos diretamente e estas propriedades herdades podem ser alteradas em tempo de execução. Além disto, pelo *Javascript* ser uma linguagem interpretada, faz com que ela não possua um compilador para auxiliar os programadores a evitarem erros. Por estas características sistemas feitos em Javascript tendem a possuir muitos *bad smells*, que são más decisões de design que pode impactar negativamente a qualidade de uma aplicação. Por isso, desenvolvedores web que não possuem um profundo conhecimento da linguagem tendem a ter dificuldades em escrever códigos manuteníveis [5].

A projeção do aumento do número de postos de trabalho — que resulta num maior número de profissionais sem experiência — aliado com a dificuldade inerente da área pode ser ambiente perfeito para a introdução de *bad smells* em diversas aplicações.

Um típico *bad smell* é o uso de *Callback Aninhado*[10]. Um *Callback* é uma função passada como parâmetro para outra função. Desenvolvedores utilizam funções *callbacks* para atividades assincronas, isto é que precisam do resultado de outra função para serem executadas. Um *Callback Aninhando*, como o próprio nome diz, é o uso de múltiplos callbacks para resolver uma atividade. Entretanto, o uso de múltiplos *callbacks* pode resultar num código difícil de ler e manter. Um típico exemplo desse *bad smell* pode ser visto abaixo.

UFCG '19, Junho, Campina Grande, PB

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

```
db.getUserByEmail({email: "usuario@exemplo.com"},
     function (user) {
    instagramAPI.getPhotos({handle: user.instagram},
         function (photos){
        sendPhotosByEmail(photos, function(response){
            if(response){
                console.log("Email enviado com sucesso")
            }
        })
    })
})
```

Listagem 1: Exemplo de CallBack Aninhando

Um possível refatoramente deste trecho de código é utilizar um encadeamento de *Promises* [2] e/ou utilizar as novas features do EcmaScript 7 (ES7) como o *async/await* [1].

```

db.getUserByEmail({email: "usuario@exemplo.com"})
    .then(function(user){
        return instagramAPI.getPhotos({handle: user.
            instagram})
    })
    .then(function (photos) {
        return sendPhotosByEmail(photos)
    })
    .then(function (response) {
        if(response){
            console.log("Email enviado com sucesso")
        }
    })
```

Listagem 2: Uso de Promises para resolver tarefas assíncronas

É notório o aumento da qualidade entre o trecho de código da Listagem 1 e 2. No primeiro trecho de código, há cinco níveis de indentação contra apenas dois níveis do segundo trecho. Muitos níveis de identação significa que há mais escopos aninhados e logo mais compexidade dificultando a leitura e consequentemente a manutenabilidade do código [10].

Muitos são os agentes - artigos e cursos - que oferecem a oportunidade do desenvolvedor estudar e assim mitigar as lacunas do conhecimento. Este trabalho tem o objetivo de servir como suporte a estes agentes.

Semelhante a matemática, programar exige prática. A falta de um ambiente interativo para por em prática os conhecimentos passados por estes agentes tráz três problemas em que raramente põe-se energia para resolvê-los:

(1) O próprio desenvolvedor que está estudando tem que organizar seu *workstation*.
(2) A falta de indicação de exercícios para fixação dos conhecimentos; e uma vez propostos
(3) A falta de um corretor para avaliar o desempenho do aprendizado.

Para melhorar este cenário, os agentes descritos acima, utilizam uma coleção, já existente, de ferramentas web que servem como ambiente dinâmico para o ensino de Javascript. Dentre as ferramentas disponíveis no mercado, destacam-se o Repl.it, JSFiddle e o ideone. O primeiro problema destacado é solucionado, porém, os problemas restantes não.

Para atacar o segundo ponto levantado, é comum que os agentes utilizem alguma dessas ferramentas acima com um template de exercício já adicionado para facilitar o aprendizado. Essa abordagem tráz outro problema: se houver mais de um exercício será necessário o gerenciamento destes, uma vez que, cada exercício tem uma URL associada.

O terceiro problema é aquele em que menos se põe energia para solucioná-lo. Se já é dificil haver material com indicações de exercícios, tê-los ainda com corretores online é mais.

Diante disto, o objetivo deste trabalho é criar uma sistema web em que sirva como um ambiente de estudo de programação para JavaScript. Ou seja, um arcabouço em que seja simples inserir questões de programação bem como a possibilidade de corrigi-las. Espera-se, que os produtores de conhecimento voltados a Javascript utilizem o *Javascript Academy* como forma de propor exercícios variados e de compartilhá-los de maneira simples através de uma URL.

