UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
ESPECIALIZAÇÃO EM BANCO DE DADOS

JEAN VALDIR BUZZELLO

PROTÓTIPO DE UM SISTEMA COLABORATIVO PARA IDENTIFICAÇÃO DE FOCOS DO MOSQUITO AEDES AEGYPTI

MONOGRAFIA DE ESPECIALIZAÇÃO

PATO BRANCO
2017

JEAN VALDIR BUZZELLO

PROTÓTIPO DE UM SISTEMA COLABORATIVO PARA IDENTIFICAÇÃO DE FOCOS DO MOSQUITO AEDES AEGYPTI

Trabalho de Conclusão de Curso, apresentado ao II Curso de Especialização em Banco de Dados, da Universidade Tecnológica Federal do Paraná, campus Pato Branco, como requisito parcial para obtenção do título de Especialista.

Orientador(a): Prof.(a). Bruno Ribas.

PATO BRANCO
2017

TERMO DE APROVAÇÃO

PROTÓTIPO DE UM SISTEMA COLABORATIVO PARA IDENTIFICAÇÃO DE FOCOS DO MOSQUITO *AEDES AEGYPTI*.

por

JEAN VALDIR BUZZELLO

Este Trabalho de Conclusão de Curso foi apresentado em 22 fevereiro de 2017 como requisito parcial para a obtenção do título de Especialista em Banco de Dados. O(a) candidato(a) foi arguido(a) pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____
Bruno Cesar Ribas
Prof.(a) Orientador(a)

_____
Luciene de Oliveira Marin
Membro titular

_____
Beatriz Terezinha Borsoi
Membro titular

"O Termo de Aprovação assinado encontra-se na Coordenação do Curso"

RESUMO

BUZZELLO, Jean Valdir. Protótipo de um Sistema colaborativo para identificação de focos do mosquito *Aedes Aegypti*. 2017. 46 f. Monografia (II Curso de Especialização em Banco de Dados) - Universidade Tecnológica Federal do Paraná. Pato Branco, 2017.

A Dengue, a Chikungunya e a Zika, doenças causadas por vírus e que são transmitida por várias espécies de mosquito do gênero Aedes, acometem milhares de brasileiros todos os anos, e são de difícil controle. A dificuldade de controle dessas doenças vem do fato de que os locais que fornecem as condições para a reprodução do mosquito transmissor serem abundantes, alguns de difícil acesso ou identificação pelos programas de controle de saúde pública que buscam eliminar esses pontos. Visando diminuir a dificuldade de identificar os focos de criadouros de mosquito, viu-se a oportunidade de desenvolver um protótipo de um sistema colaborativo para identificação de focos do mosquito, que se justifica pela urgência e o crescimento da doença no Brasil. O setor público se beneficia com a ferramenta, encaminhando os agentes de saúde com mais agilidade aos pontos que contenham o mosquito. Uma das características principais do sistema é a participação da comunidade na tarefa de cadastrar os pontos com focos de mosquitos, através de uma interface no formato do mapa do município de Pato Branco, e guardadas no banco de dados como pontos geográficos.

Palavras-chave: *MongoDB. Google Maps API. Aedes Aegypti. Node.js.*

ABSTRACT

BUZZELLO, Jean Valdir. A prototype of a colaborative software to identify evidences of Aedes Aegypti. 2017. 46 p. Monography (II Specialization Course in Database) - Federal University of Technology - Parana. Pato Branco, 2017.

Dengue, Chikungunya and Zika diseases caused by viruses and transmitted by various species of mosquito of the genus Aedes, affect thousands of Brazilians every year, and are difficult to control. The difficulty of controlling these diseases comes from the fact that the sites that provide the conditions for the reproduction of the transmitting mosquito are abundant, some of them difficult to access or identified by the public health control programs that seek to eliminate these points. Aiming to reduce the difficulty of identifying mosquito breeder outbreaks, an opportunity was developed to develop a prototype of a collaborative system for identifying mosquito outbreaks, which is justified by the urgency and growth of the disease in Brazil. The public sector benefits from the tool, directing health workers more quickly to the points that contain the mosquito. One of the main characteristics of the system is the participation of the community in the task of registering the points with mosquito pictures, through an interface in the format of the map of the municipality of Pato Branco, and stored in the database as geographic points.

Palavras-chave: MongoDB. Google Maps API. Aedes Aegypti. Node.js.

LISTA DE SIGLAS

AJAX	*Asynchronous JavaScript and XML*
API	*Application Programming Interface*
CRUD	*Create, Read, Update e Delete*
CSS	*Cascading Style Sheets*
GPS	*Global Positioning System*
HTML	*HyperText Markup Language*
HTTP	*HyperText Transfer Protocol*
I/O	*input/output*
JSON	*JavaScript Object Notation*
MVC	*Model-view-controller*
RAM	*Random Access Memory*
URL	*Uniform Resource Locator*

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE CÓDIGOS

SUMÁRIO

1 INTRODUÇÃO

1.1 CONSIDERAÇÕES INICIAIS

O *Aedes Aegypti* é uma espécie de mosquito que habita as regiões tropicais e subtropicais. Originário da África, sua disseminação a outros continentes ocorreu através de barris de água transportados nos barcos durante as primeiras explorações e colonizações europeias (NELSON, 1986).

Essa espécie é a principal responsável transmissão da Dengue, Chikungunya e Zika no Brasil. A transmissão desses vírus acontecem pela picada da fêmea do mosquito, que se alimenta de sangue humano ou de animais domésticos. A alimentação acontece principalmente nas primeiras horas do dia, ao meio da manhã ou ao anoitecer. Depois da cópula a fêmea precisa se alimentar de sangue para o desenvolvimento dos ovos. Geralmente o intervalo entre a alimentação sanguínea e a postura dos ovos é de 3 dias e o número de ovos em torno de 100. Na desova, a fêmea deposita seus ovos em recipientes naturais ou artificiais, geralmente com pouca luminosidade e com paredes duras. Os ovos se fixam nas paredes úmidas do recipiente, acima da superfície da água, onde permanecem até eclodirem e darem origem as larvas (NELSON, 1986).

Devido a uma estreita relação com o homem, o Aedes é um mosquito essencialmente urbano. Isso fica claro quando analisamos os dados do Ministério da Saúde que descreve cerca de 70% dos casos notificados da Dengue no país se concentram em Municípios com mais de 50.000 habitantes (MINISTÉRIO PÚBLICO DO ESTADO DO PARANÁ, 2016).

