UNIVERSIDADE FEDERAL DE SANTA CATARINA

ABORDAGEM DE DESENVOLVIMENTO UTILIZANDO JAVASCRIPT ISOMÓRFICO

Diego da Silva Marques

Florianópolis - SC

2016/2

UNIVERSIDADE FEDERAL DE SANTA CATARINA

DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

ABORDAGEM DE DESENVOLVIMENTO UTILIZANDO JAVASCRIPT ISOMÓRFICO

Diego da Silva Marques

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Ciências da Computação

Florianópolis - SC

2016/2

Diego da Silva Marques

ABORDAGEM DE DESENVOLVIMENTO UTILIZANDO JAVASCRIPT ISOMÓRFICO

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Ciências da Computação

Orientador: Prof. Dr. Frank Augusto Siqueira

Banca examinadora

Prof. Dr. Elder Rizzon Santos

Prof. Dr. Leandro José Komosinski

SUMÁRIO

SUMÁRIO **4**

RESUMO **5**

1. INTRODUÇÃO **6**

1.2 Objetivos 7

1.3 Motivação e Justificativa 7

1.4 Método de Pesquisa 8

2. TECNOLOGIAS, CONCEITOS E FRAMEWORKS **9**

2.1 JavaScript 9

2.2 React e JSX 10

2.3 Flux e Redux 11

2.4 Node.js e Express 13

2.5 Angular 13

2.6 Outras Tecnologias 14

3. JAVASCRIPT ISOMÓRFICO **16**

3.1 O que é JavaScript Isomórfico? 16

3.2 Por quê JavaScript Isomórfico? 16

3.3 Como desenvolver utilizando JavaScript Isomórfico? 22

2.3.1 Frameworks do Cliente 23

3.3.2 Frameworks do Servidor 24

2.4 JavaScript Isomórfico X JavaScript Universal 24

4. A APLICAÇÃO DE EXEMPLO **26**

4.1 Descrição 26

4.2 Modelagem 26

4.2.1 Casos de Uso 27

4.2.2 Requisitos Funcionais e Não-Funcionais 32

REFERÊNCIAS **35**

RESUMO

Este trabalho tem como finalidade mostrar o que é JavaScript Isomórfico, quais suas finalidades, quais as características que o diferem de uma abordagem de desenvolvimento tradicional, diferenciar os diversos termos utilizados para definir seu conceito ou conceitos correlatos, quais ferramentas podem ser utilizadas para construir aplicações e, por fim, demonstrar através de um exemplo como funciona o processo de desenvolvimento de uma aplicação utilizando JavaScript Isomórfico com as ferramentas propostas anteriormente. A aplicação desenvolvida se trata de um sistema onde é possível consultar uma API que contém registros de ocorrências policiais do estado de São Paulo, disponibilizadas publicamente, que passaram por um tratamento dos dados visando um enriquecimento das informações contidas nos mesmos. Além de ser implementada a aplicação utilizando JavaScript Isomórfico, foi implementada uma aplicação do tipo Single Page Application utilizando uma abordagem tradicional de desenvolvimento, visando demonstrar as diferenças entre as duas abordagens.

Palavras-chave: JavaScript, MEAN Stack, ReactJS, AngularJS, NodeJS, ExpressJS, Javascript Isomórfico, Javascript Universal, Single Page Application, SPA, SEO, Web, Server-rendered Application.

1. INTRODUÇÃO

O desenvolvimento de aplicações Web completas utilizando a linguagem de programação JavaScript vem se tornando cada vez mais popular, pois seus benefícios para o usuário final vão muito além dos observados antigamente, quando JavaScript era utilizado apenas para tarefas simples, como validação de formulários. Um dos modelos de aplicação mais comum utilizando a linguagem, até a publicação deste trabalho, chama-se *Single Page Application* (SPA), que segundo MIKOWSKI e POWELL (2013) traz inúmeras vantagens para a experiência do usuário da aplicação, porém também apresenta suas desvantagens. Algumas dessas desvantagens podem ser resolvidas através de uma abordagem de desenvolvimento chamada de JavaScript Isomórfico, que pode ser resumida como "aplicação JavaScript que roda tanto no servidor como no cliente" (STRIMPEL, 2016), o que traz diversos benefícios que serão discutidos posteriormente.

Além disso, existem diferentes termos para definir tais conceitos, como "JavaScript Isomórfico", "JavaScript Universal", "Server-rendered application", cada um tendo seu próprio significado, com todos eles sendo apresentados no capítulo 3, chegando a um consenso para o escopo deste trabalho de qual o significado de JavaScript Isomórfico.

1.2 Objetivos

O objetivo geral deste trabalho, como citado anteriormente, é demonstrar as diferenças entre a abordagem de desenvolvimento tradicionalmente empregada para desenvolvimento de aplicações Web e a abordagem baseada em JavaScript Isomórfico. Essas diferenças serão observadas na prática, através do desenvolvimento de uma mesma aplicação utilizando as duas abordagens. Para tal, espera-se que ambas as aplicações propostas estejam implementadas ao final deste trabalho, com todo o processo de desenvolvimento documentado, além de ambas estarem hospedadas na internet para posterior acesso e validação. Além disso, é esperado que as características definidas na fundamentação dos conceitos de uma aplicação isomórfica possam ser avaliadas, demonstrando suas diferenças para a aplicação tradicional.

1.3 Motivação e Justificativa

A principal motivação deste trabalho é servir como uma introdução para pesquisas aprofundadas sobre JavaScript Isomórfico, que serve para resolver alguns dos principais problemas que as aplicações web desenvolvidas utilizando JavaScript com os principais frameworks disponíveis no mercado enfrentam, visto a escassez de publicações acadêmicas e artigos científicos sobre a mesma e sobre técnicas relacionadas.

1.4 Método de Pesquisa

A pesquisa será feita através da listagem das particularidades do desenvolvimento utilizando Javascript Isomórfico, e da elaboração de um sistema em que seja possível a demonstração de tais particularidades. A definição do sistema, assim como sua modelagem, se dará após a pesquisa e revisão do estado da arte e do levantamento de tudo que precisa ser demonstrado.