2 ARQUITETURA E PROJETO DA SOLUÇÃO

2.1 Visão Geral da Solução

O *Javascript Academy* é uma ferramenta web que tem por objetivo servir de suporte para produtores de conteúdo ensinarem Javascript de maneira descentralizada e dinâmica. A plataforma permite que um usuário qualquer possa adicionar exercícios em um repositório e daí criar e compartilhar Lições[1], bem como obter informações sobre as respostas.

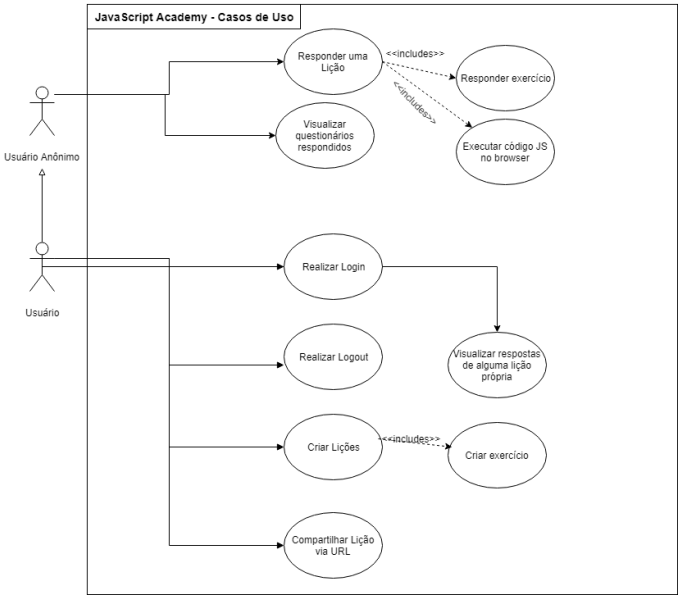

Figura 1: Diagrama de Caso de Uso

Um grande diferencial do *Javascript Academy* é a possibilidade de um usuário anônimo interagir com o sistema. Um usuário anônimo é aquele que não precisa realizar login. Mesmo sem possuir credenciais, este usuário vai poder responder e visualizar lições bem como enviar para o servidor as métricas das respostas.

Outras ações podem ser observadas na Figura 1 que representa os casos de uso do sistema.

[1]Chamamos de lição um agrupamento de exercícios

2.1.1 Dados recolhidos. No início desta seção, foi dito que recolheríamos dados durante a solução da lição e os compartilharíamos com o criador da mesma. Esta funcionalidade tem o objetivo de servir como incentivo para que produtores adotem nossa plataforma. Estes dados podem ser utilizados tanto para pesquisas acadêmicas quanto para avaliar um possível candidato em uma seleção de empregos. A seguir são descritos os dados recolhidos.

- $Tempo\ Total(l)$: A duração total, em milissegundos, para a resolução dos exercícios da *lição* l;
- $Numero\ de\ visualizacoes(l)$: Inteiro que identifica a quantidade de vezes que a lição l foi visualizada. Ou seja, diz quantas vezes tentaram responder a lição l;
- $Numero\ de\ respostas(l)$: Inteiro que identifica a quantidade de vezes que a lição l foi respondida;
- $Codigo\ Submetido(e, l)$: Código submetido do exercício e da lição l;
- $Tentativas(e, l)$: Indica a quantidade de tentativas de resolver o exercício e da lição l;
- $Tempo(e, l)$: Indica a quantidade de tempo, em milissegundos, para resolver o exercício e da lição l;

2.2 Tecnologias Utilizadas

Como o *Javascript Academy* se trata de um sistema web foi escolhido o conceito de *Arquitetura REST* para modelar o sistema. E como é de praxe, o sistema foi dividido em duas camadas principais: o *Frontend* e o *Backend*.

No *Frontend* foi escolhido o framework *React Hooks* com o apoio do *Material-UI* - biblioteca de componentes em React largamente utilizada pela comunidade. A adoção do *React* se deu pela afinidade do autor e pela maturação do framework que já está na versão de número 16.6.0.