Atualmente, 80% dos brasileiros vivem em cidades. O grande fluxo de migração da área rural para as cidades teve como consequências o crescimento desordenado e falta de condições básicas de saneamento, facilitando a proliferação do mosquito (BRAGA; VALLE, 2007).

Segundo boletim epidemiológico do ministério da saúde, em 2016, até a semana epidemiológica 37 (3/1/2016 a 17/09/2016), foram registrados 1.438.624 casos prováveis de Dengue 200.465 casos prováveis de febre pelo vírus Zika e 38.332 casos prováveis de febre de Chikungunya no país. No Paraná ocorreram 43.034 casos de casos prováveis de Dengue em 2015 contra 63.899 em 2016, 17 casos prováveis de febre de Chikungunya em 2015 contra 983 em 2016, além de 1.043 novos casos febre pelo vírus Zika em 2016 (SECRETARIA DE VIGILÂNCIA EM SAÚDE, 2016).

A Dengue é uma doença febril aguda que se apresenta nas formas de Dengue clássica e febre hemorrágica. Atualmente existem 4 tipos de vírus identificados. A Dengue clássica tem como principais sintomas febre alta, dores de cabeça, dores musculares, debilidade física, dores nas articulações, perda de peso, dor ao redor dos olhos, náuseas, vômitos, erupções cutâneas, dores abdominais, diarreia, desidratação (FUNDAÇÃO NACIONAL DE SAÚDE, 2016). A doença tem duração de 5 a 7 dias, mas a recuperação pode demorar várias semanas. Na forma de febre hemorrágica, os sintomas são semelhantes ao da Dengue clássica, porém com um agravamento entre o terceiro e o quarto dia, com o aparecimento de hemorragia e colapso no sistema circulatório. Quando o mosquito infecta uma pessoa por um dos quatro tipos de vírus, seu organismo desenvolve anticorpos para aquele tipo específico de vírus. Se houver reincidência da doença, mas o tipo do vírus for diferente do que ela havia contraído inicialmente, o organismo utilizará os anticorpos do primeiro tipo para combater ao vírus atual. Devido a esse fato, algumas células liberam substâncias que enfraquecem as paredes dos vasos sanguíneos, ocorrendo a perda do plasma, e como consequência o sangue tem um aumento de concentração das hemácias, deixando-o mais espesso, como consequência a pressão arterial cai e deixa de irrigar algumas partes do corpo, causando falência circulatória, o que pode levar ao óbito em 12 a 24 horas (CONSELHO FEDERAL DE FARMÁCIA, 2002).

A febre Chikungunya embora tenha sintomas em comum com a Dengue, se distingue por gerar dores fortes nas articulações. A doença passa por 3 fases: aguda, subaguda e crônica. Na fase aguda pode ocorrer febre, dores musculares, dores de cabeça, náusea e fadiga, e durar em média 7 dias. Na fase subaguda a febre desaparece e acontece o agravamento das dores nas articulações. Essa fase pode ter duração de até 3 meses. Após 3 meses acontece a fase crônica, com a persistência dos sintomas de dores nas articulações e musco esquelética, com limitação dos movimentos. Apesar de a letalidade ocorrer mais comumente em pessoas com idade avançada e quando em combinação com outra patologia, tem caráter epidêmico e com redução de produtividade e qualidade de vida (SECRETARIA DE VIGILÂNCIA EM SAÚDE, 2015).

A Zika é uma doença febril de duração de 3 a 6 dias. Se caracteriza por febre baixa, conjuntivite, dor de cabeça, dor nas articulações das mãos e pés, alguns casos podem gerar inflamações nas articulações, fadiga, vômitos e diarreia. A doença é considerada sem muita gravidade, e em média 80% das pessoas infectadas não geram as manifestações clínicas. Entretanto, foi confirmado pelo Ministério da Saúde a relação entre o Zika e a microcefalia. A transmissão do vírus no período de gestação pode causar a malformação congênita em que o

cérebro não se desenvolve de maneira adequada, o que em 90% dos casos leva ao retardo mental (SECRETARIA DE ATENÇÃO BÁSICA, 2016).

Como ainda não foram desenvolvidas vacinas ou tratamentos específicos para a Dengue, Zika e Chikungunya (MINISTÉRIO DA SAÚDE, 2016), a maior parte dos esforços se concentra em reduzir o índice de infestação pelo Aedes Aegypti. Uma das principais ações tomadas é a visita dos agentes de saúde em imóveis, orientando os moradores sobre medidas de prevenção e na identificação de focos.

De acordo com informações contidas no blog do Ministério da Saúde, aproximadamente 35% dos imóveis brasileiros foram visitados pelos agentes de saúde, cerca de 23,8 milhões dos 67 milhões de imóveis estimados (Blog da Saúde, 2016). Diante desses dados, fica evidente a dificuldade em mapear todos os locais que dão as condições para a reprodução do mosquito transmissor. Estes focos podem ter uma infinidade de formas de ocorrer, como caixas destampadas, lixo acumulado, garrafas, laje com acumulo de água, pratos de vasos com plantas, pneus velhos, calhas e fossas.

Visando desenvolver uma nova ferramenta no mapeamento de focos do mosquito e auxiliar os agentes de saúde, foi desenvolvido um protótipo de um sistema colaborativo, em que a população cadastra os focos de criadouro do mosquito. Cada foco cadastrado é mostrado como um ponto em um mapa, e cada ponto contém a descrição do local, endereço e foto.

1.2 OBJETIVOS

O objetivo geral será o desenvolvimento de um protótipo de aplicativo para mapeamento dos focos de criadouro do mosquito Aedes Aegypti no município de Pato Branco. Os objetivos específicos se referem a como os dados armazenados serão visualizados e sobre a integração de componentes que usam a da linguagem JavaScript na construção de aplicações *web*.

1.2.1 Objetivo Geral

O objetivo geral consiste em implementar um sistema *web* para guardar dados georreferenciados cadastrados pela população e auxiliar a saúde pública a identificar os focos do mosquito Aedes Aegypti dentro do município de Pato Branco.

1.2.2 Objetivos Específicos

Usar os dados georreferenciados armazenados no banco de dados para criar pontos em um mapa com componentes de interface com o usuário.

Exemplificar o uso do JavaScript como linguagem principal na construção de aplicações *web*, mostrando a integração entre tecnologias que usam o JavaScript na escrita de seu código.

