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

DAVI LAERTE NUNES SABINO NASCIMENTO

VisuED:

UMA FERRAMENTA PARA IMPLEMENTAÇÃO E VISUALIZAÇÃO DE LISTAS LIGADAS

CAMPINA GRANDE - PB

2019

DAVI LAERTE NUNES SABINO NASCIMENTO

VisuED:

UMA FERRAMENTA PARA IMPLEMENTAÇÃO E VISUALIZAÇÃO DE LISTAS LIGADAS

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. João Arthur Brunet Monteiro.

CAMPINA GRANDE - PB

2019

N244v Nascimento, Davi Laerte Nunes Sabino do.
VisuED: uma ferramenta para implementação e visualização de listas ligadas. / Davi Laerte Nunes Sabino do Nascimento. – 2019.

9 f.

Orientador: Prof. Dr. João Arthur Brunet Monteiro.
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. Listas ligadas. 2. Estruturas de dados. 3. Registro de dados. 4. Ferramenta online. 5. VisuED – listas ligadas. I. Monteiro, João Arthur Brunet. II. Título.

CDU:004(045)

Elaboração da Ficha Catalográfica:

Johnny Rodrigues Barbosa
Bibliotecário-Documentalista
CRB-15/626

DAVI LAERTE NUNES SABINO NASCIMENTO

VisuED:

UMA FERRAMENTA PARA IMPLEMENTAÇÃO E VISUALIZAÇÃO DE LISTAS LIGADAS

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. João Arthur Brunet Monteiro
Orientador – UASC/CEEI/UFCG

Professor Dr. Jorge César Abrantes de Figueiredo
Examinador – UASC/CEEI/UFCG

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

Trabalho aprovado em: 25 de novembro 2019.

CAMPINA GRANDE - PB

VisuED: uma ferramenta para implementação e visualização de listas ligadas

Trabalho de Conclusão de Curso

Davi Laerte Nunes Sabino Nascimento
Universidade Federal de Campina Grande
Campina Grande, Paraíba, Brasil
davi.nascimento@ccc.ufcg.edu.br

RESUMO

Os estudantes de computação normalmente têm dificuldade para entender as listas ligadas, que consistem em um conjunto de registro de dados (nós) ligados entre si e organizados por referências de forma que cada nó tem uma referência que aponta para o próximo. A maior dificuldade é no momento em que é necessário desenvolver algoritmos relacionados às operações em uma dessas listas, pois os mesmos sentem dificuldade em entender como funcionam as conexões entre os nós. Isso muitas vezes se deve a falta de uma melhor visualização do funcionamento das operações dessas listas. Portanto, este trabalho, visa o desenvolvimento de uma ferramenta que vai permitir ao estudante de computação ter uma visualização dinâmica do comportamento dos nós durante a manipulação de referências que as operações básicas dessas listas realizam.

REPOSITÓRIOS

https://github.com/davilaerte/visualization-server
https://github.com/davilaerte/ed-visualization

1 INTRODUÇÃO

Uma estrutura de dados é um modo particular de armazenamento e organização de dados em computador [11] de modo que esses dados possam ser usados de forma eficiente, facilitando sua busca e manutenção. As estruturas de dados são temas fundamentais da Ciência da Computação, sendo utilizadas nas mais diversas áreas de conhecimento e tendo diversas possíveis aplicações. Dada a devida importância desse assunto [13], observa-se que ele constitui um grande divisor de águas nos cursos de Computação, pois causa um grande impacto na formação acadêmica.

Dentro do conjunto de estruturas de dados estudadas, as listas ligadas são uma das estruturas que os discentes mais sentem dificuldade. Esse tipo de estrutura consiste em um conjunto de registro de dados (nós) ligados entre si e organizados por referências de forma que cada nó tem uma referência que aponta para o próximo. Os estudantes sentem dificuldade em entender e implementar a manipulação de referências nessas estruturas, devido ao fato das manipulações serem complexas e difíceis de serem abstraídas.

