Diego da Silva Marques

ABORDAGEM DE DESENVOLVIMENTO UTILIZANDO JAVASCRIPT ISOMÓRFICO

Florianópolis

2017

Diego da Silva Marques

ABORDAGEM DE DESENVOLVIMENTO UTILIZANDO JAVASCRIPT ISOMÓRFICO

Trabalho de Conclusão do Curso de Graduação em Ciências da Computação, do Departamento de Informática e Estatística, do Centro Tecnológico da Universidade Federal de Santa Catarina, requisito parcial à obtenção do título de Bacharel em Ciências da Computação.

Orientador: Prof. Dr. Frank Augusto Siqueira

Florianópolis

2017

Diego da Silva Marques

ABORDAGEM DE DESENVOLVIMENTO UTILIZANDO JAVASCRIPT ISOMÓRFICO

Trabalho de conclusão de curso submetido ao Departamento de Informática e Estatística da Universidade Federal de Santa Catarina para a obtenção do Grau de Bacharelado em Ciências da Computação.

Orientador:

_____

Prof. Dr. Frank Augusto Siqueira
Orientador

Banca Examinadora:

_____

Prof. Dr. Elder Rizzon Santos
Avaliador

_____

Prof. Dr. Leandro José Komosinski
Avaliador

AGRADECIMENTOS

Primeiramente gostaria de agradecer a minha família, em especial a minha mãe, por ter passado todo esse processo me apoiando e me ajudando no que foi possível.

Aos meus amigos que souberam compreender minha ausência nesta etapa final mas que nunca me abandonaram.

Ao Professor Frank por ter abraçado a ideia que levei até ele e ter me ajudado durante todo o processo de formulação da ideia, pela orientação e por ter permitido que eu fizesse o trabalho no meu ritmo.

A comunidade de desenvolvimento Web no geral, por manter as tecnologias e os conceitos sempre em constante renovação, apaixonando cada vez mais desenvolvedores ao redor do mundo.

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, definir o que é a abordagem 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. Com base em alterações nas configurações da aplicação no servidor, foi disponibilizada uma versão que ilustra o que seria uma aplicação tradicional e feita a comparação entre ambas, mostrando que as diferenças propostas existem porém com um pequeno grau de relevância em alguns casos.

Palavras-chaves: JavaScript, ReactJS, NodeJS, ExpressJS, Javascript Isomórfico, Javascript Universal, Single Page Application, SPA, SEO, Web, Server Side Rendered, Meteor.

ABSTRACT

This research aims to show what is Isomorphic JavaScript, what it does, what make it different from a traditional development approach, define what is a traditional development approach, differentiate the many terms used to define their concepts or related concepts, which tools can be used to build applications with it and show by an example how the Isomorphic JavaScript development process works using these tools. The developed application is a system where it is possible to query an API that contains police criminal reports data from São Paulo state, publicly found, that were previously data enriched. With some changes on the server side configuration, a version that simulates a traditional application has been made available to compare them both, showing that the differences between them do exists, but some of them had a low relevance at the end.

Keywords: JavaScript, ReactJS, NodeJS, ExpressJS, Isomorphic JavaScript, Universal JavaScript, Single Page Application, SPA, SEO, Web, Server Side Rendered, Meteor.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
CSS	Cascade Style Sheet
DOM	Document Object Model
ECMA	European Computer Manufacturers Association
ES2015	ECMAScript 2015
ES5	ECMAScript 5
ES6	ECMAScript 6
GB	Gigabyte
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
JSON	JavaScript Object Notation
Mbps	Megabits Per Second
MVC	Model-View-Controller
NPM	Node Package Manager
NVM	Node Version Manager
REST	REpresentational State Transfer
SEO	Search Engine Optimization
SPA	Single Page Application
URL	Unified Resource Locator
XML	eXtensible Markup Language

SUMÁRIO

1. INTRODUÇÃO

O desenvolvimento de aplicações Web 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 Side Rendered", 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.

1.1 Objetivos

O objetivo geral deste trabalho 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 podem ser observadas na prática, através do desenvolvimento de uma aplicação configurada para funcionar de ambas as formas. Para tal, espera-se que a aplicação proposta esteja implementada ao final deste trabalho, com todo o processo de desenvolvimento documentado, além de 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.

1.2 Motivação e Justificativa

A principal motivação deste trabalho é experimentar o desenvolvimento de uma aplicação que utilize JavaScript Isomórfico, onde seja possível reunir informações sobre tal abordagem e comparar o seu comportamento com base em suas premissas, visto a escassez de publicações acadêmicas e artigos científicos sobre a mesma e sobre técnicas relacionadas.

1.3 Método de Pesquisa

A pesquisa foi 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 deu após a pesquisa e revisão do estado da arte e do levantamento de tudo que precisa ser demonstrado.

Após a definição do sistema, ocorreu sua modelagem. Com o sistema definido e modelado, teve início a etapa de desenvolvimento e testes, além da hospedagem da aplicação com ambas configurações ao final desta etapa.

Com ambas aplicações desenvolvidas e hospedadas na internet, foi possível 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 são citadas as tecnologias, os conceitos e os frameworks que são utilizados no escopo deste trabalho, durante o processo de desenvolvimento da aplicação.

2.1 Aplicação Web

Primeiramente é importante definir alguns conceitos relacionados ao que é uma aplicação Web, como o que é uma aplicação Web e alguns de seus padrões de projeto e modelos.

Como pode ser visto em SEBESTA (2012), uma aplicação Web pode ser definida como uma aplicação distribuída que utiliza a arquitetura cliente-servidor, onde existe um servidor capaz de receber requisições que utilizam o protocolo HTTP e tratá-las conforme seu algoritmo interno e retornar o resultado deste processamento através de uma resposta HTTP para o cliente que efetuou a requisição.

Aplicações Web utilizam primariamente o HTML como o principal recurso para exibir os dados de sua resposta, porém conforme a tecnologia foi evoluindo, existem outros tipos de resposta comum, como a notação JSON, utilizada principalmente para aplicações conhecidas como Web Services, que são serviços que tem como principal função fornecer dados ao cliente requisitante através de um sistema de rotas que utiliza a URL para se localizar.

Dentro das aplicações Web, de maneira geral, a arquitetura de projeto mais utilizada segundo SEBESTA (2012) é a Arquitetura Multicamadas, onde a aplicação é dividida em diversas camadas e cada uma tem um papel dentro do ecossistema da aplicação. O número de camadas e a maneira como se comunicam são as mais variadas possíveis, tendo normalmente uma quantidade entre 2 e 4 camadas, além da comunicação entre duas camadas poder ocorrer em um sentido ou em ambos sentidos, dependendo do padrão de projeto utilizado.

O padrão de projeto mais utilizado em aplicações Web é o modelo MVC (Model-View-Controller), onde a interação do usuário é separada da representação dos dados, e sua lógica é separada entre três principais componentes:

- Model: Representa os dados da aplicação.
- View: Modelo de representação da interface visual da aplicação para o usuário.
- Controller: Componente responsável por gerenciar a interação do usuário com a aplicação.

Apesar dos padrões e modelos citados, a Web se mantém em constante evolução, e existem diversos outros tipos de padrões, modelos, arquiteturas e paradigmas sendo utilizados para desenvolver aplicações Web.

2.2 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 efetuar 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).

2.3 SEO

SEO, acrônimo para Search Engine Optimization, são uma série de técnicas para otimizar o posicionamento de uma página ou aplicação Web nos mecanismos buscadores da internet, como o Google. Segundo é mostrado em SMARTY (2009), temos mais de 200 variáveis que podem alterar a maneira como uma página ou aplicação irá ser buscada, e consequentemente mostrada, por um mecanismo deste tipo. Apesar da fonte fazer um levantamento de 200 variáveis, levando em conta o quanto a tecnologia evoluiu desde sua publicação, podemos imaginar que o número tenha aumentado com o passar do tempo.

O SEO nos dias de hoje tem um impacto muito relevante sobre páginas e aplicações Web, pois a maioria dos usuários que utilizam tais tecnologias fazem uso dos mecanismos de busca constantemente e de forma natural, o que torna tal técnica imprescindível a qualquer aplicação que deseja aumentar sua visibilidade para seu público-alvo.

2.4 ES6 e Babel

A partir de 2015, uma nova definição para o ECMAScript foi proposta, que muda radicalmente a sintaxe e as capacidades da versão anterior. Esta versão ficou conhecida como ECMAScript 6, ou conhecida também por sua abreviação ES6, ou até mesmo ES2015. Por se tratar de uma mudança drástica, a maioria dos Interpretadores de JavaScript no momento da publicação deste trabalho ainda não dá suporte total ao ES6, por questões de compatibilidade das aplicações que já existem, o que leva a

necessidade de ocorrer uma compilação da mesma para a versão antiga do ECMAScript, conhecida como ES5, que é totalmente suportada pelos interpretadores.

Uma das ferramentas utilizadas para a compilação do ES6 para o ES5 é a ferramenta Babel, que através de diversos *presets* ganha a capacidade de transformar a sintaxe nova para a antiga. Além de transformar ES6 para ES5, o Babel também é responsável por compilar o padrão conhecido como JSX, utilizado pela biblioteca React, para algo que possa ser reconhecido e interpretado, e que será definido a seguir.

2.5 React, JSX e React Router

"*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, 2016e).

React permite ao usuário criar diversos componentes de interface gráfica 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, 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, 2016c). 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, 2016d).

A biblioteca React Router tem o intuito de prover a capacidade de navegação e roteamento para uma aplicação escrita em React. Como visto em REACTROUTER (2017c), ela é flexível o suficiente para fornecer a mesma funcionalidade para o caso da aplicação ser processada no lado do servidor, o que torna a mesma essencial para atingir os objetivos aqui propostos.

2.6 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, 2016b). 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 1.

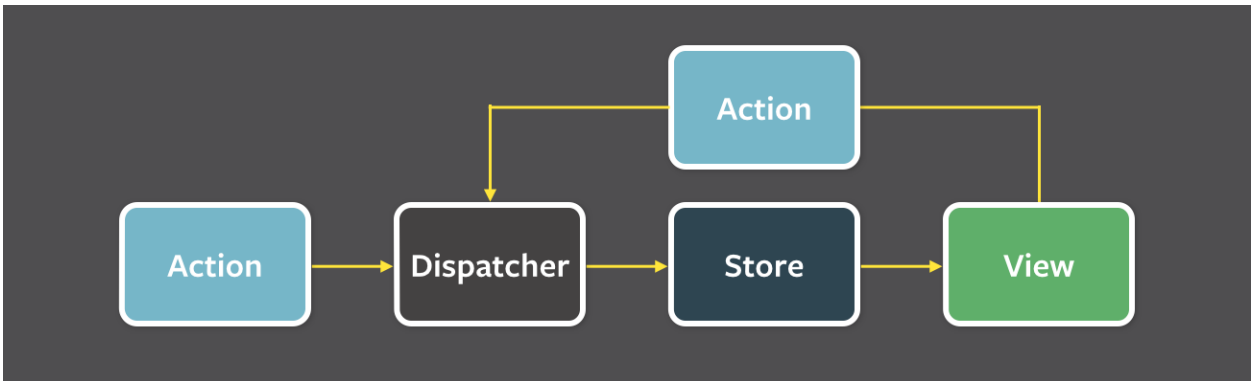

Figura 1 - Fluxo de dados da arquitetura Flux [FACEBOOK, 2016b]

Apesar de ser baseada no Flux, o Redux tem algumas peculiaridades que não seguem à risca a arquitetura, como um *Store* único e centralizado e o uso de *Action Creators*, que são os responsáveis pela criação das *Actions*, além de introduzir o conceito de *Reducers*, entidades com o propósito de manipular o *Store* com base nas *Actions* criadas pelos *Action Creators*.

2.7 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.

2.8 Outras Tecnologias

Aqui são 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. Ainda na questão do auxílio, também será utilizada a biblioteca Lodash, uma biblioteca repleta de funcionalidades com relação ao tratamento de listas e objetos em JavaScript, dentre outras.

Webpack é uma ferramenta utilizada para entregar uma versão modular de todo o código desenvolvido para a aplicação, além de incluir todas as dependências necessária e executar tarefas, como compilar a sintaxe ES6 para a conhecida ES5 através da ferramenta Babel.

Para lidar com o download e instalação das dependências previamente citadas, será utilizado o NPM (Node.js Package Manager), uma ferramenta capaz de baixar e instalar dentro do contexto a aplicação uma gama enorme de bibliotecas e frameworks disponibilizados em sua central através do envio das mesmas por seus criadores.

Além do que já foi citado anteriormente, foi escolhida a *template engine* EJS para desenvolver a única view disponível na aplicação, além de uma implementação do Google Maps em forma de componente preparada para aplicações que utilizam JavaScript Isomórfico, segundo ISTARKOV (2017).

Existem diversas outras dependências que só tem algum sentido se utilizadas em conjunto com dependências maiores, como as diversas bibliotecas para complementar React e Redux, e este tipo de dependência será citada apenas no Apêndice A, que contém uma lista de todas as dependências utilizadas no projeto, junto com suas respectivas versões.

Grande parte das dependências citadas nesta seção foram escolhidas sem uma pesquisa aprofundada sobre soluções semelhantes, e foram escolhidas simplesmente por afinidade ou facilidade no uso por parte do autor.

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 no lado do servidor da aplicação, 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 2.

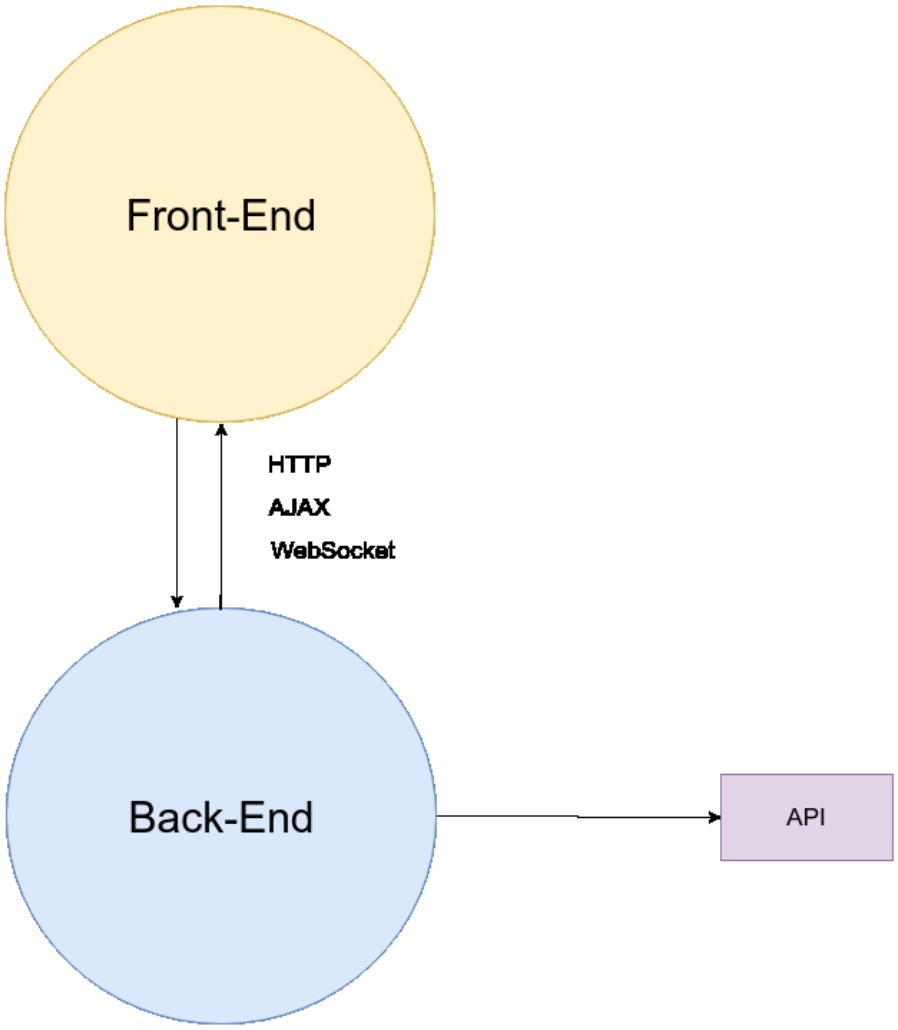

Figura 2 - Configuração de uma *Single Page Application* tradicional

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 3.

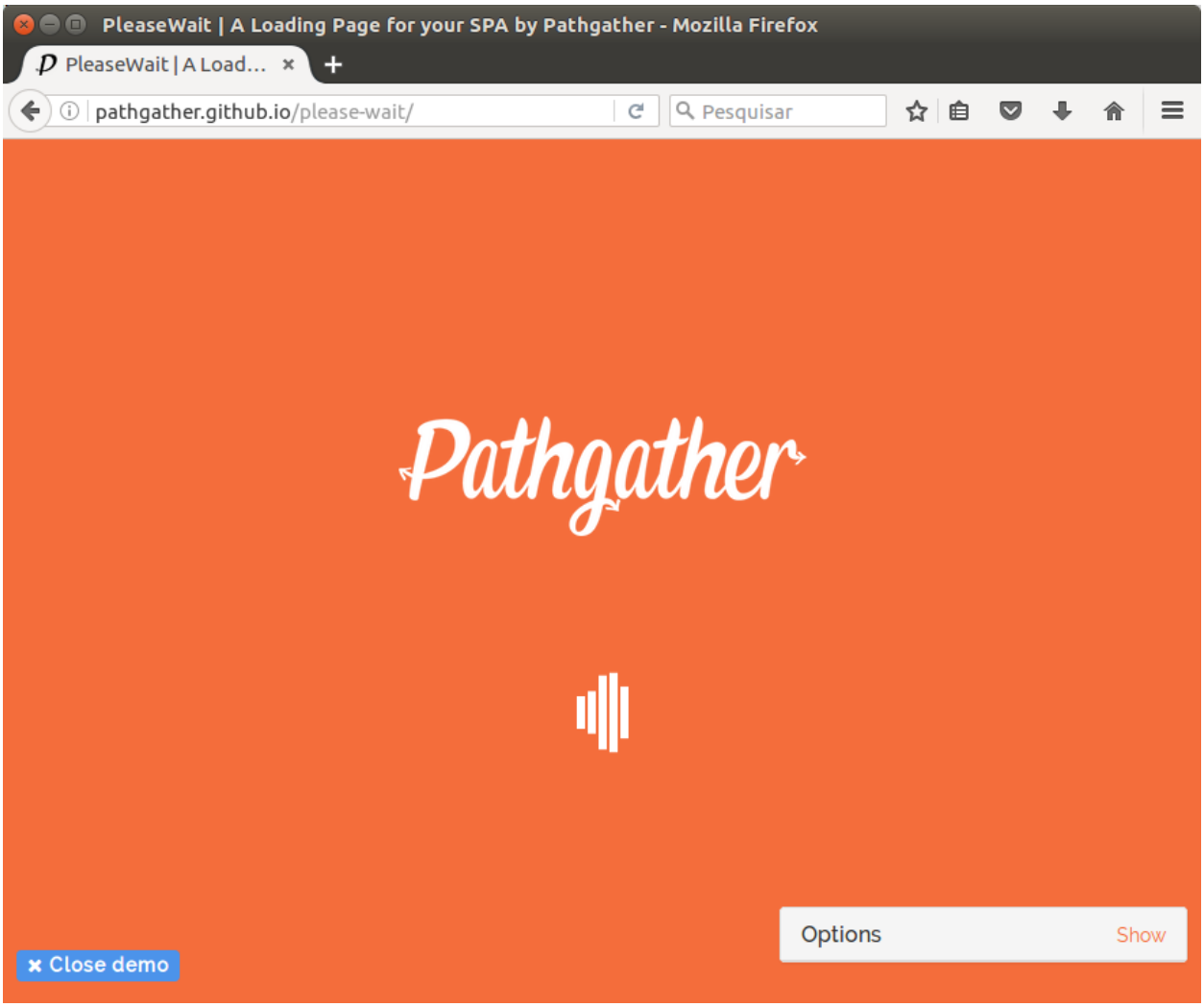

Figura 3 - Exemplo de página de carregamento de uma Single Page Application

Em uma SPA tradicional, o conteúdo semanticamente relevante da página não se encontra disponível na mesma no momento da primeira requisição ao servidor, 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 4 - 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 4, 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 (Search Engine Optimization), 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.

Outro problema inerente às SPAs tradicionais é um problema de roteamento, pois aplicações no modelo SPA utilizam tecnologias de roteamento disponíveis no cliente, o que causa um problema ao acesso direto as URLs.

Para exemplificar este problema, suponhamos que a aplicação esteja disponível através do endereço *exemplo.com* e que a mesma tenha uma segunda página que possa ser acessada através da URL example.com/page2. Ao efetuar uma requisição ao servidor, o mesmo irá retornar o Entry Point da aplicação, e o cliente ao receber o Entry Point sem nenhuma informação sobre a rota /page2 irá carregar o contexto da página inicial. Tal problema é comumente resolvido através da tecnologia conhecida como Hash History, como visto em TSONEV (2016), onde o cliente consegue manter informações sobre a rota acessada pelo usuário através de uma formatação especial na URL, que no nosso exemplo seria algo como *exemplo.com/#/page2*. O problema do uso de Hash History é a adição do caractere '#' ao URL, o que pode causar problemas ao SEO e dificultar o processo de aprendizagem e lembrança da mesma por parte dos usuários, além do cliente reter o conteúdo que segue o caractere ao efetuar requisições HTTP como visto em TSONEV (2016), o que torna o servidor alheio ao roteamento da aplicação, o que novamente, é uma das características das SPAs 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. Com relação às rotas, o servidor tem a capacidade completa de trabalhar as mais diversas rotas e mudar o contexto da aplicação dependendo das mesmas, visto que o mesmo processa a aplicação antes de enviar a resposta, conseguido assim alterar seu contexto.

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 5.

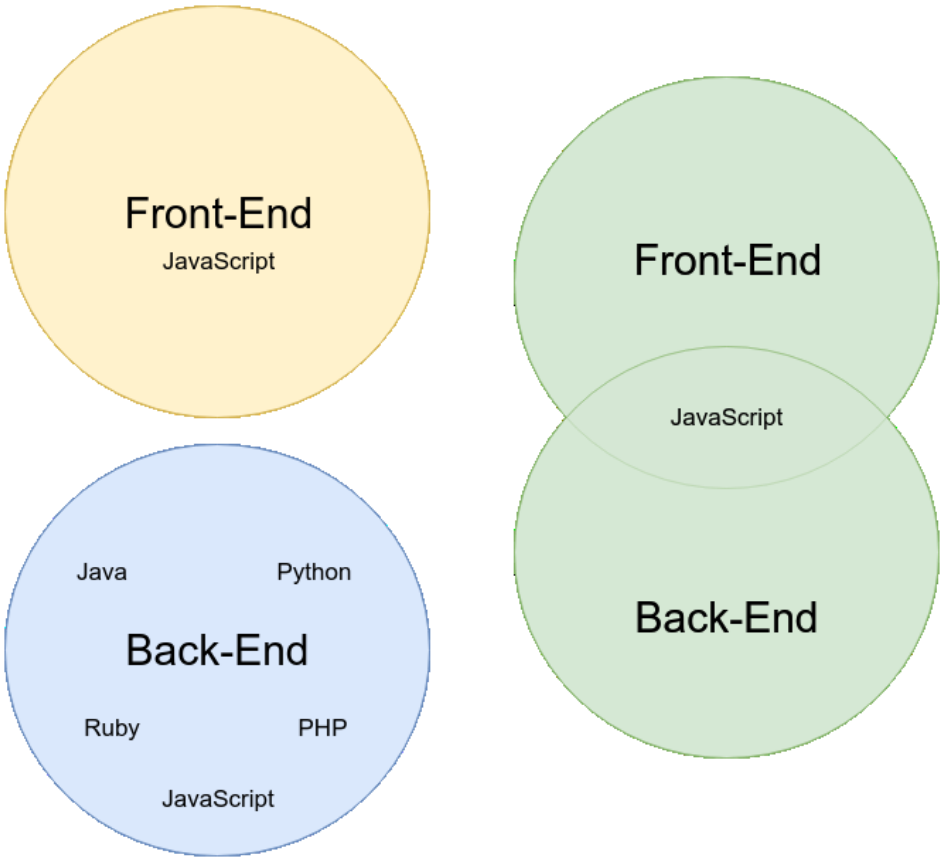

Figura 5 - Configuração de uma Single Page Application tradicional X JavaScript Isomórfico

3.3 Desenvolvimento de Aplicações Web com JavaScript Isomórfico

Para desenvolver aplicações Web empregando JavaScript Isomórfico, é necessário que as características da abordagem sejam implementadas na aplicação, utilizando tecnologias e frameworks que auxiliem neste processo. 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.