Após a definição do sistema, deverá ocorrer sua modelagem. Com o sistema definido e modelado, terá início o desenvolvimento e testes do ponto de acesso à API, onde as aplicações poderão consumir os dados fornecidos pela mesma. Com o ponto de acesso pronto e testado, será dado início à implementação e aos testes de ambas as aplicações.

Com ambas aplicações desenvolvidas e hospedadas na internet, pode se iniciar o processo de análise de resultados, para checar se as características propostas pela definição podem ser observadas.

2. TECNOLOGIAS, CONCEITOS E FRAMEWORKS

Neste capítulo, serão citadas as tecnologias, os conceitos e os frameworks que serão utilizadas no escopo deste trabalho, durante o processo de desenvolvimento de aplicações.

2.1 JavaScript

Segundo FLANAGAN (2006), JavaScript é uma linguagem de programação interpretada, com capacidade de programação orientada a objetos (POO) que tem uma sintaxe muito parecida com a sintaxe de linguagens como C++/C e Java, além de diversas inspirações na linguagem Perl. O nome oficial da linguagem, segundo a especificação ECMA-262, é ECMAScript, pois a linguagem foi padronizada e estabilizada pela associação European Computer Manufacturer's Association (ECMA), e conta com diversas implementações do padrão.

Nos primórdios da linguagem, os principais navegadores da época tinham um interpretador de JavaScript embutido, possibilitando a execução de scripts da linguagem, que eram feitos com a principal finalidade de executar ações de manipulação do *Document Object Model* (DOM) de uma página, como validar dados de um formulário e executar pequenos cálculos sem a necessidade de uma requisição ao servidor HTTP.

Conforme a tecnologia foi evoluindo, diversas capacidades foram adicionadas à linguagem, como capacidade de requisições assíncronas (AJAX), padronização de uma notação para objetos (JSON), entre outras, o que trouxe a criação de diversos frameworks e bibliotecas para facilitar e aprimorar o desenvolvimento utilizando a mesma.

Com o advento do interpretador de alto desempenho Chrome V8, desenvolvido pela Google para o seu navegador Google Chrome, foi criada uma plataforma de execução chamada Node.js, que possibilitou a execução de scripts da linguagem JavaScript do lado do servidor, e criou uma nova gama de possibilidades para a linguagem como, por exemplo, poder criar um servidor HTTP utilizando JavaScript (GOOGLE, 2016).

2.2 React e JSX

"*React é um framework de JavaScript para criar interfaces de usuário desenvolvido pelo Facebook e pelo Instagram. Muitas pessoas escolhem imaginar o React como o V do padrão MVC"* (FACEBOOK, 2016b).

React permite ao usuário criar diversos componentes que podem ser reutilizados de forma simples, e têm seu estado gerenciado, por padrão, pelo próprio framework, o que significa que quando um dado utilizado dentro de um componente for atualizado, o componente se atualizará automaticamente. Tal funcionalidade é possível pois o React trabalha utilizando um conceito chamado Virtual DOM, onde é calculado em quais nodos do DOM existem alterações, e somente tais nodos são atualizados, gerando

assim um grande ganho de performance, visto que não é necessário atualizar toda a árvore de objetos a cada atualização de um nodo.

Além disso, React tem a capacidade de ser executado do lado do servidor, graças aos interpretadores de JavaScript que tem tal finalidade, o que traz uma série de benefícios que serão demonstrados posteriormente. Tal prática é comumente conhecida como JavaScript Isomórfico, ou JavaScript Universal, como visto em BREHM (2013).

Existem duas formas de se representar elementos no React: a forma tradicional, na qual se pede ao objeto responsável pela interface de acesso à biblioteca para criar um elemento através de um método definido em sua API; ou utilizando uma sintaxe chamada JSX (FACEBOOK, 2016d). O JSX é um “açúcar sintático” para a forma tradicional citada acima, onde os elementos podem ser declarados com uma sintaxe que se assemelha muito ao HTML, beneficiando desenvolvedores que estão acostumados com esse tipo linguagem para criar suas aplicações. Além de permitir declarar elementos do próprio HTML, como “div” ou “span”, utilizando a sintaxe de *tags* do HTML, ele permite representar os componentes criados utilizando o React com a mesma sintaxe (FACEBOOK, 2016c).

2.3 Flux e Redux

“*Redux é um container de estados previsível para aplicativos Javascript. Ele ajuda você a escrever aplicações que se comportam consistentemente, rodam em diferentes ambientes (cliente, servidor, nativo) e são fáceis de testar*” (REDUX, 2016).

Redux será utilizado em combinação com React, visando um controle consistente do estado da aplicação e do fluxo da execução de operações da aplicação desenvolvida neste trabalho. O Redux é uma implementação da arquitetura Flux, que de acordo com KIM (2015), trata os estados da aplicação de forma imutável, criando um novo estado para cada alteração.

O Flux é uma arquitetura para construção de interfaces para usuários, proposta pelo Facebook e comumente adotada como a arquitetura padrão do React (FACEBOOK, 2016e). O Flux separa a aplicação em diversas camadas, e propõe um fluxo de dados em uma única direção, onde as *Views* da aplicação geram *Actions* e essas são encaminhadas ao *Dispatcher*, que avisa a todas as *Stores* que estavam registradas com ele sobre a ação. As *Stores*, por sua vez, lidam com a ação desejada e emitem um evento para a *View* que criou a ação com o intuito da mesma se atualizar, gerando um fluxo de dados em uma única direção, como pode ser visto na Figura 6.