Como o *Javascript Academy* é um ambiente de estudo de programação, é exigido pouco cálculo computacional porém, o gargalo acontece nas operações de I/O. O *NodeJS* se comporta muito bem neste cenário, graças ao seu *Event Loop* que não bloqueia execução e permite que outras *requests* sejam enviadas enquanto se espera pelo resultado da *request* inicial. Logo foi escolhido como tecnologias do Backend a stack de *NodeJS*, *MongoDB* e *Express.js*.

2.3 Principais Decisões Arquiteturais

Esta é a seção a mais importante deste documento. Por isso, para facilitar a leitura, a abordagem das principais decisões arquiteturais será feita a partir de perguntas.

2.3.1 Onde o código é avaliado? Executar um código externo e não-validado é uma tarefa perigosa para qualquer organização por causa de um motivo principal: *injeção de código*. Um código mal intencionado executado no servidor pode trazer grandes prejuízos financeiros, por exemplo: um simples *while(true)* poderia fazer o processo ser executado indefinitavamente elevando a mensalidade da *cloud*. Assim, aproveitando que a linguagem é Javascript e o usuário estará executando o sistema necessariamente num browser, foi decidido avaliar o código no lado do usuário com suas próprias permissões.

2.3.2 Como o código é avaliado? Para executar códigos *Javascript* é utilizado uma função nativa da linguagem chamada *eval()*. Esta função é o centro de grandes discussões em diversos fóruns da comunidade de *Javascript* e quando o cenário é a utilização dela no lado do cliente dois pontos importantes são levantados.

O primeiro, é a *injeção de código* que foi comentado anteriormente. Mesmo isolando o servidor de códigos maliciosos ainda é possível ocorrer brechas de segurança no browser. Segundo a OWASP (*The Open Web Application Security Project*), no seu último relatório [12], foi mostrado que ataques via XSS (Cross-site scripting) ainda estão estão no Top 10 das maiores vulnerabilidades de segurança. A essência de um ataque via XSS é a injeção de scripts no cliente por alguém não autorizado para executá-los se passando pelo cliente. Porém, no caso do *JavaScript Academy* o código avaliado seria o que o usuário inseriu na mesma página, logo, não é possível ocorrer nenhum dano.

Já o segundo, diz respeito ao desempenho do *eval()*. Uma grande desvantagem da função é o esforço computacional desgantante que a chamada faz para procurar as variáveis utilizadas a partir do escopo local até o global. Porém, existe uma ótima alternativa para executar códigos *Javascript*: *Function*. Como funções são cidadãs de primeira-ordem da linguagem, é possível passá-las como argumento e recebê-las como retorno.

Assim, basta passar como parâmetro do construtor de *Function* a string referente ao código de uma função a ser executada e receber como retorno a função correspondente aquela string. A grande diferença é que enquanto um utiliza todo o espaço amostral de variáveis possíveis a serem resolvidas, o outro utiliza apenas as variáveis que estão declaradas no escopo global.

Abaixo é possível notar como isto ocorre.

```
1  function evaluateCode(code) {
2      return eval(`(${code})`)
3  }
4
5  const input = 'function triplo(arr){ arr.map(i => i*3)}'
6
7  const aux = evaluateCode(input)
8  console.log(aux([1,2,3]))
```

Listagem 3: Com *eval()*

No trecho de código acima a função *evaluateCode* retorna a avaliação do código passado por parâmetro[2]. A seguir no trecho abaixo, é possível notar que o código a ser avaliado é concatenado com o retorno de um escopo com modo estrito - o que permite utilizar as melhorias de perfomance das engines - e passado como parâmetro do construtor de *Function*. Porém, a alternativa do uso do construtor de *Function* só é válido se o código a ser executado é uma função.

```
1  function evaluateCode(code) {
2      return Function('"use strict";return (' + code + ')')
           ()
3  }
4
5  const input = 'function triplo(arr){ arr.map(i => i*3)}'
6  const aux = evaluateCode(input)
7  console.log(aux([1,2,3]))
```

Listagem 4: Sem *eval()*

[2]O sufixo "("e prefixo ")"servem para indicar que *eval()* vai avaliar a string como uma função

O objetivo do *JavaScript Academy* é proporcionar ao usuário um ambiente dinâmico para estudar Javascript e isso não pode ser alcançado tendo apenas a possibilidade de executar funções. É necessário também executar trechos(*snippets*) de código. Assim, para atingir este objetivo, a função *eval* pode ser invocada indiretamente, proporcionando, que o código gerado será acessível apenas para as variáveis globais.

```
1 function test() {
2   var x = 2, y = 4;
3   console.log(eval('x + y'));  // Chamada direta, utiliza
        o escopo local, resultado    6
4   var geval = eval; // equivalente a chamar o eval a
        partir do escopo global
5   console.log(geval('x + y')); // Chamada indireta,
        utiliza o escopo global, e ocorre um erro, pois 'x
        ' n o    definido no escopo global.
6 }
```