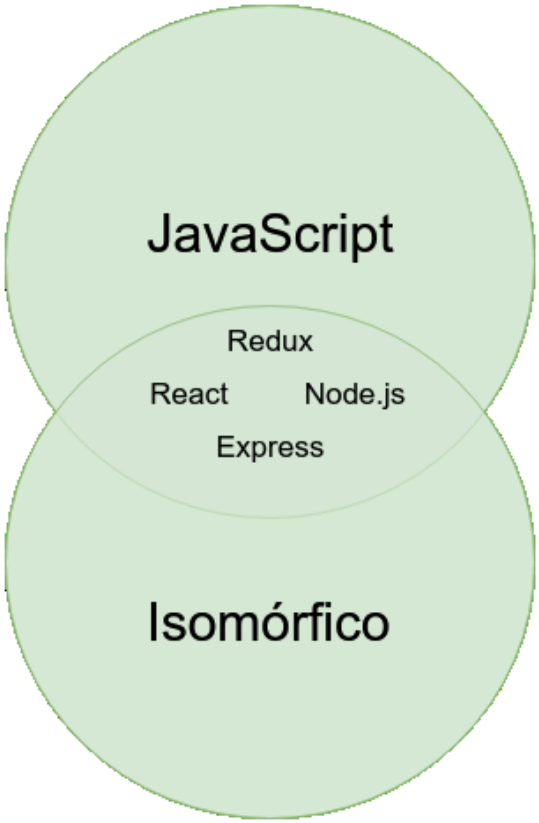

Figura 6 - Principais bibliotecas utilizadas no desenvolvimento da aplicação isomórfica

3.3.1 Tecnologias 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.

A principal motivação para a escolha destes dois frameworks em particular, além da sinergia entre ambos no processo de desenvolvimento, se dá por ambos terem a capacidade de funcionar de maneira excepcional em diversos tipos de ambiente, o que é crucial para que este tipo de aplicação possa funcionar. Tanto React como Redux tem mecanismos que, nativamente ou através de bibliotecas que enriquecem suas funcionalidades, dão a capacidade do cliente e do servidor trabalharem de forma conjunta, através de um compartilhamento de estado (por parte do Redux) e pela renderização de seus componentes sem a necessidade específica de um browser (no caso do React).

3.3.2 Tecnologias 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.

O Node.js foi escolhido por ser a ferramenta que possibilita a execução de códigos JavaScript fora do contexto de um browser (como citado anteriormente), e o Express por ser o framework mais utilizado e de maior suporte por parte da comunidade de desenvolvedores dentre suas alternativas, além de ser razoavelmente simples de se configurar e utilizar, possibilitando um rápido processo de desenvolvimento.

3.3.4 Tecnologias alternativas

As tecnologias aqui mencionadas são apenas um conjunto possível de ferramentas para se desenvolver uma aplicação isomórfica. Nesta seção serão citadas tecnologias alternativas para se alcançar o mesmo objetivo.

Como estamos lidando com o conceito de um servidor capaz de executar código JavaScript, nossas opções são restritas neste quesito. Se tratarmos o conceito de aplicação isomórfica diferente do conceito adotado neste trabalho, e permitirmos que a tecnologia responsável pelo servidor de aplicação utilize qualquer linguagem de programação, as possibilidades são enormes, porém não estão no escopo deste trabalho. Considerando este fato e o estado da arte atual, existem poucas alternativas ao NodeJS, visto que é a principal tecnologia utilizada para executar JavaScript fora do Browser, segundo CAPAN (2013).

A plataforma Meteor, que apesar de ser escrita em cima de NodeJS segue outra filosofia, pode ser utilizada para desenvolver aplicações isomórficas por sua natureza de utilizar JavaScript de ponta a ponta, segundo METEOR (2017). Meteor também funciona em conjunto com outras bibliotecas como React, o que o torna uma opção bastante viável.

Para o conjunto React e Redux, existem diversas alternativas disponíveis, dentre elas o Vue.js, como visto em VUEJS (2017), onde um esquema semelhante ao apresentado neste trabalho pode ser implementado, ou o Angular com alguma

tecnologia auxiliar como a solução proposta em ANGULARJS (2017), onde é possível alcançar os objetivos do isomorfismo. Se considerarmos o uso de tecnologias auxiliares, uma infinidade de outras tecnologias podem ser citada, por isso serão citadas apenas as anteriores.

É importante esclarecer que uma comparação entre as tecnologias citadas e as tecnologias utilizadas neste trabalho estão fora do escopo do mesmo, as tecnologias aqui utilizadas foram escolhidas puramente por experiência de desenvolvimento do autor, para facilitar o processo de desenvolvimento e demonstrar o processo de forma satisfatória.

3.4 JavaScript Isomórfico X JavaScript Universal X Server Side Rendered

Os termos JavaScript Isomórfico, JavaScript Universal e Server Side Rendered 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 de executar em múltiplos ambientes, como servidores, browsers, ou qualquer que seja o ambiente com a capacidade de executá-la, enquanto o JavaScript Universal se refere a uma aplicação escrita estritamente utilizando JavaScript como sua linguagem de programação e sendo assim, tenha a capacidade de executar em diversos ambientes, desde que o ambiente esteja preparado para o mesmo. 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, pois o conceito da palavra Isomorfismo é de que:

> “ 1 - Que tem forma igual.
>
> 2 - Que cristalizou sob formas iguais ou idênticas.”
>
> AURÉLIO (2017).

E que tal conceito leva ao entendimento de que são coisas diferentes, porém com a mesma forma, o que não representa o resultado ao qual se almeja com tal abordagem,

e muito menos “Universal”, e que o termo ideal seria “Server-Rendered”, ou como mais utilizado atualmente, “Server Side 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 executa no servidor (Server Side Rendered), tem a capacidade de executar em diversos ambientes (JavaScript Isomórfico) e tem como linguagem de programação estritamente o JavaScript (JavaScript Universal).

4. PROJETO DA APLICAÇÃO DE EXEMPLO

Este capítulo descreve a aplicação que foi desenvolvida como forma de experimentar os conceitos relacionados ao funcionamento das aplicações isomórficas discutidos anteriormente.

4.1 Descrição

A aplicação 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 são fornecidos através da Web API proposta por OLIVEIRA *et al.* (2016), e conta com um mapa interativo para prover ricas experiências de navegabilidade e visualização dos dados, além de ser possível criar modelos para impressão utilizando as consultas executadas.

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 é representada através de um modelo simplificado de Casos de Uso, referenciado como Caso de Uso Adaptado. Este modelo tenta extrair o essencial da estrutura do Modelo de Casos de Uso proposto em SCOTT (2003). 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 Adaptados

Número	1a
Nome do Caso de Uso	Pesquisa Simples
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 um único critério pelo qual ele gostaria de utilizar para filtrar sua pesquisa. 4. Se usuário não selecionar nenhum critério, será utilizado “Nome” como critério. 5. Sistema executa a pesquisa e exibe os resultados em uma tela contendo um mapa através de pontos.
Condição de Saída	1. Sistema termina 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 “Pesquisa Simples”

Número	1b
Nome do Caso de Uso	Pesquisa Avançada
Ator	Usuário
Condição de Entrada	Usuário deve ter acesso ao sistema.
Fluxo de Eventos	1. Usuário clica no botão para trazer o menu de opções. 2. Usuário é apresentado com os diversos filtros que pode utilizar para efetuar a pesquisa, sendo possível combiná-los. 3. Usuário insere o valor pelo qual deseja pesquisar nos campos. 4. Sistema executa a pesquisa e exibe os resultados em uma tela contendo um mapa através de pontos.
Condição de Saída	1. Sistema termina 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 2 - Caso de Uso “Pesquisa Avançada”

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 sai da tela. 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 3 - Caso de Uso “Navegar pelo mapa”

Número	3
Nome do Caso de Uso	Exibir Resultados
Ator	Usuário
Condição de Entrada	Caso de uso (1)
Fluxo de Eventos	1. Usuário clica para exibir o menu de opções. 2. Usuário seleciona a opção Resultados. 3. Sistema exibe os resultados da pesquisa para o usuário.
Condição de Saída	1. Usuário fecha a tela de exibição
Exceções	1. Caso nenhum resultado esteja disponível, o usuário deve ser informado.
Requisitos Especiais	Nenhum.

Tabela 4 - Caso de Uso “Exibir Resultados”

Número	4
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 5 - Caso de Uso “Detalhar Ocorrência”

Número	5
Nome do Caso de Uso	Exportar Ocorrência para Impressão
Ator	Usuário
Condição de Entrada	Caso de uso (4)
Fluxo de Eventos	1. Usuário seleciona opção de relatório 2. Sistema exibe as informações da ocorrência em um formato amigável para impressão
Condição de Saída	1. Usuário pressiona o botão retornar.
Exceções	Nenhuma
Requisitos Especiais	Nenhum

Tabela 6 - Caso de Uso “Exportar Ocorrência para Impressão”

Número	6
Nome do Caso de Uso	Gerar Relatório de Ocorrências
Ator	Usuário
Condição de Entrada	Caso de uso (3)
Fluxo de Eventos	1. Usuário clica na opção de Relatório na tela de resultados. 2. Sistema exibe os detalhes de todas as ocorrências que foram resultado da pesquisa.
Condição de Saída	1. Usuário clica no botão Retornar.
Exceções	1. Caso a pesquisa não tenha fornecido nenhum resultado, o botão de relatórios deve estar inacessível. 2. Um grande volume de dados retornados pela pesquisa pode deteriorar a performance da geração do relatório desejado pelo usuário.
Requisitos Especiais	Nenhum

Tabela 7 - 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. 2. Deve ser capaz de se comunicar com a API fornecida para o experimento. 3. Deve ser capaz de mostrar conteúdo ao usuário exibindo novas telas de visualização.
Requisitos Não-Funcionais	1. 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 URL inserida em um browser. 4. A aplicação deve ser escrita utilizando JavaScript, no modelo Single Page Application.

Tabela 8 - Requisitos

4.3 Desenvolvimento

Neste tópico será reportado todo o processo de desenvolvimento da aplicação, desde a instalação de suas dependências à implantação da mesma em um servidor.

4.3.1 Instalando dependências

Para iniciar o desenvolvimento da aplicação, é necessário inicialmente instalar todas as dependências que serão utilizadas durante o processo de desenvolvimento. Muitas das dependências já foram citadas anteriormente, e uma lista completa de dependências pode ser consultada no Apêndice A.

A ferramenta utilizada para gerenciar as dependências do projeto é o NPM, e antes mesmo de se instalar as dependências é necessário inicializar um novo projeto com o NPM através do comando *npm init*. Após o levantamento de todas as dependências necessárias, as mesmas podem ser instaladas inserindo em uma linha de comando o conteúdo do Quadro 1.

```
npm i -D axios babel-cli babel-core babel-loader babel-preset-es2015 babel-preset-
react babel-preset-stage-0 babel-register ejs eslint-config-rallycoding eslint_d express
google-map-react lodash mocha nodemon react react-dom react-hot-reloader@3.0.0-
beta.6 react-redux react-router@3.0.2 redux redux-form@6.6.3 redux-thunk
webpack@2.3.2 webpack-dev-server@2.4.2
```

Quadro 1 - Comando de instalação das dependências

Algumas das dependências utilizadas no comando acima tem uma versão explicitada (através da concatenação do caractere ‘@’), pois ocorreram mudanças drásticas entre as versões descritas e as versões posteriores, o que faria o desenvolvimento da aplicação mudar dependendo do momento da instalação.

Com as dependências devidamente instaladas e o projeto iniciado, podemos definir a estrutura de arquivos do projeto.

4.3.2 Estrutura de diretórios

Por se tratar de um projeto isomórfico, a estrutura de diretórios do projeto é muito importante para refletir conceitos relacionados à abordagem. Inicialmente, após

se iniciar o projeto com o NPM e instalar as dependências necessárias, o diretório do projeto deve conter apenas o arquivo *package.json* e o diretório *node_modules*, que são respectivamente o arquivo de controle do NPM e a pasta onde o mesmo instalou todas as dependências.

Levando em conta os fatos apresentados, a estrutura de diretórios apresentada no Quadro 2 é proposta como a estrutura do projeto.

- public: Diretório que contém recursos estáticos
  - assets/icons: Diretório que contém arquivos relacionados a ícones
  - css: Diretório que contém bibliotecas CSS e o arquivo de CSS criado para o projeto
  - fonts: Diretório que contém as fontes utilizadas no projeto
  - js: Diretório que contém bibliotecas externas e dependências JavaScript utilizadas no cliente

- src: Diretório que contém o código-fonte
  - client: Diretório com arquivos relacionados ao cliente (Front-end)
  - server: Diretório com arquivos relacionados ao servidor (Back-end)
  - shared: Diretório com arquivos utilizados por ambas partes

Quadro 2 - Estrutura de Diretórios na Raíz do Projeto

Com a estrutura de diretórios principal do projeto definida, é necessário ainda definir a estrutura de diretórios dentro do diretório *src*. Para o diretório *client*, levando em consideração todo o fluxo de uma aplicação que utiliza React e Redux, foi definida a estrutura de arquivos do Quadro 3.

- actions: Diretório que armazena os *Action Creators*
- components: Diretório que contém os *Dumb Components* do projeto
- containers: Diretório que contém os *Smart Components* do projeto
- reducers: Diretório que contém os *Reducers* do projeto
- constants.js: Arquivo que contém as constantes utilizadas pelo cliente
- index.js: Arquivo ponto de entrada para a aplicação
- store.js: Arquivo responsável por inicializar o *Store* da aplicação

Quadro 3 - Estrutura de arquivos do diretório *client*

Os conceitos de *Dumb Components* e *Smart Components*, segundo ABRAMOV (2015), estão intimamente atrelados a uma aplicação desenvolvida utilizando React e a arquitetura Flux. São ditos *Dumb Components* os componentes que têm a função de apenas mostrar algum conteúdo sem se preocupar com o estado da aplicação, enquanto os *Smart Components* são componentes que lidam com alterações no estado da aplicação diretamente. Jargões como *Action Creators*, *Reducers* e *Store* podem ter seu significado conferido na seção 2.5. Para o diretório *server*, é proposta a estrutura do Quadro 4.

- views: Diretório que contém os *templates* da aplicação
    - index.ejs: *Template* principal da aplicação, que utiliza EJS
- isomorphic.js: Arquivo responsável por tratar o roteamento e execução do cliente do lado do servidor
- server.js: Arquivo responsável por configurar e iniciar um servidor de aplicação utilizando Node e Express

Quadro 4 - Estrutura de arquivos do diretório *server*

O papel de cada arquivo deste diretório será descrito na próxima seção, porém vale ressaltar que em uma aplicação tradicional, na maioria dos casos por se tratar apenas de um arquivo HTML que utiliza um arquivo JavaScript com toda a lógica e visualização da aplicação, não existe a necessidade de se utilizar mecanismos de templates, enquanto em uma aplicação isomórfica eles têm um papel fundamental para prover a possibilidade do conceito ser colocado em prática.

O diretório *shared* contém o arquivo de rotas da aplicação e um arquivo chamado *util.js* que contém funções que são utilizadas tanto no cliente como no servidor, como a transformação de um objeto JavaScript para o formato de parâmetros da URL, por exemplo.

4.3.3 A configuração do servidor

Como dito anteriormente, para que seja possível que a abordagem isomórfica seja posta em prática, é necessária uma configuração cuidadosa na parte do servidor (Back-end) da aplicação para que todos os conceitos discutidos anteriormente sejam válidos. Em termos práticos, é necessário que o servidor tenha a capacidade de processar a parte do cliente antes de enviar a resposta HTTP, enviando na mesma o resultado do processamento junto com os demais dados oriundos da requisição. O React contém um módulo especial para este tipo de necessidade, que se faz disponível através de uma função conhecida como *renderToString*, disponível através do pacote *server* da biblioteca *ReactDOM.*

O funcionamento desta função, juntamente com outros conceitos, pode ser analisado através do trecho de código mostrado no Quadro 5, retirado do arquivo *isomorphic.js*, citado na seção 4.3.2

```
import React from 'react';
import { renderToString } from 'react-dom/server';
import { Provider } from 'react-redux';
import { match, RouterContext } from 'react-router';

import routes from '../shared/routes';
import store from '../client/store';

export default (app) => {
  app.get('*', (req, res) => {
    match({ routes, location: req.url }, (error, redirect, props) => {
      if (error) {
        res.status(500).send(error.message);
      } else if (redirect) {
        res.redirect(redirect.pathname + redirect.search);
      } else if (props) {
        const reactOutput = renderToString(
          <Provider store={store}>
            <RouterContext {...props} />
          </Provider>
        );
        res.render('index.ejs', { reactOutput });
      } else {
        res.status(404).send('Not found!');
      }
    });
  });
};
```

Quadro 5 - Código-fonte do arquivo isomorphic.js

Neste arquivo, é definido o tratamento de rota para uma requisição HTTP do tipo GET, utilizando o caractere ‘*’ para representar qualquer rota enviada ao servidor. Após o tratamento de erros e redirecionamentos, a função *renderToString* trata de executar a aplicação do cliente no lado do servidor, levando em consideração o contexto de rotas da biblioteca *React Router* e o estado da aplicação da biblioteca *Redux*. Ao final da execução, a função envia como resposta o resultado da execução da mesma inserido dentro do template *index.ejs* na variável *reactOutput.*

Este arquivo, em conjunto com o template, é o principal responsável por interligar todas as tecnologias e conceitos citados até o momento e tornar o desenvolvimento de uma aplicação isomórfica possível. Vale ressaltar que essa é apenas uma implementação possível destes conceitos, e que podem existir outras alternativas para se alcançar o mesmo resultado.

O trecho importante do template, responsável por mostrar o resultado esperado e realizar a ligação com a função *renderToString*, pode ser conferido no Quadro 6:

```
<body>
  <div id="root"><%- reactOutput %></div>
</body>
```

Quadro 6 - Template da página principal da aplicação

4.3.4 Página Inicial

Com o servidor preparado e configurado para lidar com a aplicação, inicia-se o processo de desenvolvimento da parte do cliente (Front-end).

Para cumprir com os requisitos apresentados, são necessárias basicamente três páginas, sendo que o conceito de página no escopo deste trabalho, como já discutido anteriormente, não está ligado ao conceito de três arquivos HTML como em outras aplicações. Tais páginas podem ser definidas como:

- Mapa: a página de mapa deve ser a página principal da aplicação, onde o usuário terá contato com um mapa interativo e todas as funcionalidades do mesmo, como descritos na seção 4.2

- Relatório: uma página na qual o usuário será apresentado apenas com dados referentes a pesquisas realizadas na página de interação da aplicação.
- Pagina Inicial: uma página que será apresentada ao usuário explicando sobre a aplicação e fornecendo um campo para pesquisa simples, que levará à pagina de Mapa. O resultado visual pode ser conferido na Figura 7.

Figura 7 - Página Inicial da Aplicação

4.3.5 Mapa

Como dito na seção 4.3.4, a principal interface de interação entre o usuário e a aplicação é através de uma página chamada de Mapa.

Por se tratar do componente principal da aplicação, o componente de Mapa é um *Smart Component* que utiliza internamente um componente de mapa interativo fornecido pela biblioteca *google-map-react*, como visto em ISTARKOV (2017). Tal biblioteca se trata de um invólucro ao serviço Google Maps, e foi escolhida dentre tantas outras por estar preparada para trabalhar de maneira isomórfica sem problemas

de compatibilidade. Os problemas de compatibilidade entre a abordagem isomórfica e algumas bibliotecas serão discutidos na seção 4.4.

Em posse de uma tecnologia capaz de apresentar os mapas interativos ao usuário, o próximo passo é criar componentes para representar visualmente no mapa os dados retornados nas consultas realizadas pelo usuário, através da aplicação, para a API proposta anteriormente. Foram criados componentes para representar pontos no mapa e informações de forma parcial sobre cada ponto, e o resultado dos mesmos podem ser conferidos na Figura 8.

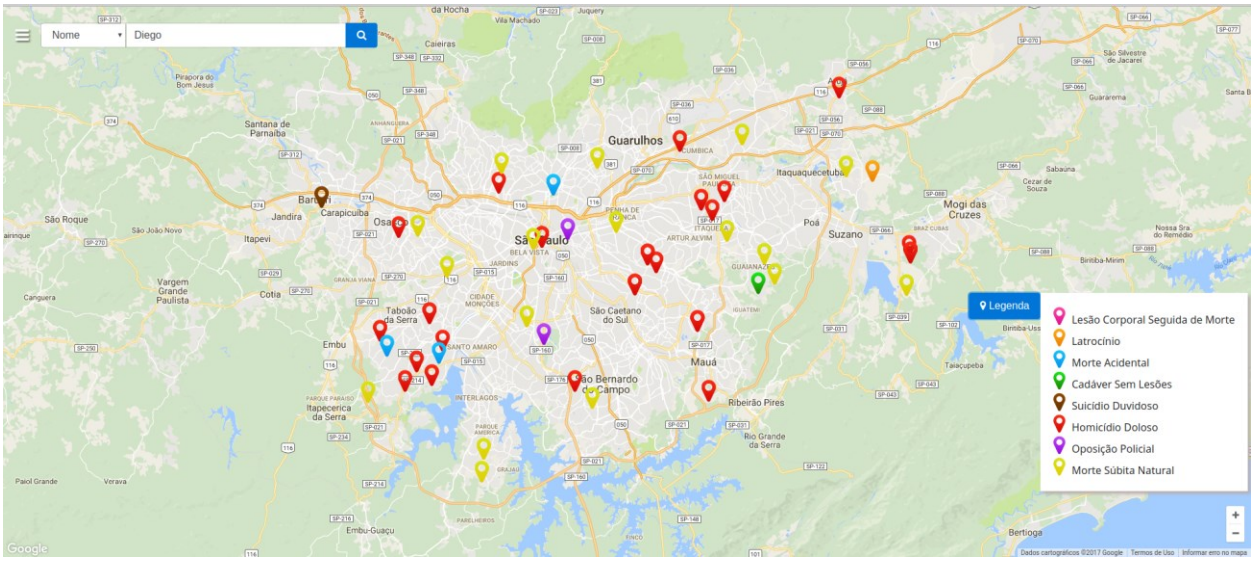

Figura 8 - Página principal da aplicação

4.3.6 Integração com a API

Com o componente responsável por apresentar a interface principal ao usuário, é necessário alimentar o mesmo com os dados da API. Para tal, é necessário entender como a API funciona e como devem ser executadas consultas na mesma. A API proposta se trata de uma RESTful API que, através de requisições HTTP do tipo GET a rotas predefinidas com parâmetros propostos em OLIVEIRA *et al.* (2016), fornece os dados através de um objeto que utiliza a notação JSON.

