

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Augusto Duarte de Melo

**Desenvolvimento de uma Solução Web Para
Aplicação de Bateria Informatizada de
Avaliação de Idosos: Visualização de Resultados
de Questionários.**

Uberlândia, Brasil

2016

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Augusto Duarte de Melo

**Desenvolvimento de uma Solução Web Para Aplicação
de Bateria Informatizada de Avaliação de Idosos:
Visualização de Resultados de Questionários.**

Trabalho de conclusão de curso apresentado
à Faculdade de Computação da Universidade
Federal de Uberlândia, Minas Gerais, como
requisito exigido parcial à obtenção do grau
de Bacharel em Ciências da Computação.

Orientador: Fabiano Azevedo Dorça

Universidade Federal de Uberlândia – UFU

Faculdade de Ciência da Computação

Bacharelado em Ciências da Computação

Uberlândia, Brasil

2016

Augusto Duarte de Melo

Desenvolvimento de uma Solução Web Para Aplicação de Bateria Informatizada de Avaliação de Idosos: Visualização de Resultados de Questionários.

Trabalho de conclusão de curso apresentado
à Faculdade de Computação da Universidade
Federal de Uberlândia, Minas Gerais, como
requisito exigido parcial à obtenção do grau
de Bacharel em Ciências da Computação.

Trabalho aprovado. Uberlândia, Brasil, 15 de dezembro de 2016:

Fabiano Azevedo Dorça
Orientador

Professor

Professor

Uberlândia, Brasil
2016

Dedico esse trabalho a minha família, a minha namorada, meus amigos, e todas as pessoas que de alguma forma colaboraram para realização e concretização deste trabalho.

Agradecimentos

Agradeço a Universidade Federal do Triângulo Mineiro (UFTM), Faculdade de Computação (FACOM), ao professor Fabiano Dorça e a professora Sabrina por podermos realizar esse trabalho.

Resumo

Os sistemas web tem como objetivo facilitar o desenvolvimento de uma aplicação, já que por sua vez eles rodam em navegadores que existem em todas as plataformas (tanto desktop qnto mobile). A Universidade Federal de Uberlândia (UFU), juntamente com a Universidade Federal do Triângulo Mineiro (UFTM), decidiram construir uma sistema que visa produzir uma bateria informatizada de avaliação de idosos. Com isso decidimos então criar um sistema web para facilitar a aplicação dessa bateria de avaliação, o uso da tecnologia nesse cenário irá trazer os seguintes benefícios: facilidade na aplicação, eliminar grandes quantidade de formulários impressos, entre outros. Este trabalho de conclusão de curso visa criar um aplicação, com as soluções mais atuais (HTML5, JavaScript, CSS, VueJS, MaterializeCSS), que será responsável por visualizar os resultados e respostas dessa bateria de avaliação cognitiva.

Palavras-chave: Sistemas-web, visualizador, protocolo de análise cognitivo.

Lista de ilustrações

Figura 1 – Vuex [14]	17
Figura 2 – Webpack [21]	19
Figura 3 – Classe anotação	20
Figura 4 – Exemplo XML classe anotação	20
Figura 5 – Exemplo classe anotação XML Schema	21
Figura 6 – Exemplo objeto anotação JSON	22
Figura 7 – Modelo em Cascata	24
Figura 8 – Bateria de testes	24
Figura 9 – Visualizador	25
Figura 10 – XML Schema	26
Figura 11 – Tela e XML Schema	26
Figura 12 – Pacote NPM	27
Figura 13 – Vue-router	28
Figura 14 – Tela principal em um dispositivo com tela grande	28
Figura 15 – Tela principal em um dispositivo mobile	29
Figura 16 – Tela de upload de arquivos	29
Figura 17 – Tabela de questionários	30
Figura 18 – Questionário completo	30
Figura 19 – Componente de paginação	31
Figura 20 – Erros do Eslint	32
Figura 21 – Commits realizados	32
Figura 22 – Repositório no github	33

Lista de abreviaturas e siglas

HTML	<i>HyperText Markup Language</i>
CSS	<i>Cascading Style Sheets</i>
HTTP	<i>Hypertext Transfer Protocol</i>
SPA	<i>Single-Page Application</i>
ES6	<i>ECMA Script 6</i>
ES5	<i>ECMA Script 5</i>
XML	<i>Extensible Markup Language</i>
JSON	<i>JavaScript Object Notation</i>
X2JS	<i>XML to JSON</i>
SCM	<i>Source Control Manager</i>

Sumário

1	INTRODUÇÃO	11
1.1	Contextualização	11
1.2	Justificativa	12
1.3	Objetivo	12
1.3.1	Objetivo Geral	12
1.3.2	Objetivo Específico	12
1.3.2.1	Requisitos Funcionais	13
1.3.2.2	Requisitos Não Funcionais	13
2	FERRAMENTAS UTILIZADAS	14
2.1	JavaScript	14
2.1.1	ECMAScript 6	14
2.1.2	ESlint	14
2.2	HTML5	15
2.2.1	LocalStorage	15
2.3	CSS	15
2.4	VueJS 2	15
2.4.1	Vue-router	16
2.4.2	Vuex	16
2.4.3	Vue-loader	16
2.5	BabelJS	17
2.6	MaterializeCSS	17
2.7	Git	18
2.7.1	GitHub	18
2.8	Webpack	18
2.9	NPM	19
2.10	XML	19
2.10.1	XML Schema	20
2.11	JSON	21
2.11.1	X2JS	21
3	DESENVOLVIMENTO	23
3.1	Implementação e Testes	24
3.2	Homologação	34
3.3	Dificuldades e Desafios Encontrados	34

4	CONCLUSÃO	35
	REFERÊNCIAS	36

1 Introdução

1.1 Contextualização

Os sistemas web são sistemas que de forma geral são projetados para utilização através do (tanto *mobile* quanto *desktop*). Eles são projetados utilizando tecnologias como HTML (*HyperText Markup Language*), JavaScript e CSS (*Cascading Style Sheets*), depois esses arquivos são disponibilizados em um servidor e podem ser acessados utilizando o protocolo HTTP (*Hypertext Transfer Protocol*). Eles também são um dos mais utilizados hoje, pois é notado que atualmente a maioria dos computadores que estão conectados a internet possuem um navegador.

Na atualidade o modelo de desenvolvimento de páginas é o SPA (*single-page application*). Basicamente o SPA visa codificar menos no lado do servidor e mais no lado do cliente, por conseguinte temos que quase toda a aplicação estará no lado do cliente fazendo que as transições ocorram sem necessidade de fazer o recarregado na página. Alguns benefícios de uma aplicação SPA são:

- Melhor experiência do usuário: Já que a página não irá ter *refreshes* então o usuário terá a sensação de estar em uma aplicação desktop.
- Performance: Porque o sistema é carregado inteiro na primeira requisição de forma assíncrona.
- Responsabilidade do *client-side*.
- Facilidade de manutenção: Fica bem dividido o que é *front-end* e o que *back-end* na hora do desenvolvimento.

Seguindo essa idéia de sistemas web, notou-se como que o uso da tecnologia na área da psicologia, principalmente na hora avaliar uma pessoa, melhorar e agilizar o processo de diagnóstico. Os conhecimentos que são adquiridos através da percepção, memória, raciocínio, entre outros; estão extremamente relacionados com as nossas funções cognitivas que também está ligado ao processo de envelhecimento. Os estudos sobre envelhecimento demonstram que ocorre um declínio significativo em funções como atenção, memória e funções executivas de planejamento mesmo em idosos não- acometidos por doenças [1]. Atualmente não é possível evitar completamente as perdas cognitivas, porém existem atividades que combatem esse processo de perda. Pesquisas sobre intervenções cognitivas demonstram que o treino cognitivo pode ocasionar aumento do desempenho ou permitir a manutenção de habilidades cognitivas em idosos saudáveis [2]. No Brasil o grupo que

mais cresce e que apresenta o maior foco de pesquisas é o dos idosos, e a maioria dessas pesquisas é voltada a melhorar a qualidade de vida desse grupo. Os problemas de memória representam uma das principais queixas dos idosos [3] e alguns trabalhos, em geral estudos de caso desenvolvidos por psicólogos cognitivos e neuropsicólogos mostraram potencial para auxiliar a conservar essa capacidade por meio de treinamento cognitivo [4].

1.2 Justificativa

Seguindo essa linha de pensamento, a Universidade Federal de Uberlândia (UFU) em parceria com a Universidade Federal do Triângulo Mineiro (UFTM), propuseram o desenvolvimento de uma ferramenta virtual na qual através de uma série de atividades e questionários o idoso possa treinar suas funções cognitivas, além disso essa ferramenta virtual iria realizar um diagnóstico e seria mostrado a possibilidade do idoso saudável vir a desenvolver uma patologia neuropsiquiátricas.

Desta forma, este trabalho vem a trazer benefícios na aplicação de uma bateria de testes para diagnóstico e prevenção de problemas cognitivos em idosos, já que o ambiente informatizado na plataforma web permite que esta bateria de testes seja facilmente aplicada pelos psicólogos, já que elimina grandes quantidades de formulários impressos, facilitando a análise dos resultados e também a minimizando a perda de informações. Além disto, os resultados podem ser facilmente compartilhados entre os psicólogos aplicadores da bateria e pesquisadores da área (uma vez que o sistema foi projetado visando este cenário), já que o sistema foi implementado em plataforma web.