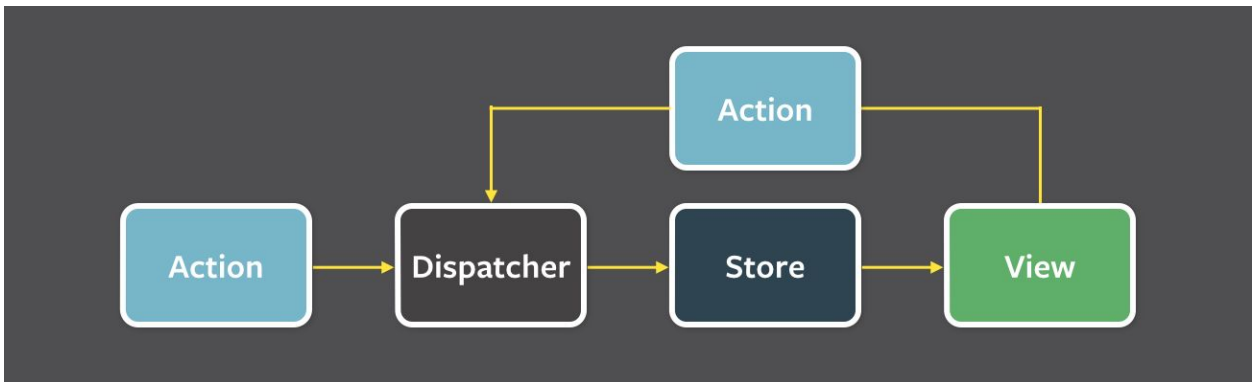

Figura 6 - Fluxo de dados da arquitetura Flux [FACEBOOK, 2016e]

2.4 Node.js e Express

Node.js é um ambiente de execução capaz de executar códigos escritos em JavaScript fora de um *browser*, baseado no interpretador Chrome V8, o mesmo interpretador utilizado pelo *browser* Google Chrome. Seu principal objetivo é permitir a criação de aplicações de rede escaláveis utilizando um modelo orientado a eventos, em contraste de outras soluções que utilizam um modelo orientado a threads, segundo TILKOV e VINOSKI (2010) e NODEJS (2016).

Express, por sua vez, é um framework que tem como objetivo permitir a criação de um servidor para aplicações Web utilizando o Node.js, ajudando a tratar o fluxo de requisições e respostas da aplicação e facilitando o desenvolvimento através de abstrações de tarefas como roteamento de páginas, tratamento de *cookies* e sessões, entre outros.

2.5 Angular

"AngularJS é um framework MVC client-side que trabalha com tecnologias já estabelecidas: HTML, CSS e JavaScript. Criado na Google e liberado como projeto open-source para o público em 2009, seu foco reside na criação de Single Page Applications (SPA), embora isso não o impeça de ser utilizado em outros tipos de aplicação. AngularJS é um high opinionated framework, o que significa que ele guia o desenvolvedor de maneira implacável na organização de seu código.

Miško Hevery, pai do framework, procurou trazer os mesmos benefícios do modelo MVC aplicados no lado do servidor para o navegador como a facilidade de manutenção, reusabilidade através de componentes e testabilidade, uma vez que a lógica é desacoplada de elementos do DOM. Ainda há recursos como injeção de dependências!" (ALMEIDA, 2015).

Ao contrário do React, a proposta do Angular não é de ser apenas uma biblioteca para criação de interfaces do usuário, mas sim ser um framework completo para criação de aplicações através da extensão do HTML. *"Angular é o que o HTML deveria ter sido, se o mesmo tivesse sido pensado para a criação de aplicações"* (ANGULARJS, 2016b). Angular será utilizado neste trabalho para implementar a aplicação que não tem os recursos definidos como recursos de uma aplicação que utiliza JavaScript Isomórfico, em outras palavras, será utilizado para a aplicação definida como tradicional.

2.6 Outras Tecnologias

Aqui serão apresentadas outras ferramentas e tecnologias que não impactam diretamente no funcionamento das aplicações que serão desenvolvidas, porém auxiliam no processo de desenvolvimento e entrega da aplicação.

Bootstrap e jQuery são ferramentas para auxiliar na criação de interfaces de usuários, com diversos componentes estilizados prontos e manipulação do DOM e tratamento de eventos JavaScript facilitados.

SASS, por sua vez, é uma extensão ao CSS, que adiciona diversas capacidades como o reuso e extensão de estilos, definição de variáveis, entre outros.

NPM, Gulp, Bower e Webpack são ferramentas responsáveis por gerenciar todas as bibliotecas utilizadas nas aplicações, além de compactar o código onde for necessário para diminuir o consumo de dados necessários para utilizar a aplicação e controlar a instalação e entrega da aplicação ao servidor.

O NPM (*Node.js Package Manager*) e o Bower são ferramentas para o gerenciamento de dependências, sendo que o Bower ficará responsável por gerenciar apenas as bibliotecas utilizadas pela interface do usuário. Já o Gulp e o Webpack são responsáveis por compilar os diversos códigos JavaScript da aplicação e as suas bibliotecas para um único arquivo comprimido, além de compilar o SASS, rodar os testes unitários necessários e configurar o ambiente de execução para o desenvolvimento.

3. JAVASCRIPT ISOMÓRFICO

Este capítulo descreve o conceito de JavaScript Isomórfico e mostra como ele é utilizado para o desenvolvimento de aplicações Web.

3.1 O que é JavaScript Isomórfico?

JavaScript Isomórfico é o nome atribuído ao conceito de desenvolvimento para Web, no modelo cliente-servidor, onde o mesmo código, escrito em JavaScript, é executado em ambas partes. O conceito de JavaScript Isomórfico se tornou uma possibilidade a partir do momento em que foi possível executar um código escrito em JavaScript do lado do servidor da aplicação, apesar de que os primeiros indícios de seu uso datam de uma época mais recente, tendo se popularizado com a publicação de BREHM (2013).

3.2 Por quê JavaScript Isomórfico?

Para justificar o uso, é necessário primeiro entender os problemas que comumente existem em uma aplicação desenvolvida utilizando uma abordagem de desenvolvimento tradicional. Assume-se neste trabalho que uma aplicação tradicional é definida como uma aplicação Web, modelo cliente-servidor, onde existe um *Front-End* que se trata de uma SPA escrito utilizando algum framework baseado em JavaScript, e um *Back-End* em qualquer linguagem de programação, como pode ser visto na Figura 1.