1.3 JUSTIFICATIVA

A criação de um protótipo de um sistema para o controle do Aedes Aegypti se fundamenta nos números divulgados pelo Ministério da Saúde sobre casos das doenças transmitidas pelo mosquito. Além das doenças com transmissão tipicamente relacionadas ao mosquito Aedes, como Dengue, Zika e Chikingunya, está a febre amarela.

No começo de 2017 um novo surto do vírus causador da febre amarela teve início em algumas cidades de Minas Gerais. A febre amarela tem um ciclo de transmissão silvestre e um urbano. No ciclo transmissor silvestre existe como hospedeiro macaco e como vetor o mosquito Haemagogus. No ciclo urbano os hospedeiros são os seres humanos e o vetor o mosquito Aedes Aegypti. Desde 1942 não há registro de casos em áreas urbanas, mas especialistas temem que se ocorrer a reintrodução do vírus causador da febre amarela no ambiente urbano infestado pelo mosquito Aedes Aegypti, a doença pode ganhar proporções nunca registradas e gerar grande risco á saúde pública (PORTAL DA SAÚDE, *2016*).

O desenvolvimento de um protótipo de sistema para mapear os focos de criadouro do mosquito Aedes Aegypti tem como intuito se tornar uma ferramenta auxiliar no controle das doenças Dengue, Chikungunya e Zika, estimulando a população pato-branquense, pouco adepta dos aplicativos similares já existentes, a contribuir para o controle do mosquito. Enquanto as vacinas estão em fase de desenvolvimento, a redução da população de mosquitos se torna uma das principais formas de diminuir a incidência dessas doenças. A eliminação dos focos do mosquito, feita pelos agentes de saúde, pode ser agilizada pelo processo de mapeamento feito com ajuda da população no trabalho de cadastrar os pontos no sistema.

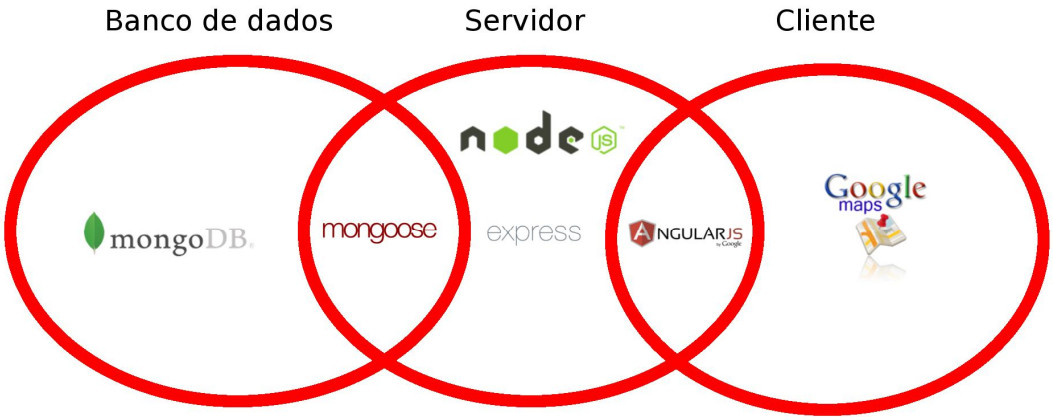
1.4 ESTRUTURA DO TRABALHO

O trabalho está dividido em 5 capítulos. O capítulo 1 aborda as considerações iniciais, objetivos e justificativas para o presente trabalho. No capítulo 2 está o referencial teórico ne-

cessário para o entendimento dos principais componentes que formam o sistema. O capítulo 3 apresenta alguns trabalhos relacionados, descrevendo funcionalidades semelhantes às encontradas a este trabalho. No capítulo 4 acontece a apresentação do sistema através de descrição e imagem das principais telas. Finalmente no capítulo 5 está a conclusão e as melhorias do sistema para trabalhos futuros.

2 REFERENCIAL TEÓRICO

As principais soluções técnicas para implementar o sistema são MongoDB, Google Maps JavaScript API, Node.js. O MongoDB é um banco de dados orientado a documentos que suporta nativamente, sem a necessidade de *plugins* adicionais, o armazenamento de dados geográficos. O Node.js é uma plataforma que proporciona a execução do código JavaScript dentro do servidor e suporta módulos para funcionalidades adicionais, como o Mongoose (padronização e métodos de acesso aos registros do banco de dados), Express (recursos comuns de um servidor HTTP) e o Angular (divisão da aplicação em camadas). O Google Maps JavaScript API fornece a parte visual de mapas e localização dos pontos cadastrados, usando os dados de latitude e longitude que estão armazenados no banco de dados. A integração dessas três tecnologias é uma boa maneira de exemplificar o uso da linguagem JavaScript em diferentes contextos, pois todas usam essa linguagem de formas distintas, seja como consulta no banco de dados em forma de JavaScript Object Notation (JSON), ao instanciar objetos da parte visual do mapa ou construir a lógica da aplicação. A Figura 1 representa como as tecnologias estão distribuídas e se encadeiam. O MongoDB guarda os registros da aplicação, os quais são criados ou acessados dentro da aplicação pelo Mongoose. O Mongoose atua sobre a plataforma Node.js dentro do servidor, assim como o Express e o Angular. Parte das funcionalidades do Angular estão no lado cliente para fornecer dados e recursos nas telas da aplicação.



Por fim o Google Maps API fornece os mapas no lado cliente.

2.1 JSON

JSON é um formato baseada em JavaScript para comunicação de dados. Apesar de ser baseado na estrutura de objetos JavaScript contém uma estrutura de linguagem independente e pode ser usada para transferir dados entre diferentes linguagens de programação. Um JSON é essencialmente uma coleção de pares de chave e valor onde todas as chaves devem ser envolvidas por aspas duplas. Os tipos de dados aceitos como valores são *String*, *Number*, *Array*, *Object*, *Boolean* e *Null (HOLMES, 2013)*.

A listagem 1 exemplifica o formato JSON .

```
{
  "nome": "João",
  "endereco": "",
  "idade": 40,
  "casado": true,
  "cidade": null,
  "filhos": [{
    "nome": "Pedro"
     }, {
    "nome": "Maria"
  }]

}
```

Listagem 1 - Exemplo JSON

2.2 MONGODB

O MongoDB (MONGODB, 2017) é um sistema gerenciador de banco de dados orientado a documentos de código aberto, designado para aplicações *web*. Um registro no MongoDB é um documento, similar ao JSON, composto por um par de chave valor. Os tipos que os campos podem receber são *Strings*, *Number*, *Date*, *Array*, *Document*, *Document Array*, entre outros. Pelo fato do MongoDB usar documentos embutidos dentro de outros documentos, é possível representar em apenas um documento JSON o que no modelo relacional estaria distribuído em diversas tabelas, tirando a necessidade de relacionamentos custosos entre registros, e reduzindo a atividade de leitura e escrita no banco de dados.