1.3 Objetivo

1.3.1 Objetivo Geral

A fim de facilitar o diagnóstico de problemas cognitivos, foi decidido criar uma ferramenta em parceria com a UFTM, o objetivo dessa ferramenta é auxiliar os psiquiatras a aplicarem o protocolo de análise cognitiva em seus pacientes, e onde o pesquisador (responsável) conseguirá visualizar o resultado e as respostas do protocolo aplicado.

1.3.2 Objetivo Específico

Este trabalho tem como foco desenvolver um sistema web utilizado o método de desenvolvimento SPA, para isso serão utilizados as seguintes ferramentas: HTML5, JavaScript, CSS; que são a base de qualquer sistema web. Decidimos utilizar os frameworks VueJS e MaterializeCSS pois são ferramentas que irão agilizar o processo de desenvolvimento. Os requisitos funcionais e não funcionais serão listados a seguir:

1.3.2.1 Requisitos Funcionais

- Tela de *upload* dos protocolos respondidos, onde será possível fazer o *upload* de múltiplos arquivos.
- Tela de visualização do protocolo de análise cognitivo.
- Lista dos protocolos respondidos.
- Download dos protocolos.

1.3.2.2 Requisitos Não Funcionais

- O sistema deve ser aberto em qualquer navegador.
- Deve ser utilizado ferramentas conceituadas no mercado.
- O código deve seguir uma padronização.
- Deve ser um sistema web, permitindo que a bateria de testes fosse aplicada por psicólogos independente de sua localização
- Deve permitir compartilhamento de informações entre pesquisadores
- Deve ser de interface visual de fácil manipulação

2 Ferramentas Utilizadas

No desenvolvimento de sistemas web existe o uso de ferramentas tanto para desenvolver o sistema no *client-side* quanto no *server-side*, porém como o nosso sistema consiste de um visualizador foi necessário utilizar somente ferramentas do *client-side* pois as tecnologias utilizadas estão muito avançadas. Para o desenvolvimento do visualizador em questão, foram utilizadas diversas ferramentas e frameworks existentes para proporcionar um sistema seguro, de qualidade e que possa ser desenvolvido no tempo adequado, entre elas podemos citar: JavaScript, HTML5, CSS3, VueJS, MaterializeCSS, dentre outras ferramentas, além de utilizar o editor NeoVim para auxiliar no desenvolvimento.

2.1 JavaScript

O JavaScript é a linguagem mais comum baseada em scripts no *client-side*, é de baixa tipagem e orientada a objetos baseada em protótipos. Por ser executada no cliente, é possível fazer alterações no HTML de maneira mais rápida, porém sem toda a liberdade que é possível no servidor, como por exemplo acesso a banco de dados [5].

2.1.1 ECMAScript 6

ECMAScript é uma especificação e padronização de linguagens script (JavaScript) feito pela *Ecma International* [6].

Na aplicação construída foi utilizada a versão 6 (ES6) que é a última versão do JavaScript que disponibiliza diversas funcionalidade, como por exemplo os imports que foram amplamente utilizados.

2.1.2 ESLint

É um ferramenta de análise de código JavaScript muito utilizado em desenvolvimento de software, o seu principal uso é para a detecção de erros e padronização [7].

Neste projeto ela foi utilizada principalmente na padronização do código, foi decidido utilizarmos o mesmo padrão utilizado pelo Google, com por exemplo é obrigatório colocar “;” ao final de cada instrução (o que é opcional no JavaScript), e também foram feitas algumas customizações, como não poder misturar tabs e espaços na indentação.

2.2 HTML5

É uma linguagem de marcação de Hipertexto interpretada pelos navegadores e fundamental para o desenvolvimento web. A linguagem usa *tags* em seu desenvolvimento e é interpretada no *client-side*. É uma linguagem de fácil acesso e aprendizado pois não precisa mais do que um navegador para executar os programas desenvolvidos em HTML [8].

Foi utilizada amplamente para criação do “esqueleto” da página, já que o posicionamento dos elementos e estilo foram responsabilidade do CSS e do MaterializeCSS.

2.2.1 LocalStorage

Antes do HTML5 a única estrutura em que era permitido salvar dados eram nos *cookies*, com a vinda do HTML5 vieram diversas estruturas dentre elas o LocalStorage que age como um banco de dados no browser, sendo possível salvar até 5 megabytes de dados [9].

Como o intuito do projeto era criar algo que fosse fácil de ser instalado e mantido, foi decidido não utilizar um banco de dados, e por causa disso decidimos utilizar o LocalStorage onde foram salvos os dados.

2.3 CSS

O CSS é uma linguagem de estilos para documentos de marcação como HTML e XML, sendo assim é possível separar o conteúdo do documento, do estilo do mesmo. É preciso ficar atento ao usar pois nem todos estilos da linguagem funcionam da mesma maneira em todos navegadores, sendo assim é preciso garantir o funcionamento em todos os navegadores que serão utilizados [10].

O CSS nesse projeto com o intuito de realizar pequenos ajustes já que o MaterializeCSS possui a maioria dos componentes já prontos.

2.4 VueJS 2

Vue é um framework JavaScript em que o foco é construir user interfaces. O Vue é dividido em várias bibliotecas como (vue-router, vuex, vue-loader) e cada uma delas traz funcionalidade específicas que falaremos em seguida. A biblioteca *core* é focada somente na camada do usuário, ou seja, na criação de componentes mais complexos e genéricos [11].

A utilização do VueJS nesse projeto foi facilitar a criação de componentes mais genéricos por exemplo o componente de paginação, este componente já existia no Materi-

alizeCSS, porém ele foi generalizado de tal forma que ao passar determinados parâmetros ele tinha um comportamento diferente e mais adequado a página que estava. Foi utilizado também para criar componentes mais complexos, como por exemplo o componente *Questionnaire* que é a junção de diversas telas para formar o protocolo de análise cognitiva.

2.4.1 Vue-router

Como dito *Single Page Application* faz com que as páginas sejam carregadas dinamicamente e router faz com que determinado componente seja carregado baseado na URL acessada [12].

Foi utilizado para determinar baseado na URL qual view e com quais componentes deveriam ser carregados, por exemplo se o usuário acessar `/upload` irá renderizar a view *UploadView* que possui o componente Upload.

2.4.2 Vuex

Vuex é um gerenciador de estado de objetos baseado na arquitetura Flux desenvolvido e utilizado pelo Facebook. Ele funciona como uma variável global para todos os componentes, porém com regras de acesso que asseguram que o objeto pode somente ser mudado através de métodos [13].

Na figura 1 podemos observar o seguinte fluxo um determinado componente quer fazer uma alteração, portanto dispara o *Dispatch* que irá chamar uma *Action* que pode ou não chamar algo do *Backend* essa *Action* irá chamar um *Commit* de uma *Mutation* (As mutações podem ser acompanhadas pelo Devtools) fazendo com que o *State* do objeto mude sendo renderizado no componente.

O Vuex foi utilizado para dividir o acesso a lista de protocolos respondidos pelos pacientes, por exemplo na view *ListQuestionnaireView* lista os protocolos respondidos, enquanto que na view *QuestionnaireView* detalha protocolo selecionado.

2.4.3 Vue-loader

Vue loader é uma biblioteca que faz com que toda a criação de componentes seja feita em um arquivo [15]. O vue-loader foi utilizado para facilitar o desenvolvimento, caso ele não tivesse sido utilizado, todo o template deveria ser declarado dentro de uma variável.

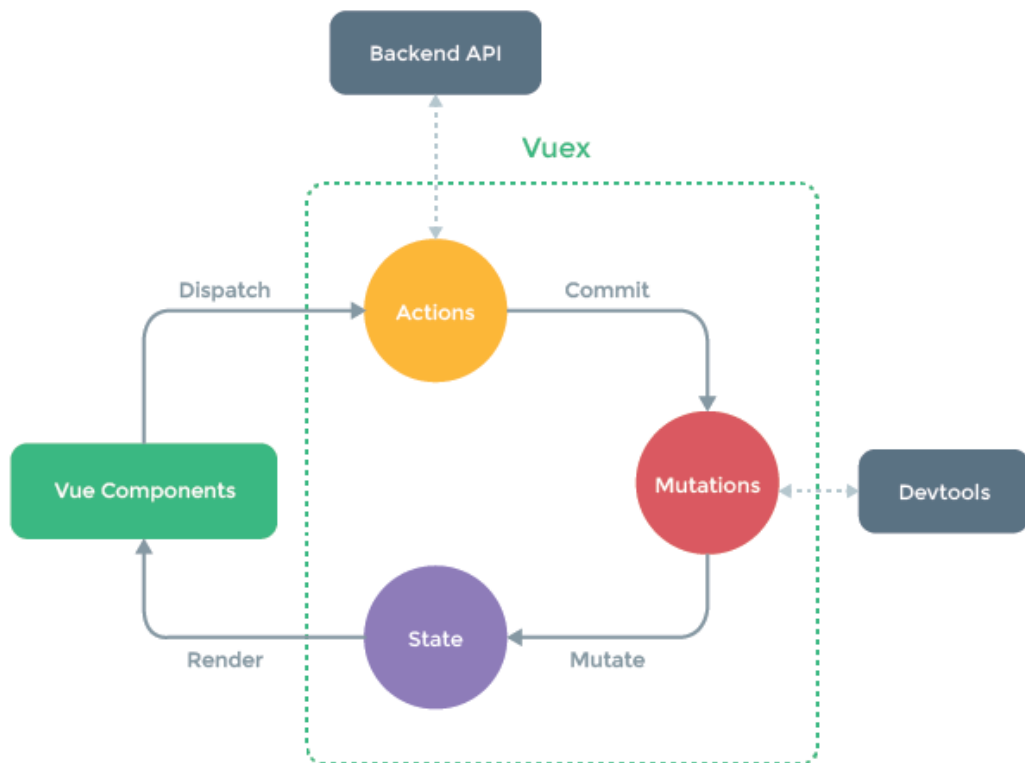


Figura 1 – Vuex [14]

2.5 BabelJS

Babel é um transpilador de ECMAScript 6 para ECMAScript 5, ou seja, com ele podemos utilizar recursos da ES6 enquanto os navegadores tem somente suporte para o ES5 [16].