Listagem 5: *eval()* declarado no escopo global

Essa abordagem apresenta um problema: apenas o resultado da última expressão do trecho de código é retornada pela função *geval*. Contudo, esta é uma limitação técnica em que se acredita não haver grandes empecilhos, uma vez que, essa avaliação só é utilizada para facilitar a codificação do usuário.

2.3.3 Como os testes são executados? Assim como a avaliação de código, a execução dos testes para corrigir um determinado exercício também é realizado no cliente. Como a solução de uma lição é de forma incremental, um exercício após o outro, o envio para o servidor das métricas colhidas durante a sessão só ocorre quando todos os exercícios forem respondidos.

Como essa é uma funcionalidade chave para o funcionamento do *Javascript Academy* é exibido na Figura **??** o diagrama de sequência referente a correção dos exercícios.

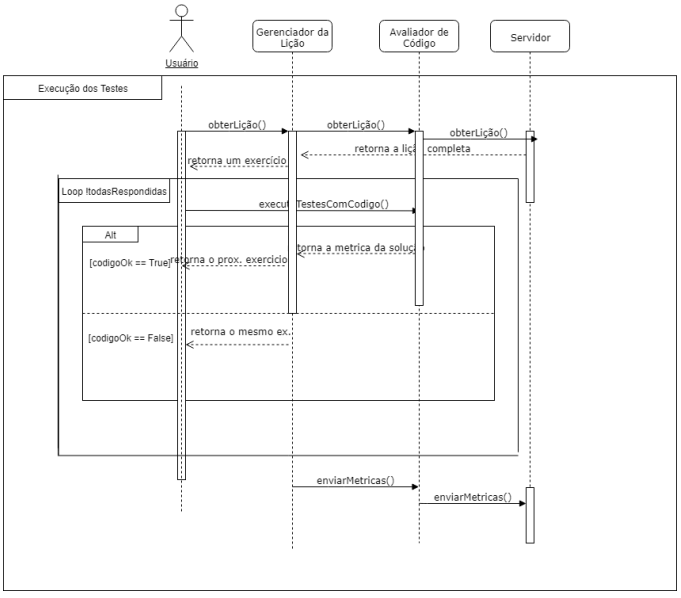

Figura 2: Diagrama de Sequência para a correção dos exercícios

2.3.4 Como foi realizada a estratégia para usuários anônimos? Todo o fluxo desta funcionalidade acontece depois da última ação no Diagrama de Sequência da Figura **??** e se baseia numa estratégia bem similar ao padrão de autenticação JWT. Para facilitar o entendimento vamos chamar a requisição que envia as métricas para o servidor de *R*.

No servidor, quando *R* chega ela é interceptada pelo *Middleware* responsável pela autenticação da sessão e seu cabeçalho é inspecionado. Caso não seja encontrado o cabeçalho de autorização do JWT e o cabeçalho referente a um usuário anônimo, é instanciado uma entidade *U* do tipo de *Usuário Anônimo* e *U* se torna o usuário da sessão, e todo o fluxo é realizado sem nenhuma alteração em relação a um usuário cadastrado.

Ao final da computação[3], é associado a um cabeçalho da resposta o *id* do usuário *U* gerado anteriormente. Quando a resposta chega no lado do cliente é verificado este cabeçalho, e caso presente, o valor do mesmo é guardado no *LocalStorage*.

No interceptor das requisições do cliente, ou seja no *Frontend*, é verificado a presença deste valor no *LocalStorage* e se presente o cabeçalho de autorização de usuário anônimo é adicionado antes de ser enviado para o servidor.

2.4 Metodologia de Avaliação de Código

O componente mais importante do *Javascript Academy* é sua engenharia de avaliação de código. Isto compreende a execução dos códigos submetidos sobre uma bateria de testes. Para cada teste são necessárias três passos:

(1) Primeiro, é o mais simples. O código é posto pra ser executado em um interpretador apropriado que no nosso caso é a engine do browser;
(2) Segundo, é verificado se o interpretador não retornou nenhum tipo de erro;
(3) E por fim é verificado se não houve um estouro de tempo ou memória na execução do teste

Normalmente, a primeira fase ocorre de maneira simples e não há muito como evitar sua execução já que nasce da vontade do usuário. As fases seguintes, 2 e 3, são mais complexas e detalhadas e será abordado indiretamente ao longo do texto.