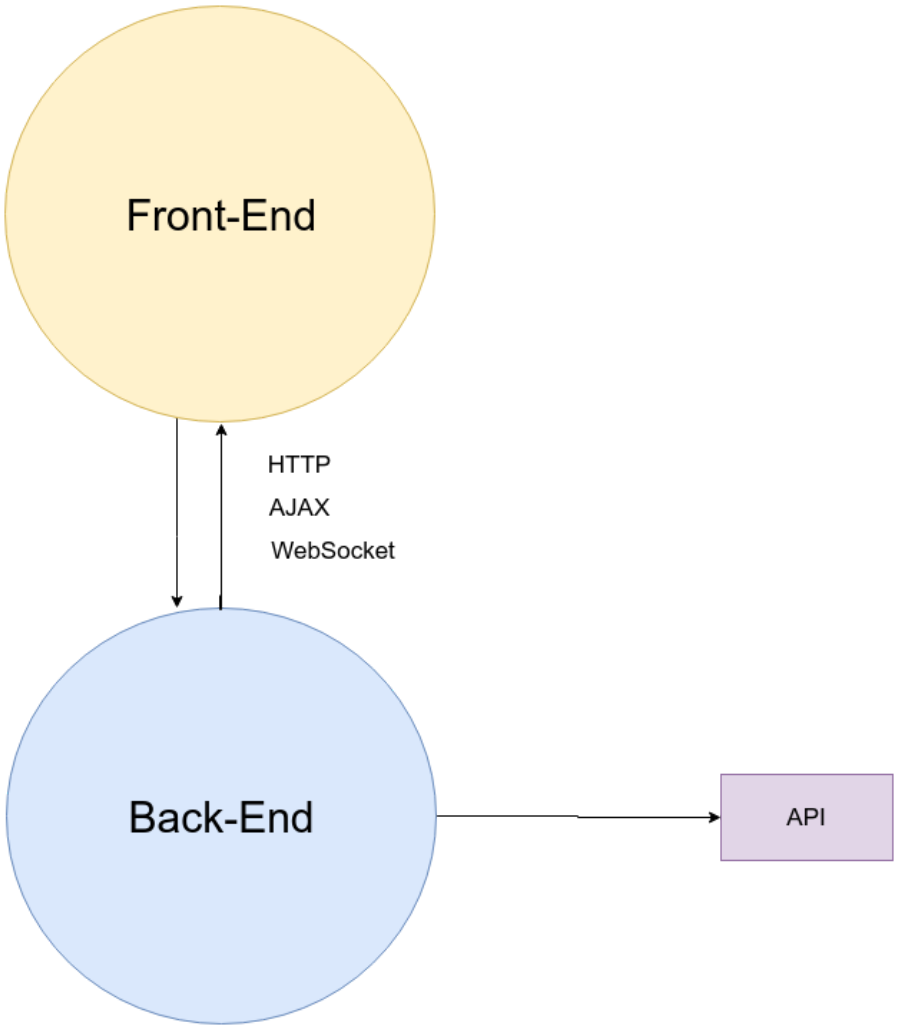

Figura 1 - Configuração de uma *Single Page Application* tradicional

O servidor, ao receber uma requisição do cliente (*browser*), processará a mesma e enviará à aplicação *Front-End* uma resposta, que será utilizada pelo *browser*. Dependendo da complexidade e do tamanho da aplicação, durante todo o tempo dessa operação será apresentada ao usuário uma tela em branco, ou caso esse problema

tenha sido tratado previamente, uma tela amigável indicando carregamento da aplicação, como o exemplo da Figura 2.