Todo documento tem obrigatoriamente um campo chamado _id que funciona como chave primária e sempre vem em primeiro na ordem dos campos do documento. Os documentos são armazenados em formato BSON, que é uma representação binária de documentos JSON, com limite de 16 *megabytes*. Com esse limite se assegura que o documento não consumirá em excesso a *Random Access Memory (*memória RAM) ou largura de banda em uma transmissão.

Cada banco de dados MongoDB pode ter múltiplas coleções (*collections*), que são o equivalente às tabelas no modelo relacional. Pelo fato do MongoDB usar esquemas dinâmicos

cada documento dentro de uma coleção pode ter diferentes campos e ainda assim manter uma função similar ou relacionada.

É possível filtrar os documentos, especificando as condições que determinam quais registros devem ser selecionados para leitura, atualização e remoção. A sintaxe de filtro segue o padrão JSON de chave e valor, com o nome do campo e o valor que se deseja filtrar. O MongoDB contém uma interface em JavaScript chamada *Mongo Shell* que permite fazer consultas e atualizações, assim como efetuar operações administrativas. Com essa interface JavaScript também é possível escrever *scripts* complexos, executando os arquivos com a extensão "*.js*" diretamente no MongoDB.

Outro importante recurso do MongoDB são os índices. Índices ajudam na execução eficiente de consultas. Sem eles o MongoDB realiza uma consulta em todos os registros do banco de dados para selecionar os documentos que se ajustam a declaração da consulta. Se o índice apropriado existe para a consulta, ele é usado para limitar o número de documentos que serão avaliados para a seleção. Além do índice _id que previne inserir dois documentos com o mesmo identificador podem ser criados índices para um único campo ou para uma combinação de campos.

As operações básicas de *create, read, update e delete* (CRUD) são feitas através da notação de JSON através do *Shell*.

O MongoDB fornece os seguintes métodos para inserção de documentos em uma coleção: *db.collection.insert()*; *db.collection.insertOne(); db.collection.insertMany()*. Todos esses métodos tem como alvo apenas uma coleção, e tem a característica da atomicidade apenas quando se insere um simples documento. A listagem 2 representa a inserção de um documento com o método *insert()*.

```
db.users.insert({
    name: "José",
    age: 26,
    status: "A"
});
```

Listagem 2 - Exemplo do uso do método db.collection.insert()

As operações de leitura servem para selecionar os documentos desejados das coleções. O MongoDB fornece o método *db.collection.find()* para fazer a seleção de documentos de uma coleção. É possível especificar filtros de consulta que identificam os documentos que serão retornados. Na listagem 3 está um exemplo do uso do método *find()*, o conteúdo entre as duas chaves na linha dois é filtro de consulta, e é utilizado como critério de seleção a idade ser maior do que 18 usando o operador de comparação $gt (*greater than*). Na linha três é defini-

do a projeção dos campos que estarão no resultado da consulta. Por fim é usado a função *limit()* para limitar o número de resultados.

```
db.users.find({
    { age: { $gt: 18 } },
    { name: 1, address: 1 }
}).limit(5);
```

Listagem 3 - Exemplo do uso do método db.collection.find()

Para modificar documentos em uma coleção os métodos disponíveis são: *db.collection.update(); db.collection.updateOne(); db.collection.updateMany(); db.collection.replaceOne()*. Por padrão as operações para atualização de documentos tem efeito sobre apenas uma coleção e é possível especificar filtros para identificar os documentos que serão atualizados. A listagem 4 exemplifica o uso do método *update()*. Na linha dois temos o filtro de seleção, na linha três aparece o operador $set, que é usado para definir o valor dos campos determinados. Para construir consultas mais complexas existe uma variedade de operadores, que servem para buscar por diferentes tipos de condições. Caso não fosse usado o operador $set, o documento seria sobreposto e teria apenas o campo *status*. A opção *multi* serve para que o MongoDB atualize múltiplos documentos.

```
db.users.update({
    { age: { $gt: 18 } },
    { $set: { status: “A” } },
    { multi: true }
});
```

Listagem 4 - Exemplo do uso do método db.collection.update()

As operações de remoção de documentos são feitas pelos métodos: *db.collection.remove(); db.collection.deleteOne(), db.collection.deleteMany()*. Na listagem 5 está um exemplo de utilização da função *remove()*.

```
db.users.remove({
    status: “D”
});
```

Listagem 5 - Exemplo do uso do método db.collection.remove()

2.3 NODE.JS

Node.js (NODE.JS, 2017) é uma plataforma construída sobre o *V8 JavaScript Engine*, uma máquina virtual desenvolvida pela Google que a possibilita a compilação do código JavaScript para código de máquina nativo e serve para construir aplicações *server side* usando a orientação a eventos e modelo de *input/output* (I/O) não bloqueante. O Node.js proporciona desenvolver tanto *server side* quanto o *client side* com a linguagem JavaScript.

A orientação a eventos é um tipo de programação em que fluxo da execução é determinado por eventos. Os eventos são tratados por *manipulador de eventos, chamado handlers,* ou por funções *callback*. Um *callback* é uma função que é invocada quando algo significante acontece, como quando os resultados de uma consulta ao banco de dados está pronta, ou quando o usuário faz um clique. Este tipo de programação tem como característica a não interrupção do fluxo do sistema quando estiver ocorrendo processos de entrada ou saída, os chamados *I/O*. As operações de *I/O* podem ocorrer em paralelo, e quando isso acontece cada processo executará o respectivo *callback* quando finalizar (HAVIV, 2014).

A programação orientada a eventos no Nodes é acompanhada por um *event loop* que realiza duas funções principais em um ciclo contínuo: detecção de eventos e acionamento dos *event handlers*. Em qualquer momento dentro do *loop*, ele deve detectar quais eventos acabaram de acontecer, e determinar qual função *callback* será invocada quando a resposta estiver pronta (CANTELON, 2014).