A fase de correção dos testes, que compreende os passos acima, são as mais caras computacionalmente. Cada teste pode levar alguns minutos e, no pior casos, pode ocasionar *overflow* na memória do browser. Abaixo algumas definições importantes para o funcionamento do *Javascript Academy*. Importante frisar que foi tomado como referencial teórico o estudo de Wasik[11].

É importante para este trabalho a definição de uma instância de teste. Seja o alfabeto $\sum$ utilizado para montar as palavras do input e output dos dados, consistindo de letras, digitos e caracteres especiais, ou seja, $\sum = [aA - zA, 0 - 8, , ...]$. A instancia de teste $t_i \in T$, onde T é o conjunto de testes do problema, t_i é definido como: $t_i = (i_i, o_i, p)$, onde i é o input, o o output e p são as restrições para execução do teste que não pode ser excedidos durante sua execução, como tempo máximo de execução e maximo uso da memória, p é definido internamente no contexto do *Javascript Academy*

O output da execução da saída da bateria de testes é a solução S que, no caso desse sistema, só assume dois valores:

[3] i.e. quando a métrica da lição é associada ao usuário e demais entidades

- Aceito: quando nenhuma das execuções dos testes falha e nenhuma restrição p_i é quebrada
- Resposta errada: quando algum teste é falho ou uma das restrições é quebrada

Normalmente juizes onlines possuem um conjunto maior de valores da solução, como *Memória ou Tempo Excedidos* e uma pontuação atrelada a correção. Como este trabalho possui um escopo curto foi decidido abstrair essas funcionalidades de um juiz online.

3 SISTEMA EM USO

3.1 Sondagem da Satisfação do Usuário

O principal objetivo do *JavaScript Academy* é se tornar um ótimo ambiente de estudos para Javascript. Porém, para isto ocorrer além de executar as suas funcionalidades é necessário que o sistema agrade ao usuário, caso contrário irá frustrá-lo no uso e consequentemente o mesmo deixará de utilizar o sistema. Para evitar isto, é necessário analisar a satisfação do usuário com o intuito de recolher dados e assim melhorar os pontos defasados. Ou seja, a satisfação do usuário com o produto deve nortear o seu desenvolvimento.

Contudo, a avaliação do usuário por si só é algo muito subjetivo e não assegura a usabilidade do sistema. Para resolver esta subjetividade é utilizado metas quantitativas de usabilidade para direcionar a avaliação. Assim, focamos na facilidade de uso do sistema como foco da avaliação de satisfação subjetiva do usuário.

Muitas são as técnicas utilizadas para realizar esta sondagem, dentre as quais tornou-se muito aderido o uso de questionários. Questionários funcionam como testes que propiciam um perfil do qual se extrai valores quantitativos e qualitativos que permitem avaliar a satisfação do usuário. É recomendado a utilização de questionários padronizados, pois permitem a comparação de resultado obtidos em diferentes sistemas, além disto, estes questionários devem apresentar opções de respostas fechadas, que é o que permite a produção de dados quantitativos e objetivos.

Muitos são os questionários desenvolvidos pela comunidade para a mensuração da usabilidade. Alguns exemplos de questionário são: SUMI (Usability Measurement Inventory), WAMMI (Website Analysis Measurement Inventory), QUIS (Questionnaire for User Interaction Satisfaction), ASQ (After-Scenario Questionnaire), PSSUQ (Post-Study System Usability Questionnaire) e CSUQ (Computer System Usability Questionnaire).

Como o principal objetivo é entender quais aspectos do sistema o usuário mais se frusta, foi escolhido o PSSUQ. Pois, o mesmo avalia características como facilidade de uso e de aprendizado, simplicidade, informação e interface com o usuário. Estes pontos compõe a avaliação global do sistema. O questionário possui 18 itens, e a cada item do questionário foi associado uma escala somativa de 7 pontos, delimitada inferiormente por *concordo fortemente* e superiormente por *discordo fortemente*. Foi utilizado o questionário padrão desenvolvido pela IBM [7].