Como a aplicação foi utilizada usando ES6 (ECMAScript 6) e a maioria dos browsers não possui o suporte total para essa versão, o babel foi um ferramenta essencial para transpilar esse código em uma versão do JavaScript (es5) em que todos os browsers (após 2009) possuem suporte.

2.6 MaterializeCSS

É um framework *open source* baseado no *Material Design* desenvolvido pelo Google. O framework utiliza CSS e JavaScript para facilitar a criação de elementos da parte visual da tela, sendo possível também alterar e estender a funcionalidade dos elementos quando for necessário. Foi utilizado amplamente no projeto, principalmente por causa

dos seus componentes já estilizados como por exemplo botões, o menu de navegação lateral [17].

O framework ajudou bastante também na hora de fazer com que o layout fosse responsivo, sendo necessário nenhum ajuste para que o mesmo layout funcionasse no *mobile*.

2.7 Git

É um controlador de versão *open source*, que é amplamente utilizado pela comunidade *open source*. O Git foi criado pelo Linus Torvalds em 2005 para o desenvolvimento do kernel do Linux [18].

O Git foi muito utilizado no projeto para gerenciar os *commits*, sendo que os *commits* representa uma funcionalidade nova ou um ajuste, também foi utilizado na hora de achar bugs, por exemplo em um determinado momento o menu lateral deixou de ser responsivo (sumir quando a tela era pequena) foi necessário reverter o código até um *commit* em que ele funcionava e entender o que foi introduzido que “quebrou” a funcionalidade.

2.7.1 GitHub

O GitHub é um repositório do Git que também é amplamente utilizado pela comunidade *open source*. Ele provê todas as funcionalidade do Git como SCM (*source control management*) e adiciona as suas funcionalidade como *bug tracking*, *feature request*, entre outros [19].

O GitHub foi utilizado principalmente com um repositório do código, não sendo utilizado as outras funcionalidade como bug tracking.

2.8 Webpack

Webpack é um *module bundler*, ou seja ele pega módulos com dependências e gera arquivo estático que representam esses módulos, como pode ser visto na figura 2 [20].

Foi utilizado principalmente para poder utilizar o vue-loader no desenvolvimento, e possibilitando assim que todo o JavaScript utilizado na aplicação fosse exportado para um arquivo somente (bundle.js).

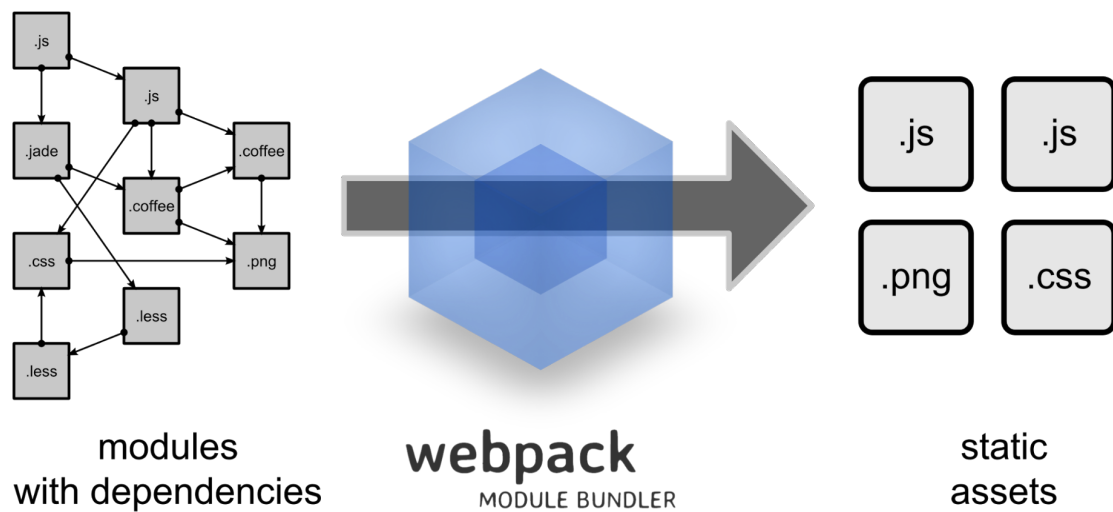


Figura 2 – Webpack [21]

2.9 NPM

O NPM é um gerenciador de dependências de *node_modules* por exemplo: Vue, MaterializeCSS, entre outros. Ele facilita a instalação e compartilhamento do código, uma vez que ele gera um arquivo de dependências do projeto [22].

Foi utilizado principalmente para gerenciar as dependências do projeto.

2.10 XML

XML (*Extensible Markup Language*) é uma linguagem de marcação e muito flexível utilizada para armazenamento de dados e transporte [23].

O XML foi utilizado para representar, armazenar e exportar os dados entre as aplicações, ou seja, os dados exportados pela aplicação são no formato XML e o formato de entrada no visualizador é XML. Isto permitiu fácil intercâmbio dos dados entre o aplicador da bateria de testes e o pesquisador que irá visualizar estes dados.

Para um melhor entendimento do que é um XML na figura 4 podemos ver um objeto que representa uma anotação, figura 3 conseguimos ver a representação da sua classe em Java. Como observado na figura 4 vemos que a primeira *tag* representa qual versão do XML está sendo utilizada, nesse caso a 1.0, na sequência vemos a *tag* Anotação que indica qual elemento está sendo descrito, posteriormente podemos ver os atributos (para, de, cabeçalho e corpo) desse elemento e seus valores, basicamente o XML descreve elementos seguindo a seguinte lógica: `<atributo> valor </atributo>`.

```
class Anotacao {  
    String Para;  
    String De;  
    String Cabecalho;  
    String Corpo;  
}
```

Figura 3 – Classe anotação

```
<?xml version="1.0" encoding="UTF-8" ?>  
<Anotacao>  
    <Para>Augusto</Para>  
    <De>Camila</De>  
    <Cabecalho>Uberlândia 09/11/12</Cabecalho>  
    <Corpo>Isto é um JSON</Corpo>  
</Anotacao>
```

Figura 4 – Exemplo XML classe anotação

2.10.1 XML Schema

É a descrição de um XML onde são especificados os tipos dos campos, com isso temos como validar tanto na hora da importação como na exportação se o XML salvo obedece às regras que foram impostas [24].

XML Schema facilitou bastante na hora de compartilhar os dados entre as aplicações já que por sua vez ele determinou a estrutura detalhada das informações a serem geradas e processadas posteriormente.

Basicamente para definir um XML Schema devemos seguir os seguintes passos:

1. Definir os atributos que podem aparecer no elemento.
2. Definir numero de o número de elementos filho.
3. Definir tipo de cada atributo.
4. Definir o os valores padrão dos elementos e dos seus atributos.

Na figura 5 podemos ver um exemplo de XML Schema, neste exemplo a primeira *tag* que podemos observar é a que define a versão do XML, atualmente a versão 1.0, na sequencia é utilizado a *tag* que define que este XML representa um XML Schema. Nesse XML Schema iremos representar uma classe Anotação, na figura 3 podemos ver a sua representação em Java, primeiramente iremos definir um elemento elemento raiz (que no caso é o Anotação) através da *tag* `xs:element`, posteriormente iremos dizer que esse elemento é compostos por outros tipo, para isso iremos utilizar a *tag* `xs:complexType`

(que indica que este elemento é composto por outros elementos) e `xs:sequence` (informa que os atributos devem aparecer nessa sequencia, essa *tag* não é obrigatória), e finalmente iremos definir os atributos da classe.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="anotacao">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="para" type="xs:string"/>
        <xs:element name="de" type="xs:string"/>
        <xs:element name="cabecalho" type="xs:string"/>
        <xs:element name="corpo" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

Figura 5 – Exemplo classe anotação XML Schema

2.11 JSON

JSON (*JavaScript Object Notation*) é um formato de armazenamento e transporte de dado e é o formato mais amplamente utilizado no desenvolvimento web [25].

Foi utilizado amplamente na hora de manipular os dados no JavaScript, já que por sua vez o JavaScript não entende XML nativamente.

O exemplo a seguir irá ajudar no entendimento de como um objeto JSON é construído, como podemos observar na figura 6 ele representa um objeto da classe da figura 3 (um objeto do tipo anotação) o JSON não possui tipos, todos os valores e atributos são representados como *strings* (como pode ser notada na figura 6), a definição de objeto é tudo que está entre chaves e todo atributo é seguido por seu valor por exemplo: “atributo”: “valor”.