Diante das dificuldades enfrentadas pelos estudantes de computação em entender a manipulação de referências em listas ligadas e visto a importância desse assunto na formação dos discentes, fica evidente a necessidade de uma ferramenta que consiga ajudá-los no entendimento e visualização das operações nesse tipo de estrutura, de forma que os mesmos consigam abstrair a complexidade das manipulações de referências que são feitas.

O objetivo deste trabalho é desenvolver uma ferramenta web que vai permitir ao estudante de computação implementar as operações básicas de uma lista ligada e visualizar dinamicamente as manipulações de referências que foram implementadas nas operações. Sendo assim, o estudante poderá entender e abstrair visualmente o funcionamento das operações que ele implementou.

O presente artigo tem o intuito de relatar sobre a experiência de desenvolvimento do VisuED, aplicação que busca auxiliar o estudo e compreensão das listas ligadas. Durante o desenvolvimento, não foi realizado nenhum experimento para avaliar a qualidade do sistema, porém um experimento de avaliação é um importante trabalho futuro a ser realizado em benefício da continuação da ferramenta.

1.1 Soluções conhecidas

Atualmente, existem algumas soluções similares a esta proposta, entre elas as mais populares são o VisuAlgo [9] e o Data Structures Visualization [3]. Essas soluções normalmente fornecem visualização animada para varias estruturas de dados e algoritmos conhecidos. Para as listas ligadas, elas oferecem visualização dinâmica das suas operações. No entanto, elas não oferecem a possibilidade do usuário criar sua própria implementação de uma lista ligada, assim ele perde a chance de aprender de forma prática sobre o funcionamento das operações nessas estruturas.

2 SOLUÇÃO

2.1 Descrição

O VisuED é uma ferramenta online que permite a implementação e visualização de listas ligadas de forma dinâmica, para assim possibilitar ao usuário um maior entendimento prático sobre as manipulações de referências das listas ligadas.

Na ferramenta, o usuário pode escolher um tipo de lista para implementar e visualizar. Após escolher uma lista é possível implementar as operações daquela lista (Figura 1) na linguagem de programação Java, usando o editor de código disponibilizado na interface. Para facilitar e simplificar a implementação da lista, é disponibilizado parte da implementação daquele tipo de lista em Java, faltando somente o código referente as suas operações, assim o usuário consegue escrever os métodos da forma que achar melhor.

Com o objetivo de viabilizar o processo de implementação e visualização e também para deixá-lo mais simples e prático, algumas funcionalidades são essenciais e foram implementadas:

- Visualização do nó da lista: permite a visualização da implementação do nó que é usado pela lista, esse nó é usado para representar um elemento e suas referências e deve ser usado na implementação das operações.
- Gerar implementação padrão: o usuário tem a opção de gerar uma implementação padrão para as operações da lista, facilitando o processo de aprendizagem, pois é possível entender melhor o funcionamento da sua estrutura ao comparar com a padrão.
- Visualizar implementação: compila a implementação da lista e abre a tela de visualização correspondente a que foi feita.

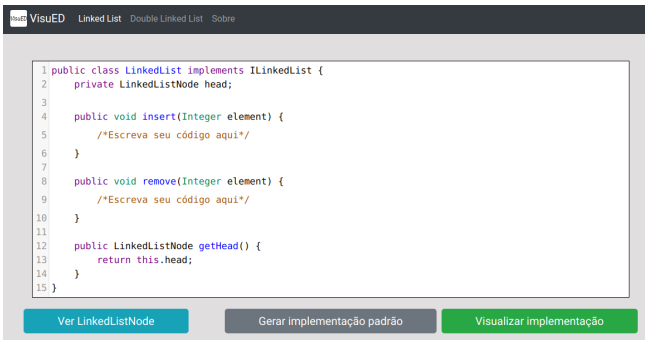

Figura 1: Editor de código

Na tela de visualização da implementação (Figura 2) é possível executar os métodos que foram implementados para a lista e depois visualizar os efeitos deles nos elementos e ligações. Para um melhor entendimento sobre a posição dos componentes da lista, a visualização possui legendas em partes importantes da lista como a cabeça, cauda e as ligações entre os elementos. Também é possível modificar a posição dos nós da visualização, movendo os nós para assim possibilitar mais praticidade e interatividade.