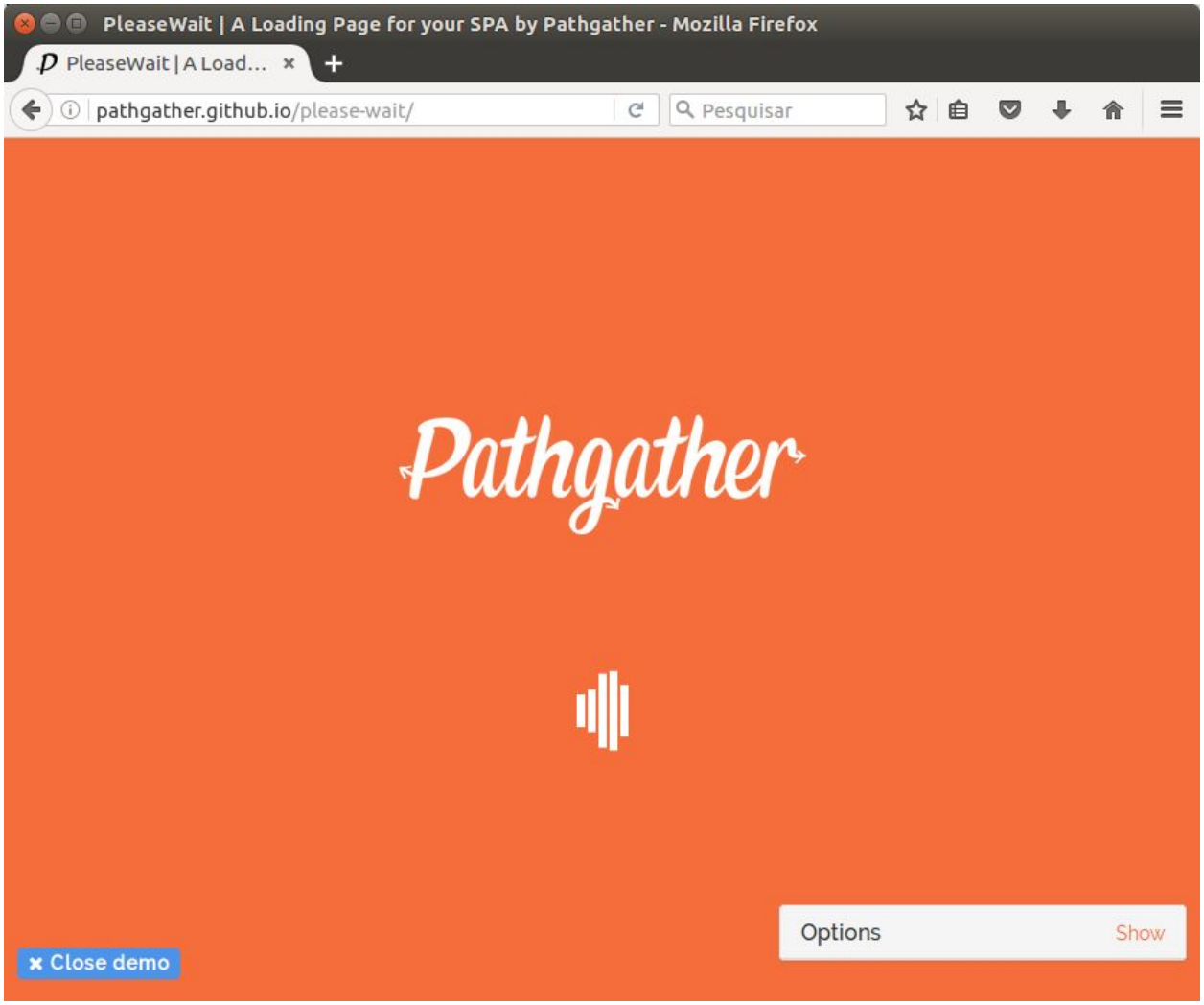

Figura 2 - Exemplo de página de carregamento de uma Single Page Application

Em uma SPA tradicional, o conteúdo da página não se encontra disponível na mesma no momento da primeira requisição ao servidor por padrão, visto que a maioria dos frameworks trabalha diretamente com manipulação do DOM, o que torna o mesmo