Para realizar a avaliação do *JavaScript Academy* por meio do questionário PSSUQ, foram avaliados 4 alunos de graduação do curso de Ciência da Computação da Universidade Federal de Campina Grande, com diferentes graus de experiência em Javascript. Tais alunos receberam informações sobre o sistema e seguiram um mesmo conjunto de passos que englobam todas as funcionalidades do sistema.

Os passos realizados para responder os questionários, foram:

(1) Criar uma conta a partir da pagina de registro;
(2) Realizar login com suas credenciais recém feitas;
(3) Responder uma das listas compartilhadas via URL;
(4) Enviar respostas da lista de exercício;
(5) Ir para a página de criar lições;
(6) Adicionar questões previamentes salvas na lição;
(7) Criar e adicionar um exercício na lição atual;
(8) Criar e compartilhar a lição[4] ;
(9) Visualizar a lição criada e, se houver, as métricas de resposta para aquela lição.

Após o uso do sistema, foi passado um link com o questionário onde os participantes responderam de forma online. É importante frisar que durante o processo de respostas o pesquisador se afastou do usuário com o intuito de diminuir qualquer enviesamento. Abaixo é possível observar as médias das respostas como um gráfico de barras. A avaliação da satisfação do usuário foi obtida de acordo o valor da média de cada item que varia de 1 a 4.5.

Segundo a escala Lickert, essa variação mostra que a sondagem não foi um sucesso completo. Pode-se destacar que os pontos das questões *As mensagens de erros do sistema foram claras o suficiente para me ajudar na correção de erros*, *As informações (como ajuda online, na tela de mensagens e outros documentos) fornecidas com este sistema foram claras*, *Sempre que eu cometi algum erro, eu pude recuperar de forma fácil e rápida* e *Este sistema tem todas as funções e capacidades que eu esperava que ele tivesse* foram as piores avaliadas. Desta maneira, a avaliação desses itens propiciaram a análise de que o sistema de fato precisa melhorar suas mensagens de erros para que ajudem os usuários a entender os problemas ocorridos bem como possam ser solucionados.

É importante frisar que no item *Este sistema tem todas as funções e capacidades que eu esperava que ele tivesse* os usuários de testes comentaram que esperavam haver um ambiente em que fosse possível manipular HTML e Javascript. Contudo, esta funcionalidade não faz parte do escopo deste projeto e por isso não pode ser interpretada como um aspecto negativo do teste.

Os pontos melhores avaliados são os dos itens *A organização das informações nas telas do sistema é clara*, *A interface deste sistema é agradável*, *A organização das informações nas telas do sistema é clara*, com valores de 1 a 1.5 na escala Lickert.

É possível notar que o sistema se comportou bem para a grande maioria dos itens do questionário e por isso o sistema atendeu as expectativas do usuário. Exceto nos pontos destacados anteriormente que precisam de melhorias. Logo, é possível inferir que a versão inicial do *Javascript Academy* atende parcialmente as necessidades dos usuários entrevistados.

4 EXPERIÊNCIA E LIÇÕES APRENDIDAS

4.1 Da idéia ao desenvolvimento

O *Javascript Academy* foi inicialmente concebido com uma idéia bastante ambiciosa. O projeto seria o desenvolvimento de uma plataforma de estudos com foco direcionado nos desenvolvedores de

[4] A URL do passo (3) é criada no passo (8) do usuário de testes anterior. Para o primeiro usuário foi utilizado uma lição previamente salva