2.4 GOOGLE MAPS JAVASCRIPT API

O Google Maps (GOOGLE MAPS APIS, 2017) foi originalmente desenvolvida pela Where 2 Technologies, uma empresa dedicada a criação de soluções em mapeamento, e posteriormente foi adquirida pelo Google. A API possibilita a criação e visualização de mapas, pontos, rotas, entre outras funcionalidades. Os mapas são construídos utilizando *HyperText Markup Language* (HTML), *Cascading Style Sheets* (CSS) e JavaScript, e com auxílio de chamadas *Asynchronous JavaScript and XML* (AJAX) acontece o carregamento das imagens que vão compor novas partes do mapa. As imagens são inseridas em uma sessão dentro da página HTML e a medida que se navega pelo mapa, a API envia mais informações sobre as novas coordenadas através das chamadas AJAX, carregando novas imagens e arquivos (SVENNBERG, 2010).

A API consiste basicamente de arquivos JavaScript que contém classes com métodos e propriedades que são usadas para definir o comportamento do mapa. O Google Maps usa o sistema de coordenadas *World Geodetic System 84* (WGS84) para expressar as localizações exatas, usando os valores de latitude e longitude medidas em graus.

Além da parte visual dos mapas a API tem outros recursos como sugestões de endereço ao usuário enquanto estiver ocorrendo uma entrada de texto, classes especiais que fazem a conversão de pontos geográficos em um endereço específico ou vice-versa, e a possibilidade de desenhar formas geométricas sobre o mapa para delimitar áreas.

2.5 EXPRESS

O Express *(*EXPRESS*, 2017)* é um *framework* JavaScript desenvolvido por TJ Holowaychuk que atua sobre a plataforma do Node.js com recursos de um servidor *web*. Ele fornece um conjunto de recursos comuns para aplicações *web*, como manuseio de requisições e respostas, inclusão de *templates* modular e criação de um sistema de roteamento (HAVIV, 2014).

O Express apresenta 3 objetos principais (*application, request e response*), cada um contendo alguns métodos que adicionam funcionalidades de acordo com seu papel.

O objeto *application* é usado para configurar a aplicação, definindo variáveis globais de configuração, tipo do *template* e quais extensões de arquivos serão renderizadas, criação do *middleware* para manusear requisições e respostas feitas a um determinado *path*.

O objeto *request* fornece métodos com informações auxiliares sobre as requisições HTTP feitas, como as *query strings*, os parâmetros e corpo das requisições (*request body*) e o *path* da requisição.

O objeto *response* é usado para construir as respostas as requisições feitas ao aplicativo. Com esse objeto é possível enviar uma resposta contendo HTML, código de *status*, cabeçalho HTTP e fazer *redirects*. Na listagem 6 está o código para a criação do *middleware* para as requisições ao *path* raiz.

```
var Express = require('Express');
var app = Express();
app.use('/', function(req, res) {
    res.send('Hello World');
});
app.listen(3000);
console.log('Server running at http://localhost:9000/');
module.exports = app;
```

Listagem 6 - Criação de middleware Express para roteamento

2.6 ANGULARJS

O AngularJS (ANGULARJS, 2017) é um framework JavaScript *frontend* designado para construir aplicações *web* usando a arquitetura *Model-view-controller* (MVC). Esse *framework* estende as funcionalidades do HTML utilizando atributos especiais para vincular a lógica de negócios do JavaScript com os elementos HTML. Isso permite a manipulação do *layout* no lado cliente e uma comunicação bidirecional que sincroniza os dados entre o *model* e a *view (HAVIV, 2014).*

O módulo central no AngularJS é carregado com objetos que contém um conjunto de métodos que permitem as operações básicas da aplicação. As funcionalidades do AngularJS que não fazem parte no núcleo são separadas por módulos externos e podem ser adicionadas à medida que se torna necessário.

O conceito módulos é muito importante no AngularJS pois a aplicação deve estar contida dentro de um módulo, ou ainda dividida em vários módulos. Os módulos AngularJS são criados com a função angular.*module(nomeDoModulo,[dependencias], [config])*. O primeiro parâmetro é o nome do módulo criado, o segundo é um *array* com o nome de outros módulos que servirão de dependências, e o terceiro parâmetro é uma função que atua quando o modulo é registrado.

A comunicação bidirecional usada no AngularJS faz com que a aplicação mantenha os dados do *model* sempre sincronizado com o que aparece *view*. Quando um evento acontece na view automaticamente os dados são atualizados no *model* e qualquer alteração nos dados do *model* imediatamente são propagada para a *view*.

As diretivas do AngularJS são úteis para definir as funcionalidades da aplicação. Uma diretiva é uma extensão dos elementos HTML que geram novos comportamentos e são colocadas como um atributo ou como o nome de elementos. Como exemplo se pode citar a diretiva *ng-app*, definindo qual o elemento raiz da aplicação, a diretiva *ng-controller*, para definir qual *controller* deve ser usada para gerenciar a *view*, a *ng-model* que faz a ligação entre o valor mostrado na *view* e o que está registrado no *model*, o *ng-repeat* usado para fazer iterações sobre uma coleção, entre outras. Também é possível definir diretivas personalizadas, com funcionalidades definidas manualmente.

2.7 MONGOOSE

O Mongoose (MONGOOSE, 2017) é um módulo do Node.js que oferece o recurso de modelar dados e persistir como um documento no MongoDB. Pelo fato do MongoDB ser um banco de dados que não tem obrigatoriedade de seguir a mesma estrutura em cada documento, o Mongoose pode criar documentos que serão padronizadas, definindo uma lista de propriedades e restrições (*constraints*) para todos os documentos de uma coleção. O Mongoose usa esquemas para modelar as entidades, oferece validação pré-definida, validações personalizadas e definição de atributos virtuais. A listagem 7 mostra um exemplo de esquema.

```
var mongoose = require('mongoose'),
Schema = mongoose.Schema;
var UserSchema = new Schema({
    firstName: String,
    lastName: String,
    email: String,
    username: String,
    password: String
});
mongoose.model('User', UserSchema);
```

Listagem 7 - Exemplo de esquema do Mongoose

A listagem 8 descreve a utilização de um esquema. Inicialmente é carregado módulo do Mongoose para em seguida chamar o método *model*(), passando como parâmetro o nome do esquema definido, para que ele esteja disponível para a construção de uma nova instância e posterior persistência de um novo registro. O método *save*() tem como função persistir o objeto no banco de dados e recebe um parâmetro que é usado para verificar o processo ocorreu sem erros.

```
var User = require('mongoose').model('User');
exports.create = function(req, res, next) {
    var user = new User(req.body);
    user.save(function(err) {
        if (err) {
            return next(err);
        } else {
            res.json(user);
        }
    });
};
```

Listagem 8 - Exemplo de uso de um esquema do Mongoose

O método *find*() é usado para receber múltiplos documentos armazenados em uma mesma coleção e recebe 4 parâmetros:

1) Um JSON de consulta com a sintaxe idêntica a usada no *Shell* do MongoDB.
2) Projeção de campos que se deseja selecionar.
3) Lista de configurações opcionais.
4) Função *callback que* recebe como parâmetro um valor para verificação de erros e o resultado da consulta com os objetos selecionados.