2.11.1 X2JS

É uma biblioteca que utilizamos para transformarmos um XML para JSON (e vice-versa), para que assim pudéssemos manipular os dados que foram salvos em um arquivo XML [26].

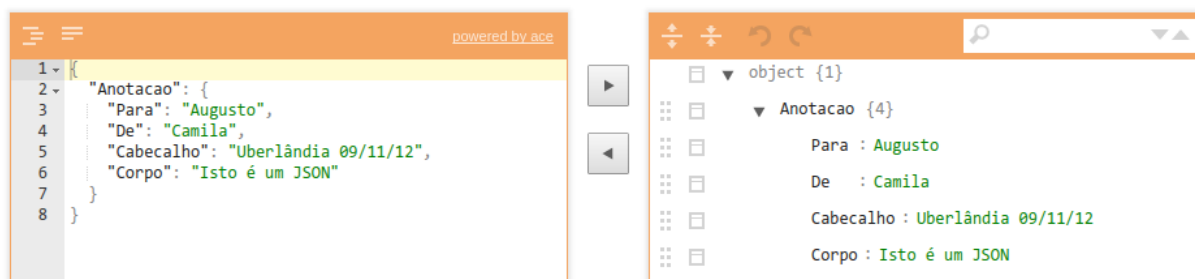


Figura 6 – Exemplo objeto anotação JSON

Esta biblioteca facilitou muito na hora de manipular os dados importados, sem ela isso não seria possível, pois ela transforma os dados XML em JSON podendo assim eles serem transformados em objetos e por conseguinte serem manipulados.

3 Desenvolvimento

A metodologia de desenvolvimento adotada para esse projeto foi a em cascata, este é um modelo sequencial no qual cada etapa deve ser concluída antes de ser dado início a outra como podemos ver na figura 7 [27]. A seguir será detalhado cada uma das fases do modelo:

1. **Requerimento:** Nesta etapa são desenvolvidos os requisitos do produtos, e como ele deve ser. Com isso nessa etapa foi gerado o documento com todas as telas que o sistema deve ter, conseqüentemente foi feito uma validação dos requisitos junto ao interessado juntamente com a prototipação das telas. Ao final dessa etapa foi observado que o sistema possuía todas as funcionalidades que o usuário desejava.
2. **Projeto:** São definidos os requisitos que permitam a codificação do que foi estabelecido no requerimento. Nesta etapa foram definidos quais frameworks, linguagens e bibliotecas seriam utilizados, e também foi definido o uso do XML Schema para a persistência dos dados.
3. **Implementação:** Nesta etapa é criada a aplicação propriamente dita. Ao final dela temos a aplicação pronta para ser testada.
4. **Verificação:** Com essa etapa podemos realizar os testes mas não com o cliente, e assim fazer os ajustes. Ao final dela temos o produto pronto para ser testado pelo cliente.
5. **Implantação:** Instalação do sistema no ambiente de uso, treinamento de usuários, utilização do sistema pelo cliente, ao final desta etapa temos o sistema homologado pelo cliente podendo ser assim entregue.
6. **Manutenção:** Consiste na correção de erros e melhorias, que não foram detectadas em outras etapas.

O desenvolvimento deste visualizador teve com previsão um intervalo de seis meses, sendo desenvolvido somente por um programador.

O sistema foi separado em funcionalidades, onde cada funcionalidade foi mapeado para uma tela e deveria ser capaz de funcionar separadamente. A análise das funcionalidades e das telas foi feito em conjunto com os pesquisadores envolvidos somente depois de consolidada a documentação e validado os protótipos foi dado início ao desenvolvimento.

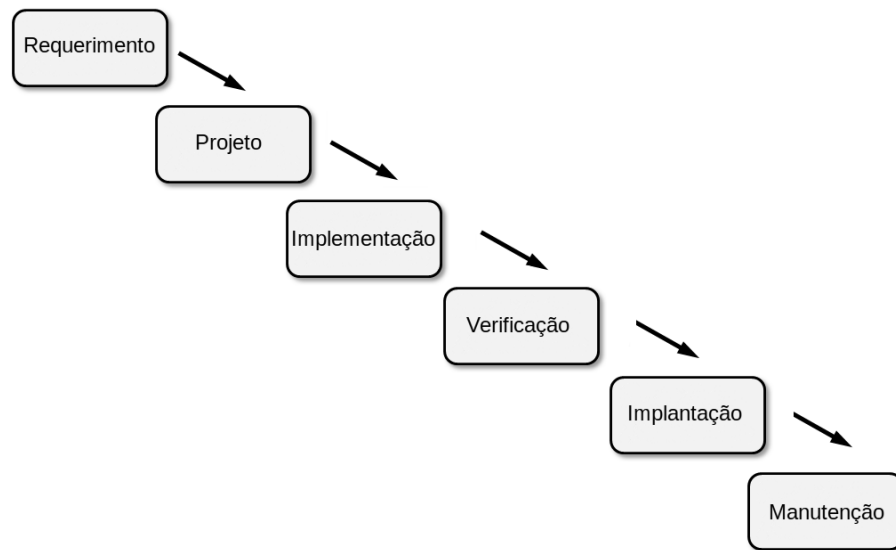


Figura 7 – Modelo em Cascata

3.1 Implementação e Testes

A fase de implementação e teste serão citadas no mesmo tópico, pois as mesmas são paralelizadas durante o desenvolvimento, ou seja, não existe uma fase separada somente para o desenvolvimento e uma somente para testes, foi definido assim pois o desenvolvimento e as correções conseguiriam ser mais rápidas e eficazes, agilizando assim o processo.

A seguir será apresentado um aparato geral de como é o fluxo da aplicação. No diagrama de atividade na figura 8 podemos ver de como é o fluxo da aplicação do questionário, ao final foi obtido o XML dos questionários aplicados que serão a entrada para a aplicação do visualizador.

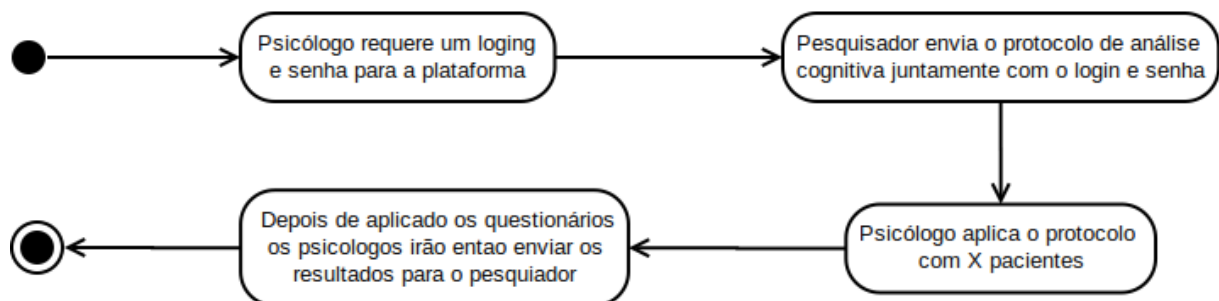


Figura 8 – Bateria de testes

Na figura 9 onde é representado o diagrama de atividade do visualizador, temos uma visão de como o sistema funciona e as suas principais funcionalidades, como: fazer *upload* dos arquivos, visualizar os protocolos respondidos, entre outras.

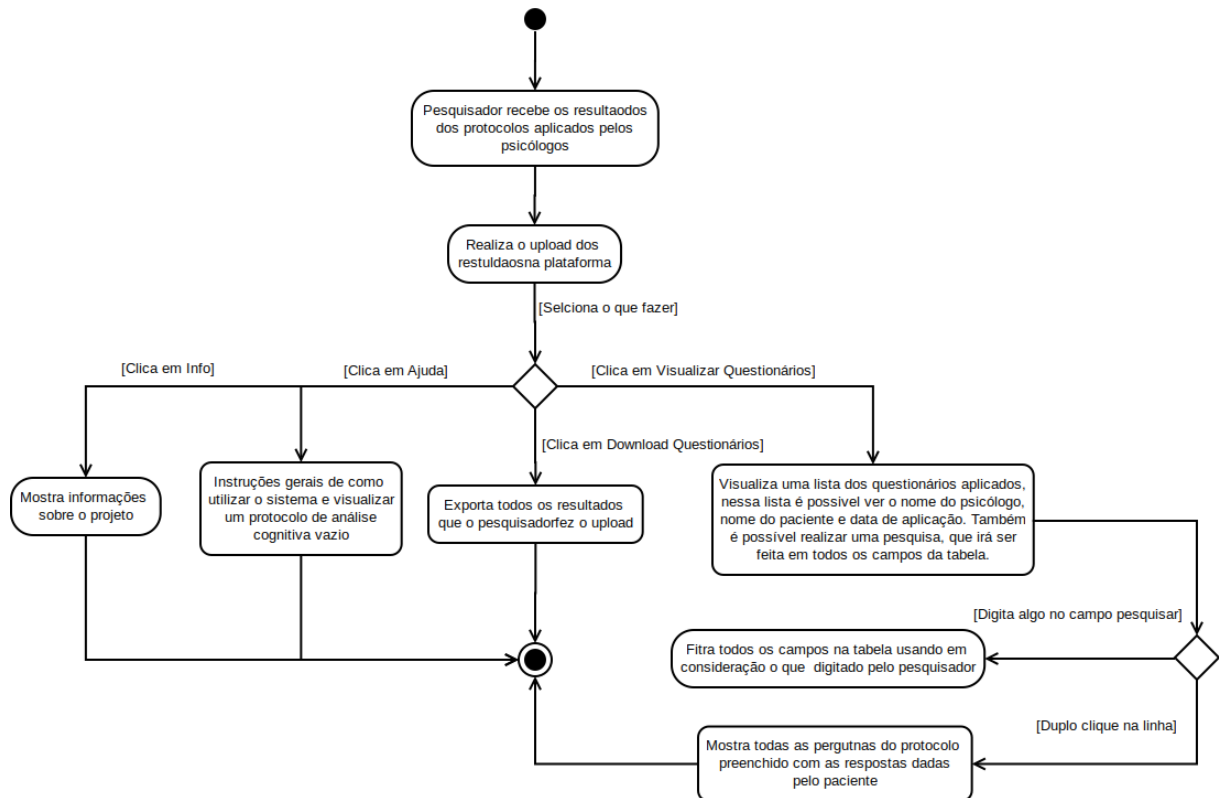


Figura 9 – Visualizador

Após a definição das telas pelos pesquisadores envolvidos, foi possível realizar o desenvolvimento de um protótipo inicial que foi validado junto aos usuários.