Conhecendo o funcionamento da API, podemos criar um componente capaz de efetuar consultas na mesma através de dados inseridos pelo usuário, e refletir o resultado da resposta da API no estado da aplicação. Por se tratar de um componente

capaz de alterar o estado da aplicação, o componente de buscas deve ser um *Smart Component*, segundo o que foi definido anteriormente.

Ao definir os componentes de mapa e pesquisa como componentes capazes de acessar o estado da aplicação, ou seja, componentes capazes de manipular e sofrerem alterações através dos mecanismos da biblioteca Redux, a atualização visual do componente de mapa para refletir o estado da aplicação com os resultados da consulta a API ocorre de maneira transparente ao desenvolvedor. Tal processo pode ser representado pela Figura 9.

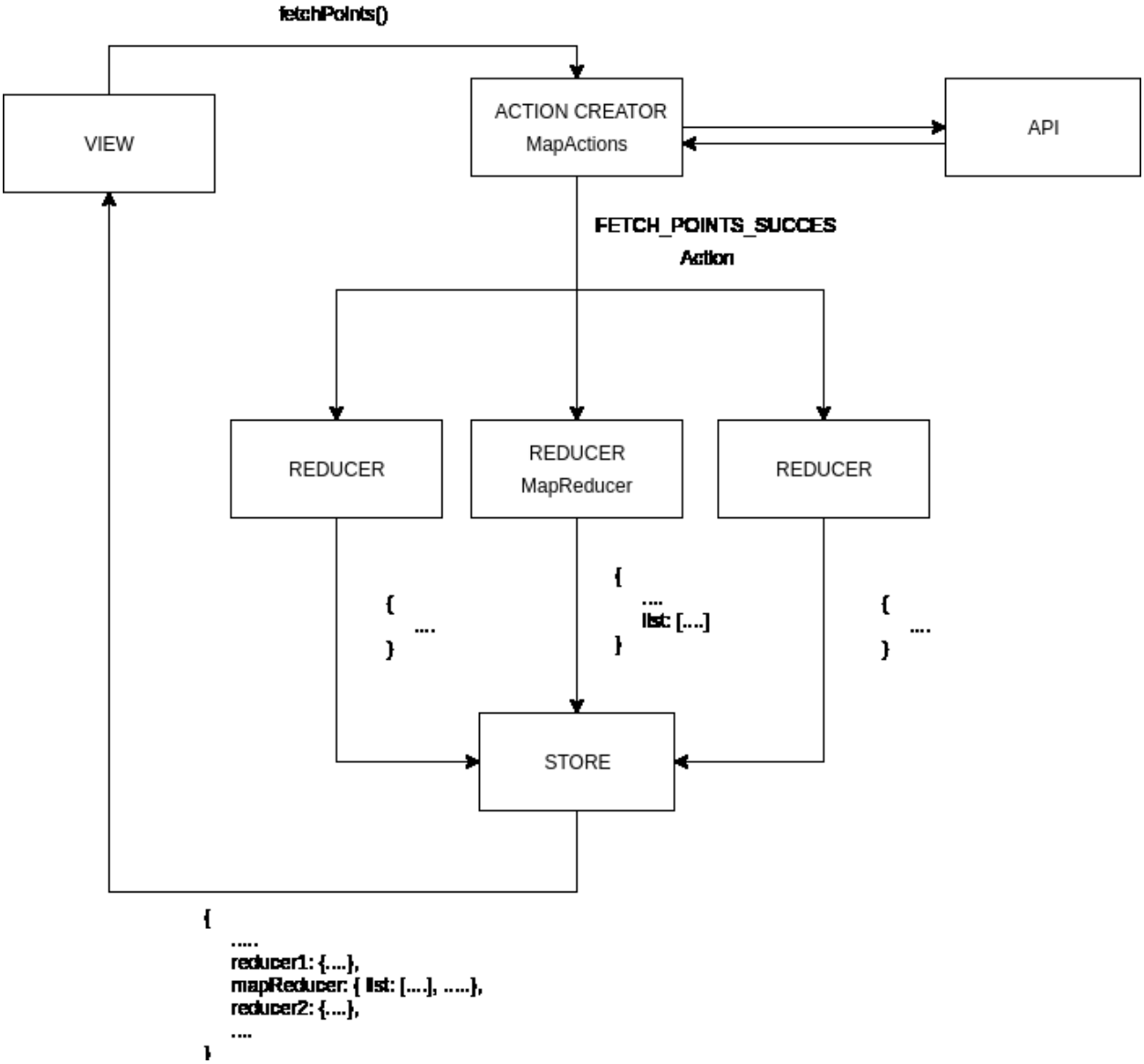

Figura 9 - Fluxo simplificado da interação entre React e Redux

Tal comportamento pode ser observado entre todos os componentes da aplicação capazes de lidar com o estado da mesma, e este é um dos principais motivos da utilização das bibliotecas React e Redux em conjunto.

Além do componente de pesquisa completo, chamado de ‘Pesquisa Avançada’, onde estão disponíveis ao usuários campos de formulário HTML correspondentes a todos os parâmetros que a API aceita para executar consultas, existe um componente de pesquisa rápida que se trata de uma barra de pesquisa (um *text input* do HTML) com um botão e um menu para selecionar um campo apenas para pesquisa, que está disponível na página de mapa e também na página inicial, como visto nas Figuras 7 e 8.

4.3.7 Exibindo resultados

Para mostrar os dados sobre cada ocorrência, de maneira parcial, foi criada na API uma nova rota que disponibiliza apenas o mínimo de dados necessários, e tais dados podem ser visualizados ao mover o *mouse* por cima de um ponto no mapa, como pode ser visto na Figura 10, ou através do menu lateral da Figura 11. Segundo os requisitos apresentados na seção 4.2, existe ainda um caso onde o usuário deseja ver todos os detalhes disponíveis sobre uma ou mais ocorrências, e para tal é necessário um novo componente capaz de armazenar tais informações. Como a única função deste componente é mostrar tais dados, o mesmo pode ser um *Dumb Component* e seu estado será mantido pelo componente principal.

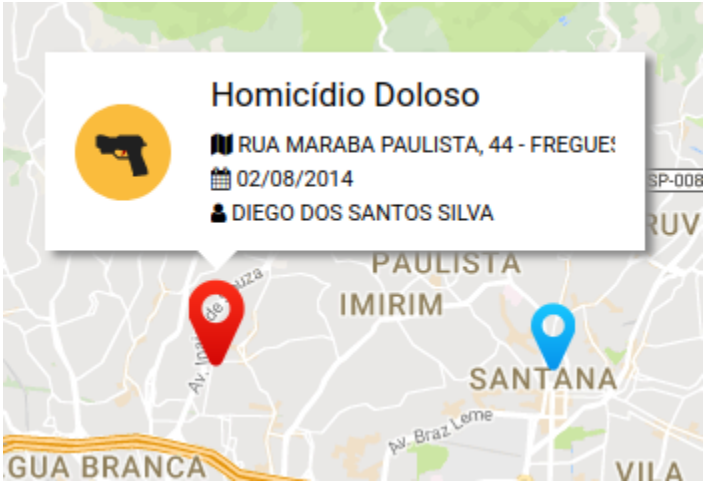

Figura 10 - Informações sobre um ponto no mapa

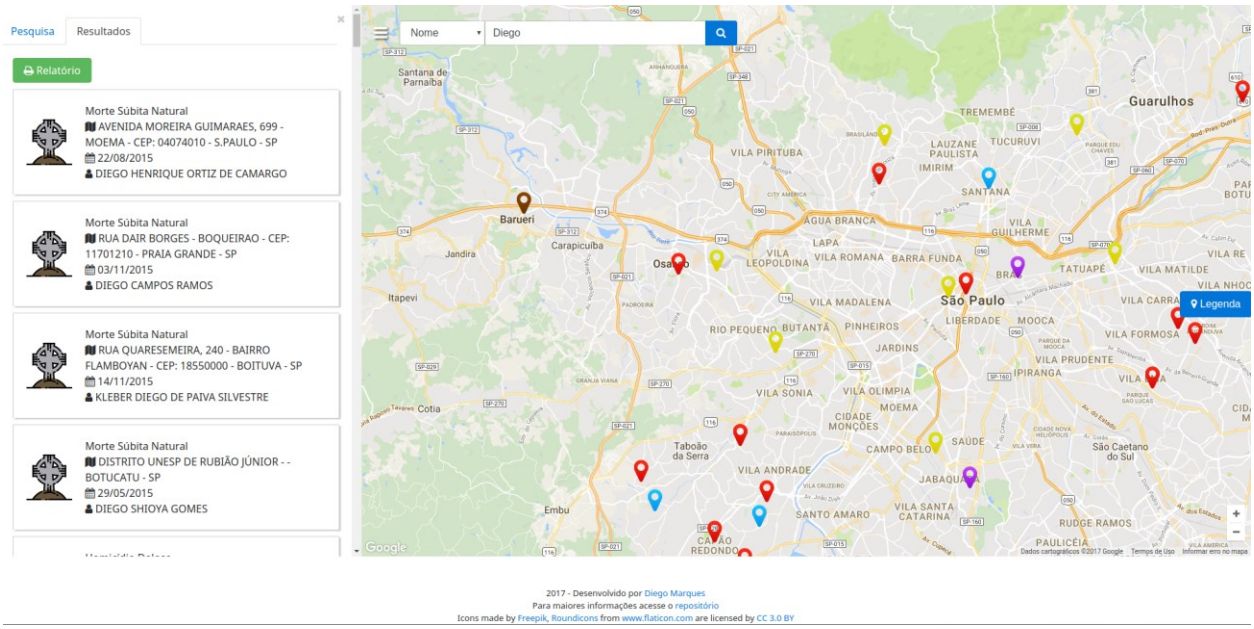

Figura 11 - Informações de todos os pontos de uma pesquisa no menu lateral

A API fornece uma rota especial que, através de um parâmetro contendo a identificação da ocorrência, parâmetro esse que é armazenado no estado da aplicação para cada ponto utilizando a rota de pesquisa com resultados parciais, retorna todos os dados sobre uma determinada ocorrência. Tal rota é suficiente para o caso de se detalhar um ponto individual, porém existe um segundo caso onde o usuário deseja detalhar todos os pontos de sua pesquisa, que deve ser tratado utilizando outra rota, também disponível pela API.

O resultado visual do detalhamento de uma ocorrência pode ser visualizado na Figura 12 e de todas as ocorrências de uma pesquisa na Figura 13.

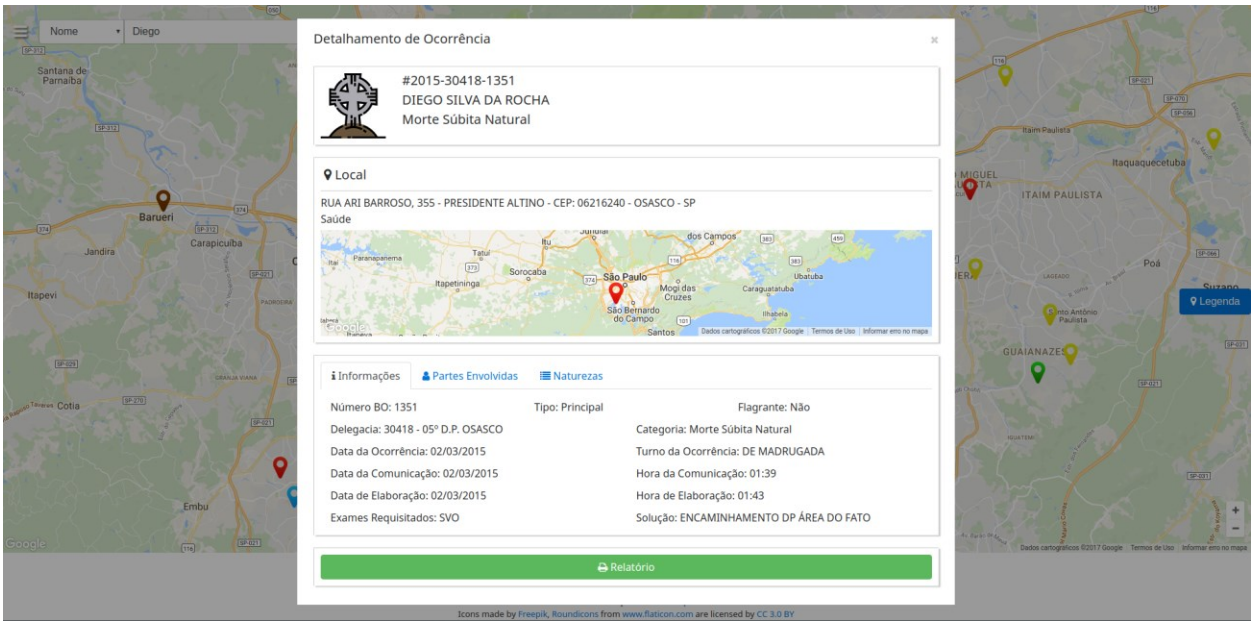

Figura 12 - Tela de detalhamento de uma ocorrência

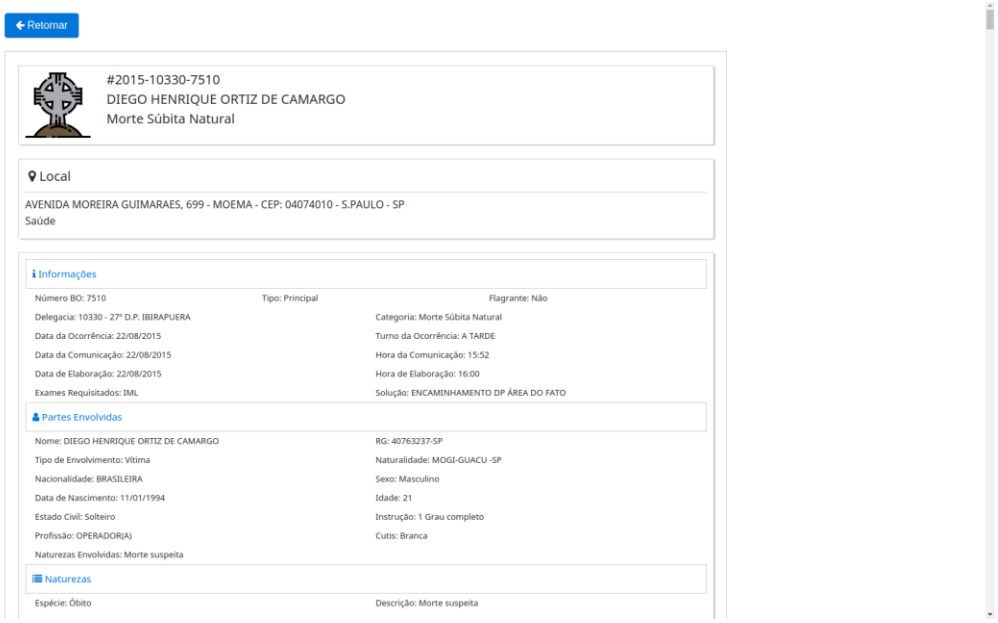

Figura 13 - Tela de lista de ocorrências detalhadas (Relatório)

4.3.8 Publicando a aplicação

Com todos os requisitos definidos implementados e com a aplicação funcionando, é possível hospedar a aplicação para acesso através da internet em

algum serviço especializado. Com base nas ferramentas utilizadas, como o Webpack e NPM, é muito simples a portabilidade do projeto e instalação do mesmo em qualquer ambiente, bastando que todas as dependências estejam instaladas. O processo de instalação e execução da aplicação, em modo ‘desenvolvimento’ e em modo ‘produção’, pode ser conferido no Apêndice C, e todo o código-fonte da mesma está disponível no Apêndice D.

Apesar da aplicação isomórfica estar implementada, para que os testes e as métricas possam ser executados comparando uma aplicação tradicional a uma aplicação isomórfica, é necessário que exista uma versão tradicional da aplicação. Como visto na seção 4.3.3, os conceitos que envolvem o JavaScript Isomórfico são tratados pelos frameworks através da união de uma *template engine* com a capacidade de processar o código do cliente no servidor. Portanto para se obter uma versão tradicional derivada da aplicação isomórfica desenvolvida até o momento, é necessário uma alteração na estrutura da aplicação nestes pontos.

Para a aplicação tradicional, o tratamento de rotas deve ser feito de uma maneira alternativa, que pode ser descrita através do trecho de código do Quadro 7, que se trata de uma alteração no arquivo *server.js*, dentro do diretório responsável pelos arquivos do servidor da aplicação. Além da alteração no arquivo *server.js*, é necessária a remoção da funcionalidade dentro do template index.ejs, mostrada no Quadro 8.

```
import isomorphic from './isomorphic';
...
// O tratamento de rotas do arquivo ‘isomorphic.js’
// não é utilizado em uma abordagem tradicional
//isomorphic(app);

app.get(‘*’, (req, res) => {
  res.render(‘index.ejs’, {});
});


….
```

Quadro 7 - Trecho do arquivo de configuração do servidor Express para a abordagem tradicional

```
  <body>
    <!-- Anteriormente, a variável ‘reactOutput’ era inserida dentro da div abaixo -->
    <div id="root"></div>
  </body>
```

Quadro 8 - Trecho do template da página para a abordagem tradicional

O resultado de ambas abordagens pode ser conferido nos endereços *http://isomorfico.tcc.diegomarques.me/* e *http://tradicional.tcc.diegomarques.me/*, respectivamente. Vale ressaltar que estes endereços estarão disponíveis durante todo o período do ano de 2017, ano da publicação deste trabalho, porém os mesmos não serão garantidos posteriormente, sendo necessário executar a aplicação de forma local. O processo de execução local pode ser conferido no Apêndice C.

4.4 Dificuldades no desenvolvimento

Apesar de todos os benefícios que foram citados anteriormente sobre a abordagem de desenvolvimento isomórfica, cabe aqui o relato de um dos principais problemas encontrados por causa do uso da mesma no processo de desenvolvimento. Muitas ferramentas e tecnologias para JavaScript não estão preparadas para lidar com aplicações de natureza isomórfica, como ferramentas que necessitam de recursos que só estão disponíveis nos *browsers*, pois apesar do servidor ter a capacidade de executar o código JavaScript, não é a intenção das tecnologias do servidor de terem as mesmas capacidades dos *browsers*, o que restringe as possibilidades de um desenvolvedor.

Por conta disso, algumas bibliotecas tiveram que ser completamente trocadas durante o processo de desenvolvimento, pois tal comportamento só pôde ser percebido após o início de seu uso. Caso as aplicações isomórficas se tornem o novo padrão adotado pela comunidade de desenvolvedores, a tendência é que estes casos comecem a se tornar isolados, porém os mesmos existem e causam problemas se não tratados de antemão.

Outro ponto não relacionado com as tecnologias, mas sim ao processo de desenvolvimento, é que o modelo da aplicação é um modelo diferente do tradicional, o que torna a organização e estruturação do projeto uma tarefa mais complexa, pois a mesma deve idealmente refletir o modelo da aplicação. Adicionalmente, há uma curva de aprendizagem maior em comparação com as aplicações tradicionais, pois é necessário lidar com novos conceitos.

5. TESTES E MÉTRICAS

Com a aplicação completamente implementada e implantada em um servidor, é possível efetuar testes que nos permitam chegar a conclusões com base nos conceitos apresentados e diversos outros conceitos correlatos. Os testes apresentados neste capítulo foram efetuados de diversas formas, utilizando desde ferramentas automatizadas até testes manuais, e têm como objetivo comparar as duas aplicações, tentando verificar se os conceitos que foram apresentados sobre as aplicações isomórficas no capítulo 3 são válidos. Segundo os conceitos vistos, a aplicação isomórfica deve demonstrar diversas melhorias em comparação com a aplicação tradicional.

5.1 A Primeira requisição

Como dito no capítulo 3, a principal diferença entre uma aplicação isomórfica e uma aplicação tradicional é o conceito da primeira requisição. A resposta à primeira requisição HTTP de uma aplicação isomórfica deve conter a estrutura da página HTML completamente disponível, ou seja, o cliente deve necessitar do mínimo de mudanças possíveis em seu DOM para que a aplicação esteja disponível visualmente ao usuário após o mesmo processar tal resposta. Isto irá depender não somente dos conceitos apresentados, mas também do processo de renderização das tecnologias escolhidas e como ambos os lados, cliente e servidor, se conciliarão para aproveitar os ganhos de performance que foram apresentados teoricamente ao longo deste trabalho.

Para testar este comportamento, é necessário efetuar apenas uma requisição a cada página e avaliar, através do conteúdo de sua resposta, se as requisições geraram conteúdos diferentes. Para que os conceitos sejam provados, é necessário ainda que a aplicação isomórfica e a aplicação tradicional apresentem o comportamento acima, onde a aplicação isomórfica deve conter o conteúdo do DOM completo em sua resposta e a aplicação tradicional deve mostrar apenas seu *Entry Point*.

Com base nesta proposta, foram efetuadas as requisições utilizando o equipamento mostrado na Tabela 9.

Processador	Intel(R) Core(TM) i7-2670QM CPU @ 2.20GHz
Memória RAM	8GB
Conexão	NET Virtua 60Mbps
Ferramenta	Navegador Google Chrome Versão 58.0.3029.110 (64-bit)
SO	Linux chernobyl 4.8.0-53-generic #56-Ubuntu SMP Tue May 16 00:23:44 UTC 2017 x86_64 x86_64 x86_64 GNU/Linux

Tabela 9 - Configurações do equipamento para testes com navegador

O resultado dos testes efetuados nesta seção podem ser observados através das Figuras 14 e 15 em suas respectivas linhas 17. Na Figura 14, podemos ver os primeiros elementos do DOM da aplicação dentro do *Entry Point* na resposta do HTTP, sendo que grande parte do seu conteúdo foi omitido por conta da maneira que a ferramenta mostra os dados. Já na Figura 15 podemos ver apenas o *Entry Point* da aplicação, portanto podemos concluir que os conceitos relacionados puramente ao DOM são verdadeiros. O único conceito apresentado neste trabalho que se encaixa nesta categoria é o conceito de melhoria no processo de SEO da página, pois os demais conceitos como balanceamento de carga de processamento necessitam de testes mais aprofundados a nível de Sistema Operacional e utilização de recursos do mesmo, e não podem ser considerados verdadeiros apenas com este teste. Tais testes podem ser conferidos na seção 5.4.