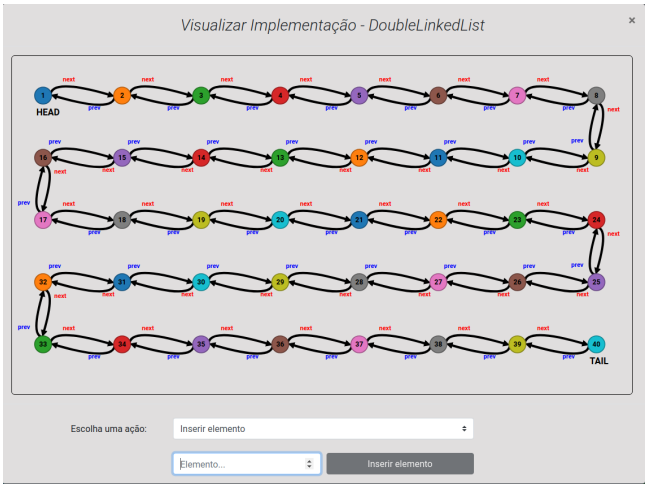

Figura 2: Tela para visualização de uma implementação

2.2 Arquitetura

O VisuED é projetado de acordo com a arquitetura Cliente-Servidor, na qual existem dois componentes do sistema. Os componentes se comunicam segundo o modelo REST (Representational State Transfer), utilizando o JSON (JavaScript Object Notation) como o modelo para transferência de dados.

Este modelo de comunicação propicia a modularização e a escalabilidade [14]. Sendo assim, os componentes podem evoluir e se modificar constantemente e de maneira independente e, assim, oferecerem apenas seus serviços para os demais através de suas interfaces, as quais podem ser acessadas por outros componentes, provendo a transferência dos dados no formato JSON.

O modelo arquitetural de Cliente-Servidor do sistema é organizado em um cliente e um servidor, permitindo a distribuição de tarefas entre os mesmos. O cliente, conhecido como front-end da aplicação, é o responsável pela interação com os usuários do sistema, enquanto que o servidor, conhecido como o back-end da aplicação, é responsável por gerenciar os dados da aplicação.

2.2.1 Tecnologias do back-end

No back-end foram utilizados o framework Spring Boot [8], para construção do servidor, e a biblioteca Java Runtime Compiler [4], para a compilação em tempo de execução do código Java. O Spring Boot é uma tecnologia que facilita o processo de configuração e publicação da aplicação, com ela é possível criar rapidamente APIs Rest, por ser relativamente simples de configurar. Devido ao fato de seu código ser escrito em Java, foi mais fácil usar as ferramentas para compilação do código de uma implementação. O Java Runtime Compiler é uma biblioteca Java que facilita a compilação de código Java em tempo de execução, sendo possível compilar uma String contendo código e retornar a classe correspondente. A maior vantagem dessas tecnologias é que elas são de código aberto, não sendo necessário pagar para utilizá-las.

2.2.2 Estrutura do back-end

Como é possível observar na Figura 3, o código é organizado em seis categorias principais: builders, controllers, exceptions, helpers, impl e models.

Os builders são responsáveis por gerar as classes que representam as listas ligadas que vão ser implementadas. Eles recebem o código referente às operações da lista e produzem uma String contendo uma classe Java que corresponde à implementação da lista.