dinâmico e mutável em tempo de execução. Graças a isso, as SPAs podem fazer a navegação entre suas páginas, na maioria dos casos, sem precisar enviar uma requisição ao servidor, o que traz uma fluidez maior para a aplicação e alivia a carga de processamento no servidor. Isso não significa que não existam requisições ao servidor após a primeira requisição, pois apesar da aplicação estar executando no cliente, a maioria das aplicações precisa consumir dados fornecidos a ela, e esse processo pode ser otimizado utilizando requisições AJAX que atualizem parcialmente o DOM, ou WebSockets para aplicações em tempo real.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>NativeIDE</title>
  </head>
  <body>
    <div id="root"></div><!-- App entry point -->
  </body>
  <script src="js/bundle.js"></script>
  <style media="screen">
    body {
      margin: 0px auto;
    }
  </style>
</html>
```

Figura 3 - Exemplo de *Entry Point* de uma *Single Page Application*

Apesar de trazer ganhos em questão de performance para o servidor, reduzindo o número de requisições e processamento de sua parte, aplicações de grande porte podem sofrer problemas relativos ao uso dos recursos na máquina do cliente onde ela

está rodando, visto que não existe um balanceamento da carga de processamento nesse modelo onde toda a aplicação depende dos recursos da máquina do cliente.

Além disso, por não carregar o conteúdo semanticamente relevante da página no momento da primeira requisição, como pode ser observado na Figura 3, pois as SPAs dependem de um elemento chamado *Entry Point* para iniciar sua criação através da mutação do DOM, as mesmas têm por padrão um problema envolvendo SEO, visto que os Web Crawlers necessitam das informações contidas na página para funcionarem. Tal problema pode ser contornado com um tratamento especial dependente da requisição, porém não é uma coisa que é tratada por padrão quando se utiliza um conjunto de frameworks tradicionais.

Todos os problemas citados acima podem ser resolvidos utilizando uma abordagem isomórfica. Na abordagem isomórfica, quando um cliente faz uma requisição completa ao servidor, a mesma aplicação está rodando em ambos os lados, logo o servidor tem a capacidade de processar a requisição utilizando a aplicação e responder com uma página com conteúdo estático ao cliente, enquanto todas as funcionalidades de navegação, rotas, atualizações dinâmicas, etc estão sendo preparadas pelo cliente, o que gera um balanceamento de carga entre ambos, além de fornecer o conteúdo semanticamente importante na resposta, que pode ser aproveitado pelos *Web Crawlers*. Com isso, o conteúdo da página já pode ser exibido de imediato, sem a necessidade de avisar ao usuário sobre o carregamento da mesma.

Além dos problemas citados anteriormente, em muitos projetos é comum cliente e servidor serem escritos em linguagens de programação distintas, e que muitas vezes

podem até pertencer a paradigmas de programação diferentes, o que leva a sérios problemas de manutenibilidade de código, além de ser necessário muitas vezes replicar o mesmo código entre ambos, e tal problema também é resolvido utilizando a abordagem isomórfica, pois o código da aplicação é único para ambos, além de toda a aplicação ser escrita na mesma linguagem de programação, o que facilita a manutenção da mesma, como representado na Figura 4.

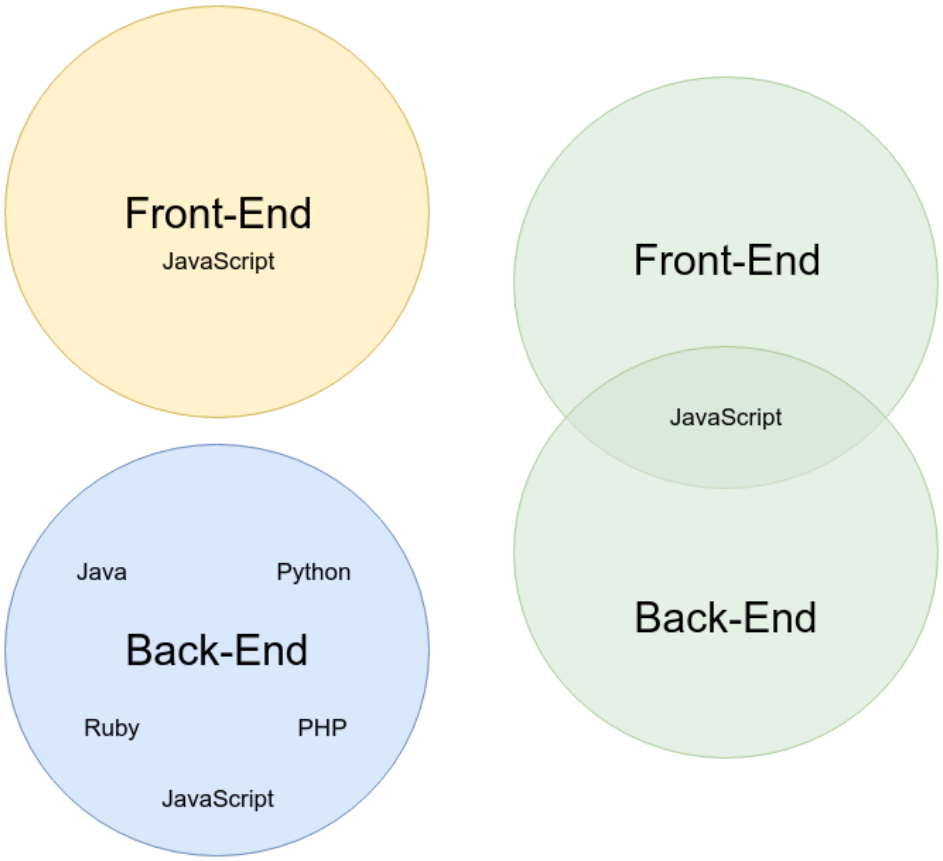

Figura 4 - Configuração de uma Single Page Application tradicional X JavaScript Isomórfico

3.3 Como desenvolver utilizando JavaScript Isomórfico?

Para desenvolver utilizando JavaScript Isomórfico, é necessário um conjunto de frameworks que permitem que uma aplicação tenha o comportamento esperado por esta abordagem. Existem diversos tipos de frameworks que permitem completar tal objetivo, porém neste trabalho os principais frameworks que serão utilizados para demonstrar o desenvolvimento de uma aplicação serão o React em conjunto com Redux e o Node.js em conjunto com o Express, além de outros frameworks auxiliares que irão ser utilizados em conjunto com esses para os mais diversos fins.

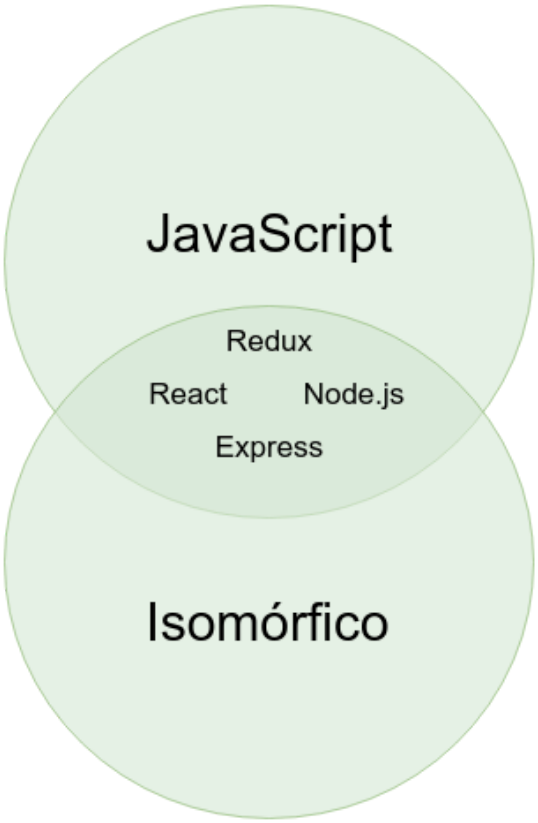

Figura 5 - Principais bibliotecas utilizadas no desenvolvimento da aplicação isomórfica

2.3.1 Frameworks do Cliente

Dentre os diversos frameworks responsáveis por prover a capacidade de desenvolver uma aplicação isomórfica, o React foi escolhido por sua forma de lidar com o DOM, além de sua capacidade de trabalhar usando JSX, que torna confortável o desenvolvimento para um desenvolvedor Web que está acostumado com HTML. Além disso, o React possui uma comunidade adepta ao seu uso muito grande e que está em constante crescimento, o que torna qualquer problema encontrado no processo de desenvolvimento mais fácil de se resolver.

Porém o React é um framework, como visto em FACEBOOK (2016a), que tem como propósito criar interfaces de usuário, apesar de prover meios de controlar estado de seus componentes e tratar a lógica da aplicação. No escopo deste trabalho, toda a lógica da aplicação, suas operações e seu estado serão tratados pelo Redux, de forma que o React será utilizado unicamente para criar as interfaces de usuário, que é seu principal propósito. O Redux é uma implementação da arquitetura Flux, que trata os estados da aplicação de forma imutável e por isso, cria uma cópia do estado atual a cada mudança que ocorre e organiza as mesmas através de um grafo, o que torna a depuração da aplicação muito simples e garante a capacidade de voltar a um estado anterior, no caso de um erro, sem causar muitos problemas, justamente pela habilidade de percorrer um grafo dos estados.

3.3.2 Frameworks do Servidor