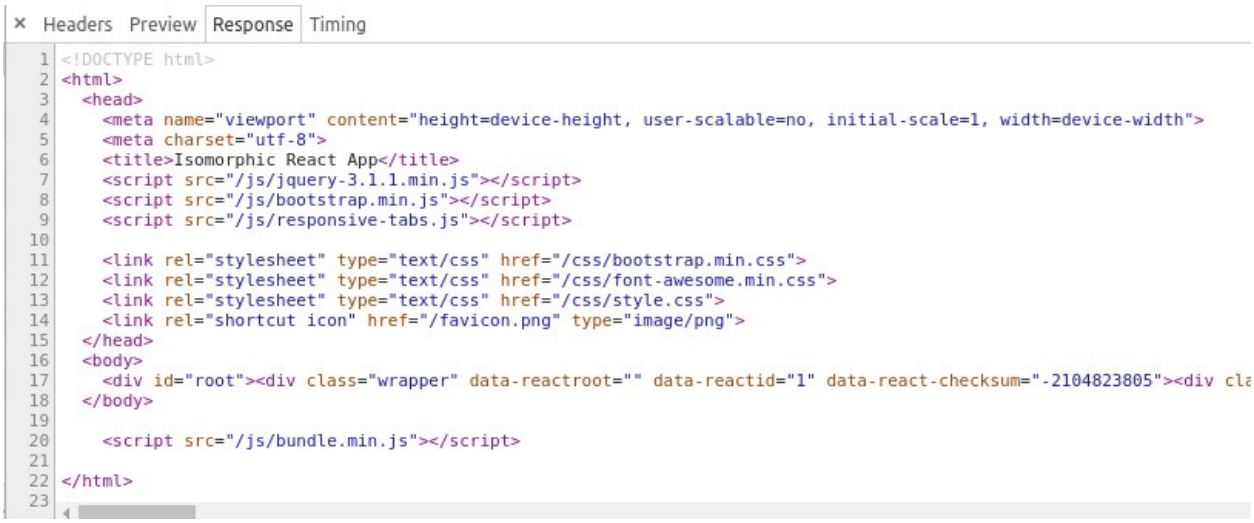

Figura 14 – Resposta a uma requisição da aplicação isomórfica

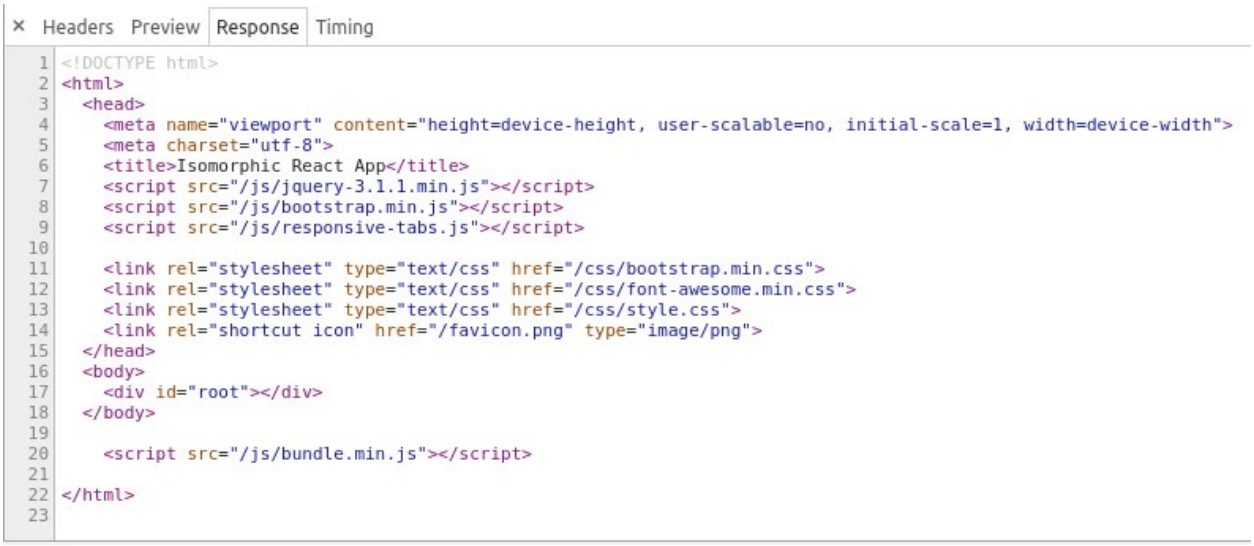

Figura 15 – Resposta a uma requisição da aplicação tradicional

5.2 Roteamento com acesso direto

Como já visto na seção 3.2, um dos problemas comumente encontrados nas aplicações tradicionais é o problema de roteamento através do acesso direto, que é um problema relacionado a frameworks e configurações do tratamento de rotas. Basicamente as aplicações tradicionais encontram problemas para tratar o contexto e o roteamento da aplicação quando é inserida uma rota diretamente na URL, como por exemplo *http://tradicional.tcc.diegomarques.me/mapa*

Para efetuar o teste deste problema, é necessário apenas efetuar uma requisição em cada aplicação e verificar se o conteúdo visual da página é o mesmo ou se a aplicação tradicional retorna algum tipo de erro.

Com base nos resultados que podem ser conferidos nas Figuras 16 e 17, as páginas apresentam o mesmo conteúdo, contradizendo tal premissa. Isto pode ser explicado com base nas tecnologias utilizadas, especialmente na biblioteca React Router. Como visto em REACTROUTER (2017a) e REACTROUTER (2017b), a biblioteca React Router trata as rotas pelo lado do cliente de forma que esta deficiência seja suprida sem a necessidade de uma aplicação isomórfica.

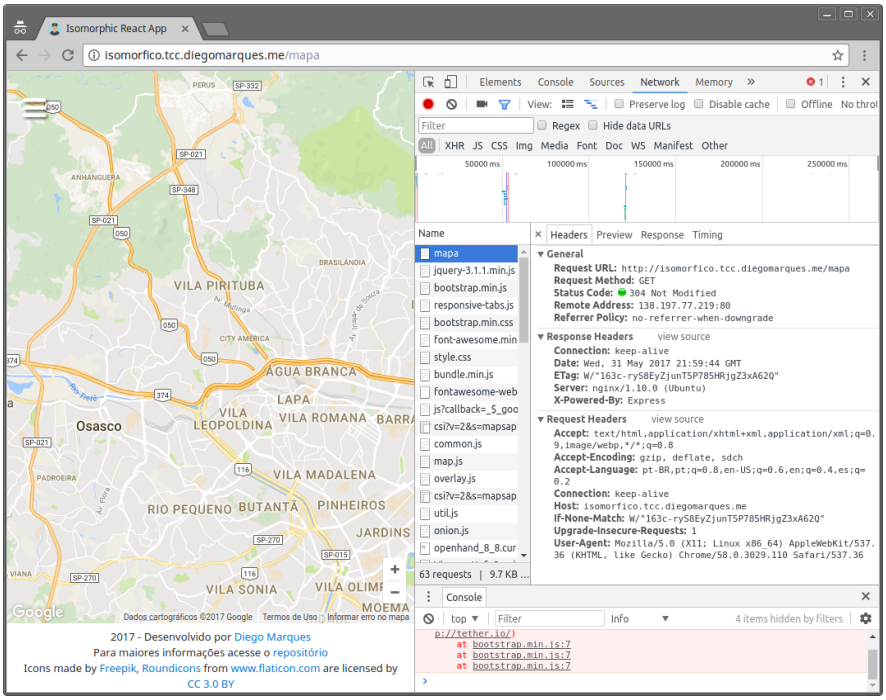

Figura 16 - Acesso direto na aplicação isomórfica

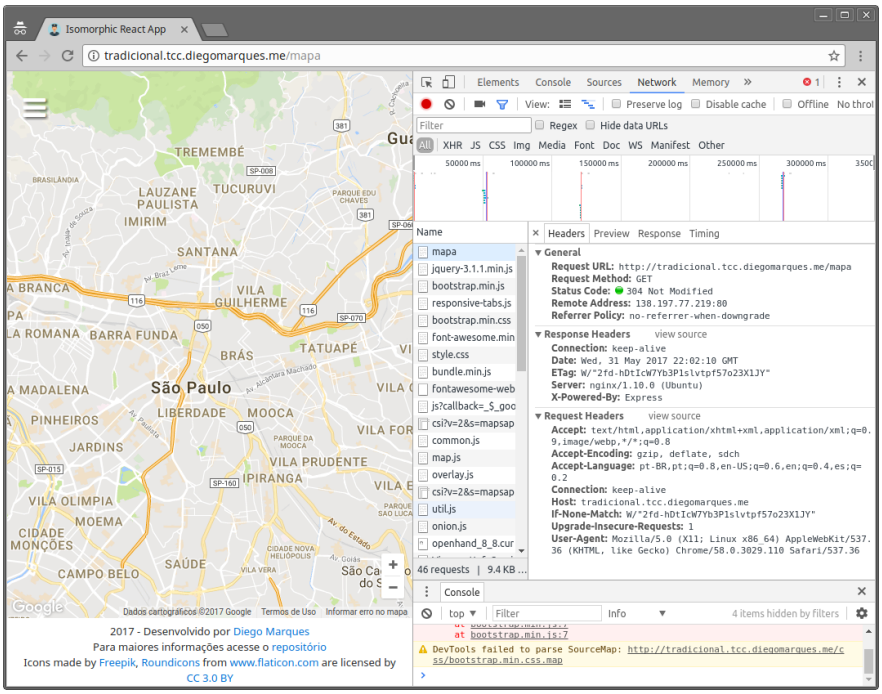

Figura 17 - Acesso direto na aplicação tradicional

5.3 Tempo de carregamento

Nesta seção é discutida a questão dos tempos de carregamento de ambas aplicações. Antes de serem apresentados os detalhes dos testes e suas conclusões, é necessário definir o escopo do mesmo, pois a métrica de tempo de carregamento é muito abrangente e sem um escopo definido, torna-se difícil chegar a qualquer conclusão.

O tempo de carregamento considerado neste teste se trata do tempo entre o cliente enviar a requisição e a página estar carregada, ou seja, sua resposta processada e o DOM totalmente gerado, além de diversos tempos parciais como o tempo para o carregamento do DOM, o tempo para obter apenas a resposta, entre outros que poderão ser visto na Tabela 11.

Para este teste, ao contrário dos testes anteriores, precisamos de diversas requisições para chegar a um tempo médio e o desvio padrão entre as amostragens. Para efetuar um grande número de requisições, foi utilizada a ferramenta automatizada Phantomas, que como visto em PHANTOMAS (2017), tem a capacidade de fornecer muitos dos dados necessários para o escopo deste teste. O resultado da execução de

100 requisições em ambas as aplicações pode ser conferido na Tabela 11, e o equipamento utilizado pode ser conferido na Tabela 9, com a mudança da ferramenta para a ferramenta citada anteriormente.

O número 100 foi escolhido para as requisições devido à observação do desvio padrão ser consideravelmente baixo na maioria das métricas, assim não havendo a necessidade de uma amostragem maior. Antes de se apresentar as conclusões, é importante definir o que cada métrica da Tabela 11 representa. A Tabela 10 mostra o significado de cada métrica, que foram extraídos com base em PHANTOMAS (2017).

Requisições	Número total de requisições HTTP feitas
Tamanho do Conteúdo (bytes)	Tamanho do conteúdo de todas as respostas (baseado no cabeçalho Content-Length)
Tráfico HTTP Completo (ms)	Tempo que levou para receber o último byte da última requisição HTTP
Tempo para o Primeiro Byte (ms)	Tempo que levou para receber o primeiro byte da primeira resposta que não era um redirecionamento
Tempo para o Último Byte (ms)	Tempo que levou para receber o último byte da primeira resposta que não era um redirecionamento
Total de elementos no DOM	Número total de nodos de elemento HTML
Tempo para DOM Interativo (ms)	Tempo que levou para interpretar o HTML e construir o DOM
Tempo para DOM Completo (ms)	Tempo que levou para carregar todos os recursos da página
Tempo no Backend (%)	Tempo até o primeiro byte comparado ao tempo de carregamento total

Tempo no Frontend (%)	Tempo até o evento onLoad do objeto window comparado ao tempo de carregamento total
Resposta mais lenta (ms)	Tempo para o último byte da resposta mais lenta (levando em consideração as requisições de cada acesso)
Resposta mais rápida (ms)	Tempo para o último byte da resposta mais rápida (levando em consideração as requisições de cada acesso)
Maior latência (ms)	Tempo para o primeiro byte da resposta mais lenta (levando em consideração as requisições de cada acesso)
Menor latência (ms)	Tempo para o primeiro byte da resposta mais rápida (levando em consideração as requisições de cada acesso)

Tabela 10 - Descrição das métricas do teste de tempo de carregamento

	Isomórfica		Tradicional	
	Média	Desv. Padr.	Média	Desv. Padr.
Tráfico HTTP Completo (ms)	1645.09	36.76	1539.02	48.59
Tempo para o Primeiro Byte (ms)	330.32	8.73	325.56	36.89
Tempo para o Último Byte (ms)	353.3	9.61	348.13	37.23
Total de elementos no DOM	76	0	2	0
Tempo para DOM Interativo (ms)	1314.8	30.52	1314.59	27.86
Tempo para DOM Completo (ms)	1322.99	27.08	1316.47	27.9
Tempo no Backend (%)	20.02	0.71	19.8	1.45
Tempo no Frontend (%)	79.98	0.71	80.2	1.45
Resposta mais lenta (ms)	1195.87	26.7	1194.37	26.34
Resposta mais rápida (ms)	303.23	7.19	305.57	7.86
Maior latência (ms)	475.99	28.17	478.93	31.77
Menor latência (ms)	156.57	5.35	161.18	6.25

Tabela 11 - Resultados dos testes de Tempo de Carregamento

Com base nas métricas apresentadas e em seus resultados, devemos levar em consideração ainda que o teste ocorreu em apenas um ambiente de rede, que neste

caso foi o ambiente de rede onde se encontrava o equipamento descrito na Tabela 9, para que diferenças relacionadas a latência de conexão entre diferentes ambientes pudessem ser desprezadas nesse escopo.

Após todos os pontos levantados nesta seção, podemos chegar à conclusão de que o impacto da abordagem isomórfica sobre a abordagem tradicional no quesito de tempo de carregamento pode ser considerado praticamente irrelevante, pois a diferença na média entre ambos é muito baixa, algo na grandeza de poucos bytes ou poucos milissegundos, porém como era de se esperar o tamanho do DOM da resposta das duas aplicações demonstra uma diferença expressiva, visto que o mesmo já é recebido pelo cliente completamente carregado na abordagem isomórfica. Os dados individuais de cada requisição, bem como suas estatísticas podem ser conferidos em *https://github.com/dmarquesdev/isomorphic-javascript/raw/master/docs/isomorphic_stats.json* para a versão isomórfica e *https://github.com/dmarquesdev/isomorphic-javascript/raw/master/docs/standard_stats.json* para a versão tradicional.

Vale ressaltar que apesar das médias das métricas entre as duas aplicações serem parecidas, como dito anteriormente, o desvio padrão da maioria das métricas na aplicação tradicional se mostra bem elevado em comparação ao desvio padrão da aplicação isomórfica, o que pode nos levar a crer que a aplicação tradicional se comporta de maneira mais instável em relação às métricas apresentadas do que a aplicação isomórfica.

5.4 Performance do Cliente

Medir a performance do cliente talvez seja a métrica mais difícil de ser avaliada pela falta de ferramentas disponíveis para tal fim utilizando JavaScript. A melhor ferramenta disponível é a ferramenta embutida no navegador Google Chrome, que exibe diversas informações sobre a performance no carregamento de uma página. Por conta deste problema, não foi possível coletar um grande número de amostras por se tratar de um processo manual envolvendo a leitura dos dados por olho humano.

Levando em consideração uma única amostra de cada aplicação, porém, alguns pontos-chave podem ser observados. Dentre estes pontos, podemos observar através

das Figuras 18 e 19 que a aplicação isomórfica tende a consumir menos recursos computacionais do que a aplicação tradicional. Além disso, podemos observar nas Figuras 20 e 21, que representa a árvore de chamadas de função da página, que enquanto a aplicação isomórfica passa mais tempo interpretando o HTML, a aplicação tradicional passa mais tempo executando código JavaScript.

Figura 18 - Consumo de recursos da aplicação isomórfica em uma requisição

Figura 19 - Consumo de recursos da aplicação tradicional em uma requisição

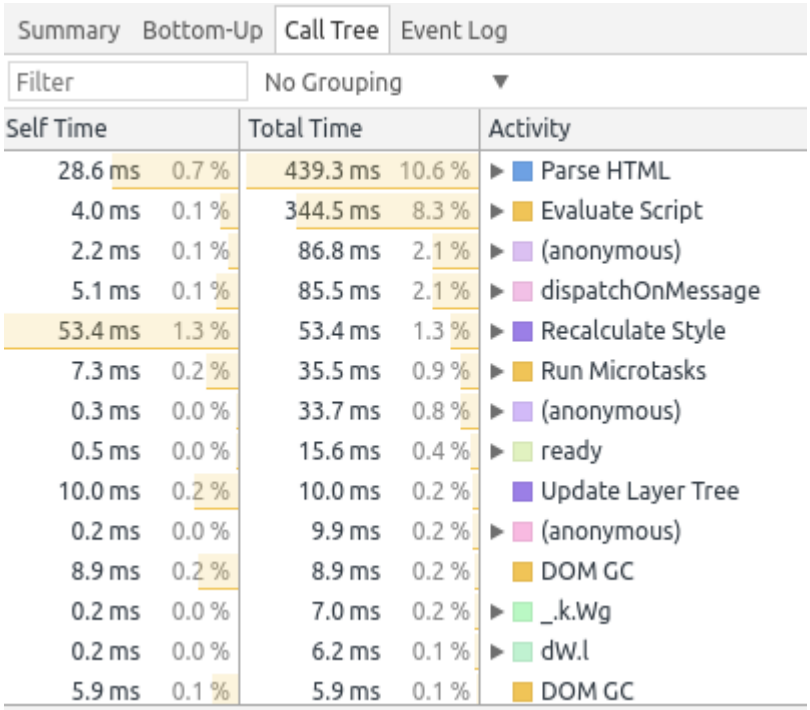

Figura 20 - Árvore de chamadas da aplicação isomórfica

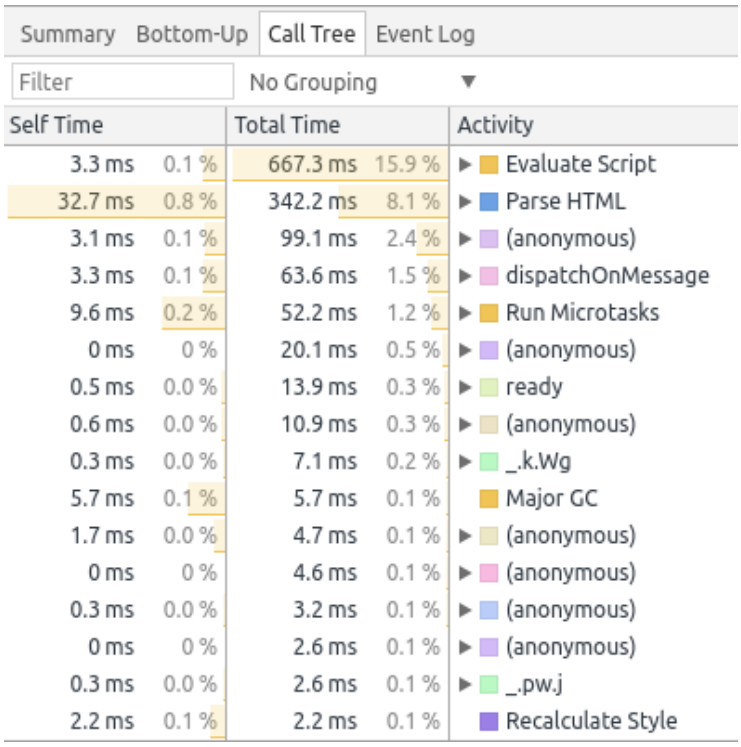

Figura 21 - Árvore de chamadas da aplicação tradicional

Tais comportamentos eram esperados levando em consideração os conceitos aqui apresentados, porém não se pode chegar a nenhuma conclusão concreta pelo fato do espaço amostral que contém apenas uma amostra ser praticamente irrelevante. Com base nisso, foram coletadas mais 10 amostras em cada aplicação e em todos os casos, apesar dos tempos e quantidades variarem entre si, o mesmo comportamento pôde ser observado, o que nos leva a concluir que a diferença de performance entre as duas abordagens existe, porém em uma aplicação do porte da aplicação proposta ou menor, provavelmente não haverá um grande impacto na experiência do usuário. Os dados das amostras adicionais podem ser conferidos em *https://github.com/dmarquesdev/isomorphic-javascript/blob/master/docs/performance*

5.5 Considerações Finais

Após todos os testes descritos neste capítulo podemos observar que as diferenças propostas pelo conceito de uma aplicação isomórfica existem, principalmente quando se trata de características do servidor, enquanto que as

características ligadas ao cliente, como performance e tempo de carregamento, podem ser observadas em uma proporção que pode ser considerada irrelevante para uma aplicação com o porte da aplicação proposta neste trabalho.

6. CONCLUSÕES

Neste trabalho foram apresentados os diversos conceitos ligados à abordagem de desenvolvimento isomórfica utilizando JavaScript, como seus benefícios e características, assim como a definição de uma abordagem análoga à abordagem isomórfica, definida como abordagem tradicional, para fins de comparação. Com os conceitos em mãos, foi possível encontrar um conjunto de tecnologias e ferramentas capaz de botar em prática tudo que havia sido discutido, através da implementação de uma aplicação utilizando tais tecnologias.

Durante o processo de desenvolvimento, foram encontrados alguns problemas relacionados às tecnologias escolhidas, por conflitos com a abordagem isomórfica, que necessitaram de mudanças durante o processo, o que mostra que apesar de todos os seus benefícios, a abordagem isomórfica traz consigo alguns problemas.

Após o desenvolvimento e publicação da aplicação, da forma isomórfica e da forma tradicional, foram executados diversos testes em cima de ambas. Através dos resultados obtidos é possível concluir que diferenças existem, porém em alguns casos em uma proporção muito baixa que pode tornar sua relevância questionável.

Mesmo obtendo resultados inesperados com base nas premissas, a abordagem de desenvolvimento utilizando JavaScript Isomórfico continua sendo muito promissora, por trazer melhorias e correções a um modelo que contém algumas deficiências. Apesar das dificuldades enfrentadas ao se incluir novos conceitos dentro da rotina e do modelo de desenvolvimento de aplicações para Web nos dias atuais, no geral os benefícios que se obtém em troca são muito proveitosos. Mesmo que em alguns aspectos tais benefícios não sejam tão expressivos, eles ainda existem.

Além dos benefícios ligados ao funcionamento da aplicação, o processo de desenvolvimento utilizando somente uma linguagem de programação para desenvolver toda a aplicação traz benefícios tanto para desenvolvedores experientes, que no caso podem focar nas peculiaridades de apenas uma sintaxe de linguagem, como para desenvolvedores menos experientes, que ganham a capacidade de desenvolver uma aplicação completa dominando apenas um tipo de linguagem. Portanto, a abordagem isomórfica pode ser considerada uma ótima opção para se desenvolver uma aplicação para a Web.

6.1 Trabalhos Futuros

Como trabalhos futuros, de forma a incrementar, dar continuidade ou suprir uma deficiência observada na pesquisa aqui iniciada, são propostos os seguintes temas:

- Avaliação de SEO em aplicações que utilizam JavaScript Isomórfico.
- Avaliação do impacto no tempo de desenvolvimento em aplicações que utilizam JavaScript Isomórfico.
- Testes e métricas de usabilidade em aplicações desenvolvidas utilizando JavaScript Isomórfico.
- Avaliação de aplicações que utilizam JavaScript Isomórfico em dispositivos móveis ou de baixo poder computacional.