Um exemplo de utilização do método *find() do* Mongoose é exemplificado pelo código na Listagem 9.

```
exports.list = function(req, res, next) {
    User.find({}, function(err, users) {
        if (err) {
            return next(err);
        } else {
            res.json(users);
        }
    });
};
```

Listagem 9 - Exemplo do método find() do Mongoose

3 TRABALHOS RELACIONADOS

Nesse capítulo serão expostos os projetos que tem as mesmas propostas e características do presente trabalho, isso é, registro dos pontos geográficos de criadouro do mosquito e exibição desses pontos em um mapa.

Os trabalhos similares analisados abrangem todo o território nacional para cadastro dos focos do mosquito enquanto o presente trabalho tem a intenção de mapear os focos de mosquito dentro do município de Pato Branco. O desenvolvimento de um aplicativo específico para o município visa reforçar a participação da população Pato Branquense no combate ao mosquito, devido baixo número de cadastros nos aplicativos que abrangem todo o território nacional.

O presente trabalho difere também em relação aos demais por ser de código aberto, possibilitando a alteração e melhoras no código fonte da aplicação por diferentes desenvolvedores que tiverem interesse no projeto.

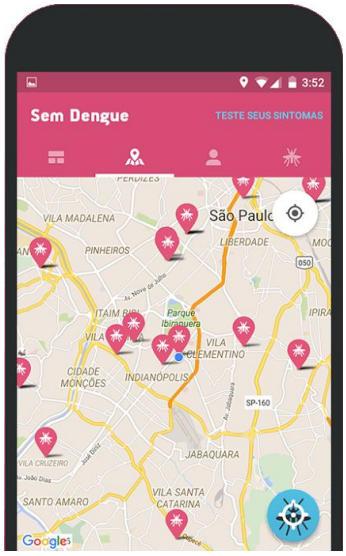
3.1 CAÇA MOSQUITO

O aplicativo Caça Mosquito foi desenvolvido em parceria com a Companhia de Processamento de Dados do Estado da Bahia (Prodeb) para a plataforma Android e tem o objetivo de mapear zonas com focos do mosquito Aedes Aegypti. O mapeamento é feito por meio de geolocalização, utilizando *Global Positioning System* (GPS) do aparelho celular, e é possível enviar uma fotografia e informações dos locais com possíveis criadouros do mosquito. Essas informações são coletadas e transmitidas para os órgãos municipais, que encaminham agentes de endemias aos locais para a eliminação dos focos. A Figura 2 mostra a localização dos criadouros de mosquito Aedes Aegypti dentro do aplicativo (SECRETARIA DA SAÚDE DA BAHIA, 2016).

Figura 2: Caça Mosquito

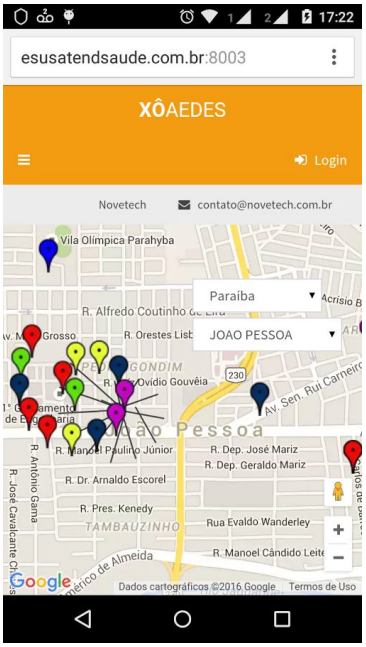
3.2 SEM DENGUE

O aplicativo *Sem Dengue* é uma iniciativa do Colab.re em parceria com diversas prefeituras e órgãos de governo por todo o país. A população pode publicar focos do Aedes Aegypti fotografando o local e confirmar o endereço que o GPS identificou. Todas as publicações são enviadas a uma plataforma acessada pelo setor público, para o controle e as tomadas de ação.O aplicativo foi desenvolvido para as plataformas Android e IOS. A tela de publicações no aplicativo Sem Dengue é mostrada na Figura 3 (BRASIL SEM DENGUE, 2016).



3.3 XÔ AEDES

Desenvolvido pela empresa Novetec, o aplicativo roda na plataforma Android e tem como objetivo informar com a Vigilância Ambiental os locais onde existam possíveis focos de reprodução do mosquito Aedes Aegypti. O usuário tira uma fotografia e faz um comentário sobre o possível foco. O endereço é capturado pelo GPS do aparelho e a ocorrência pode ser acompanhada através do site www.xoaedes.com.br. Os focos cadastrados aparecem na tela descrita na Figura 4 (NOVETECH, 2016).



3.4 OBSERVATÓRIO DO AEDES AEGYPTI

O Observatório do Aedes Aegypti é uma plataforma Online/Móvel que possibilita que a população denuncie a suspeita de focos e casos de suspeita de Dengue, Zika e Chikungunya de forma georreferenciada através de um sistema *web* para o cadastro e um aplicativo móvel que permite que os agentes de endemias verifiquem as denúncias através das rotas de visitas geradas pelo aplicativo. A Figura 5 exibe a tela da parte Online do sistema, onde é possível cadastrar os focos (OBSERVATÓRIO DA DENGUE, 2016).