Em seguida foi definido o ponto mais importante da plataforma, o XML Schema, que é o modelo dados utilizado pela bateria de testes e pelo visualizador. Se não fosse definido este elemento primeiramente, teria acarretado em sérios problemas na hora de compartilhar os dados da bateria de testes com o visualizador.

Para a comunicação entre as aplicações foi escolhido utilizar a linguagem XML pois possui uma tipagem mais forte que no caso o JSON, uma outra alternativa, para que a tipagem dos dados fosse mantida entre as aplicações foi o utilizado o XML Schema, na figura 10 podemos observar o XML Schema utilizado pela aplicação, onde cada tela tem seus atributos e dentro dela existem os atributos que representam as respostas das questões que existem naquela tela.

Na figura 11 podemos observar mais claramente como cada atributo está mapeado para um campo na tela, e seus respectivos tipos. Em outras telas podemos encontrar outros tipos de dados como byte, date, entre outros. O único problema de utilizar o XML e o XML Schema foi que o JavaScript não tem suporte nativo a eles, por isso utilizamos uma biblioteca chamada X2JS que pega um objeto no formato XML e converte para um JSON no qual o JavaScript tem total suporte.

```

<xs:element name="Pacientes">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Paciente" maxOccurs="unbounded" minOccurs="0">
        <xs:complexType>
          <xs:sequence>
            <xs:element type="xs:string" name="Nome"/>
            <xs:element type="xs:byte" name="TelaAtual"/>
            <xs:element type="xs:byte" name="Flag"/>
            <xs:element name="Tela2">
              <xs:complexType>
                <xs:sequence>
                  <xs:element type="xs:date" name="dataAplicacao"/>
                  <xs:element type="xs:string" name="nomeAvaliado"/>
                  <xs:element type="xs:date" name="dataNascimento"/>
                  <xs:element type="xs:string" name="lateralidade"/>
                  <xs:element type="xs:string" name="escolaridade"/>
                  <xs:element type="xs:string" name="trabalho"/>
                  <xs:element type="xs:string" name="trabalhoAtual"/>
                  <xs:element type="xs:string" name="naturalidade"/>
                  <xs:element type="xs:string" name="sexo"/>
                  <xs:element type="xs:string" name="limitacaoCorporal"/>
                  <xs:element type="xs:string" name="problemaVisual"/>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
            <xs:element name="Tela3">
              </xs:element>
            <xs:element name="Tela4">
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  ...

```

Figura 10 – XML Schema

Você consegue realizar as seguintes atividades de vida diárias com independência?

Tomar banho?	Vestir-se	Comer
Sim	Sim	Sim

Tomar medicação	Locomover-se dentro de casa	Ir a algum lugar fora de casa
Não	Sim	Não

Preparar a própria refeição	Fazer compras	Resolver problemas financeiros
Sim	Sim, com ajuda	Sim, com ajuda

Você tem obrigações diárias na rotina familiar/nas tarefas de casa?

Não

Descrição: N/A

Você tem obrigações diárias fora de casa?

Não

Descrição: N/A

Navigation: < 1 2 3 4 5 6 7 8 9 10 >

Figura 11 – Tela e XML Schema

Posteriormente foram definidas quais as tecnologias a ser utilizadas no sistema (capítulo 2), juntamente com o detalhamento das telas será explicado porquê de cada tecnologia escolhida.

A ferramenta que facilitou bastante esse projeto foi o NPM, pois ele age como

um gerenciador de dependências para o projeto, facilitando assim o compartilhamento do mesmo, ele também facilita na hora de instalar as dependências, já que por sua vez ele mantém a versão que foi utilizada no desenvolvimento, como podemos ver o JSON de configuração do NPM na figura 12. Podemos ver todas as dependências do projeto que estão dentro do objeto *dependencies* e suas respectivas versões, dentro do objeto *devDependencies* são encontradas as dependências de desenvolvimento, ou seja, programas que são necessários para dar *build* no projeto.

```
{
  ...
  "dependencies": {
    "material-design-icons": "^3.0.1",
    "materialize-css": "^0.97.8",
    "npm": "^3.10.10",
    "vue": "^2.0.5",
    "vue-router": "^2.0.1"
  },
  "devDependencies": {
    "babel-core": "^6.18.2",
    "babel-loader": "^6.2.7",
    "babel-plugin-transform-runtime": "^6.1.2",
    "babel-preset-es2015": "^6.18.0",
    "babel-preset-stage-0": "^6.1.2",
    "babel-runtime": "^6.18.0",
    "css-loader": "^0.25.0",
    "eslint": "^3.9.1",
    "eslint-config-google": "^0.6.0",
    "eslint-plugin-vue": "^0.1.1",
    "image-webpack-loader": "^3.0.0",
    "node-sass": "^3.11.2",
    "sass-loader": "^4.0.2",
    "style-loader": "^0.13.0",
    "vue-hot-reload-api": "^2.0.6",
    "vue-html-loader": "^1.0.0",
    "vue-loader": "^9.8.1",
    "webpack": "^1.13.3",
    "webpack-dev-server": "^1.12.0"
  }
}
```

Figura 12 – Pacote NPM

Posteriormente foi dado início ao desenvolvimento das telas. Inicialmente foram feitos os menus laterais como pode ser visto na figura 14 e na figura 15. O MaterializeCSS teve grande participação nessa parte. Sendo ele o responsável por garantir a responsividade em seus componentes como pode ser observado na figura 15,, em telas com resoluções menores, por exemplo em um dispositivo mobile, a tela esconde o menu lateral. O framework que agilizou o desenvolvimento foi o VueJS, principalmente o *vue-router* que ajudou a mapear as rotas com as *views* que possuem os componentes, como pode ser observado na figura 13. Outra responsabilidade do *MaterializeCSS* foi garantir o estilo das páginas, como pode ser observado ele segue diretamente as diretrizes do *material design* isso pode ser observado nas cores escolhidas, nos ícones, posicionamento dos componentes, entre

outros.

```
import Vue from 'vue';
import VueRouter from 'vue-router';

import Home from '../view/HomeView.vue';
import Upload from '../view/UploadView.vue';
import ListQuestionnaire from '../view/ListQuestionnaireView.vue';
import Questionnaire from '../view/QuestionnaireView.vue';

Vue.use(VueRouter);

export default new VueRouter({
  routes: [
    { path: '/', component: Home },
    { path: '/upload', component: Upload },
    { path: '/list', component: ListQuestionnaire },
    { path: '/questionnaire', component: Questionnaire }
  ]
});
```

Figura 13 – Vue-router

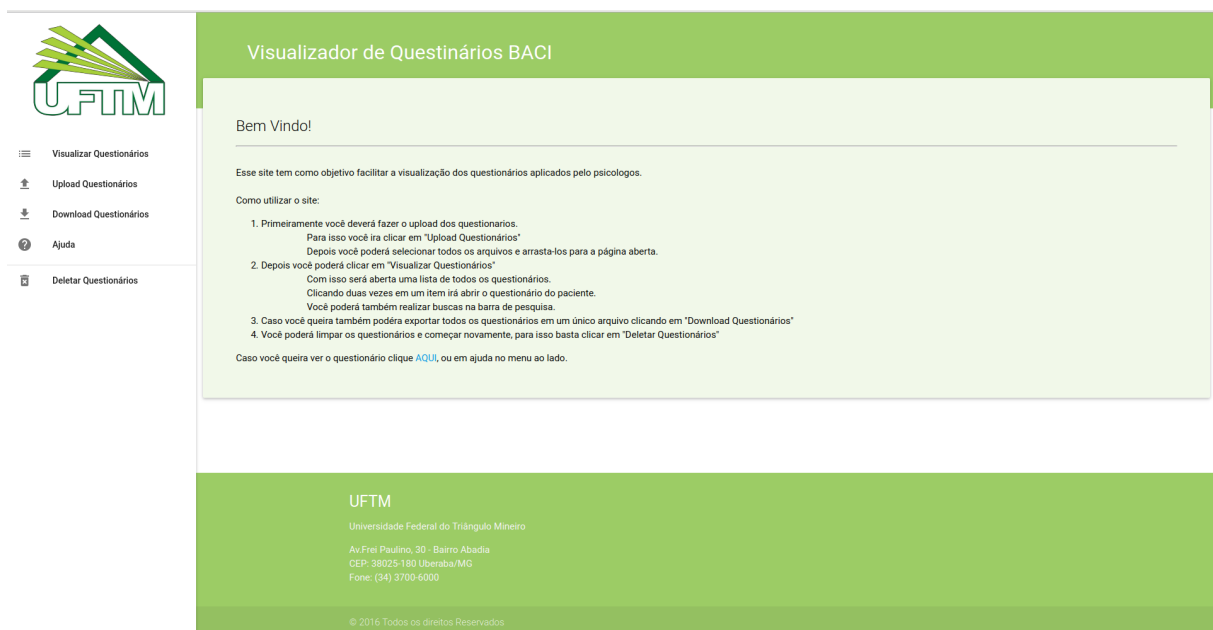


Figura 14 – Tela principal em um dispositivo com tela grande

Após finalizado o menu lateral, foi desenvolvido a tela principal *Home*, como também pode ser observada na figura 14 onde é ensinado como deverá ser utilizado o sistema, e qual é o intuito do sistema.