APÊNDICE A - LISTA DE DEPENDÊNCIAS DO PROJETO

A seguinte lista de dependências foi retirada do arquivo *package.json,* o arquivo de configurações do NPM, e contém todas as dependências utilizadas no projeto, junto com suas respectivas versões e uma breve descrição.

```
"axios": "^0.16.0", # Executa requisições HTTP e trata através de Promises
"babel-cli": "^6.24.0", # Utilitário do Babel para linha de comando
"babel-core": "^6.24.0", # Core da ferramenta Babel
"babel-loader": "^6.4.1", # Loader da ferramenta Babel para Webpack
"babel-preset-es2015": "^6.24.0", # Preset do Babel para compilar ES6
"babel-preset-react": "^6.23.0", # Preset do Babel para compilar JSX
"babel-preset-stage-0": "^6.22.0", # Preset do Babel para melhorias na sintaxe do ES6
"babel-register": "^6.24.0", # Auxilia o desenvolvimento para Node utilizando ES6
"ejs": "^2.5.6", # Template engine para Express
"eslint-config-rallycoding": "^3.2.0", # Configuração do ESlint fornecida por https://www.rallycoding.com/
"eslint_d": "^4.2.5", # Versão otimizada do ESlint, um linter para JavaScript
"express": "^4.15.2", # Framework para criação de servidor web com NodeJS
"google-map-react": "^0.23.0", # Implementação do Google Maps para React
"lodash": "^4.17.4", # Biblioteca para manipulação de dados em JavaScript
"nodemon": "^1.11.0", # Utilitário para reiniciar o NodeJS ao detectar mudanças.
"react": "^15.4.2", # Biblioteca para criação de interfaces com o usuário
"react-dom": "^15.4.2", # Versão da biblioteca React para o DOM
"react-hot-loader": "^3.0.0-beta.6", # Loader que recarrega componentes de React no Webpack Dev Server
"react-redux": "^5.0.3", # Conector entre as bibliotecas React e Redux
"react-router": "^3.0.2", # Biblioteca para tratamento de rotas utilizando React
"redux": "^3.6.0", # Implementação da arquitetura Flux para tratar estado da aplicação
"redux-form": "^6.6.3", # Utilitário para manipular formulários utilizando Redux
```

```
    "redux-thunk": "^2.2.0", # Utilitário para lidar com requisições assíncronas utilizando Redux
    "webpack": "^2.3.2", # Empacotador de dependências que gera pacotes de código prontos para utilização
    "webpack-dev-server": "^2.4.2" # Servidor capaz de atualizar a página do browser quando uma alteração nos arquivos é detectada
```

APÊNDICE B - RELATÓRIO DE TEMPO DE DESENVOLVIMENTO DO PROJETO

Neste apêndice serão apresentados os resultados do tempo de desenvolvimento gasto durante o projeto, com uma breve descrição de cada tarefa. Foi utilizada uma ferramenta chamada Toggl, que como vista em TOGGL (2017), tem a capacidade de marcar a quantidade de tempo gasto durante o desenvolvimento de diversas tarefas de forma totalmente flexível. Segundo a ferramenta, foram gastas 115 horas e 35 minutos durante a implementação, e tal resultado parcial pode ser conferido visualmente através da Figura 22 ou de forma completa através da URL *https://github.com/dmarquesdev/isomorphic-javascript/raw/master/docs/development-time-report.pdf*

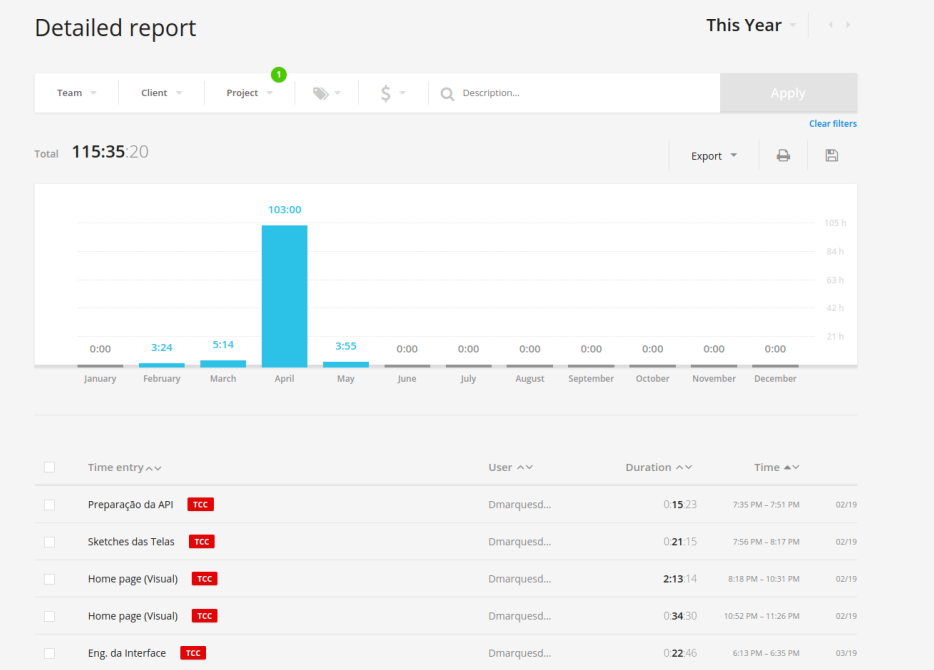

Figura 22 - Relatório de tempo de desenvolvimento (Parcial)

APÊNDICE C - EXECUTANDO O PROJETO E SUA API

Todo código-fonte da aplicação está disponível no repositório do GitHub que pode ser acessado através da URL *https://github.com/dmarquesdev/isomorphic-javascript*, e sua API através do repositório *https://github.com/dmarquesdev/criminal-report-api*. A versão da API disponível neste repositório se trata de uma versão modificada da versão proposta por OLIVEIRA *et al.* (2016) para atender os requisitos da aplicação desenvolvida neste trabalho. Neste apêndice não será abordado o funcionamento da ferramenta de versionamento git ou o próprio GitHub, e sim como executar a aplicação e sua API.

Neste guia de exemplo, consideraremos como Sistema Operacional utilizado o sistema Linux através da distribuição Ubuntu. O processo completo para a instalação e execução da aplicação é o seguinte:

1) Instale a ferramenta *git* e as tecnologias *node* e *npm* através do *NVM* (Node Version Manager). Caso já os tenha instalado, pula para a próxima etapa.

```
sudo apt-get update

sudo apt-get install build-essential libssl-dev git

curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.33.2/install.sh |
bash

source ~/.bashrc

nvm install node
```

2) Clone ambos repositórios para a máquina

```
git clone https://github.com/dmarquesdev/isomorphic-javascript
git clone https://github.com/dmarquesdev/criminal-report-api
```

3) Instale as dependências necessárias para o projeto *isomorphic-javascript*

```
cd isomorphic-javascript
npm install
cd ..
```

4) Verifique a instalação do Java

```
java -version
```

Este comando deve gerar uma saída semelhante a essa:

```
openjdk version "1.8.0_131"
OpenJDK Runtime Environment (build 1.8.0_131-8u131-b11-0ubuntu1.16.10.2-b11)
OpenJDK 64-Bit Server VM (build 25.131-b11, mixed mode)
```

Caso gere alguma saída informando algo relacionado a ausência do Java, o mesmo pode ser instalado através do seguinte comando:

```
sudo apt-get install openjdk-8-jdk
```

5) Conceda permissão de execução ao *Shell Script* responsável por executar a API

```
cd criminal-report-api
chmod +x start.sh
```

6) Extraia os arquivos de dados da API

```
unzip data.zip
```

7) Execute a API

```
./start.sh
```

8) Gere uma chave de API para Google Maps utilizando sua conta Google. Em outro terminal, acesse a pasta do projeto *isomorphic-javascript* e crie o arquivo de configuração

```
cd isomorphic-javascript/src/shared
mv config.sample.js config.js
```

Abre o arquivo config.js com um editor de texto de sua preferência e insira sua chave de API no campo especificado.

9) Compile a aplicação

```
npm run build
```

10) Execute o servidor da aplicação

```
npm run start
```

11) Acesse a aplicação no navegador através do endereço *http://localhost:3000*

Para acessos posteriores, apenas as etapas 7, 10 e 11 são necessárias.

APÊNDICE D - CÓDIGO-FONTE DA APLICAÇÃO DE EXEMPLO

Neste apêndice serão listado todos os código-fontes criados para a aplicação.

package.json

```
{
  "name": "isomorphic-javascript",
  "version": "0.1.0",
  "description": "Isomorphic Javascript Example",
  "main": "index.js",
  "directories": {
    "doc": "docs"
  },
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "build": "webpack --config webpack.prod.config.js",
    "dev-start": "nodemon --watch src/server --watch src/shared --exec babel-node
src/server/server",
    "start": "NODE_ENV=production babel-node src/server/server",
    "dev-server": "webpack-dev-server --hot --inline --port 3001"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/dmarquesdev/isomorphic-javascript.git"
  },
  "author": "Diego Marques",
  "license": "MIT",
  "bugs": {
    "url": "https://github.com/dmarquesdev/isomorphic-javascript/issues"
  },
  "homepage": "https://github.com/dmarquesdev/isomorphic-javascript#readme",
  "devDependencies": {
    "axios": "^0.16.0",
    "babel-cli": "^6.24.0",
    "babel-core": "^6.24.0",
    "babel-loader": "^6.4.1",
    "babel-preset-es2015": "^6.24.0",
    "babel-preset-react": "^6.23.0",
    "babel-preset-stage-0": "^6.22.0",
    "babel-register": "^6.24.0",
    "ejs": "^2.5.6",
    "eslint-config-rallycoding": "^3.2.0",
```

```
    "eslint_d": "^4.2.5",
    "express": "^4.15.2",
    "google-map-react": "^0.23.0",
    "lodash": "^4.17.4",
    "nodemon": "^1.11.0",
    "react": "^15.4.2",
    "react-dom": "^15.4.2",
    "react-hot-loader": "^3.0.0-beta.6",
    "react-redux": "^5.0.3",
    "react-router": "^3.0.2",
    "redux": "^3.6.0",
    "redux-form": "^6.6.3",
    "redux-thunk": "^2.2.0",
    "webpack": "^2.3.2",
    "webpack-dev-server": "^2.4.2"
  }
}
```

webpack.config.js

```
const webpack = require('webpack');
const path = require('path');

module.exports = {
  devServer: {
    contentBase: path.join(__dirname, '/public'),
    headers: { 'Access-Control-Allow-Origin': '*' }
  },
  entry: [
    'webpack-dev-server/client?http://localhost:3001',
    'webpack/hot/only-dev-server',
    path.join(__dirname, '/src/client/index.js')
  ],
  output: {
    path: path.join(__dirname, '/public/js'),
    filename: 'bundle.js',
    publicPath: 'http://localhost:3001/js/',
  },
  module: {
    rules: [
      {
        test: /\.jsx?$/,
        loaders: ['react-hot-loader/webpack'],
        exclude: /node_modules/
```

```
    },
    {
      loader: 'babel-loader',
      test: /\.jsx?$/,
      exclude: /node_modules/,
      query: {
        presets: ['stage-0', 'react', 'es2015']
      }
    }
  ]
 },

 plugins: [
  new webpack.DefinePlugin({
    'process.env': {
      NODE_ENV: JSON.stringify('development')
    }
  })
 ]
};
```

webpack.prod.config.js

```
const webpack = require('webpack');
const path = require('path');

module.exports = {
        entry: path.join(__dirname, '/src/client/index.js'),
        output: {
                path: path.join(__dirname, '/public/js'),
                filename: 'bundle.min.js',
                publicPath: '/'
        },

        devtool: 'source-map',

        module: {
                rules: [
                        {
                                test: /\.jsx?/,
                                exclude: /node_modules/,
                                loader: 'babel-loader',
                                query: {
                                        presets: ['es2015',
```

```
'react', 'stage-0']
                                                                        }
                                                                }
                                                        ]
                                        },

                                        plugins: [
                                                        new webpack.DefinePlugin({
                                                                'process.env': {
                                                                        NODE_ENV:
JSON.stringify('production')
                                                                }
                                                        }),
                                                        new webpack.optimize.UglifyJsPlugin({ minimize: true })
                                        ]
};
```

public/css/style.css

```
/* General */
.center-content {
  display: flex;
  justify-content: center;
  align-content: center;
  align-items: center;
}

/* App */
.wrapper {
  min-height: 100vh;
  display: flex;
  justify-content: space-between;
  align-items: stretch;
  flex-direction: column;
  max-height: 100vh;
  max-width: 100vw;
}

/* Footer */
.footer {
  align-self: flex-end;
}

.footer-text {
```

```
  display: block;
  text-align: center;
  font-size: 0.8rem;
}

/* Home */
.home-container {
  flex-grow: 3;
  display: flex;
  flex-direction: column;
  justify-content: center;
}

.home-container .jumbotron {
  text-align: center;
}

.home-container .jumbotron .container > * {
  padding: 1rem;
}

@media (max-width: 576px) {
  .home-container .description .icon {
    padding-bottom: 1.5rem;
  }
}

/* Map */
.map-container {
  width: 100%;
  height: 89vh;
  display: flex;
  align-items: stretch;
  position: relative;
  overflow: hidden;
}

.map {
  flex-grow: 8;
}

.marker-icon {
  color: transparent;
  font-size: 3.5em;
  cursor: pointer;
  background-image: #a90329;
```

```
  background-image: -moz-linear-gradient(top, #ff3019 0%, #cf0404 100%);
  background-image: -webkit-linear-gradient(top, #ff3019 0%, #cf0404 100%);
  background-image: linear-gradient(to bottom, #ff3019 0%, #cf0404 100%);
  filter: progid:DXImageTransform.Microsoft.gradient( startColorstr='#ff3019',
endColorstr='#cf0404',GradientType=0 );
  color: transparent;
  -webkit-background-clip: text;
  background-clip: text;
  transition: all .2s ease-in-out;
}

.marker-icon.__red {
  background-image: #a90329;
  background-image: -moz-linear-gradient(top, #ff3019 0%, #cf0404 100%);
  background-image: -webkit-linear-gradient(top, #ff3019 0%, #cf0404 100%);
  background-image: linear-gradient(to bottom, #ff3019 0%, #cf0404 100%);
  filter: progid:DXImageTransform.Microsoft.gradient( startColorstr='#ff3019',
endColorstr='#cf0404',GradientType=0 );
}

.marker-icon.__yellow {
  background-image: #d7e516;
  background-image: -moz-linear-gradient(top, #d7e516 0%, #e2cc04 100%);
  background-image: -webkit-linear-gradient(top, #d7e516 0%, #e2cc04 100%);
  background-image: linear-gradient(to bottom, #d7e516 0%, #e2cc04 100%);
  filter: progid:DXImageTransform.Microsoft.gradient( startColorstr='#d7e516',
endColorstr='#e2cc04',GradientType=0 );
}

.marker-icon.__green {
  background-image: #18f218;
  background-image: -moz-linear-gradient(top, #18f218 0%, #1f7702 100%);
  background-image: -webkit-linear-gradient(top, #18f218 0%, #1f7702 100%);
  background-image: linear-gradient(to bottom, #18f218 0%, #1f7702 100%);
  filter: progid:DXImageTransform.Microsoft.gradient( startColorstr='#18f218',
endColorstr='#1f7702',GradientType=0 );
}

.marker-icon.__blue {
  background-image: #19c9ff;
  background-image: -moz-linear-gradient(top, #19c9ff 0%, #048eea 100%);
  background-image: -webkit-linear-gradient(top, #19c9ff 0%, #048eea 100%);
  background-image: linear-gradient(to bottom, #19c9ff 0%, #048eea 100%);
  filter: progid:DXImageTransform.Microsoft.gradient( startColorstr='#19c9ff',
endColorstr='#048eea',GradientType=0 );
}
```

```
.marker-icon.__orange {
  background-image: #ffae19;
  background-image: -moz-linear-gradient(top, #ffae19 0%, #ea7f04 100%);
  background-image: -webkit-linear-gradient(top, #ffae19 0%, #ea7f04 100%);
  background-image: linear-gradient(to bottom, #ffae19 0%, #ea7f04 100%);
  filter: progid:DXImageTransform.Microsoft.gradient( startColorstr='#ffae19',
endColorstr='#ea7f04',GradientType=0 );
}

.marker-icon.__pink {
  background-image: #ff4fa7;
  background-image: -moz-linear-gradient(top, #ff4fa7 0%, #e0048f 100%);
  background-image: -webkit-linear-gradient(top, #ff4fa7 0%, #e0048f 100%);
  background-image: linear-gradient(to bottom, #ff4fa7 0%, #e0048f 100%);
  filter: progid:DXImageTransform.Microsoft.gradient( startColorstr='#ff4fa7',
endColorstr='#e0048f',GradientType=0 );
}

.marker-icon.__purple {
  background-image: #c24efc;
  background-image: -moz-linear-gradient(top, #c24efc 0%, #9004d6 100%);
  background-image: -webkit-linear-gradient(top, #c24efc 0%, #9004d6 100%);
  background-image: linear-gradient(to bottom, #c24efc 0%, #9004d6 100%);
  filter: progid:DXImageTransform.Microsoft.gradient( startColorstr='#c24efc',
endColorstr='#9004d6',GradientType=0 );
}

.marker-icon.__brown {
  background-image: #914600;
  background-image: -moz-linear-gradient(top, #914600 0%, #633201 100%);
  background-image: -webkit-linear-gradient(top, #914600 0%, #633201 100%);
  background-image: linear-gradient(to bottom, #914600 0%, #633201 100%);
  filter: progid:DXImageTransform.Microsoft.gradient( startColorstr='#914600',
endColorstr='#633201',GradientType=0 );
}

.marker-icon:hover {
  transform: scale(1.3);
}

/* Search Bar */
.map-search {
  position: absolute;
  top: 3%;
  width: 60%;
```

```
  display: flex;
  justify-content: space-around;
}

@media(min-width: 1200px) {
  .map-search {
    width: 30%;
  }
}

.search-bar-type {
  flex-grow: 1 !important;
}

.search-bar-value {
  flex-grow: 3 !important;
}

.map-search-bar {
  flex-grow: 3;
}

.menu-btn {
  flex-grow: 1;
  color: #fff !important;
  text-align: center;
  font-size: 1.5em;
  text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.6);
  cursor: pointer;
}

@media(max-width: 768px) {
  .menu-btn {
    padding-left: 4vw;
    text-align: left;
    font-size: 2em;
  }
}

.menu-btn:hover, .menu-btn:active, .menu-btn:visited {
  color: #fff !important;
}

/* Loading */
.overlay {
  position: fixed;
```

```
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background-color: rgba(0, 0, 0, 0.4);
  z-index: 9999;
  display: flex;
  justify-content: center;
  align-items: center;
}

.loading-icon {
  font-size: 6em;
  color: #f0ad4e;
}


/* Crime Preview */
.preview-tip {
  width: 300px;
  height: auto;
  display: flex;
  flex-direction: row;
  justify-content: space-around;
  align-items: center;
  background-color: white;
  position: relative;
  top: -155px;
  left: -75px;
  padding: 10px;
  box-shadow: 4px 3px 5px #888888;
  -moz-box-shadow: 4px 3px 5px #888888;
  -webkit-box-shadow: 4px 3px 5px #888888;
  z-index: 9999;
}

.preview-tip:after {
  content: "";
  position: absolute;
  bottom: -10px;
  left: 76px;
  display: block;
  width: 0;
  border-width: 10px 10px 0;
  border-style: solid;
  border-color: #FFF transparent;
```

```
  z-index: 9999;
}

.preview-tip-icon {
  padding-right: 10px;
}

.preview-tip-text {
  width: 202px;
  font-size: 1em;
  color: black;
  text-overflow: ellipsis;
  overflow: hidden;
  white-space: nowrap;
  min-width: 200px;
}

.preview-tip-text ul {
  padding-left: 0;
  list-style: none;
  margin: 0 auto;
}

.preview-tip-text ul li {
  padding: 2px 0;
}

.preview-tip-text .title {
  font-size: 1.5em;
  margin-bottom: 5px;
}

/* SideBar */
.sidebar {
  flex-grow: 1;
  overflow-y: auto;
  z-index: 9999;
  background-color: #fff;
}

@media(min-width: 1200px) {
  .sidebar {
    width: 20vw;
  }
}
```

```
@media(max-width: 1200px) {
  .sidebar {
    width: 40vw;
  }
}

@media(max-width: 768px) {
  .sidebar {
    width: 60vw;
  }
}

@media(max-width: 550px) {
  .sidebar {
    width: 80vw;
  }
}

@media(max-width: 550px) {
  .sidebar {
    width: 95vw;
  }
}

.sidebar-header {
  padding: 10px;
}

.sidebar-title {
  display: inline;
}

.sidebar-close {
  float: right;
  color: #ccc !important;
  cursor: pointer;
}

/* Dialog */
.dialog {
  background-color: #fff;
  padding: 10px 25px;
  max-height: 95vh;
  overflow-y: auto;
}
```

```
.dialog-header {
  padding: 10px 0px;
}

.dialog-title {
  display: inline;
}

.dialog-close {
  float: right;
  color: #ccc !important;
  cursor: pointer;
}

/* Card */
.mycard {
  display: flex;
  flex-direction: column;
  padding: 10px;
  border-top: 1px solid #ccc;
  border-left: 1px solid #ccc;
  margin: 2% auto;
  box-shadow: 3px 2px 1px 0px #ccc;
}

.mycard .header {
  padding: 5px;
  border-bottom: 1px solid #ccc;
  margin-bottom: 10px;
}

/* Crime Detail */
.crime-detail {
  min-width: 50vw;
}

.crime-detail-resume {
  flex-direction: row;
}

.crime-detail-resume .resume {
  padding-left: 25px;
}

.crime-detail-map {
  height: 30vh;
```

```
}

@media(max-width: 768px) {
  .crime-detail-map {
    height: 45vh;
  }
}

.crime-detail-tabs .nav {
  margin-bottom: 10px;
}

.crime-detail-tabs .tab-content .row {
  padding: 5px 0px;
}

.crime-detail-tabs .tab-content .tab-header a{
  display: block;
  border: 1px solid #ccc;
  padding: 10px;
  font-size: 1.2em;
}

.crime-detail-tabs .tab-content .tab-header a:hover{
  text-decoration: none;
}

/* Map Caption */
.map-caption {
  position: absolute;
  bottom: 9vh;
  display: flex;
  flex-direction: row;
}

.__hidden, .__visible {
  transition: all 0.5s ease;
}

.map-caption.__hidden {
  right: -319px;
}

.map-caption.__visible {
  right: 5px;
}
```

```
.map-caption ._list {
  list-style: none;
  background-color: #fff;
  padding: 20px;
  box-shadow: 4px 4px 6px 0px #bbafaf;
}

.map-caption ._list .marker-icon {
  font-size: 2em;
  padding-right: 10px;
}

.map-caption ._button {
  height: 40px;
  cursor: pointer;
}

/* Report */
.report {
  border: 1px solid #ccc;
  padding: 0px 20px;
  margin: 20px auto;
}

#report-container {
  padding-top: 20px;
}

/* Crime List */
.crime-list {
  padding: 20px;
}

.crime-list .crime-card {
  display: flex;
  flex-direction: row;
  justify-content: center;
  align-items: center;
  cursor: pointer;
}

.crime-list .crime-card img {
  margin: 10px;
}
```

```
.crime-card-info {
  list-style: none;
  margin: 0px auto;
  padding-left: 0px;
  padding: 10px;
}

@media(max-width: 550px) {
  .crime-list .crime-card {
    flex-direction: column;
  }
}

/* Message */
.message-bar {
  display: block;
  width: 100%;
  color: #fff;
  background-color: #cc2b2b;
  text-align: center;
  padding: 10px;
}
```

src/client/constants.js

```
// API Constants values
export const Categories = {
  lesao_corporal_seguida_de_morte: "Lesão Corporal Seguida de Morte",
  latrocinio: "Latrocínio",
  morte_acidental: "Morte Acidental",
  cadaver_sem_lesoes: "Cadáver Sem Lesões",
  suicidio_duvidoso: "Suicídio Duvidoso",
  homicidio_doloso: "Homicídio Doloso",
  oposicao_policial: "Oposição Policial",
  morte_subita_natural: "Morte Súbita Natural"
};

export const CategoryIcon = {
  lesao_corporal_seguida_de_morte: 'sword',
  latrocinio: 'burglar',
  morte_acidental: 'man-falling-off-a-precipice',
  cadaver_sem_lesoes: 'photo',
  suicidio_duvidoso: 'rope',
  homicidio_doloso: 'blaster',
```

```
  oposicao_policial: 'police',
  morte_subita_natural: 'grave'
};

export const CategoryColor = {
  LESAO_CORPORAL_SEGUIDA_DE_MORTE: '__pink',
  LATROCINIO: '__orange',
  MORTE_ACIDENTAL: '__blue',
  CADAVER_SEM_LESOES: '__green',
  SUICIDIO_DUVIDOSO: '__brown',
  HOMICIDIO_DOLOSO: '__red',
  OPOSICAO_POLICIAL: '__purple',
  MORTE_SUBITA_NATURAL: '__yellow'
};

export const Genders = {
  masculino: 'Masculino',
  feminino: 'Feminino'
};

export const SkinColors = [
  'Vermelha',
  'Parda',
  'Preta',
  'Amarela',
  'Branca',
  'Outros'
];

export const Educations = [
  'Analfabeto',
  '1 Grau incompleto',
  '1 Grau completo',
  '2 Grau incompleto',
  '2 Grau completo',
  'Superior incompleto',
  'Superior completo'
];

// Field types
export const FIELD_TEXT = 'text';
export const FIELD_SELECT = 'select';
export const FIELD_NUMERIC = 'numeric';

// Search
export const SearchTypes = {
```

```
name: {
  label: 'Nome',
  type: FIELD_TEXT
},

idReport: {
  label: 'Cód. BO',
  type: FIELD_TEXT
},

year: {
  label: 'Ano',
  type: FIELD_NUMERIC,
  minValue: 2013,
  maxValue: 2015
},

personId: {
  label: 'RG',
  type: FIELD_TEXT
},

category: {
  label: 'Categoria',
  type: FIELD_SELECT,
  defaultValue: Categories.lesao_corporal_seguida_de_morte,
  values: Categories
},

gender: {
  label: 'Gênero',
  type: FIELD_SELECT,
  defaultValue: Genders.masculino,
  values: Genders
},

age: {
  label: 'Idade',
  type: FIELD_NUMERIC,
  minValue: 18,
  maxValue: 130
},

skin: {
  label: 'Cor da Pele',
  type: FIELD_SELECT,
```

```
    defaultValue: SkinColors[0],
    values: SkinColors
  },

  education: {
    label: 'Escolaridade',
    type: FIELD_SELECT,
    defaultValue: Educations[0],
    values: Educations
  }
};

export const ReportTypes = {
  COLLECTION: 'COLLECTION',
  SINGLE: 'SINGLE'
};
```

src/client/index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import { Router, browserHistory } from 'react-router';

import { Provider } from 'react-redux';

import routes from '../shared/routes';
import store from './store';