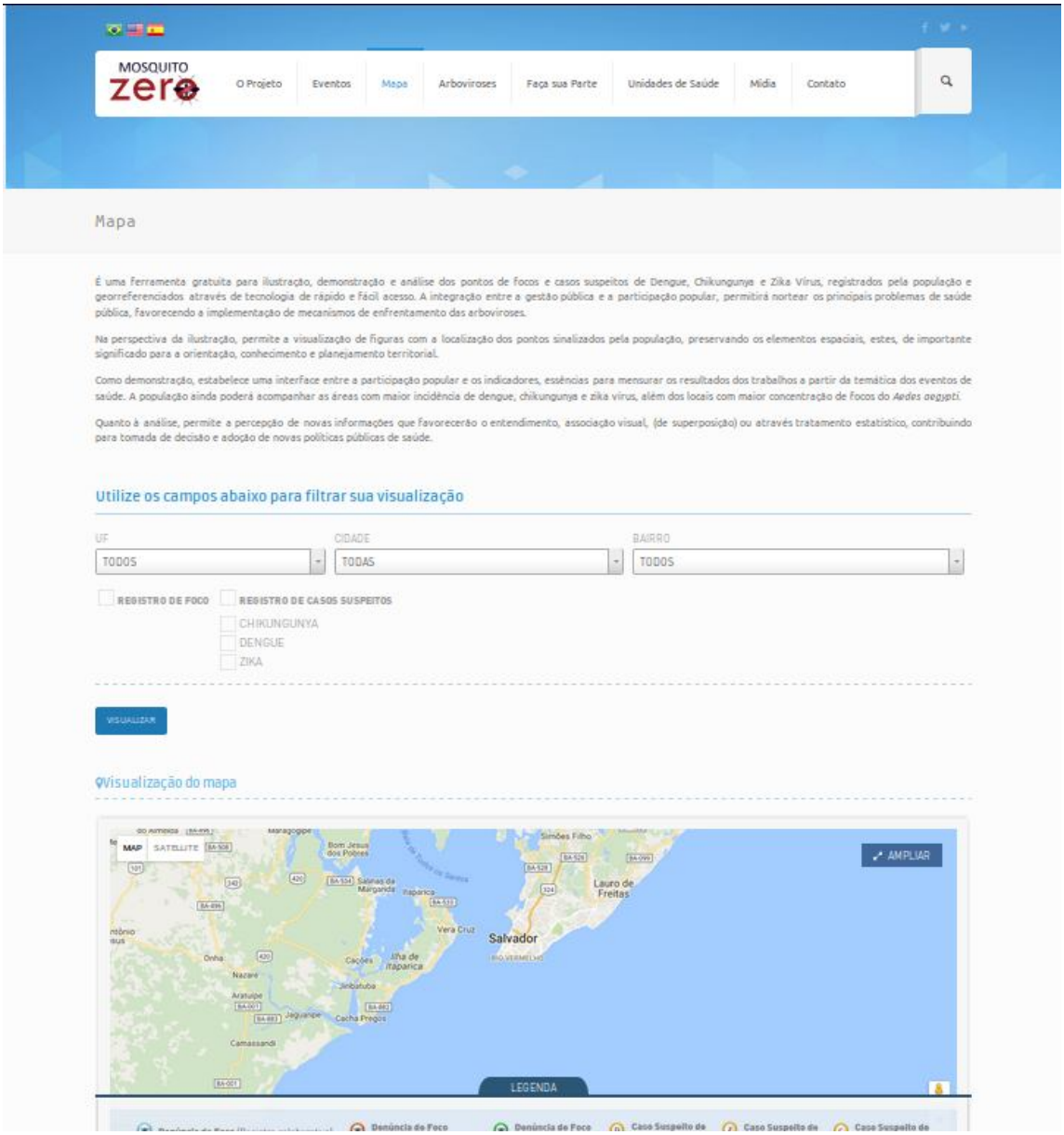


Figura 5: Observatório da Dengue

3.5 MOSQUITO ZERO

Mosquito Zero é um sistema desenvolvido pelo Núcleo de Tecnologia da Informação da Secretaria Municipal da Saúde de Salvador para as plataformas Android e IOS, além da existência de um portal. Em 2014, o aplicativo foi vencedor do concurso Ideias Inovadoras da FAPESB e recebeu recursos do Ministério da Saúde.

Através do aplicativo e portal é possível denunciar os focos do *Aedes Aegypti*, enviado as coordenadas geográficas, foto e dados do local, para a uma Central de Monitoramento, onde as informações passam por triagem e são encaminhadas aos órgãos competentes para as devidas resoluções. É possível também identificar às condições de saneamento básico da própria residência e consultar as unidades de saúde mais próximas através de tecnologia de georreferenciamento. A tela de cadastro de um novo foco é demonstrada na Figura 6 (MOSQUITO ZERO, 2016).

3.6 VOCÊ NA RPC

Você na RPC é um aplicativo para Android e IOS da Rede Paranaense de Comunicação. Além das funcionalidades referentes a programação da rede de televisão, é possível acessar a seção “Mapa do mosquito” para incluir alerta de focos de reprodução do Aedes Aegypti no mapa. Na Figura 7 é possível visualizar as opções de busca e cadastro de alerta de focos (RPC, 2016).

O Quadro 1 abaixo traz um resumo das características de cada aplicativo para efeito de comparação.

Aplicativo	**Desktop**	**Mobile**	**Atua em conjunto com órgãos públicos**	**Regiões de abrangência**
Caça Mosquito	Não	Sim	Sim	Todo o país
Sem Dengue	Não	Sim	Sim	Todo o país
Xô Aedes	Não	Sim	Sim	Todo o país
Observatório do Aedes Aegypti	Sim	Sim	Sim	Todo o país
Mosquito Zero	Sim	Sim	Sim	Todo o país
Você na RPC	Não	Sim	Sim	Todo o país
Mapa Aedes Pato Branco	Sim	Não	Sim	Município de Pato Branco

Quadro 1: Comparação entre aplicativos

4 RESULTADO

4.1 APRESENTAÇÃO DO SISTEMA

O Mapa Aedes Pato Branco é um sistema de mapeamento de focos do criadouro do mosquito Aedes Aegypti dentro da área do município de Pato Branco. É um sistema colaborativo em que a população terá ampla participação, cadastrando os pontos com as suspeitas de foco. O sistema é de código aberto e está disponível para download e alterações no GitHub atraveś do link https://github.com/jeanbuzzello/MAPA-AEDES-PATO-BRANCO.

O cadastro dos focos acontece selecionando um ponto no mapa, ou digitando o endereço específico do local. Para completar o cadastro é necessário ainda fazer uma breve descrição do local, fotografar e enviar uma foto e completar o formulário para que os dados sejam gravados no banco de dados.

Cada foco cadastrado é exibido por um marcador dentro de um mapa. Esse marcador tem como recurso a exibição das informações ao se fazer o clique com o botão direito do mouse sobre ele. As informações exibidas são o endereço do local, o comentário que foi feito e a imagem cadastrada.