Para executar a aplicação do lado do servidor, foi escolhido o Express, um framework para aplicações Web que roda em cima do Node.js e permite tratar as requisições e respostas HTTP. Por utilizar o Node.js como base, torna-se possível executar o código JavaScript da aplicação e tratar seu resultado da forma esperada. Ambos frameworks foram escolhidos para esse trabalho simplesmente por familiaridade com o processo de desenvolvimento utilizando os mesmos, além de contarem com diversos plugins e frameworks auxiliares que agilizam o desenvolvimento ou adicionam novas funcionalidades às existentes.

2.4 JavaScript Isomórfico X JavaScript Universal

Os termos JavaScript Isomórfico e JavaScript Universal são, muitas vezes, usados como sinônimos. Porém, como é proposto em RAUSCHMEYER (2015), HENGEVELD (2015) e STRIMPEL (2016), existe uma diferenciação entre os mesmos. Segundo as fontes citadas, Isomorfismo vai além do JavaScript e se refere à aplicação que tem a funcionalidade rodar em múltiplas plataformas, enquanto o JavaScript Universal se refere à capacidade de um código escrito em JavaScript de rodar em múltiplas plataformas. Ainda, segundo ROBBESTAD (2016), nenhum dos termos está correto, e diz que o significado da palavra “Isomorfismo” não se adequa aos contexto de um código que executa em múltiplos ambientes, e muito menos “Universal”, e que o termo ideal seria “Server-Rendered”.

Neste trabalho, apesar de serem levadas em consideração todas as definições para o que está sendo proposto, o termo “JavaScript Isomórfico” tem um significado que engloba todos os três termos apresentados, sendo utilizado para expressar a ideia de uma aplicação que roda no servidor de aplicação (Server-Rendered), é escrita em JavaScript de uma forma que possa rodar em múltiplas plataformas (JavaScript Universal) e portanto irá rodar tanto no cliente quanto no servidor (Isomorfismo/JavaScript Isomórfico).

4. A APLICAÇÃO DE EXEMPLO

Este capítulo descreve a aplicação que será desenvolvida de duas maneiras: utilizando a abordagem tradicional; e com JavaScript Isomórfico.

4.1 Descrição

A aplicação escolhida a ser desenvolvida para a demonstração das técnicas e conceitos citados anteriormente se trata da implementação de um sistema baseado no exemplo apresentado por OLIVEIRA *et al.* (2016), onde o usuário poderá consultar dados de relatórios policiais publicados pela Secretaria de Segurança Pública de São Paulo (SSP/SP) e pelo Tribunal de Justiça de São Paulo (TJSP), por meio de filtros que incluem região, gênero, escolaridade, entre outros. Tais dados serão fornecidos através da Web API proposta por OLIVEIRA *et al.* (2016), e contará com um mapa interativo para prover ricas experiências de navegabilidade e visualização dos dados, além de ser possível criar relatórios utilizando as consultas executadas, com modelos como gráficos e tabelas para facilitar seu entendimento.

4.2 Modelagem

Levando em consideração a proposta da aplicação, é necessário definir o escopo da mesma e sua modelagem. A modelagem das funcionalidades do sistema será representada através de um modelo simplificado de Casos de Uso. Após as funcionalidades estarem modeladas, podemos extrair os Requisitos Funcionais e

Não-Funcionais da aplicação e com isso definir onde cada ferramenta descrita no capítulo 2 será aplicada.

4.2.1 Casos de Uso

Número	1
Nome do Caso de Uso	Pesquisar Ocorrências
Ator	Usuário
Condição de Entrada	Usuário deve ter acesso ao sistema.
Fluxo de Eventos	1. Usuário clica no campo para inserir uma pesquisa. 2. Usuário insere o texto pelo qual deseja pesquisar. 3. Usuário seleciona quais critérios ele gostaria de utilizar para filtrar sua pesquisa. 4. Se usuário não selecionar nenhum critério, será utilizado “Nome” como critério. 5. Usuário seleciona filtros adicionais para a pesquisa. 6. Sistema executa a pesquisa e exibe os resultados em uma tela contendo um mapa.
Condição de Saída	1. Sistema termina a pesquisa. 2. Usuário cancela a pesquisa.

Exceções	1. Caso usuário não insira nenhum conteúdo para pesquisar, mostrar uma mensagem pedindo para que informe o conteúdo da pesquisa.
Requisitos Especiais	Nenhum.

Tabela 1 - Caso de Uso “Pesquisar Ocorrências”

Número	2
Nome do Caso de Uso	Navegar pelo mapa
Ator	Usuário
Condição de Entrada	Caso de Uso (1)
Fluxo de Eventos	1. Sistema apresenta os Pontos de Interesse (POI) resultantes da pesquisa em um mapa, utilizando sua localização. 2. Usuário pode executar ações com o mapa: a. Aproximar a visualização (Zoom In). b. Afastar a visualização (Zoom out). c. Mover a visualização do mapa. d. Destacar um POI para obter informações

	resumidas (Highlighting). e. Selecionar um POI para obter um relatório completo (Selection).
Condição de Saída	1. Usuário cancela a pesquisa atual. 2. Usuário realiza uma nova pesquisa.
Exceções	1. Caso a pesquisa não forneça nenhum resultado, o mapa deve ser exibido em conjunto com uma mensagem avisando tal situação.
Requisitos Especiais	A biblioteca utilizada para implementar a funcionalidade de mapas deve ser fluida, visando a melhor experiência de uso ao usuário, visto que a maior interação do usuário com o sistema será através da mesma.

Tabela 2 - Caso de Uso “Navegar pelo mapa”

Número	3
Nome do Caso de Uso	Detalhar Ocorrência
Ator	Sistema
Condição de Entrada	Caso de uso (2), evento 2.e

Fluxo de Eventos	1. Sistema prepara as informações completas sobre a ocorrência selecionada. 2. Sistema exibe uma tela com todas as informações.
Condição de Saída	1. Usuário fecha a tela de exibição.
Exceções	Nenhuma.
Requisitos Especiais	Nenhum.

Tabela 3 - Caso de Uso “Detalhar Ocorrência”

Número	4
Nome do Caso de Uso	Exportar Ocorrência para Impressão
Ator	Usuário
Condição de Entrada	Caso de uso (3)
Fluxo de Eventos	1. Usuário seleciona opção de exportar informações. 2. Usuário seleciona o formato de arquivo desejado. 3. Sistema converte as informações para o formato de arquivo selecionado pelo usuário. 4. Sistema disponibiliza o arquivo para o usuário através de download.

	5. Usuário efetua download do arquivo.
Condição de Saída	1. Usuário termina o download do arquivo. 2. Usuário cancela o download do arquivo.