Os controllers são encarregados pela parte lógica do servidor. O controller *DSVisualizationRestController* processa as chamadas HTTP do sistema, já os controllers *DSVisualizationController* e *DSHashImplsController* gerenciam a lógica de criação de uma nova implementação e execução de operações das listas. Também existe o *VisualizationServiceErrorAdvice* que é responsável por verificar e gerenciar possíveis erros que possam ser causados durante a execução ou criação de uma implementação, caso ocorra algum erro esse controller retorna a exceção na requisição HTTP.

As exceptions é onde são definidos os tipos de erros que podem ocorrer durante a lógica de criação e execução de uma implementação. Atualmente há três tipos de exceptions que estão definidas, são elas: o *CompilationImplErrorException* que acontece devido a alguma falha na compilação do código de uma nova lista, o *RunImplErrorException* que ocorre por causa de algum erro na execução

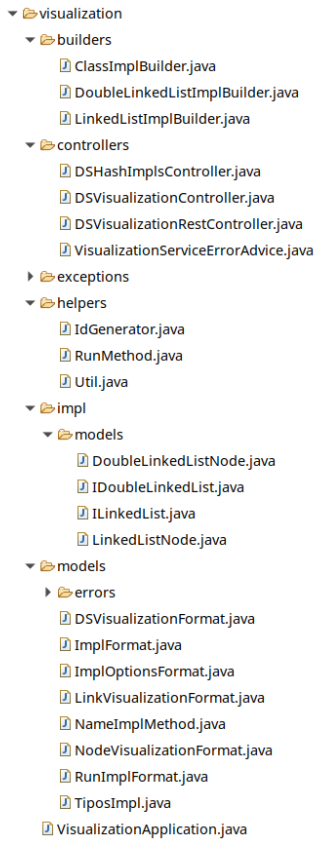

Figura 3: Estrutura de diretórios do servidor

de uma operação em uma lista e por último existe o *TimeoutRunMethodException* que ocorre quando a execução da operação demora muito tempo para terminar, o que poderia indicar um possível laço infinito no código da operação.

Os helpers, como o próprio nome sugere, possuem classes que auxiliam à aplicação. Na classe *IdGenerator* é onde o id de uma nova implementação é gerado, já a classe *RunMethod* é usada para executar um método de uma lista com um tempo limite, e por último existe a classe *Util* onde existe uma variedade de funções que ajudam na lógica da aplicação, como por exemplo a função isValidString que verifica se a String passada é nula ou vazia.

As categorias impl e models são responsáveis pela definição de objetos que são usados para auxiliar à lógica da aplicação. Na categoria impl é onde estão definidas as interfaces e os nós utilizados na implementação de uma nova lista, já nos models é onde são definidos os objetos usados na comunicação, via JSON, com o cliente.

2.2.3 *Tecnologias do front-end*

Para o desenvolvimento do cliente foi utilizado o React [6], que é uma biblioteca JavaScript de código aberto que permite a criação de interfaces de usuário, ou seja, propicia a construção de uma camada de visualização em um sistema web. A mesma segue um modelo arquitetural baseado em componentes, os quais em conjunto definem a camada de visualização da aplicação. Esta tecnologia foi utilizada no sistema com o intuito de desenvolver a interface do usuário, de maneira a prover um sistema com módulos mais reusáveis e escaláveis [10].

Além do React, foi utilizado a biblioteca de código aberto CodeMirror [1], que é um editor de texto versátil implementado em JavaScript, ele é especializado em edição de código e possui um grande número de linguagens suportadas, ele foi usado na aplicação com o intuito de facilitar a edição das implementações. Seu uso no sistema foi feito através do react-codemirror [7], que é um componente React que oferece as funcionalidades do CodeMirror.

Também foi usado o D3.js [2], que é uma biblioteca JavaScript de código aberto usada para produzir visualizações de dados dinâmicas e interativas em navegadores web. Ela foi utilizada na aplicação para produzir a visualização dos nós e referências da lista. Seu uso facilitou e simplificou a manipulação dos elementos da visualização.