ReactDOM.render(
    <Provider store={store}>
      <Router
        history={browserHistory}
        routes={routes}
      />
    </Provider>
  , document.getElementById('root')); // eslint-disable-line no-undef
```

src/client/store.js

```
import ReduxThunk from 'redux-thunk';
import { createStore, applyMiddleware } from 'redux';
```

```
import reducers from './reducers';

const INITIAL_STORE = {};
const store = createStore(reducers, INITIAL_STORE,
applyMiddleware(ReduxThunk));

export default store;
```

src/client/actions/index.js

```
export * from './PointsActions';
export * from './SearchActions';
export * from './ReportActions';
```

src/client/actions/PointActions.js

```
import axios from 'axios';

import { objectToURLParameters } from '../../shared/util';
import config from '../../shared/config';

import {
  FETCH_POINTS_SUCCESS,
  FETCH_POINTS_FAILURE,
  FETCH_POINT_SUCCESS,
  FETCH_POINT_FAILURE,
  START_API_CALL,
  END_API_CALL
} from './types';

const API_URL = process.env.NODE_ENV === 'production' ?
  config.PROD_API_URL :
  config.DEV_API_URL;

export const fetchPoints = (properties) => {
  const params = objectToURLParameters(properties);

  return (dispatch) => {
    dispatch({
      type: START_API_CALL
```

```
    });
    axios
      .get(`${API_URL}/points?${params}`)
      .then((response) => {
        if (response.data.length) {
          dispatch({
            type: FETCH_POINTS_SUCCESS,
            payload: response.data
          });
        } else {
          dispatch({
            type: FETCH_POINTS_FAILURE,
            payload: 'Nenhum resultado encontrado!'
          });
        }
        dispatch({
          type: END_API_CALL
        });
      })
      .catch((error) => {
        dispatch({
          type: FETCH_POINTS_FAILURE,
          payload: 'Erro ao processar consulta!'
        });
        dispatch({
          type: END_API_CALL
        });
      });
  };
};

export const fetchPoint = (id, callback) => (dispatch) => {
  dispatch({
    type: START_API_CALL,
  });
  axios
    .get(`${API_URL}/report?idReport=${id}`)
    .then((response) => {
      dispatch({
        type: FETCH_POINT_SUCCESS,
        payload: response.data[0]
      });
      dispatch({
        type: END_API_CALL
      });
      if(callback) {
```

```
        callback();
      }
    })
    .catch((error) => {
      dispatch({
        type: FETCH_POINT_FAILURE,
        payload: 'Erro ao processar dados da ocorrência!'
      });
      dispatch({
        type: END_API_CALL
      });
    });
};
```

src/client/actions/ReportActions.js

```
import axios from 'axios';

import { objectToURLParameters } from '../../shared/util';
import config from '../../shared/config';

import {
  FETCH_REPORT_DATA_SUCCESS,
  FETCH_REPORT_DATA_FAILURE,
  START_API_CALL,
  END_API_CALL,
  CHANGE_REPORT_TYPE
} from './types';

const API_URL = process.env.NODE_ENV === 'production' ?
  config.PROD_API_URL :
  config.DEV_API_URL;

export const fetchReportData = (properties) => {
  const params = objectToURLParameters(properties);

  return (dispatch) => {
    dispatch({
      type: START_API_CALL
    });
    axios
      .get(`${API_URL}/report?${params}`)
      .then((response) => {
        dispatch({
```

```
          type: FETCH_REPORT_DATA_SUCCESS,
          payload: response.data
        });
        dispatch({
          type: END_API_CALL
        });
      })
      .catch((error) => {
        dispatch({
          type: FETCH_REPORT_DATA_FAILURE,
          payload: 'Erro ao processar pesquisa!'
        });
        dispatch({
          type: END_API_CALL
        });
      });
  }
};

export const changeReportType = (type, id = null) => {
  return (dispatch) => {
    dispatch({
      type: CHANGE_REPORT_TYPE,
      payload: { type, id }
    });
  };
};
```

src/client/actions/SearchActions.js

```
import {
  CHANGE_SEARCH_BAR_TERM,
  CHANGE_SEARCH_BAR_TYPE,
  SET_COMPLETE_SEARCH
} from './types';

export const changeSearchTerm = (term) => {
  return (dispatch) => {
    dispatch({
      type: CHANGE_SEARCH_BAR_TERM,
      payload: term
    });
  }
};
```

```
export const changeSearchType = (type) => {
  return (dispatch) => {
    dispatch({
      type: CHANGE_SEARCH_BAR_TYPE,
      payload: type
    });
  }
};

export const setSearch = (search) => {
  return (dispatch) => {
    dispatch({
      type: SET_COMPLETE_SEARCH,
      payload: search
    });
  }
};
```

src/client/actions/types.js

```
// Points
export const FETCH_POINTS_SUCCESS = 'FETCH_POINTS_SUCCESS';
export const FETCH_POINTS_FAILURE = 'FETCH_POINTS_FAILURE';
export const FETCH_POINT_SUCCESS = 'FETCH_POINT_SUCCESS';
export const FETCH_POINT_FAILURE = 'FETCH_POINT_FAILURE';

// Loading
export const START_API_CALL= 'START_API_CALL';
export const END_API_CALL = 'END_API_CALL';

// Search Bar
export const CHANGE_SEARCH_BAR_TERM = 'CHANGE_SEARCH_BAR_TERM';
export const CHANGE_SEARCH_BAR_TYPE = 'CHANGE_SEARCH_BAR_TYPE';
export const SET_COMPLETE_SEARCH = 'SET_COMPLETE_SEARCH';

// Report
export const FETCH_REPORT_DATA_FAILURE =
'FETCH_REPORT_DATA_FAILURE';
export const FETCH_REPORT_DATA_SUCCESS =
'FETCH_REPORT_DATA_SUCCESS';
export const CHANGE_REPORT_TYPE = 'CHANGE_REPORT_TYPE';
```

src/client/components/CrimeDetail.js

```
import React, { Component, PropTypes } from 'react';
import { Link } from 'react-router';

import {
  FlatIcon,
  Icon,
  Dialog,
  Card
} from './common';

import {
  GoogleMap,
  Marker
} from './map';

import {
  Categories,
  CategoryIcon
} from '../constants';

class CrimeDetail extends Component {
  componentDidUpdate() {
    responsiveTabs();
  }

  renderButtons() {
    return this.props.layout === 'dialog' && (
      <Card className="crime-detail-buttons">
        <Link to="/report" className="btn btn-success">
          <Icon name="print" /> Relatório
        </Link>
      </Card>
    );
  }

  render() {
    const { crime, layout } = this.props;

    if (!crime) {
      return false;
    }

    const marker = [];
    const lat = parseFloat(crime.lat);
```

```
const lon = parseFloat(crime.lon);
if (!isNaN(lat) && !isNaN(lon)) {
  marker.push(
    <Marker
      id={crime.idBO}
      key={crime.idBO}
      lat={lat}
      lng={lon}
      center={[lat, lon]}
      onClick={() => {}}
    />
  );
}

const categoria = crime.categoria.toLowerCase();

const hiddenClass = (layout === 'dialog') ?
  'hidden-sm-down' : 'hidden-xl-down';

const mapComponent = (layout === 'dialog') ?
  <GoogleMap markers={marker} zoom={8} /> :
  null;

const mapContainerClass = (layout === 'dialog') ?
  'crime-detail-map' : null;

return (
  <div className={`crime-detail layout-${layout}`}>
    <Card className="crime-detail-resume">
      <FlatIcon name={CategoryIcon[categoria]} width={100} height={100} />
      <div className="resume">
        <h5>{`#${crime.idBO}`}</h5>
        <h5>{crime.partesEnvolvidas[0].nome}</h5>
        <h5>{Categories[categoria]}</h5>
      </div>
    </Card>

    <Card className={mapContainerClass}>
      <div className="header">
        <h5><Icon name="map-marker" /> Local</h5>
      </div>
      <div className="info">
        <h6>{crime.local}</h6>
        <h6>{crime.tipoLocal}</h6>
      </div>
      {mapComponent}
```

```
</Card>

<Card className="crime-detail-tabs">
  <ul className={`nav nav-tabs ${hiddenClass}`} role="tablist">
    <li className="nav-item">
      <a
        className="nav-link active"
        data-toggle="tab"
        href="#infos"
        role="tab"
      >
        <Icon name="info" /> Informações
      </a>
    </li>
    <li className="nav-item">
      <a
        className="nav-link"
        data-toggle="tab"
        href="#partes-envolvidas"
        role="tab"
      >
        <Icon name="user" /> Partes Envolvidas
      </a>
    </li>
    <li className="nav-item">
      <a
        className="nav-link"
        data-toggle="tab"
        href="#naturezas"
        role="tab"
      >
        <Icon name="list" /> Naturezas
      </a>
    </li>
  </ul>

  <div id="crime-detail-infos" className="tab-content big">
    <div
      className="tab-pane active"
      id="infos"
      role="tabpanel"
    >
      <div className="container-fluid">
        <div className="row">
          <div className="col-xs-12 col-md-4">
            <span>Número BO: {crime.numero}</span>
```

```
          </div>
          <div className="col-xs-12 col-md-4">
            <span>Tipo: {crime.tipoBoletim}</span>
          </div>
          <div className="col-xs-12 col-md-4">
            <span>Flagrante: {crime.flagrante}</span>
          </div>
        </div>

        <div className="row">
          <div className="col-xs-12 col-md-6">
            <span>
              Delegacia:
              {` ${crime.idDelegacia} - ${crime.dependencia}`}
            </span>
          </div>
          <div className="col-xs-12 col-md-6">
            <span>
              Categoria: {Categories[crime.categoria.toLowerCase()]}
            </span>
          </div>
        </div>

        <div className="row">
          <div className="col-xs-12 col-md-6">
            <span>
              Data da Ocorrência: {crime.dataOcorrencia}
            </span>
          </div>
          <div className="col-xs-12 col-md-6">
            <span>
              Turno da Ocorrência: {crime.turnoOcorrencia}
            </span>
          </div>
        </div>

        <div className="row">
          <div className="col-xs-12 col-md-6">
            <span>
              Data da Comunicação: {crime.dataComunicacao}
            </span>
          </div>
          <div className="col-xs-12 col-md-6">
            <span>
              Hora da Comunicação: {crime.horaComunicacao}
            </span>
```

```
          </div>
        </div>

        <div className="row">
          <div className="col-xs-12 col-md-6">
            <span>
              Data de Elaboração: {crime.dataElaboracao}
            </span>
          </div>
          <div className="col-xs-12 col-md-6">
            <span>
              Hora de Elaboração: {crime.horaElaboracao}
            </span>
          </div>
        </div>

        <div className="row">
          <div className="col-xs-12 col-md-6">
            <span>
              Exames Requisitados: {crime.examesRequisitados}
            </span>
          </div>
          <div className="col-xs-12 col-md-6">
            <span>
              Solução: {crime.solucao}
            </span>
          </div>
        </div>

      </div>
    </div>

    <div
      className="tab-pane"
      id="partes-envolvidas"
      role="tabpanel"
    >
      {renderPartesEnvolvidas(crime.partesEnvolvidas)}
    </div>

    <div
      className="tab-pane"
      id="naturezas"
      role="tabpanel"
    >
      <div className="container-fluid">
```

```
              {renderNaturezas(crime.naturezas)}
            </div>
          </div>
        </div>
      </Card>
      {this.renderButtons()}
    </div>
  );
 }
}

const renderNaturezas = (naturezas) => {
 return naturezas.map((natureza, i) => {
  return (
   <div className="row" key={i}>
    <div className="col-xs-12 col-md-6">
     <span>Espécie: {natureza.especie}</span>
    </div>
    <div className="col-xs-12 col-md-6">
     <span>Descrição: {natureza.descricao}</span>
    </div>
   </div>
  );
 });
}

const renderPartesEnvolvidas = (partes) => {
 return partes.map((parte, i) => {
  return (
   <div key={parte.rg} className="container-fluid">
    <div className="row">
     <div className="col-xs-12 col-md-6">
      <span>Nome: {parte.nome}</span>
     </div>
     <div className="col-xs-12 col-md-6">
      <span>RG: {parte.rg}</span>
     </div>
    </div>

    <div className="row">
     <div className="col-xs-12 col-md-6">
      <span>Tipo de Envolvimento: {parte.tipoEnvolvimento}</span>
     </div>
     <div className="col-xs-12 col-md-6">
      <span>Naturalidade: {parte.naturalidade}</span>
     </div>
```

```
      </div>

      <div className="row">
        <div className="col-xs-12 col-md-6">
          <span>Nacionalidade: {parte.nacionalidade}</span>
        </div>
        <div className="col-xs-12 col-md-6">
          <span>Sexo: {parte.sexo}</span>
        </div>
      </div>

      <div className="row">
        <div className="col-xs-12 col-md-6">
          <span>Data de Nascimento: {parte.dataNascimento}</span>
        </div>
        <div className="col-xs-12 col-md-6">
          <span>Idade: {parte.idade}</span>
        </div>
      </div>

      <div className="row">
        <div className="col-xs-12 col-md-6">
          <span>Estado Civil: {parte.estadoCivil}</span>
        </div>
        <div className="col-xs-12 col-md-6">
          <span>Instrução: {parte.instrucao}</span>
        </div>
      </div>

      <div className="row">
        <div className="col-xs-12 col-md-6">
          <span>Profissão: {parte.profissao}</span>
        </div>
        <div className="col-xs-12 col-md-6">
          <span>Cutis: {parte.cutis}</span>
        </div>
      </div>

      <div className="row">
        <div className="col-xs-12 col-md-6">
          <span>Naturezas Envolvidas: {parte.naturezasEnvolvidas}</span>
        </div>
      </div>
    </div>
  );
});
```

```
}

CrimeDetail.propTypes = {
  crime: PropTypes.object,
  layout: PropTypes.string
};

CrimeDetail.defaultProps = {
  layout: 'dialog'
};

export default CrimeDetail;
```

src/client/components/CrimePreview.js

```
import React, { Component } from 'react';

import { FlatIcon } from './common';

import {
  Categories,
  CategoryIcon
} from '../constants';

const CrimePreview = ({
  date,
  address,
  category,
  person
}) => (
  <div className="preview-tip">
    <div className="preview-tip-icon">
      <FlatIcon name={CategoryIcon[category.toLowerCase()]} width={48} height={48}
/>
    </div>

    <div className="preview-tip-text">
      <ul>
        <li className="title">{Categories[category.toLowerCase()]}</li>
        <li><i className="fa fa-map"></i> {address}</li>
        <li><i className="fa fa-calendar"></i> {date}</li>
        <li><i className="fa fa-user"></i> {person}</li>
      </ul>
    </div>
```

```
  </div>
);


export default CrimePreview;
```

src/client/components/Footer.js

```
import React from 'react';

const Footer = () => (
  <div className="container-fluid footer">
    <span className="footer-text">
      2017 - Desenvolvido por <a href="mailto:dmarquesdev@gmail.com">Diego
Marques</a>
    </span>
    <span className="footer-text">
      Para maiores informações acesse o <a
href="https://github.com/dmarquesdev/isomorphic-javascript">repositório</a>
    </span>
    <span className="footer-text">
      Icons made by <a href="http://www.freepik.com" title="Freepik">Freepik</a>, <a
href="http://www.flaticon.com/authors/roundicons"
title="Roundicons">Roundicons</a> from <a href="http://www.flaticon.com"
title="Flaticon">www.flaticon.com</a> are licensed by <a
href="http://creativecommons.org/licenses/by/3.0/" rel="noopener noreferrer"
title="Creative Commons BY 3.0" target="_blank">CC 3.0 BY</a>
    </span>
  </div>
);

export default Footer;
```

src/client/components/Home.js

```
import React, { Component, PropTypes } from 'react';

import { FlatIcon } from './common';
import { SearchBar } from '../containers';

class Home extends Component {
  onSearch() {
```

```
  this.context.router.push('/mapa');
 }

 render() {
  return (
   <div className="home-container">
    <div className="jumbotron">
     <div className="container">
      <FlatIcon name="sao-paulo" width={200} height={200} />
      <div>
       <h1>São Paulo</h1>
       <h1>Criminal Records</h1>
      </div>
      <SearchBar
       onSearch={this.onSearch.bind(this)}
      />
     </div>
    </div>

    <div className="container description">
     <div className="row">
      <div className="col-xs-12 col-md-2 center-content icon">
       <FlatIcon name="police" width={120} height={120} />
      </div>

      <div className="col-xs-12 col-md-10">
       <p>
        Esta aplicação tem o intuito de fornecer uma pesquisa em uma base de
dados
        com dados relativos a Boletins de Ocorrência registrados entre 2013 e
2015
        no estado de São Paulo. Para maiores informações sobre os dados acesse
o
        artigo acadêmico relativo a ele em
        <a
href="https://figshare.com/articles/A_Platform_of_Police_Reports/3856074/5">
          A Platform of Police Reports - ICWI 2016
        </a>. Todas as informações e tecnologias aqui utilizadas estão disponíveis
        em domínio público. Para entrar em contato com o autor desta aplicação,
        acesse os links disponíveis no rodapé desta página.
       </p>
      </div>
     </div>
    </div>
   </div>
  );
```

```
  }
}

Home.contextTypes = {
  router: PropTypes.object.isRequired
};

export default Home;
```

src/client/components/index.js

```
export { default as CrimePreview } from './CrimePreview';
export { default as Footer } from './Footer';
export { default as Home } from './Home';
export { default as CrimeDetail } from './CrimeDetail';
export { default as Message } from './Message';
export * from './common';
export * from './map';
```

src/client/components/Message.js

```
import React from 'react';

const Message = ({ text }) => (
  <div className="message-bar">
    {text}
  </div>
);

export default Message;
```

src/client/components/common/Card.js

```
import React from 'react';

const Card = ({ children, title, className, onClick }) => (
  <div className={`mycard ${className}`} onClick={onClick}>
    {children}
  </div>
);
```

```
export default Card;
```

src/client/components/common/Dialog.js

```
import React, { Component } from 'react';

import Icon from './Icon';

class Dialog extends Component {
  constructor(props) {
    super(props);
    this.state = { visible: false };
  }

  toggle() {
    this.setState({ visible: !this.state.visible });
  }

  onClose() {
    if(this.props.onClose) {
      this.props.onClose();
    }

    this.setState({ visible: false });
  }

  render() {
    const {
      title,
      children
    } = this.props;

    const style = (!this.state.visible) ? { display: 'none' } : null;

    return (
      <div
        className="overlay"
        style={style}
      >
        <div className="dialog">
          <div className="dialog-header">
            <h5 className="dialog-title">{title}</h5>
            <a
```

```
            className="dialog-close"
            onClick={this.onClose.bind(this)}
          >
            <Icon name="close" />
          </a>
        </div>

        <div className="dialog-content">
          {children}
        </div>
      </div>
    </div>
  );
 }
}

export default Dialog;
```

src/client/components/common/FlatIcon.js

```
import React, { PropTypes } from 'react';

const FlatIcon = ({ name, width, height }) => (
  <img
    alt={name}
    className="flat-icon"
    src={`/assets/icons/${name}.svg`}
    style={{ width, height }}
  />
);

FlatIcon.propTypes = {
  name: PropTypes.string.isRequired,
  width: PropTypes.number,
  height: PropTypes.number,
};

FlatIcon.defaultProps = {
  width: 50,
  height: 50,
};

export default FlatIcon;
```

src/client/components/common/Icon.js

```
import React from 'react';

const Icon = ({ name, className }) => (
  <i className={`fa fa-${name} ${className}`} />
);

export default Icon;
```

src/client/components/common/index.js

```
export { default as Icon } from './Icon';
export { default as Dialog } from './Dialog';
export { default as FlatIcon } from './FlatIcon';
export { default as Loading } from './Loading';
export { default as SideBar } from './SideBar';
export { default as Card } from './Card';
```

src/client/components/common/Loading.js