Em seguida foi desenvolvido a tela de *Upload*, como pode ser observado na figura 16, onde o principal objeto desta tela é realizar o upload das respostas da bateria de testes, para fazer o *upload* os arquivos podem ser arrastados, caso seja suportado pelo navegador, ou selecionados (todo o navegador possui suporte a esse tipo de *upload*). Nesse momento foi decidido utilizar o *vuex* para centralizar o acesso a informação do resultado

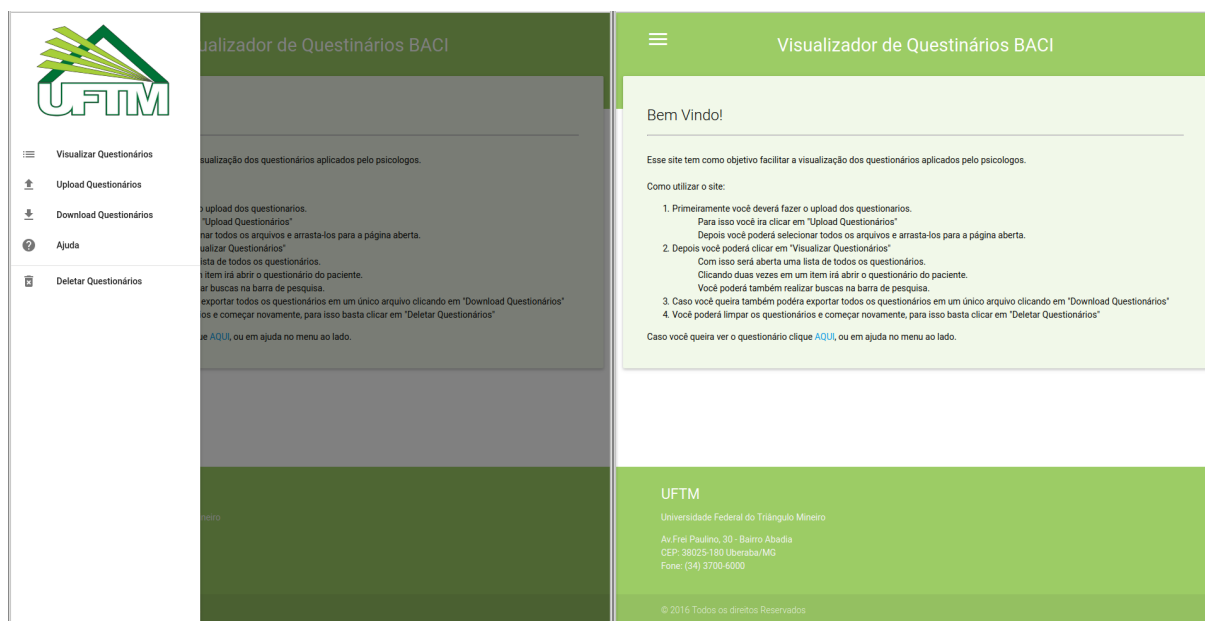


Figura 15 – Tela principal em um dispositivo mobile

dos questionários, e também foi utilizado o `LocalStorage`, quando é feito o *upload* do arquivo automaticamente o arquivo é salvo no `LocalStorage`, decidimos utilizar isso para que quando o usuário fechasse a tela não fosse necessário fazer o *upload* do XML novamente.

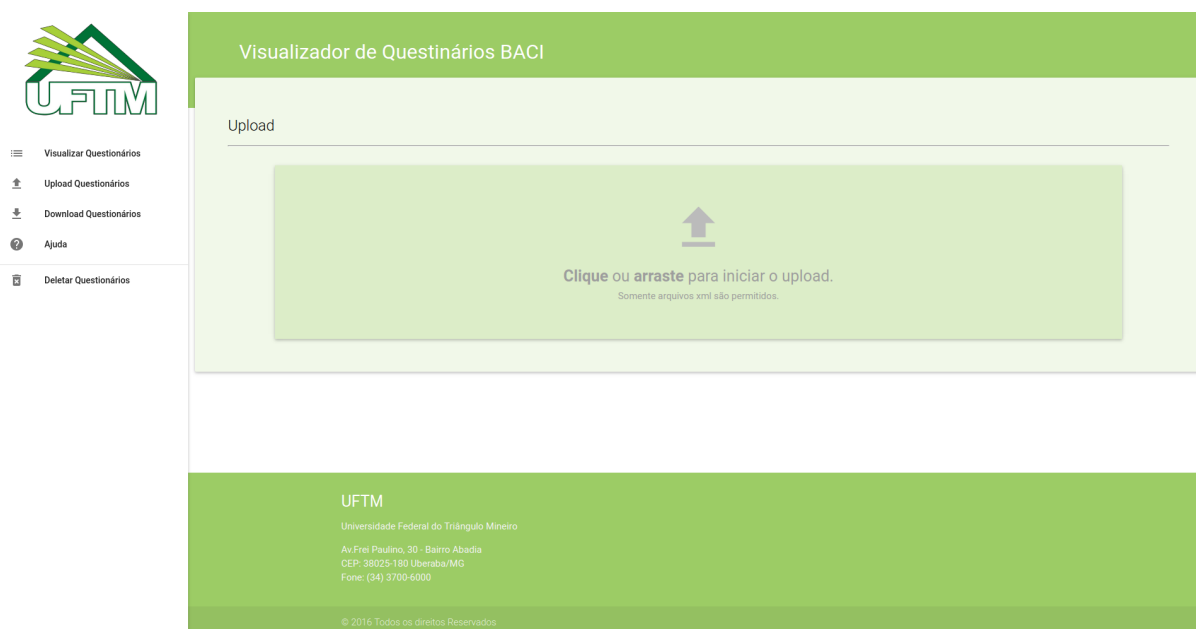
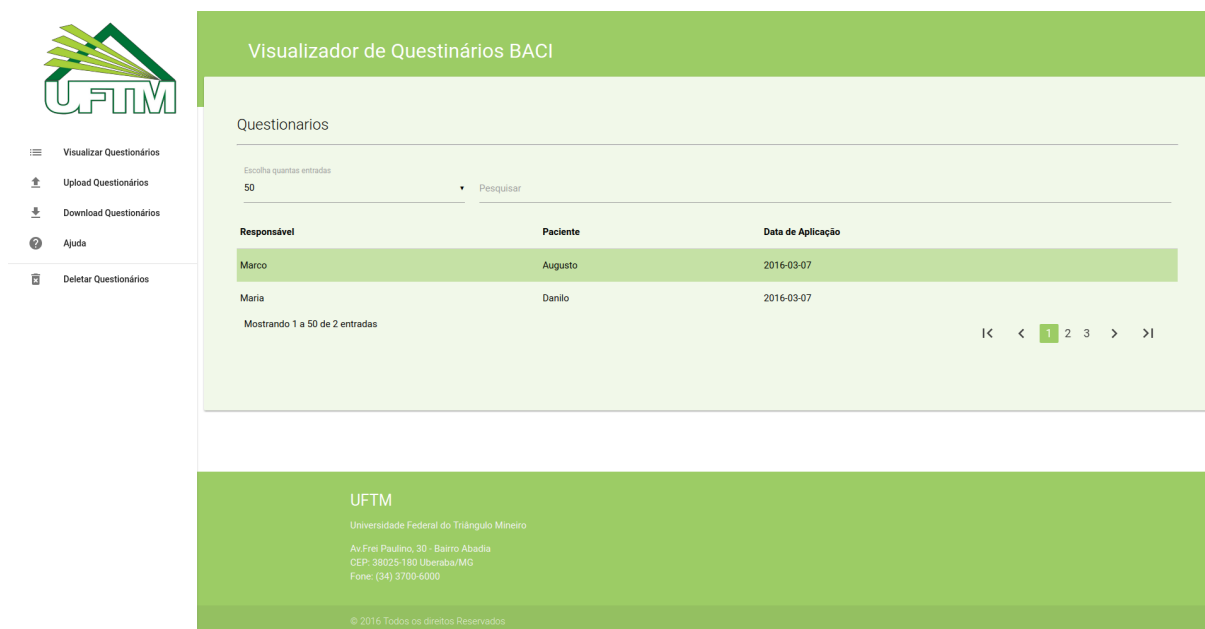