2.2.4 *Estrutura do front-end*

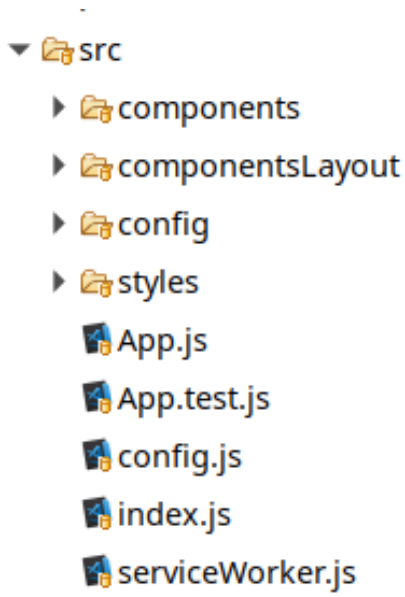

Figura 4: Estrutura de diretórios do cliente

A Figura 4 mostra como é organizado o código do lado cliente. Nas pastas *components* e *componentsLayout* é onde ficam os componentes React usados na aplicação, eles contêm a maior parte da lógica do front-end. Na parte de *components* ficam os componentes que são responsáveis por tarefas específicas do sistema, como por exemplo o componente *Visualization* que é responsável pela visualização das estruturas implementadas. Já na parte de *componentsLayout* ficam os componentes responsáveis pela criação das telas da aplicação, como por exemplo o *HomeLayout* que é responsável pela caracterização da tela inicial da ferramenta.

Os *config* são códigos responsáveis por tarefas de configuração do sistema, como por exemplo o arquivo history.js que configura a ferramenta encarregada de gerenciar o histórico de sessões.

Os *style* são onde se localizam as folhas de estilo (CSS) da aplicação. As folhas de estilo são responsáveis por organizar e estilizar as páginas HTML que são geradas pelos componentes React.

2.2.5 *Lógica da aplicação*

Para criar uma nova implementação de lista ligada e ver sua visualização, o usuário escolhe o tipo de lista que deseja implementar e escreve o código referente às operações daquela lista no editor. Em seguida, ele clica no botão de visualizar implementação, quando isso é feito o código é enviado para o servidor da aplicação. Quando essa nova implementação chega no servidor é gerado um identificador (id) para ela, depois os builders geram, usando o código passado, a String que representa a classe correspondente ao tipo de lista requisitada. A String gerada é compilada em tempo de execução usando a biblioteca Java Runtime Compiler e caso não haja erro de compilação a classe retornada é salva em um Hash em que a chave é o id da implementação, depois disso o id é retornado para o cliente, onde ele é salvo para ser usado durante a execução dos métodos da lista criada.

Após criar a implementação, a tela de visualização é mostrada, nela o usuário escolhe a operação da lista que deseja executar, quando a operação é executada, o nome do método e o id da implementação são enviados para a API. Esses dados são recebidos pelo servidor, que usa o id para recuperar a classe correspondente a implementação e executar o método escolhido. Esse método é executado com um tempo limite de 10s, caso a execução estoure o tempo limite, um erro do tipo *TimeoutException* é lançado. Logo após a execução da operação, caso não haja erro de execução e nem de timeout, os nós e ligações da lista são percorridos e a partir deles são gerados objetos do tipo *NodeVisualizationFormat* e *LinkVisualizationFormat* respectivamente. Os nós da lista são percorridos usando um algoritmo de busca em largura, que é mostrado na Figura 5, que vai garantir que todo nó acessível seja visitado [12]. Isso é feito com o intuito de fornecer o máximo de informação possível sobre o estado dos nós e ligações da lista na visualização. Depois de percorrer a lista é criado um objeto do tipo *DSVisualizationFormat* que contém os objetos *NodeVisualizationFormat* e *LinkVisualizationFormat* que foram gerados, ele é enviado para o cliente, onde será usado na visualização.

```
DSVisualizationFormat formatedLinkedList = new DSVisualizationFormat();
Map<LinkedListNode, Integer> visitedNodes = new HashMap<LinkedListNode, Integer>();
LinkedList<LinkedListNode> queue = new LinkedList<LinkedListNode>();
if (head != null) {
    queue.addLast(head);
    Integer counterId = 1;
    visitedNodes.put(head, counterId);
    while(!queue.isEmpty()) {
        LinkedListNode currNode = queue.removeFirst();
        Integer currNodeId = visitedNodes.get(currNode);
        formatedLinkedList.nodes.add(new NodeVisualizationFormat(currNodeId, String.valueOf(currNode.data)));

        if (currNode.next != null) {
            if (!visitedNodes.containsKey(currNode.next)) {
                queue.addLast(currNode.next);
                visitedNodes.put(currNode.next, ++counterId);
            }
            formatedLinkedList.links.add(new LinkVisualizationFormat(currNodeId, visitedNodes.get(currNode.next), LABEL_NEXT));
        }
    }
}
```