```
import React from 'react';

import Icon from './Icon';

const Loading = (props) => (
  <div className="overlay">
    <Icon name="refresh" className="loading-icon fa-spin" />
  </div>
);

export default Loading;
```

src/client/components/common/SideBar.js

```
import React, { Component } from 'react';

import Icon from './Icon';

class SideBar extends Component {
  constructor(props) {
    super(props);
```

```
    this.state = { visible: false };
  }

  show() {
    this.setState({ visible: true });
  }

  hide() {
    this.setState({ visible: false });
  }

  toggle() {
    this.setState({ visible: !this.state.visible });
  }

  render() {
    const { children, title } = this.props;

    return this.state.visible && (
      <div className="sidebar">
        <div className="sidebar-header">
          <a
            onClick={this.toggle.bind(this)}
            className="sidebar-close"
          >
            <Icon name="close" />
          </a>
        </div>
        <div className="sidebar-content">
          {children}
        </div>
      </div>
    );
  }
}

export default SideBar;
```

src/client/components/map/GoogleMap.js

```
import React from 'react';
import GoogleMapReact from 'google-map-react';
```

```
import config from '../../../shared/config';

const GoogleMap = ({ center, zoom, markers }) => (
    <GoogleMapReact
      center={center}
      zoom={zoom}
      minZoom={8}
      bootstrapURLKeys={{ key: config.GMAPS_API_KEY }}
    >
      {markers}
    </GoogleMapReact>
);

GoogleMap.defaultProps = {
  center: [-23.533773, -46.625290],
  zoom: 12,
  markers: []
};

export default GoogleMap;
```

src/client/components/map/index.js

```
export { default as GoogleMap } from './GoogleMap';
export { default as Marker } from './Marker';
export { default as MapCaption } from './MapCaption';
```

src/client/components/map/MapCaption.js

```
import React, { Component } from 'react';
import _ from 'lodash';

import { Icon } from '../common';
import {
  Categories,
  CategoryColor
} from '../../constants';

class MapCaption extends Component {
  constructor(props) {
    super(props);
    this.state = { visible: false };
```

```
  }

  renderMarkers() {
    return _.map(Categories, (v, k) => {
      const color = CategoryColor[k.toUpperCase()];
      return (
        <li key={color}>
          <Icon
            name="map-marker"
            className={`marker-icon ${color}`}
          />
          <span>{v}</span>
        </li>
      );
    });
  }

  toggle() {
    this.setState({ visible: !this.state.visible });
  }

  render() {
    const visible = (this.state.visible) ? '__visible' : '__hidden';
    return (
      <div className={`map-caption ${visible} hidden-sm-down`}>
        <button
          onClick={this.toggle.bind(this)}
          className="btn btn-primary _button"
        >
          <Icon name="map-marker" />
          <span> Legenda</span>
        </button>
        <ul className="_list">
          {this.renderMarkers()}
        </ul>
      </div>
    );
  }
}

export default MapCaption;
```

src/client/components/map/Marker.js

```
import React, { Component, PropTypes } from 'react';

import { Icon } from '../common';

class Marker extends Component {
  constructor(props) {
    super(props);
    this.state = { hover: false };
    this.hoverToggle = this.hoverToggle.bind(this);
  }

  hoverToggle() {
    this.setState({ hover: !this.state.hover });
  }

  render() {
    const { onClick, children, id, color } = this.props;
    return (
      <div
        className="marker"
        onClick={() => onClick(id)}
        onMouseEnter={this.hoverToggle}
        onMouseLeave={this.hoverToggle}
      >
        <Icon className={`marker-icon ${color}`} name="map-marker" />
        {this.state.hover && children}
      </div>
    );
  }
}

Marker.propTypes = {
  onClick: PropTypes.func.isRequired,
  lat: PropTypes.number.isRequired,
  lng: PropTypes.number.isRequired
};

export default Marker;
```

src/client/containers/App.js

```
import React, { Component } from 'react';
import { connect } from 'react-redux';
```

```
import { Footer, Loading } from '../components';

class App extends Component {
  render() {
    return (
      <div className="wrapper">
        {this.props.children}
        <Footer />
        {this.props.loading && <Loading />}
      </div>
    );
  }
}

const mapStateToProps = ({ loading }) => {
  return { loading: loading.show };
};

export default connect(mapStateToProps)(App);
```

src/client/containers/CrimeList.js

```
import React, { Component } from 'react';
import { connect } from 'react-redux';
import { Link } from 'react-router';

import {
  Card,
  FlatIcon,
  Icon
} from '../components';

import {
  Categories,
  CategoryIcon
} from '../constants';

class CrimeList extends Component {
  renderCrimeList() {
    return this.props.points.map((crime, i) => {
      const categoria = crime.categoria.toLowerCase();
      return (
        <Card
          key={crime.idBO}
```

```
          className="crime-card"
          onClick={() => this.props.onCardClick(crime.idBO)}
        >
          <FlatIcon name={CategoryIcon[categoria]} width={70} height={70} />
          <ul className="crime-card-info">
            <li>{Categories[categoria]}</li>
            <li><Icon name="map" /> {crime.local}</li>
            <li><Icon name="calendar" /> {crime.dataOcorrencia}</li>
            <li><Icon name="user" /> {crime.nome}</li>
          </ul>
        </Card>
      )
    });
  }

  render() {
    return (
      <div className="crime-list">
        <Link to="/report" className="btn btn-success">
          <Icon name="print" /> Relatório
        </Link>
        <div className="cards">
          {this.renderCrimeList()}
        </div>
      </div>
    );
  }
}

const mapStateToProps = ({ points }) => {
  return {
    points: points.list
  };
};

export default connect(mapStateToProps)(CrimeList);
```

src/client/containers/index.js

```
export { default as Map } from './Map';
export { default as App } from './App';
export { default as SearchBar } from './SearchBar';
export { default as Report } from './Report';
```

src/client/containers/Map.js

```
import React, { Component } from 'react';
import { connect } from 'react-redux';

import {
  GoogleMap,
  Marker,
  MapCaption,
  CrimePreview,
  Icon,
  SideBar,
  CrimeDetail,
  Dialog,
  Message
} from '../components';

import SearchBar from './SearchBar';
import SearchForm from './SearchForm';
import CrimeList from './CrimeList';

import {
  fetchPoint,
  setSearch,
  changeReportType
} from '../actions';

import {
  CategoryColor,
  ReportTypes
} from '../constants';

class Map extends Component {
  onMarkerClick(id) {
    this.props.changeReportType(ReportTypes.SINGLE, id);

    this.props.fetchPoint(id, () => {
      this.refs.detail.toggle();
    });
  }

  goToResults() {
    $('.nav-tabs a[href="#dados-tab"]').tab('show');
  }

  onDetailClose() {
```

```
  this.props.changeReportType(ReportTypes.COLLECTION);
}

markerList(points) {
  return points.map((point) => {
    const lat = parseFloat(point.lat);
    const lon = parseFloat(point.lon);

    if (!isNaN(lat) && !isNaN(lon)) {
      return (
        <Marker
          id={point.idBO}
          key={point.idBO}
          lat={lat}
          lng={lon}
          color={CategoryColor[point.categoria]}
          onClick={this.onMarkerClick.bind(this)}
        >
          <CrimePreview
            date={point.dataOcorrencia}
            address={point.local}
            category={point.categoria}
            person={point.nome}
          />
        </Marker>
      )
    }
  });
}

render() {
  const msg = this.props.error && (<Message text={this.props.error} />);
  return (
    <div>
      {msg}
      <div className="map-container">
        <SideBar
          ref="leftSidebar"
        >
          <ul className="nav nav-tabs" role="tablist">
            <li className="nav-item">
              <a
                href="#pesquisa-tab"
                className="nav-link active"
                data-toggle="tab"
                role="tab"
```

```
            >
              Pesquisa
            </a>
          </li>
          <li className="nav-item">
            <a
              href="#dados-tab"
              className="nav-link"
              data-toggle="tab"
              role="tab"
            >
              Resultados
            </a>
          </li>
        </ul>
        <div className="tab-content">
          <div id="pesquisa-tab" className="tab-pane active" role="tabpanel">
            <SearchForm onSearch={this.goToResults.bind(this)} />
          </div>

          <div id="dados-tab" className="tab-pane" role="tabpanel">
            <CrimeList onCardClick={this.onMarkerClick.bind(this)} />
          </div>
        </div>
      </SideBar>
      <div className="map">
        <GoogleMap markers={this.markerList(this.props.points)} />
        <div className="map-search">
          <a
            className="menu-btn"
            onClick={() => this.refs.leftSidebar.toggle()}
          >
            <Icon name="bars" />
          </a>
          <SearchBar onSearch={this.goToResults.bind(this)} className="map-
search-bar hidden-sm-down" />
        </div>
        <MapCaption />
      </div>

      <Dialog
        ref="detail"
        title="Detalhamento de Ocorrência"
        onClose={this.onDetailClose.bind(this)}
      >
        <CrimeDetail crime={this.props.selected} />
```

```
          </Dialog>
        </div>
      </div>
    );
  }
}

const mapStateToProps = (state) => {
  const { points } = state;
  return {
    points: points.list,
    selected: points.selected,
    error: points.error
  };
};

export default connect(mapStateToProps, {
  fetchPoint,
  setSearch,
  changeReportType
})(Map);
```

src/client/containers/Report.js

```
import React, { Component, PropTypes } from 'react';
import _ from 'lodash';
import { connect } from 'react-redux';

import { CrimeDetail, Icon } from '../components';
import { fetchReportData } from '../actions';
import { ReportTypes } from '../constants';

class Report extends Component {
  componentWillMount() {
    if(!_.isEmpty(this.props.search) || this.props.id) {
      if(this.props.type === ReportTypes.SINGLE) {
        const search = {};
        search['idReport'] = this.props.id;
        this.props.fetchReportData(search);
      } else {
        this.props.fetchReportData(this.props.search);
      }
    } else {
      this.context.router.push('/');
```

```
    }
  }

  componentDidUpdate() {
    responsiveTabs();
  }

  renderReport() {
    const { data } = this.props;
    return data.map((crime, i) => {
      return (
        <div key={crime.idBO} className="report">
          <CrimeDetail crime={crime} layout="report" />
        </div>
      );
    });
  }

  render() {
    return (
      <div id="report-container" className="container">
        <button
          onClick={() => { this.context.router.push('/mapa') }}
          className="btn btn-primary"
        >
          <Icon name="arrow-left" /> Retornar
        </button>
        {this.props.error ? this.props.error : this.renderReport()}
      </div>
    );
  }
}

Report.contextTypes = {
  router: PropTypes.object.isRequired
}

const mapStateToProps = (state) => {
  return {
    ...state.report,
    search: state.search.properties
  };
};

export default connect(mapStateToProps, { fetchReportData })(Report);
```

src/client/containers/SearchBar.js

```
import React, { Component, PropTypes } from 'react';
import { connect } from 'react-redux';
import _ from 'lodash';

import { Icon } from '../components';
import {
  fetchPoints,
  changeSearchTerm,
  changeSearchType,
  setSearch
} from '../actions';

import {
  SearchTypes,
  FIELD_SELECT
} from '../constants';

class SearchBar extends Component {
  constructor(props) {
    super(props);

    this.onSubmit = this.onSubmit.bind(this);
    this.renderSearchField = this.renderSearchField.bind(this);
    this.renderSearchTypes = this.renderSearchTypes.bind(this);
  }

  onSubmit(event, callback) {
    event.preventDefault();

    const { type, term } = this.props;
    const search = {};

    search[type] = term;

    this.props.setSearch(search);
    this.props.fetchPoints(search);
    if(callback) {
      callback();
    }
  }

  onSearchChange(event) {
    const term = event.target.value;
    this.props.changeSearchTerm(term);
```

```
}

onTypeChange(event) {
  const type = event.target.value;
  this.props.changeSearchType(type);
  this.props.changeSearchTerm(SearchTypes[type].defaultValue);
}

renderSearchTypes() {
  return (
    <select
      className="form-control search-bar-type"
      value={this.props.type}
      onChange={this.onTypeChange.bind(this)}
    >
      {
        _.map(SearchTypes, (v, k) => {
          return <option key={k} value={k}>{v.label}</option>
        })
      }
    </select>
  )
}

renderSearchField() {
  const searchType = SearchTypes[this.props.type];
  const fieldType = searchType.type;

  switch (fieldType) {
    case FIELD_SELECT:
      return (
        <select
          value={this.props.term}
          className="form-control search-bar-value"
          onChange={this.onSearchChange.bind(this)}
        >
          {
            _.map(searchType.values, (v, k) => {
              let keyVal = k;
              if (!isNaN(k)) {
                keyVal = v;
              }
              return <option key={k} value={keyVal}>{v}</option>
            })
          }
        </select>
```

```
      );
    default:
      return (
        <input
          type="text"
          className="form-control search-bar-value"
          placeholder="Insira uma pesquisa..."
          value={this.props.term}
          onChange={this.onSearchChange.bind(this)}
        />
      );
  }
 }

 render() {
  const { onSearch, className } = this.props;

  let cls = 'search-bar';
  if(className) {
    cls += ' ' + className;
  }

  return (
    <form className={cls}
      onSubmit={(event) => this.onSubmit(event, onSearch)}
    >
      <div className="form-group">
        <div className="input-group">
          {this.renderSearchTypes()}
          {this.renderSearchField()}
          <div className="input-group-btn">
            <button
              className="btn btn-primary"
              type="submit"
            >
              <Icon name="search" />
            </button>
          </div>
        </div>
      </div>
    </form>
  );
 }
}

SearchBar.propTypes = {
```

```
  onSearch: PropTypes.func
};

const mapStateToProps = ({ search }) => {
  return { type: search.type, term: search.term };
}

export default connect(
  mapStateToProps, {
    fetchPoints,
    changeSearchTerm,
    changeSearchType,
    setSearch
  })(SearchBar);
```

src/client/containers/SearchForm.js

```
import React, { Component } from 'react';
import { connect } from 'react-redux';
import _ from 'lodash';
import { reduxForm, Field } from 'redux-form';

import { Icon } from '../components';

import {
  SearchTypes,
  FIELD_TEXT,
  FIELD_SELECT,
  FIELD_NUMERIC
} from '../constants';

import { fetchPoints, setSearch } from '../actions';

class SearchForm extends Component {
  constructor(props) {
    super(props);

    this.onSubmit = this.onSubmit.bind(this);
    this.renderFields = this.renderFields.bind(this);
  }

  onSubmit(search, callback) {
    this.props.setSearch(search);
    this.props.fetchPoints(search);
```

```
  if(callback) {
    callback();
  }
}


renderFields(fieldConfig, field) {
  if (fieldConfig.type === FIELD_SELECT) {
    return (
      <Field
        key={field}
        component={renderSelect}
        name={field}
        label={fieldConfig.label}
      >
        <option>-Selecione-</option>
        {
          _.map(fieldConfig.values, (v, k) => {
            let keyVal = k;
            if(!isNaN(k)) {
              keyVal = v;
            }
            return <option key={k} value={keyVal}>{v}</option>
          })
        }
      </Field>
    );
  } else {
    return (
      <Field
        key={field}
        component={renderInput}
        type="text"
        name={field}
        label={fieldConfig.label}
      />
    );
  }
}

render() {
  const { handleSubmit, submitting, onSearch } = this.props;
  return (
    <form
      onSubmit={handleSubmit((props) =>
```

```
        this.onSubmit(props, onSearch))}
        className="container-fluid"
      >
        {_.map(SearchTypes, this.renderFields)}
        <button type="submit" className="btn btn-primary" disabled={submitting}>
          <Icon name="search"></Icon>
          <span> Pesquisar</span>
        </button>
      </form>
    )
  }
}

const renderInput = (field) => (
  <div
    className={`form-group ${field.meta.touched &&
      field.meta.invalid ? 'has-danger' : ''}`}
    >
      <label className="form-control-label">{field.label}</label>
      <input
        className="form-control"
        {...field.input}
      />
      <div className="form-control-feedback">
        {field.meta.touched ? field.meta.error : null}
      </div>
    </div>
);

const renderSelect = (field) => (
  <div
    className={`form-group ${field.meta.touched &&
      field.meta.invalid ? 'has-danger' : ''}`}
    >
      <label className="form-control-label">{field.label}</label>
      <select
        className="form-control"
        children={field.children}
        {...field.input}
      />
      <div className="form-control-feedback">
        {field.meta.touched ? field.meta.error : null}
      </div>
    </div>
);
```

```
const validate = (values) => {
  const errors = {};

  _.each(SearchTypes, (config, field) => {
    const value = values[field];

    if(config.type === FIELD_NUMERIC &&
      value != undefined) {
      if (isNaN(Number(value))) {
        errors[field] = `${config.label} deve ser um número!`;
      } else if(Number(value) < config.minValue ||
          Number(value) > config.maxValue
      ) {
        errors[field] = `O valor de '${config.label}'
            deve ser entre ${config.minValue} e ${config.maxValue}`;
      }
    }
  });

  return errors;
};

export default connect(null, { fetchPoints, setSearch })(reduxForm({
  form: 'SearchForm',
  fields: _.keys(SearchTypes),
  validate
})(SearchForm));
```

src/client/reducers/index.js

```
import { combineReducers } from 'redux';
import { reducer as FormReducer } from 'redux-form';

import PointsReducer from './PointsReducer';
import LoadingReducer from './LoadingReducer';
import SearchReducer from './SearchReducer';
import ReportReducer from './ReportReducer';

const rootReducer = combineReducers({
  points: PointsReducer,
  loading: LoadingReducer,
  search: SearchReducer,
  report: ReportReducer,
  form: FormReducer
```

```
});

export default rootReducer;
```

src/client/reducers/LoadingReducers.js

```
import {
  START_API_CALL,
  END_API_CALL
} from '../actions/types';

const INITIAL_STATE = { show: false };

export default (state = INITIAL_STATE, action) => {
  switch(action.type) {
    case START_API_CALL:
      return { ...state, show: true };
    case END_API_CALL:
      return { ...state, show: false };
    default:
      return state;
  }
}
```

src/client/reducers/PointsReducer.js

```
import {
  FETCH_POINTS_SUCCESS,
  FETCH_POINTS_FAILURE,
  FETCH_POINT_SUCCESS,
  FETCH_POINT_FAILURE
} from '../actions/types';

const INITIAL_STATE = { list: [], selected: null, error: null };

export default (state = INITIAL_STATE, action) => {
  switch (action.type) {
    case FETCH_POINTS_SUCCESS:
      return { ...INITIAL_STATE, list: action.payload };
    case FETCH_POINTS_FAILURE:
      return { ...INITIAL_STATE, error: action.payload };
    case FETCH_POINT_SUCCESS:
```

```
      return { ...state, selected: action.payload, error: null };
    case FETCH_POINT_FAILURE:
      return { ...state, error: action.payload };
    default:
      return state;
  }
};
```

src/client/reducers/ReportReducer.js

```
import {
  FETCH_REPORT_DATA_FAILURE,
  FETCH_REPORT_DATA_SUCCESS,
  CHANGE_REPORT_TYPE
} from '../actions/types';

import { ReportTypes } from '../constants';

const INITIAL_STATE = {
  type: ReportTypes.COLLECTION,
  id: null,
  data: [],
  error: null
};

export default (state = INITIAL_STATE, action) => {
  switch (action.type) {
    case FETCH_REPORT_DATA_SUCCESS:
      return { ...state, data: action.payload, error: null };
    case FETCH_REPORT_DATA_FAILURE:
      return { ...INITIAL_STATE, error: action.payload };
    case CHANGE_REPORT_TYPE:
      return { ...state, ...action.payload };
    default:
      return state;
  }
};
```

src/client/reducers/SearchReducer.js

```
import {
  CHANGE_SEARCH_BAR_TERM,
```

```
  CHANGE_SEARCH_BAR_TYPE,
  SET_COMPLETE_SEARCH
} from '../actions/types';

const INITIAL_STATE = { type: 'name', term: '', properties: {} };

export default (state = INITIAL_STATE, action) => {
  switch (action.type) {
    case CHANGE_SEARCH_BAR_TERM:
      return { ...state, term: action.payload };
    case CHANGE_SEARCH_BAR_TYPE:
      return { ...state, term: '', type: action.payload };
    case SET_COMPLETE_SEARCH:
      return { ...state, properties: action.payload };
    default:
      return state;
  }
};
```

src/server/isomorphic.js

```
import React from 'react';
import { renderToString } from 'react-dom/server';
import { Provider } from 'react-redux';
import { match, RouterContext } from 'react-router';

import routes from '../shared/routes';
import store from '../client/store';

export default (app) => {
  app.get('*', (req, res) => {
    match({ routes, location: req.url }, (error, redirect, props) => {
      if (error) {
        res.status(500).send(error.message);
      } else if (redirect) {
        res.redirect(redirect.pathname + redirect.search);
      } else if (props) {
        const reactOutput = renderToString(
          <Provider store={store}>
            <RouterContext {...props} />
          </Provider>
        );
        res.render('index.ejs', { reactOutput });
      } else {
        res.status(404).send('Not found!');
      }
```

```
    });
  });
};
```

src/server/server.js

```
import express from 'express';
import path from 'path';

import isomorphic from './isomorphic';

const app = express();
const PORT = 3000;

app.use('/', express.static(path.join(__dirname, '../../public')));
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs');

isomorphic(app);

app.listen(PORT, () => {
  console.log('Express is running on port %s', PORT);
});
```

src/server/views/index.ejs

```
<!DOCTYPE html>
<html>
  <head>
    <meta name="viewport" content="height=device-height, user-scalable=no, initial-
scale=1, width=device-width">
    <meta charset="utf-8">
    <title>Isomorphic React App</title>
    <script src="/js/jquery-3.1.1.min.js"></script>
    <script src="/js/bootstrap.min.js"></script>
    <script src="/js/responsive-tabs.js"></script>

    <link rel="stylesheet" type="text/css" href="/css/bootstrap.min.css">
    <link rel="stylesheet" type="text/css" href="/css/font-awesome.min.css">
    <link rel="stylesheet" type="text/css" href="/css/style.css">
    <link rel="shortcut icon" href="/favicon.png" type="image/png">
  </head>
```

```
  <body>
    <div id="root"><%- reactOutput %></div>
  </body>
  <%if (settings.env === 'production') { %>
    <script src="/js/bundle.min.js"></script>
  <% } else { %>
    <script src="http://localhost:3001/js/bundle.js"></script>
  <% } %>
</html>
```

src/shared/config.sample.js

```
const config = {
  GMAPS_API_KEY: 'YOUR KEY HERE',
  DEV_API_URL: 'http://localhost:8080/criminal-report-api',
  PROD_API_URL: 'http://tcc.diegomarques.me/api'
};

export default config;
```

src/shared/routes.js

```
import React from 'react';
import { Route, IndexRoute } from 'react-router';

import { App, Map, Report } from '../client/containers';
import { Home } from '../client/components';

export default (
  <Route path="/" component={App}>
    <IndexRoute component={Home} />
    <Route path="mapa" component={Map} />
    <Route path="report" component={Report} />
  </Route>
);
```

src/shared/util.js