Figura 16 – Tela de upload de arquivos

Na figura 17 é observado uma tabela com todos os protocolos aplicados, nesta tela conseguimos também realizar a pesquisa por determinado valor em toda a tabela, por exemplo, ao digitar 'Augusto' no *input* de pesquisar, e será feito a pesquisa em todos os campos da tabela, ou seja, todos os campos que tiverem o valor 'Augusto' serão mostrados, também é possível selecionar quantos elementos serão mostrados por página, no caso o

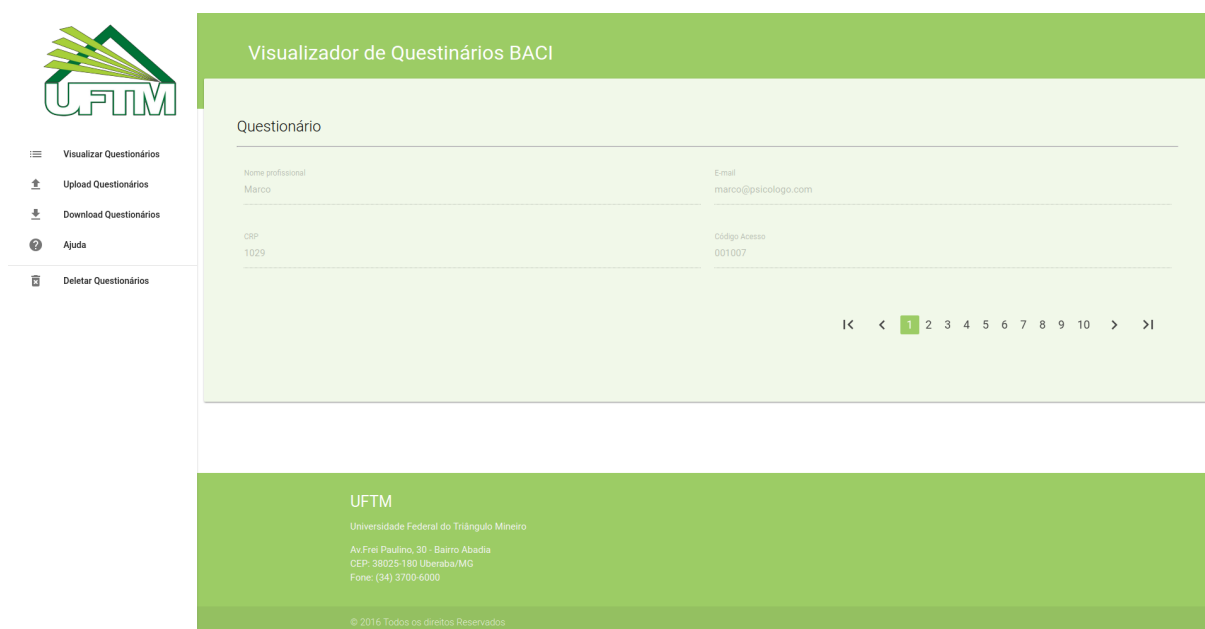
valor padrão é 50, mas poderia ser 10 ou 100. Ao realizar um clique duplo em qualquer uma das linhas da tabela iremos para a tela que irá mostrar o protocolo respondido detalhadamente com todas as questões, como demonstrado observar na figura 18.



The screenshot shows the 'Visualizador de Questionários BACI' interface. On the left is a sidebar with the UFTM logo and navigation links: Visualizar Questionários, Upload Questionários, Download Questionários, Ajuda, and Deletar Questionários. The main area displays a table of questionnaires. At the top, there's a search bar with 'Escolha quantas entradas' set to 50 and a 'Pesquisar' button. The table has three columns: Responsável, Paciente, and Data de Aplicação. It shows two entries: Marco Augusto (2016-03-07) and Maria Danilo (2016-03-07). Below the table, it says 'Mostrando 1 a 50 de 2 entradas'. At the bottom right of the table area is a pagination control: I < 1 2 3 > >I. The footer contains UFTM contact information and a copyright notice: © 2016 Todos os direitos Reservados.

Responsável	Paciente	Data de Aplicação
Marco	Augusto	2016-03-07
Maria	Danilo	2016-03-07

Figura 17 – Tabela de questionários



The screenshot shows the 'Visualizador de Questionários BACI' interface for a specific questionnaire. The sidebar is the same as in Figure 17. The main area is titled 'Questionário' and contains a form with two columns. The left column has fields for 'Nome profissional' (Marco) and 'CRP' (1029). The right column has fields for 'Email' (marco@psicologo.com) and 'Código Acesso' (001007). At the bottom right of the form area is a pagination control: I < 1 2 3 4 5 6 7 8 9 10 > >I. The footer is the same as in Figure 17.

Figura 18 – Questionário completo

A criação de componentes é parte essencial do framework VueJS, será mostrado aqui um dos componentes que foram desenvolvido o *Pagination* que foi compartilhado pelas telas de tabela de questionário e a de questionário, como podemos ver na figura 19, o componente tem diversos parâmetros um deles é o *chunkSize* que é responsável por

dizer quantas páginas serão mostradas, como visto na figura 17 são 3 e na figura 18 são 10, por isso é importante a criação de componentes, pois um componente bem escrito e estruturado pode ser reutilizado várias vezes ao longo do projeto

As bibliotecas que facilitaram o desenvolvimento desses componentes foram o vue-loader, que possibilita a criação de componentes em um único arquivo separado (arquivo com extensão vue) que conterá toda a estrutura do componente (HTML, JavaScript e CSS), o Webpack que gerou o arquivo bundle.js com todas as dependências, baseado na ordem de importação dos arquivos, e também o BabelJS que traduziu instruções escritas em ES6 (o padrão de componentes no VueJS) para o ES5 que é totalmente suportado pelos navegadores.

```
<template>
  <div class="row">
    <div class="col s12 right-align">
      <ul class="pagination">
        <li @click="go(1)" class="waves-effect"><a><i class="material-icons">first_page</i></a></li>
        <li @click="previous" class="waves-effect"><a><i class="material-icons">chevron_left</i></a></li>
        <li v-for="page in pagesDisplayed" class="waves-effect" :class="{ active: page == currPage}">
          <a @click="go(page)" > {{ page }} </a>
        </li>
        </li>
        <li @click="next" class="waves-effect"><a><i class="material-icons">chevron_right</i></a></li>
        <li @click="go(totalPages)" class="waves-effect"><a><i class="material-icons">last_page</i></a></li>
      </ul>
    </div>
  </div>
</template>

<script>
  export default {
    name: 'my-pagination',
    props: {
      startPage: {
        type: Number,
        default: 1,
        validator: function(value) {
          return value > 0;
        },
      },
      totalPages: {
        type: Number,
        required: true,
      },
      // Amount of pages to be displayed each time
      chunkSize: {
        type: Number,
        default: 3,
      },
      // This callback will be called on every page update.
      callback: {
        type: Function,
        required: true,
      },
    },
    methods: {
      ...
    },
    data: function() {
      ...
    },
    computed: {
      ...
    },
    mounted: function() {
      ...
    },
  },
};
</script>

<style lang="sass" scoped>
</style>
```

Figura 19 – Componente de paginação

Outra ferramenta muito importante utilizada no projeto foi o Eslint, que possibilitou manter uma padronização de desenvolvimento em todo o projeto e a detecção de erros, como podemos ver na figura 20, ele demonstra erros de padronização de código em cada arquivo e linha, como se fosse um compilador

```
$ eslint .

/home/augusto/workspace/tcc_fabiano/projeto/visualizador/src/main.js
  8:13  error  Expected parentheses around arrow function argument  arrow-parens
  8:24  error  Missing trailing comma                                comma-dangle

/home/augusto/workspace/tcc_fabiano/projeto/visualizador/src/router/router.js
  16:61  error  Missing trailing comma  comma-dangle
  17:6   error  Missing trailing comma  comma-dangle

/home/augusto/workspace/tcc_fabiano/projeto/visualizador/webpack.config.js
   6:30  error  Missing trailing comma  comma-dangle
  13:40  error  Missing trailing comma  comma-dangle
  17:30  error  Missing trailing comma  comma-dangle
  23:87  error  Missing trailing comma  comma-dangle
  24:18  error  Missing trailing comma  comma-dangle
  25:14  error  Missing trailing comma  comma-dangle
  26:10  error  Missing trailing comma  comma-dangle
  31:68  error  Missing trailing comma  comma-dangle
  32:10  error  Missing trailing comma  comma-dangle
  35:19  error  Strings must use singlequote  quotes
  35:29  error  Strings must use singlequote  quotes
  36:19  error  Strings must use singlequote  quotes
  36:39  error  Missing trailing comma  comma-dangle
  37:6   error  Missing trailing comma  comma-dangle

✖ 18 problems (18 errors, 0 warnings)
```