Figura 5: Algoritmo de busca em largura usado na Linked List

Se durante a criação de uma lista ou execução de algum método houver um erro de compilação, execução ou timeout, esse erro é capturado e retornado na requisição HTTP para o cliente, onde, como pode ser visto na Figura 6, ele será mostrado.

Figura 6: Erro mostrado na interface

2.2.6 *Ambiente de produção*

A aplicação foi hospedada na LocaWeb [5] usando um servidor virtual privado (VPS), nessa máquina virtual são servidos o front-end e o back-end da aplicação. Essa plataforma foi escolhida principalmente pela praticidade de poder usar uma máquina virtual Linux para realizar o deploy da aplicação. A ferramenta pode ser acessada atráves desse endereço **http://vps16522.publiccloud.com.br/**.

3 EXPERIÊNCIA

3.1 Processo de desenvolvimento

Inicialmente, foi decidido quais tecnologias seriam usadas no projeto. Essas escolhas foram baseadas, principalmente, na experiência obtida em outros projetos. Depois disso, foi feita uma análise de requisitos, onde foram definidas as funcionalidades principais do sistema. Para cada funcionalidade, foi realizado um estudo para determinar a viabilidade técnica, considerando as tecnologias escolhidas. Em seguida, foi implementado um protótipo, como é mostrado na Figura 7, com o objetivo de concretizar a viabilidade do projeto. Ele possuía apenas uma tela, onde era possível implementar o método *inserir* de uma lista ligada simples e depois visualizar o estado dos elementos após a execução da operação.

Figura 7: Única tela do protótipo do VisuED

De forma iterativa e incremental, o protótipo foi sendo aprimorado e novas funcionalidades foram sendo implementadas. Sempre que um novo recurso era implementado, eram realizados testes e correções de bugs.

3.2 Desafios

Compilar e executar o código feito pelos usuários foram as partes mais desafiadoras do projeto. A compilação de código Java em

tempo de execução drenou muito tempo de pesquisa, principalmente a respeito de ferramentas que pudessem simplificar esse processo, visto que o ferramental que a linguagem oferece é um pouco confuso e difícil de usar. Já a execução do código foi desafiadora devido às vulnerabilidades que podem ser causadas, uma delas é a possibilidade do código não parar de executar, ou seja, dele possuir laços infinitos. Para evitar isso foi necessário criar um tempo limite para a execução de um método, de forma que caso a operação não terminasse de executar nesse tempo, ela seria interrompida e uma mensagem de erro seria mostrada ao usuário.

Além disso, a visualização dos nós e ligações das listas não foi uma tarefa fácil. Foi necessário estudar bastante o D3.js e também pesquisar exemplos de visualizações parecidas, a fim de chegar em um formato de visualização. E para chegar nas fórmulas matemáticas utilizadas para determinar a posição dos nós e ligações, foi necessário fazer vários testes e ajustes, para que o comportamento esperado fosse alcançado.