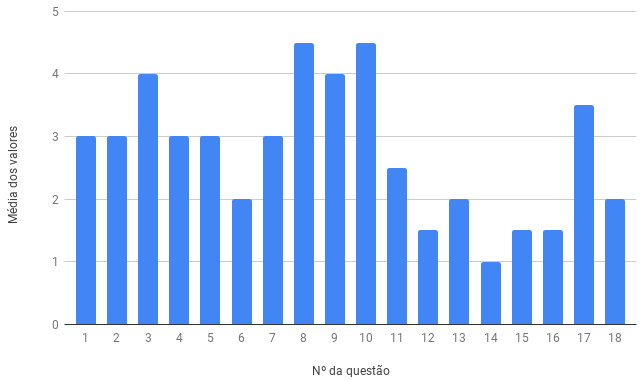

Figura 3: Médias das respostas dos itens do questionário PSSUQ

(1) No geral, estou satisfeito com o quão fácil é usar o sistema;
(2) Foi fácil utilizar este sistema.;
(3) Eu pude completar as tarefas e cenários de forma efetiva, usando este sistema;
(4) Eu fui capaz de completar as tarefas e cenários de forma rápida, usando este sistema;
(5) Eu me senti confortável usando este sistema;
(6) Foi fácil aprender a usar este sistema;
(7) Eu acredito que eu poderia me tornar produtivo rapidamente usando este sistema.;
(8) As mensagens de erros do sistema foram claras o suficiente para me ajudar na correção de erros ;
(9) Sempre que eu cometi algum erro, eu pude recuperar de forma fácil e rápida;
(10) As informações (como ajuda online, na tela de mensagens e outros documentos) fornecidas com este sistema foram claras;
(11) Foi fácil encontrar a informação que eu precisava;
(12) A informação fornecida pelo sistema é fácil de entender;
(13) A informação foi eficaz em me ajudar a completar as tarefas e cenários;
(14) A organização das informações nas telas do sistema é clara;
(15) A interface deste sistema é agradável;
(16) Eu gostei de usar a interface deste sistema;
(17) Este sistema tem todas as funções e capacidades que eu esperava que ele tivesse;
(18) No geral, estou satisfeito com este sistema;

frontend que já atuam no mercado. O desenvolvedor de *frontend* comum, aqui se entende pela pessoa que utiliza livrarias e frameworks clássicos como *AngularJS* e *JQuery*, ao fazer uso da plataforma seria introduzido aos conceitos que regem os principais frameworks, bem como a noção das boas práticas de desenvolvimento *frontend* moderno.

Toda a transição seria dada de maneira gradual. Inicialmente introduzindo conceitos e abstrações do paradigma funcional através de exercícios em Javascript e paulatinamente fazendo correlações com os frameworks modernos e as boas práticas de desenvolvimento. Para atingir esse objetivo seriam necessários três fases prévias.

(1) Criar e elaborar um ambiente onde fosse possível praticar exercícios em Javascript;
(2) Realizar revisão da literatura e organizar a ordem da apresentação dos conceitos que seriam apresentados;
(3) Criar os exercícios em Javascript de acordo com a ordem dos conceitos selecionados no fase anterior;

Contudo, essas três fases para serem concluídas com um desempenho satisfatório iria demandar mais tempo. Assim, o escopo foi decrementado apenas para o primeira fase e foi construído de modo que as fases subsequentes fossem facilmente adicionadas.

O resultado da conclusão da primeira fase é este projeto de TCC. Existiram algumas mudanças quanto ao objetivo primordial da primeira fase, mas a razão é simples: entregar um produto completo. Logo, o escopo foi redesenhado para permitir que fosse possível criar e responder listas de exercícios.

4.2 Principais desafios, suas soluções e limitações

Diversas foram as adversidades superadas durante a execução deste projeto. O nível técnico requerido para realização do proposto foi mais elevado do que o inicialmente pensado. Sobretudo, quanto à avaliação do código.

Avaliar um código não-validado é uma tarefa difícil e arriscada. No decorrer da sessão *2 - Arquitetura e Projeto da Solução* foi abordado o porquê dessa afirmação e explanada a decisão arquitetural tomada. Contudo, é importante saber qual foi o processo até chegar nesta decisão.

Basicamente, duas abordagens diferentes foram pensadas para avaliar um código. Para o cenário deste trabalho, foi decidido que uma boa heurística é aquela que:

- Sabe resolver situações em que as restrições p_i forem quebradas;
- Retorna para o usuário os erros de sintaxe e de semântica, e;
- Retorna a solução correta da computação;

A primeira abordagem foi uma variação da solução atualmente adotada. A grande diferença era que a função de avaliação do código era passada como um *callback* para um *setImmediate()*[5] de tal forma que, durante a execução, seriam observado as restrições e caso alguma fosse quebrada seria executado um *clearImmediate* para interromper a avaliação do código. Infelizmente, este fluxo de pensamento estava errado. Como Javascript é *single-thread* só é executado uma operação por vez, e no caso de um loop infinito dentro do código avaliado seria impossível que a linha que encerra a execução da operação fosse alcançada. Logo, esta não é uma abordagem ideal.