```
import _ from 'lodash';

export const objectToURLParameters = (obj) => {
  return _.map(obj, (value, key) => {
    return `${encodeURIComponent(key)}=${encodeURIComponent(value)}`;
  }).join('&');
};
```

REFERÊNCIAS

ABRAMOV, Dan. “Presentational and Container Components”. 2015. Disponível em <https://medium.com/@dan_abramov/smart-and-dumb-components-7ca2f9a7c7d0>. Acesso em: 22 mai. 2017.

ANGULARJS. Angular Universal. 2017. Disponível em <https://universal.angular.io/overview>. Acesso em: 15 mai. 2017.

AURÉLIO. Isomorfo: significado de Isomorfo. 2017. Disponível em: ‹https://dicionariodoaurelio.com/isomorfo›. Acesso em: 14 mai. 2017.

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.

CAPAN, Tomislav. Why The Hell Would I Use Node.js? A Case-by-Case Tutorial. 2013. Disponível em <https://www.toptal.com/nodejs/why-the-hell-would-i-use-node-js>. Acesso em: 14 mai. 2017.

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

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

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

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

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

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

GEITZ, Samantha. React 101. Part 2: Refactoring with Redux. 2016. Disponível em <https://blog.tighten.co/react-101-using-redux>. Acesso em: 26 mai. 2017

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

ISTARKOV. GitHub - istarkov/google-map-react: universal google map react component, allows render react components on the google map. 2017. Disponível em <https://github.com/istarkov/google-map-react>. Acesso em: 15 fev. 2017.

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.

METEOR. Introduction | Meteor Guide. 2017. Disponível em <https://guide.meteor.com>. Acesso em: 15 mai. 2017.

MIKOWSKI, Michael S.; POWELL, Josh C. "Single Page Web Applications: JavaScript End-to-End". Manning Publications. Ed. 1. 2013.

NODEJS. About | Node.js. 2016. Disponível em <https://nodejs.org/en/about/>. 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”. 15th International Conference WWW/Internet. p. 111–118. ISBN 978-989-8533-57-9. Mannheim, Germany: IADIS Press, 2016.

PHANTOMAS. PhantomJS-based web performance metrics collector and monitoring tool. 2017. Disponível em <https://github.com/macbre/phantomas>. Acesso em: 26 mai. 2017.

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.

REACTROUTER. Introduction. 2017a Disponível em <https://github.com/ReactTraining/react-router/blob/v3/docs/Introduction.md>. Acesso em: 23 mai. 2017.

REACTROUTER. Route Matching. 2017b. Disponível em <https://github.com/ReactTraining/react-router/blob/v3/docs/guides/RouteMatching.md>. Acesso em: 23 mai. 2017.

REACTROUTER. Server Rendering. 2017c. Disponível em <https://github.com/ReactTraining/react-router/blob/v3/docs/guides/ServerRendering.md>. Acesso em: 20 mai. 2017.

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

SEBESTA, Robert W. Programming the world wide web. 7th ed. Boston: Pearson, 2012. ISBN 9780132665810

SCOTT, Kendall. O Processo Unificado Explicado. Ed. Bookman, 2003.

SMARTY, Ann. Let's Try to Find All 200 Parameters in Google Algorithm. 2009. Disponível em <https://www.searchenginejournal.com/200-parameters-in-google-algorithm/15457>. Acesso em: 26 mai. 2017.

STRIMPEL, Jason; NAJIM, Maxime. "Building Isomorphic JavaScript Apps: From Concept to Implementation to Real-World Solutions". O'Reilly Media. 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.

TOGGL. Toggl Features: calculate work hours & billable hours, employee timesheets. 2017. Disponível em <https://toggl.com/features>. Acesso em: 29 mai. 2017.

TSONEV, Krasimir. Deep dive into client-side routing. 2016. Disponível <http://krasimirtsonev.com/blog/article/deep-dive-into-client-side-routing-navigo-pushstate-hash>. Acesso em: 14 mai. 2017.

VUEJS. Introduction - Vue.js. 2017. Disponível em <https://vuejs.org/v2/guide/>. Acesso em: 15 mai. 2017.

ANEXO I - ARTIGO SOBRE O TRABALHO

Abordagem de desenvolvimento utilizando JavaScript Isomórfico

Diego da Silva Marques

Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC) – Florianópolis – SC - Brasil

dmarquesdev@gmail.com

Abstract. *This research aims to show what is Isomorphic JavaScript, what it does, what make it different from a traditional development approach, define what is a traditional development approach, differentiate the many terms used to define their concepts or related concepts, which tools can be used to build applications with it and show by an example how the Isomorphic JavaScript development process works using these tools. The developed application is a system where it is possible to query an API that contains police criminal reports data from São Paulo state, publicly found, that were previously data enriched. With some changes on the server side configuration, a version that simulates a traditional application has been made available to compare them both, showing that the differences between them do exists, but some of them had a low relevance at the end.*

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, definir o que é a abordagem 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. Com base em alterações nas configurações da aplicação no servidor, foi disponibilizada uma versão que ilustra o que seria uma aplicação tradicional e feita a comparação entre ambas, mostrando que as diferenças propostas existem porém com um pequeno grau de relevância em alguns casos.*

1. Introdução

O desenvolvimento de aplicações Web 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 Side Rendered", 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.

2. Tecnologias, conceitos e frameworks

É necessário definir alguns conceitos relacionados ao trabalho, bem como algumas tecnologias e frameworks utilizados no mesmo, dentre eles:

- **Aplicação Web**: Como pode ser visto em SEBESTA (2012), uma aplicação Web pode ser definida como uma aplicação distribuída que utiliza a arquitetura cliente-servidor, onde existe um servidor capaz de receber requisições que utilizam o protocolo HTTP e tratá-las conforme seu algoritmo interno e retornar o resultado deste processamento através de uma resposta HTTP para o cliente que efetuou a requisição.
- **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.
- **SEO**: Acrônimo para *Search Engine Optimization*, são uma série de técnicas para otimizar o posicionamento de uma página ou aplicação Web nos mecanismos buscadores da internet, como o Google. Segundo é mostrado em SMARTY (2009), temos mais de 200 variáveis que podem alterar a maneira como uma página ou aplicação irá ser buscada, e consequentemente mostrada, por um mecanismo deste tipo.
- **React**: Framework que permite ao usuário criar diversos componentes de interface gráfica 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, como visto em BREHM (2013).

- **Node.js**: 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**: 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.

3. 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 no lado do servidor da aplicação, tendo se popularizado com a publicação de BREHM (2013).

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.

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.

Em uma SPA tradicional, o conteúdo semanticamente relevante da página não se encontra disponível na mesma no momento da primeira requisição ao servidor, 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.

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 4, 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 (Search Engine Optimization), 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.

Outro problema inerente às SPAs tradicionais é um problema de roteamento, pois aplicações no modelo SPA utilizam tecnologias de roteamento disponíveis no cliente, o que causa um problema ao acesso direto as URLs.

Para exemplificar este problema, suponhamos que a aplicação esteja disponível através do endereço exemplo.com e que a mesma tenha uma segunda página que possa ser acessada através da URL example.com/page2. Ao efetuar uma requisição ao servidor, o mesmo irá retornar o Entry Point da aplicação, e o cliente ao receber o Entry Point sem nenhuma informação sobre a rota /page2 irá carregar o contexto da página inicial. Tal problema é comumente resolvido através da tecnologia conhecida como Hash History, como visto em TSONEV (2016), onde o cliente consegue manter informações sobre a rota acessada pelo usuário através de uma formatação especial na URL, que no nosso exemplo seria algo como exemplo.com/#/page2. O problema do uso de Hash History é a adição do caractere ‘#’ ao URL, o que pode causar problemas ao SEO e dificultar o processo de aprendizagem e lembrança da mesma por parte dos usuários, além do cliente reter o conteúdo que segue o caractere ao efetuar requisições HTTP como visto em TSONEV (2016), o que torna o servidor alheio ao roteamento da aplicação, o que novamente, é uma das características das SPAs 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. Com relação às rotas, o servidor tem a capacidade completa de trabalhar as mais diversas rotas e mudar o contexto da aplicação dependendo das mesmas, visto que o mesmo processa a aplicação antes de enviar a resposta, conseguido assim alterar seu contexto.

4. Projeto da Aplicação de Exemplo

A aplicação 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 são fornecidos através da Web API proposta por OLIVEIRA *et al.* (2016), e

conta com um mapa interativo para prover ricas experiências de navegabilidade e visualização dos dados, além de ser possível criar modelos para impressão utilizando as consultas executadas.

Para cumprir com os requisitos citados, são necessárias basicamente três páginas, sendo que o conceito de página no escopo deste trabalho, como já discutido anteriormente, não está ligado ao conceito de três arquivos HTML como em outras aplicações. Tais páginas podem ser definidas como:

● Mapa: a página de mapa deve ser a página principal da aplicação, onde o usuário terá contato com um mapa interativo e todas as funcionalidades do mesmo.

● Relatório: uma página na qual o usuário será apresentado apenas com dados referentes a pesquisas realizadas na página de interação da aplicação..

● Pagina Inicial: uma página que será apresentada ao usuário explicando sobre a aplicação e fornecendo um campo para pesquisa simples, que levará à pagina de Mapa.

O resultado visual da página de mapa pode ser conferido na Figura 1.

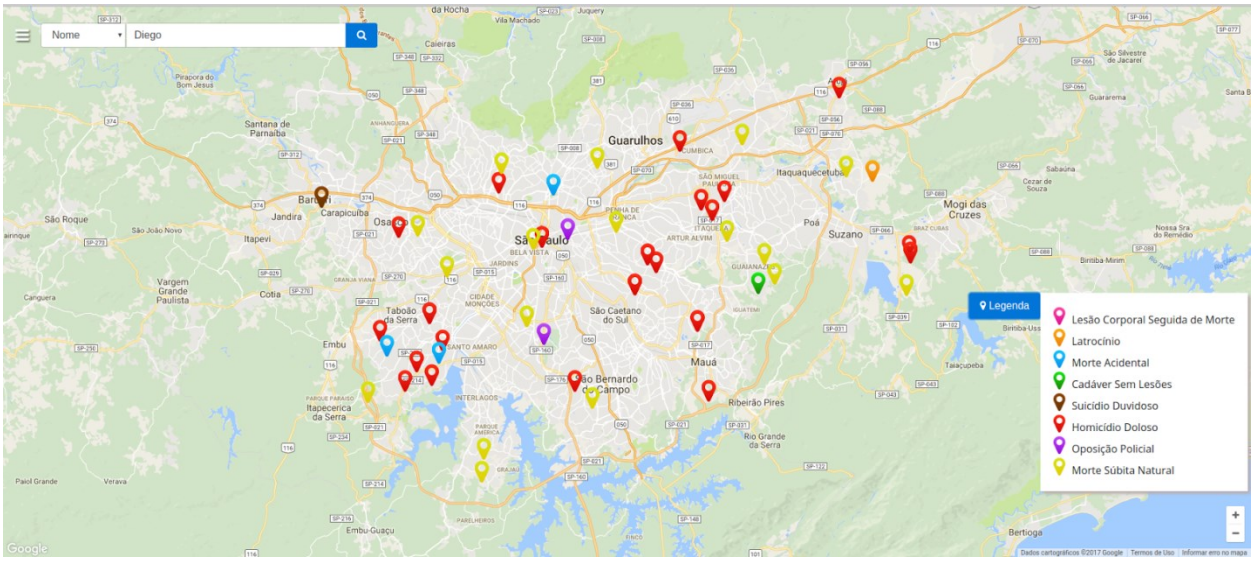

Figura 1. Página principal da aplicação

Com todos os requisitos definidos implementados e com a aplicação funcionando, é possível hospedar a aplicação para acesso através da internet em algum serviço especializado. Com base nas ferramentas utilizadas, como o Webpack e NPM, é muito simples a portabilidade do projeto e instalação do mesmo em qualquer ambiente, bastando que todas as dependências estejam instaladas

Apesar da aplicação isomórfica estar implementada, para que os testes e as métricas possam ser executados comparando uma aplicação tradicional a uma aplicação isomórfica, é necessário que exista uma versão tradicional da aplicação. Os conceitos que envolvem o JavaScript Isomórfico são tratados pelos frameworks através da união de uma *template engine* com a capacidade de processar o código do cliente no servidor. Portanto para se obter uma versão tradicional derivada da aplicação isomórfica desenvolvida, é necessário uma alteração na estrutura da aplicação nestes pontos.

O resultado de ambas abordagens pode ser conferido nos endereços http://isomorfico.tcc.diegomarques.me/ e http://tradicional.tcc.diegomarques.me/, respectivamente. Vale ressaltar que estes endereços estarão disponíveis durante todo o período do ano de 2017, ano da publicação deste trabalho, porém os mesmos não serão garantidos posteriormente.

Muitas ferramentas e tecnologias para JavaScript não estão preparadas para lidar com aplicações de natureza isomórfica, como ferramentas que necessitam de recursos que só estão disponíveis nos browsers, pois apesar do servidor ter a capacidade de executar o código JavaScript, não é a intenção das tecnologias do servidor de terem as mesmas capacidades dos browsers, o que restringe as possibilidades de um desenvolvedor. Por conta disso, algumas bibliotecas tiveram que ser completamente trocadas durante o processo de desenvolvimento, pois tal comportamento só pôde ser percebido após o início de seu uso.

5. Testes e Métricas

Os testes apresentados foram efetuados de diversas formas, utilizando desde ferramentas automatizadas até testes manuais, e têm como objetivo comparar as duas aplicações.

5.1 Primeira Requisição

A principal diferença entre uma aplicação isomórfica e uma aplicação tradicional é o conceito da primeira requisição. A resposta à primeira requisição HTTP de uma aplicação isomórfica deve conter a estrutura da página HTML completamente disponível, ou seja, o cliente deve necessitar do mínimo de mudanças possíveis em seu DOM para que a aplicação esteja disponível visualmente ao usuário após o mesmo processar tal resposta.

O resultado dos testes efetuados nesta seção podem ser observados através das Figuras 2 e 3 em suas respectivas linhas 17. Na Figura 2, podemos ver os primeiros elementos do DOM da aplicação dentro do *Entry Point* na resposta do HTTP, sendo que grande parte do seu conteúdo foi omitido por conta da maneira que a ferramenta mostra os dados. Já na Figura 3 podemos ver apenas o *Entry Point* da aplicação, portanto podemos concluir que os conceitos relacionados puramente ao DOM são verdadeiros.

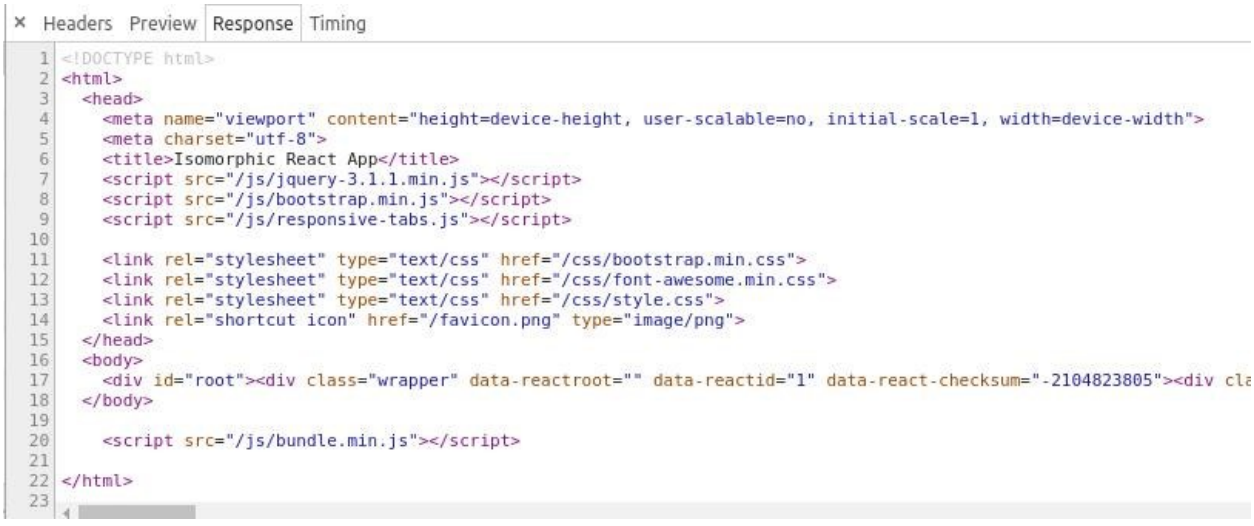

Figura 2. Resposta a uma requisição da aplicação isomórfica

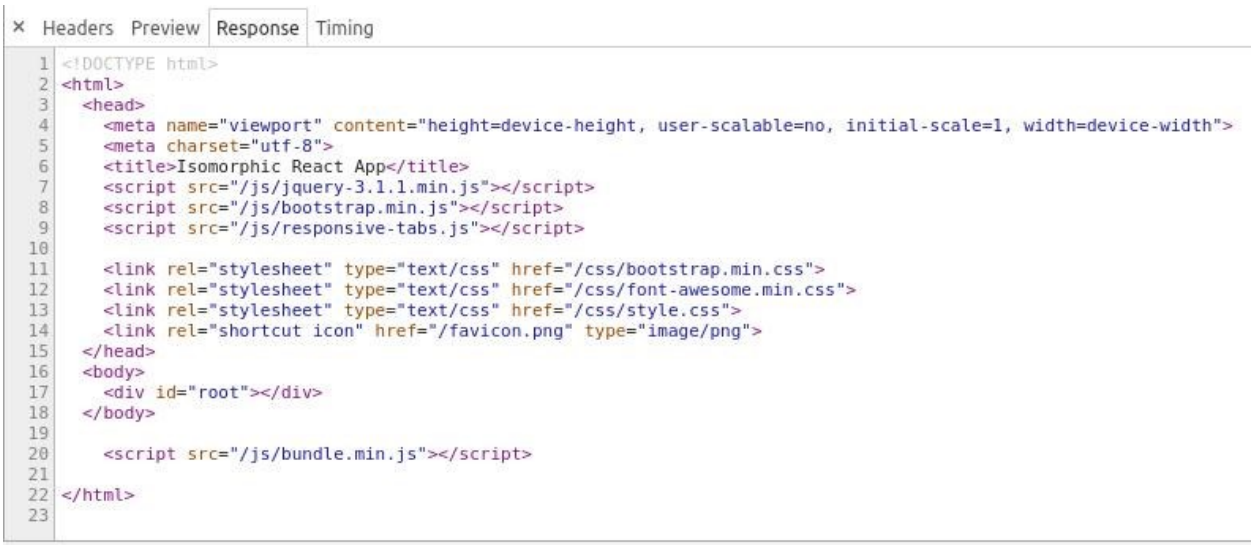

Figura 3. Resposta a uma requisição da aplicação tradicional

5.2 Tempo de Carregamento

O tempo de carregamento considerado neste teste se trata do tempo entre o cliente enviar a requisição e a página estar carregada, ou seja, sua resposta processada e o DOM totalmente gerado, além de diversos tempos parciais como o tempo para o carregamento do DOM, o tempo para obter apenas a resposta, entre outros.

Para este teste precisamos de diversas requisições para chegar a um tempo médio e o desvio padrão entre as amostragens. Para efetuar um grande número de requisições, foi utilizada a ferramenta automatizada Phantomas, que como visto em PHANTOMAS (2017), tem a capacidade de fornecer muitos dos dados necessários para o escopo deste teste. O resultado da execução de 100 requisições em ambas as aplicações pode ser conferido na Tabela 1.

	Isomórfica		Tradicional	
	Média	Desv. Padr.	Média	Desv. Padr.
Total de elementos no DOM	76	0	2	0
Tempo para DOM Interativo (ms)	1314.8	30.52	1314.59	27.86
Tempo para DOM Completo (ms)	1322.99	27.08	1316.47	27.9

Tabela 1 - Resultados dos testes de Tempo de Carregamento

Podemos chegar à conclusão de que o impacto da abordagem isomórfica sobre a abordagem tradicional no quesito de tempo de carregamento pode ser considerado praticamente irrelevante, pois a diferença na média entre ambos é muito baixa, algo na grandeza de poucos bytes ou poucos milissegundos, porém como era de se esperar o tamanho do DOM da resposta das duas aplicações demonstra uma diferença expressiva, visto que o mesmo já é recebido pelo cliente completamente carregado na abordagem isomórfica.

5.3 Performance do Cliente

Levando em consideração uma única amostra de cada aplicação, alguns pontos-chave podem ser observados. Dentre estes pontos, podemos observar que a aplicação isomórfica tende a consumir menos recursos computacionais do que a aplicação tradicional. Além disso, podemos observar que enquanto a aplicação isomórfica passa mais tempo interpretando o HTML, a aplicação tradicional passa mais tempo executando código JavaScript, o que nos leva a concluir que a diferença de performance entre as duas abordagens existe, porém em uma aplicação do porte da aplicação proposta ou menor, provavelmente não haverá um grande impacto na experiência do usuário.

6. Conclusões

Neste trabalho foram apresentados os diversos conceitos ligados à abordagem de desenvolvimento isomórfica utilizando JavaScript, como seus benefícios e características, assim como a definição de uma abordagem análoga à abordagem isomórfica, definida como abordagem tradicional, para fins de comparação. Com os conceitos em mãos, foi possível encontrar um conjunto de tecnologias e ferramentas capaz de botar em prática tudo que havia sido discutido, através da implementação de uma aplicação utilizando tais tecnologias.

Durante o processo de desenvolvimento, foram encontrados alguns problemas relacionados às tecnologias escolhidas, por conflitos com a abordagem isomórfica, que necessitaram de mudanças durante o processo, o que mostra que apesar de todos os seus benefícios, a abordagem isomórfica traz consigo alguns problemas.

Após o desenvolvimento e publicação da aplicação, da forma isomórfica e da forma tradicional, foram executados diversos testes em cima de ambas. Através dos resultados obtidos é possível concluir que diferenças existem, porém em alguns casos em uma proporção muito baixa que pode tornar sua relevância questionável.

Mesmo obtendo resultados inesperados com base nas premissas, a abordagem de desenvolvimento utilizando JavaScript Isomórfico continua sendo muito promissora, por trazer melhorias e correções a um modelo que contém algumas deficiências. Apesar das dificuldades enfrentadas ao se incluir novos conceitos dentro da rotina e do modelo de desenvolvimento de aplicações para Web nos dias atuais, no geral os benefícios que se obtém em troca são muito proveitosos. Mesmo que em alguns aspectos tais benefícios não sejam tão expressivos, eles ainda existem.

Além dos benefícios ligados ao funcionamento da aplicação, o processo de desenvolvimento utilizando somente uma linguagem de programação para desenvolver toda a aplicação traz benefícios tanto para desenvolvedores experientes, que no caso podem focar nas peculiaridades de apenas uma sintaxe de linguagem, como para desenvolvedores menos experientes, que ganham a capacidade de desenvolver uma aplicação completa dominando apenas um tipo de linguagem. Portanto, a abordagem isomórfica pode ser considerada uma ótima opção para se desenvolver uma aplicação para a Web.

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.

MIKOWSKI, Michael S.; POWELL, Josh C. "Single Page Web Applications: JavaScript End-to-End". Manning Publications. Ed. 1. 2013.

NODEJS. About | Node.js. 2016. Disponível em <https://nodejs.org/en/about/>. 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”. 15th International Conference WWW/Internet. p. 111–118. ISBN 978-989-8533-57-9. Mannheim, Germany: IADIS Press, 2016.

PHANTOMAS. PhantomJS-based web performance metrics collector and monitoring tool. 2017. Disponível em <https://github.com/macbre/phantomas>. Acesso em: 26 mai. 2017.

SEBESTA, Robert W. Programming the world wide web. 7th ed. Boston: Pearson, 2012. ISBN 9780132665810.

SMARTY, Ann. Let’s Try to Find All 200 Parameters in Google Algorithm. 2009. Disponível em <https://www.searchenginejournal.com/200-parameters-in-google-algorithm/15457>. Acesso em: 26 mai. 2017.

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

TSONEV, Krasimir. Deep dive into client-side routing. 2016. Disponível <http://krasimirtsonev.com/blog/article/deep-dive-into-client-side-routing-navigo-pushstate-hash>. Acesso em: 14 mai. 2017.