3.3 Vantagens e limitações da ferramenta

As principais vantagens da ferramenta estão relacionadas ao seu apelo prático, como o fato dela permitir que o usuário possa criar sua própria implementação de lista ligada e que depois ele visualize essa implementação de forma iterativa, sendo possível modificar facilmente a estrutura da visualização. Outra vantagem da ferramenta está no fato dela facilitar o processo de codificação, pois a aplicação disponibiliza uma parte do código já pronto, faltando apenas a implementação das operações da lista, e também a ferramenta possui uma implementação padrão para as listas, que pode ser usada como base para codificação.

A ferramenta possui algumas limitações também, umas das principais é o fato da aplicação não disponibilizar um acompanhamento passo a passo do código durante a execução de uma operação, outra limitação é a falta de uma visualização passo a passo, ou seja, a possibilidade de voltar a visualização para o estado anterior. Também existe uma limitação durante a visualização de uma lista, ela ocorre pois a animação durante a execução dos métodos é muito rápida, tornando as vezes um pouco difícil entender visualmente as mudanças que houveram em relação ao estado anterior.

3.4 Trabalhos futuros

Trabalhos futuros podem estender o VisuED de forma a competir com ferramentas para visualização de estruturas de dados já existentes e também oferecer uma melhor experiência ao usuário, isso pode ser alcançado através da implementação de mais algumas funcionalidades:

- Adicionar outras estruturas de dados, além de listas ligadas, como opções para visualizar e implementar;
- Disponibilizar a opção de visualização passo a passo, ou seja, possibilidade de voltar a visualização ao estado respectivo a operação anterior;
- Oferecer opção de exportar o código em um classe Java;
- Adicionar a opção de mostrar lentamente a visualização da execução dos métodos.

REFERÊNCIAS

[1] [n. d.]. CodeMirror Page. https://codemirror.net/
[2] [n. d.]. D3.js Page. https://d3js.org/
[3] [n. d.]. Data Structures Visualization Page. https://www.cs.usfca.edu/~galles/visualization/Algorithms.html
[4] [n. d.]. Java Runtime Compiler Repository. https://github.com/OpenHFT/Java-Runtime-Compiler
[5] [n. d.]. LocaWeb PAge. https://www.locaweb.com.br/
[6] [n. d.]. React: a JavaScript library for building user interfaces. https://reactjs.org/
[7] [n. d.]. React Codemirror Repository. https://github.com/JedWatson/react-codemirror
[8] [n. d.]. Spring Boot Page. https://spring.io/projects/spring-boot
[9] [n. d.]. VisuAlgo Page. https://visualgo.net/en
[10] Francisco Afonso. [n. d.]. Component-Based Architecture: What Is And Why Should Care. Retrieved November 18, 2019 from https://www.outsystems.com/blog/3-reasons-invest-component-based-architecture.html
[11] Zain UI Hassan. [n. d.]. Learn About Data Structures And Algorithms (DSA). Retrieved July 2, 2019 from https://www.c-sharpcorner.com/article/learn-about-data-structures-and-algorithm-dsa/
[12] David Krenkel. [n. d.]. Aplicações de Busca em Grafos. Retrieved November 17, 2019 from http://www.ceavi.udesc.br/arquivos/id_submenu/387/david_krenkel_rodrigues_de_melo.pdf
[13] Arpit Mishra. [n. d.]. Why study data structures and algorithms? Retrieved November 17, 2019 from https://www.hackerearth.com/blog/developers/study-data-structures-algorithms/
[14] Haroon Shakirat Oluwatosin. 2014. Client-Server Model. *IOSR Journal of Computer Engineering* 16, 1 (February 2014), 67–71.