Exceções	1. Usuário é desconectado do sistema. 2. Browser utilizado pelo usuário não suporta downloads.
Requisitos Especiais	O sistema deve ser capaz de converter as informações para os formatos de arquivos de documentos mais utilizados (DOC e PDF).

Tabela 4 - Caso de Uso “Exportar Ocorrência para Impressão”

Número	5
Nome do Caso de Uso	Gerar Relatório de Ocorrências
Ator	Usuário
Condição de Entrada	Caso de uso (1)
Fluxo de Eventos	1. Usuário seleciona a opção de criar um relatório. 2. Sistema mostra as opções de relatório disponíveis. 3. Usuário seleciona uma opção de relatório.

	4. Sistema gera o relatório selecionado pelo usuário. 5. Sistema exibe uma tela com os resultados.
Condição de Saída	1. Usuário fecha a tela de exibição das opções de relatório 2. Usuário cancela o relatório que está sendo gerado 3. Usuário fecha a tela com os resultados
Exceções	1. Caso a pesquisa não tenha fornecido nenhum resultado, o sistema deve mostrar uma mensagem alertando o usuário e nenhum relatório será gerado. 2. Um grande volume de dados retornados pela pesquisa pode deteriorar a performance da geração do relatório desejado pelo usuário
Requisitos Especiais	Os resultados devem ser mostrados ao usuário utilizando tabelas e gráficos.

Tabela 5 - Caso de Uso “Gerar Relatório de Ocorrências”

4.2.2 Requisitos Funcionais e Não-Funcionais

Através dos casos de uso descritos na seção anterior, podemos extrair os Requisitos Funcionais e Não-Funcionais através de uma análise das necessidades de cada caso. Além dos requisitos que extraímos dos casos de uso, podemos citar os

requisitos conceituais necessários para validar o experimento, baseados na fundamentação criada no capítulo 3. Junto com cada requisito, podemos citar ferramentas e/ou técnicas que supram tal necessidade.

Requisitos Funcionais	1. Deve ser capaz de renderizar mapas, utilizar marcações e prover navegação. Ferramenta: OpenStreetMap 2. Deve ser capaz de se comunicar com a API fornecida para o experimento. Ferramentas: AJAX, JSON 3. Deve ser capaz de mostrar conteúdo ao usuário exibindo novas telas de visualização Ferramentas: Bootstrap, jQuery (Modal) 4. Deve ser capaz de converter informações para arquivos com formato de documento como DOC e PDF Ferramentas: JasperReports
Requisitos Não-Funcionais	1. Os resultados de um relatório devem ser mostrados ao usuário utilizando tabelas e gráficos. 2. A biblioteca utilizada para prover a interface de

	mapas deve ser fluida.
Requisitos Conceituais	1. A aplicação deve poder ser executada dentro de um browser. 2. Deve existir conexão entre o servidor da aplicação e o servidor da API, caso não sejam o mesmo. 3. O sistema deve ser acessível de qualquer dispositivo com conexão com a internet através de uma URI inserida em um browser. 4. A aplicação deve ser escrita utilizando JavaScript, no modelo Single Page Application.

Tabela 6 - Requisitos

REFERÊNCIAS

BREHM, Spike. Isomorphic JavaScript The Future of Web Apps - Airbnb Engineering**.** 2013. Disponível em: <http://nerds.airbnb.com/isomorphic-javascript-future-web-apps>. Acesso em: 13 jun. 2016.

FLANAGAN, David. JavaScript: the definitive guide. O'Reilly Media. 2006.

GOOGLE. Chrome V8**.** Disponível em: <https://developers.google.com/v8/>. Acesso em: 11 set. 2016.

FACEBOOK. A JavaScript library for building user interfaces | React. 2016a. Disponível em <https://facebook.github.io/react/>. Acesso em: 11 set. 2016.

FACEBOOK. Why React?. 2016b. Disponível em <https://facebook.github.io/react/docs/why-react.html>. Acesso em: 11 set. 2016.

REDUX. Redux. Disponível em <http://redux.js.org/>. Acesso em: 11 set. 2016.

ANGULARJS. Features & Benefits – Angular 2. 2016a. Disponível em <https://angular.io/features.html>. Acesso em: 11 set. 2016.

RAUSCHMEYER, Axel. Is “Isomorphic JavaScript” a good term?. 2015. Disponível em <http://www.2ality.com/2015/08/isomorphic-javascript.html>. Acesso em: 27 set. 2016.

HENGEVELD, Gert. Isomorphism vs Universal JavaScript – Medium. 2015. Disponível em <https://medium.com/@ghengeveld/isomorphism-vs-universal-javascript-4b47fb481beb#.q7b9wh2g7>. Acesso em: 11 out. 2016.

ROBBESTAD, Sven A. Universal vs Isomorphic - Medium. 2016. Disponível em <https://medium.com/@svenarobbestad/universal-vs-isomorphic-10fc30aac39d#.buamrrh6n> . Acesso em: 17 out. 2016.

KIM, Jennie. Understanding Redux (or, How I fell in love with a JavaScript State Container) - you have to learn computers. 2015. Disponível em <http://www.youhavetolearncomputers.com/blog/2015/9/15/a-conceptual-overview-of-redux-or-how-i-fell-in-love-with-a-javascript-state-container>. Acesso em: 17 out. 2016.

FACEBOOK. JSX In Depth. 2016c. Disponível em <https://facebook.github.io/react/docs/jsx-in-depth.html>. Acesso em: 23 out. 2016.

FACEBOOK. Introducing JSX. 2016d. Disponível em <https://facebook.github.io/react/docs/introducing-jsx.html>. Acesso em: 23 out. 2016.

FACEBOOK. Flux | Application Architecture for Building User Interfaces. 2016e. Disponível em <https://facebook.github.io/flux/docs/overview.html#content>. Acesso em: 23 out. 2016.

TILKOV, Stefan; VINOSKI, Steve. Node. js: Using JavaScript to build high-performance network programs. IEEE Internet Computing, v. 14, n. 6, p. 80, 2010.

ALMEIDA, Flávio. “Mean: full stack JavaScript para aplicações web com MongoDB, Express, Angular e Node”. Casa do Código. 2015.

STRIMPEL, Jason; NAJIM, Maxime. “Building Isomorphic JavaScript Apps: From Concept to Implementation to Real-World Solutions”. O’Reilly Media. 2016.

NODEJS. About | Node.js. Disponível em <https://nodejs.org/en/about/>. Acesso em: 30 out. 2016.

ANGULARJS. AngularJS: Developer Guide: Introduction. 2016b. Disponível em <https://docs.angularjs.org/guide/introduction>. Acesso em: 30 out. 2016.

OLIVEIRA, Bruno C. N.; SALVADORI, Ivan; HUF, Alexis; SIQUEIRA, Frank. “A Platform to Enrich, Expand and Publish Linked Data of Police Reports”. 2016.

MIKOWSKI, Michael S.; POWELL, Josh C. "Single Page Web Applications." B and W. 2013.