Figura 20 – Erros do Eslint

Uma ferramenta importante em qualquer desenvolvimento de software é o *Source Control Management* (SCM), neste projeto decidimos utilizar o git, pois os desenvolvedores já possuem experiência com esse SCM e está sendo amplamente utilizada. Na figura 21 é observado alguns *commits* realizados no decorrer projeto por exemplo no *commit* e298858 podemos ver que foi introduzido o componente de paginação, basicamente cada *commit* irá representar uma funcionalidade ou uma correção ou um *merge*, o único *merge* que o projeto possui é no *commit* f0c666b onde foi criado o repositório no github, o github foi utilizado somente como um repositório online para o projeto, foi decidido utilizar o github pois assim teríamos redundância dos arquivos fontes, podemos ver o repositório no github na figura 22

```
c0b0ee4 Fixed typo on router url.
0dd0937 Component name standardization
d156a16 Organization with views
641bedb Correct package.json
beacc94 Standardization of mounted.
e298858 Added pagination component.
756b09d Added module configuration
1bc9745 Fix table
f0c666b Merge branch 'master' of https://github.com/augustomelo/BACIVisualizer
```

Figura 21 – Commits realizados

The screenshot shows the GitHub interface for the repository 'augustomelo / BACIVisualizer'. At the top, there's a search bar and navigation links for 'Pull requests', 'Issues', and 'Gist'. Below the repository name, there are statistics: 1 Unwatch, 1 Star, and 0 Fork. The main navigation bar includes 'Code', 'Issues 0', 'Pull requests 0', 'Projects 0', 'Wiki', 'Pulse', 'Graphs', and 'Settings'. The repository title is 'BACI questionnaire visualizer. — Edit'. Below this, a summary bar shows '21 commits', '1 branch', '0 releases', '1 contributor', and 'MIT' license. A toolbar offers options like 'New pull request', 'Create new file', 'Upload files', 'Find file', and 'Clone or download'. The commit history table lists the following entries:

File	Description	Time
dist	Added vue loader, webpack, hot reload and organized the project.	2 months ago
src	Added questionnaire component	25 days ago
.eslintignore	Upload component	a month ago
.eslintrc	Added Materialize as global and changed max-len to 100	a month ago
.gitignore	Index file.	2 months ago
LICENSE	Initial commit	17 days ago
README.md	Fix table	17 days ago
index.html	Upload component	a month ago
package.json	Updated packages	23 days ago
webpack.config.js	Added questionnaire component	25 days ago

Below the commit history, there's a section for 'README.md' which contains the title 'BACI XML Visualizer'.

Figura 22 – Repositório no github

3.2 Homologação

Finalizada a etapa de implementação da tela, ela está pronta para ser homologada pelos pesquisadores envolvidos, porém dada pela indisponibilidade e urgência no desenvolvimento do projeto a homologação foi deixada para somente quando o projeto fosse concluído.

3.3 Dificuldades e Desafios Encontrados

Por mais que o desenvolvedor tivesse experiência com as ferramentas escolhidas, diversos conhecimentos e experiências foram agregadas durante o desenvolvimento da aplicação. Todas as dificuldades que foram encontradas, foram resolvidas com pesquisas na internet, já que por sua vez a aplicação foi desenvolvida somente por um programador.

Mesmo o programador tendo experiência com frameworks parecidos (AngularJS) ao utilizar o VueJS, foi encontrado algumas dificuldades, por exemplo enquanto o AngularJS tenta resolver todos os problema de uma vez o VueJS tenta ser mais específico, sendo necessário adicionar bibliotecas conforme os problemas forem encontrados, tendo como exemplo a criação de rotas, enquanto no AngularJS já possui um módulo para isso e ele já faz o relacionamento entre um controller e uma view no VueJS é necessário adicionar a biblioteca vue-router que permite a criação de rotas.

Na construção de telas também houveram algumas dificuldades, já que o desenvolvedor possui experiência com outro framework (Bootstrap), mesmo eles sendo parecidos existiram algumas dificuldades na utilização e customização de alguns componentes.

Todos os desafios encontrados foram superados em sua totalidade, o que proporcionou um aprendizado enorme, o que com certeza serão aproveitados novamente.

4 Conclusão

O objeto deste trabalho que era construir um sistema web para melhorar e facilitar a visualização de questionários (BACI) respondidos por idosos, pode-se dizer que esse objetivo foi plenamente atingido, todas as telas e funcionalidades requisitadas pelo pesquisador foram feitas. Alguns benefícios que esse sistema proporcionou ao pesquisador foram: facilidade na manipulação e visualização dos dados, e também na hora de poder comparar as respostas dos questionários. Tanto a UFTM quanto a pesquisadora avaliaram o sistema de maneira positiva.

A maioria das dificuldade deste trabalho estão relacionadas com a definição e utilização das tecnologias, que por mais que o desenvolvedor tivesse experiência na área de desenvolvimento web, o mesmo não tinha um conhecimento profundo do framework escolhido (VueJS e o MaterializeCSS). Outra dificuldade encontrada foi na hora de definir o XML Schema que seria compartilhado pelas aplicações: formulário utilizado pelos psiquiatras e o visualizador utilizado pelo pesquisador, pois cada modelava o XML Schema de um jeito, ocasionalmente não solucionando a dificuldade que a outra aplicação possuía.

Como trabalhos futuros seria muito importante realizar uma análise estatística profunda nos dados e com isso tentar achar uma forma de determinar que certos grupos de pessoas com determinadas características tem a tendência de desenvolver certos problemas. Da forma que o projeto foi estruturado acreditamos que será mais fácil adicionar essa funcionalidade já que os resultados dos pacientes são manipulados em somente em um local, e caso outro programa queira fazer a leitura desses dados também será fácil já que utilizamos o XML Schema.

Pode-se concluir que ao fim do projeto, pude aprender diversas técnicas novas de programação para a web que não são passadas durante a graduação e pode também pôr em prática conhecimentos adquiridos durante o curso de computação tais como programação web, desenvolvimento orientado a objeto, padrões de projeto, arquitetura de um sistema, dentre outras.

Referências

- [1] IRIGARAY, T. Q.; FILHO, I. G.; SCHNEIDER, R. H. Effects of an attention, memory and executive functions training on the cognition of healthy elderly people. v. 25, n. 1, p. 182–187. ISSN 0102-7972. Disponível em: <http://www.scielo.br/scielo.php?script=sci_abstract&pid=S0102-79722012000100023&lng=en&nrm=iso&tlng=pt>. 11
- [2] BISOGGIO, J. et al. Cognitive enhancement through action video game training: great expectations require greater evidence. v. 5. ISSN 1664-1078. Disponível em: <<http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3928536/>>. 11
- [3] LINDÔSO, Z. C. L. et al. Subjective memory perception and manual ability among elderly from a workshop of digital inclusion. v. 14, n. 2, p. 303–317. ISSN 1809-9823. Disponível em: <http://www.scielo.br/scielo.php?script=sci_abstract&pid=S1809-98232011000200011&lng=en&nrm=iso&tlng=pt>. 12
- [4] CHIAPPE, D. et al. Improving multi-tasking ability through action videogames. v. 44, n. 2, p. 278–284. ISSN 1872-9126. 12
- [5] JavaScript Web APIs - W3C. 2016. Disponível em: <<https://www.w3.org/standards/webdesign/script>>. 14
- [6] ECMA International. 2016. Disponível em: <<http://www.ecma-international.org/>>. 14
- [7] ESLint - Pluggable JavaScript linter. 2016. Disponível em: <<http://eslint.org/>>. 14
- [8] W3C HTML. 2016. Disponível em: <<https://www.w3.org/html/>>. 15
- [9] LOCALSTORAGE. 2016. Disponível em: <<https://www.w3.org/TR/webstorage/>>. 15
- [10] CASCADING Style Sheets. 2016. Disponível em: <<https://www.w3.org/Style/CSS/>>. 15
- [11] VUE.JS. 2016. Disponível em: <<https://vuejs.org/>>. 15
- [12] VUE-ROUTER. 2016. Disponível em: <<https://router.vuejs.org/>>. 16
- [13] VUEX. 2016. Disponível em: <<https://vuex.vuejs.org/en/>>. 16
- [14] VUEX-IMAGE. 2016. Disponível em: <<https://raw.githubusercontent.com/vuejs/vuex/dev/docs/en/images/vuex.png>>. 7, 17
- [15] VUE-LOADER. 2016. Disponível em: <<http://vue-loader.vuejs.org/en/>>. 16
- [16] BABEL. 2016. Disponível em: <<https://babeljs.io/>>. 17
- [17] MaterializeCSS. 2016. Disponível em: <<http://materializecss.com/>>. 18
- [18] GIT. 2016. Disponível em: <<https://git-scm.com/>>. 18
- [19] GitHub. 2016. Disponível em: <<https://github.com>>. 18

-
- [20] WEBPACK module bundler. 2016. Disponível em: <<https://webpack.github.io/>>. 18
- [21] WEBPACK-IMAGE. 2016. Disponível em: <<https://webpack.github.io/assets/what-is-webpack.png>>. 7, 19
- [22] NPM. 2016. Disponível em: <<https://www.npmjs.com/>>. 19
- [23] EXTENSIBLE Markup Language (XML). 2016. Disponível em: <<https://www.w3.org/XML/>>. 19
- [24] W3C XML Schema. 2016. Disponível em: <<https://www.w3.org/XML/Schema>>. 20
- [25] JSON. 2016. Disponível em: <<http://www.json.org/>>. 21
- [26] X2JS. 2016. Disponível em: <<https://github.com/abdmob/x2js>>. 21
- [27] PRESSMAN, R. *Software Engineering: A Practitioner's Approach*. 7. ed. [S.l.]: McGraw-Hill, Inc., 2010. ISBN 978-0-07-337597-7. 23