A APÊNDICES

A.1 Imagens da aplicação

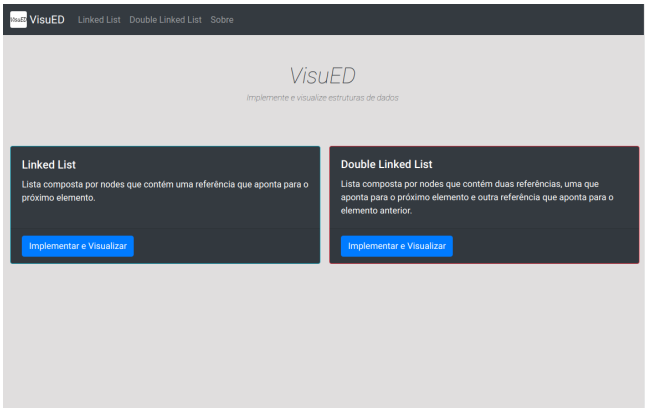

Figura 8: Tela inicial do VisuED

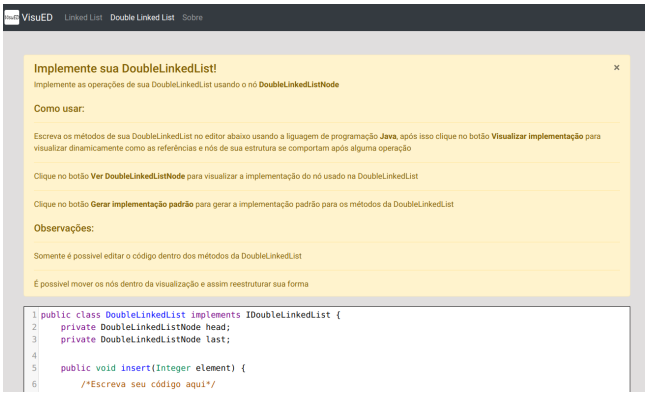

Figura 9: Alerta com informações sobre o uso da ferramenta

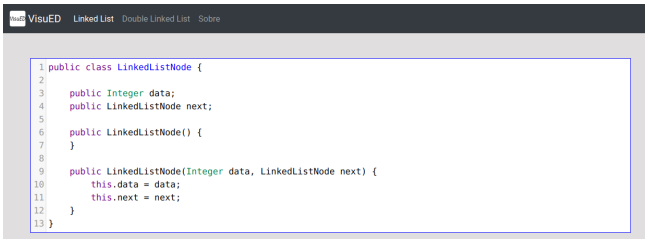

Figura 10: Editor com informações sobre o nó usado na lista

```java
public class LinkedList implements ILinkedList {
    private LinkedListNode head;

    public void insert(Integer element) {
        if (this.head == null) {
            this.head = new LinkedListNode(element, null);
        } else {
            LinkedListNode node = this.head;
            while(node.next != null) {
                node = node.next;
            }
            node.next = new LinkedListNode(element, null);
        }
    }

    public void remove(Integer element) {
        if (this.head != null) {
            if (this.head.data.equals(element)) {
                this.head = this.head.next;
            } else {
                LinkedListNode nodePrev = this.head;
                LinkedListNode node = this.head.next;
                while(node != null && !node.data.equals(element)) {
                    nodePrev = node;
                    node = node.next;
                }
                if (node != null) {
                    nodePrev.next = node.next;
                }
            }
        }
    }
```

Figura 11: Implementação padrão da Linked List

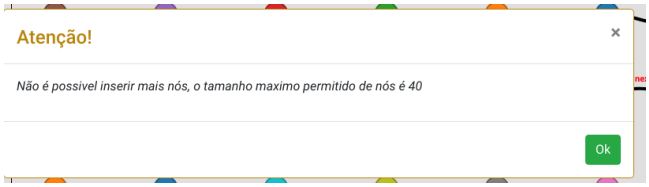

Figura 12: Alerta exibido quando o número máximo de nós é ultrapassado