A segunda abordagem fez uso de *Web Workers*. Estes são mecanismos que permitem que uma operação de um script *Javascript* seja executada em uma thread diferente da thread principal[3]. Isso faz com que o código submetido pelo usuário seja executado de maneira isolada do sistema permitindo que a UI não seja bloqueada

[5] *setImmediate* é usado para interromper operações de longa duração e executar uma função imediatamente após o browser ter concluído outras operações, como eventos e atualizações do DOM [4]

além de aumentar a segurança na thread principal. Para tal, o código do usuário inserido via UI era transformado em um *Blob*[6] e posto para ser executado num *Web Worker* de tal forma que seria possível acessar as informações via eventos. Contudo dois problemas surgem: um objeto *blob* não pode ser criado se o script passado para ele possuir algum erro de sintaxe ou semântica, pois, o erro retornado é bastante genérico e não servem para ajudar o usuário a compreender o que está acontecendo, e, quando utilizado o sistema de maneira dinâmica, isto é, de modo mais próxima do real para quem está estudando, o *Web Worker* associado precisa lhe dar com muitas operações assíncronas de tal modo que é necessário um tato mais refinado no modelo de concorrência. Infelizmente, essa abordagem também não pode ser adotada nessa versão do sistema dado o fim do prazo para conclusão do projeto.

4.3 Trabalhos futuros

Para tornar o *Javascript Academy* um verdadeiro produto a ser utilizado pela comunidade Javascript é necessário que alguns pontos sejam evoluídos, são eles:

- Retomar o desenvolvimento da abordagem que faz uso de *Web Workers* com o intuito de deixar o terminal do sistema mais próximo de um terminal nativo;
- Permitir que seja possível manipular estuturas em HTML (sandbox);
- Criar um guia prático para quem for utilizar o sistema pela primeira vez;
- Permitir cadastro com redes sociais com o intuito de abarcar mais usuários cadastrados;
- Integradar o Github com o sistema para que cada solução do usuário seja salva em um repositório escolhido previamente;
- Permitir que as métricas das lições sejam exportadas no formato *csv*;
- Permitir a alteração de lições e exercícios.

REFERÊNCIAS

[1] Mozilla Development Docs. [n. d.]. *Async Function.* Technical Report. Visitado em 21 de junho de 2018. Posted at https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise.

[2] Mozilla Development Docs. [n. d.]. *Promise.* Technical Report. Visitado em 21 de junho de 2018. Posted at https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise.

[3] Mozilla Development Docs. [n. d.]. *Web Workers API.* Technical Report. Visitado em 21 de junho de 2018. Posted at https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API.

[4] Mozilla Development Docs. [n. d.]. *Window.setImmediate().* Technical Report. Visitado em 21 de junho de 2018. Posted at https://developer.mozilla.org/en-US/docs/Web/API/Window/setImmediate.

[5] Amin Milani Fard and Ali Mesbah. 2013. JSNOSE: Detecting JavaScript Code Smells.. In *SCAM*. IEEE Computer Society, 116–125. http://dblp.uni-trier.de/db/conf/scam/scam2013.html#FardM13

[6] Githut. 2016. *Discover languages in Github.* Technical Report. Visitado em 21 de junho de 2018. Posted at https://githut.info/.

[7] James Lewis and James R. 1993. IBM Computer Usability Satisfaction Questionnaires: Psychometric Evaluation and Instructions for Use. *International Journal of Human-Computer Interaction* 7 (01 1993), 57–. https://doi.org/10.1080/10447319509526110

[8] Julia Murphy and Max Roser. 2019. Internet. *Our World in Data* (2019). https://ourworldindata.org/internet.

[9] Bureau of Labor Statistics. 2016. *Occupational Outlook Handbook.* Technical Report. U.S. Department of Labor, Washington, DC, USA. Visitado em 09 de maio de 2018. Posted at https://www.bls.gov/ooh/computer-and-information-technology/web-developers.htm.

[10] Amir Saboury, Pooya Musavi, Foutse Khomh, and Giuliano Antoniol. 2017. An empirical study of code smells in JavaScript projects. *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)* (2017), 294–305.

[11] Szymon Wasik, Maciej Antczak, Jan Badura, Artur Laskowski, and Tomasz Sternal. 2018. A Survey on Online Judge Systems and Their Applications. *ACM Comput. Surv.* 51, 1, Article 3 (Jan. 2018), 34 pages. https://doi.org/10.1145/3143560

[12] J Williams and D Wichers. 2014. Owasp, top 10, the ten most critical web application security risks. *Technical report* (2014).

[6] Um Blob representa um objeto do tipo arquivo, com dados brutos imutáveis.