Com os locais exatos dos possíveis focos, os agentes de endemias podem deslocar-se até eles e tomarem as medidas para eliminar os focos. À medida que os focos vão sendo cadastrados pode-se ter uma noção real dos locais do município com mais casos, e através de um estudo mais aprofundado desses locais, entender os motivos que facilitam a criação de focos de mosquito.

4.2 MODELAGEM DO SISTEMA

O diagrama de regras de negócio descrito na figura 8 representa o fluxo básico da aplicação na criação de um foco no mapa.

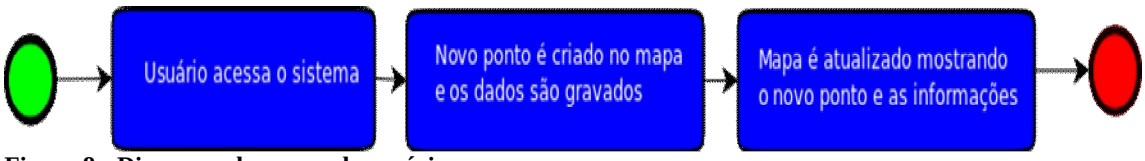


Figura 8 - Diagrama de regras de negócio

Requisitos funcionais:

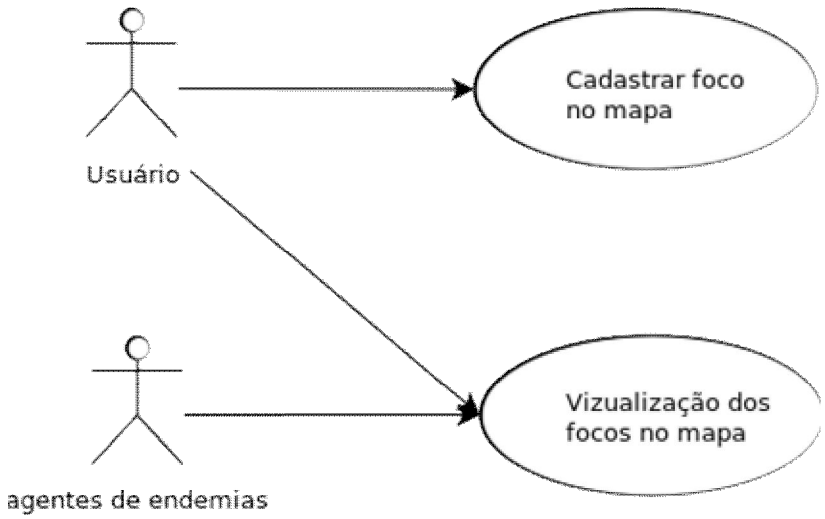
A) O sistema deve permitir o cadastro de um ponto em uma mapa, com informações adicionais de endereço, comentário e o *upload* de uma imagem.

B) Os pontos adicionados devem ser visualizados no mapa junto com as informações relativas ao mesmo.

Requisitos não funcionais:

A) O sistema deve ter layout HTML responsivo e funcionar nos navegadores Chrome, Firefox.
B) Os pontos cadastrados no mapa devem ser salvos no banco de dados com informações de latitude e longitude.
C) A entrada de texto do formulário para receber o endereço deve ter o recurso de auto completar.
D) O formulário só pode ser enviado quando todos os campos estiverem completos.
E) Ao clicar sobre o mapa, o sistema deve completar o campo com o endereço referente a localização geográfica.

A Figura 9 representa os casos de uso de acordo com os requisitos funcionais definidos.



Na Listagem 10 está descrito o esquema do Mongoose usado para definir a estrutura dos documentos persistidos no banco de dados para representar os focos de criadouros do mosquito cadastrados no sistema. O uso de esquemas se torna necessário pois por padrão o

MongoDB tem esquema dinâmico de dados, isto é, documentos de uma mesma coleção podem variar de tipo e quantidade de atributos, o que pode ocasionar erros no fluxo do sistema.

Cada uma das chaves do esquema define uma propriedade do documento e será convertida para o tipo definido na opção *type* contida dentro do objeto de configuração, que também define a obrigatoriedade da existência de um valor para a propriedade quando for criado um novo documento. As propriedades comentário (descrição do local), endereço e *file (*nome do arquivo de imagem registrada do local*)* serão convertidas para o tipo *String* e a propriedade *latlong será convertida para o tipo Array* e conterá elementos do tipo *Number* para representar as coordenadas geográficas de latitude e longitude.

As propriedades declaradas no esquema serão usadas para construir os documentos através da classe *mongoose.model()*, que recebe como parâmetros o nome da coleção do documento e o esquema definido. As instâncias da classe *model* do Mongoose representarão os documentos da coleção *point*, e podem ser usadas para armazenar ou receber os documentos do banco de dados com o auxilio de funções de *CRUD*.

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;

var pointSchema = new Schema({
        latlong : {type: [Number], require: true},
        comentario: {type: String, require: true},
        endereco : {type: String, required: true},
        file: {type: String, required: true}
});

mongoose.model('point', pointSchema);
```

Listagem 10 - Esquema pointSchema do Mongoose

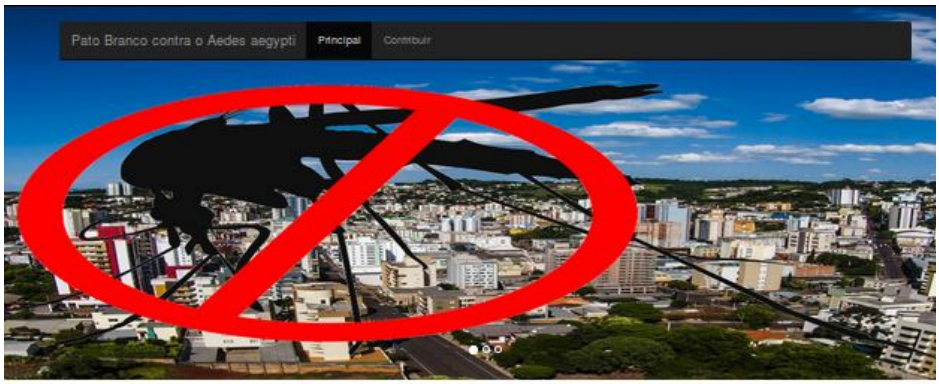
A Listagem 11 é a forma como o registro é armazenado no banco de dados. Mesmo que a propriedade _id não tenha sido representada no esquema, o Mongoose cria automaticamente e permite o acesso desta propriedade nos documentos recebidos em uma consulta.

```
{
    “id” : ObjectId(“584ec97bc17ade312d893ded”),
    “latlong” : [
        -52.706,
        -26.092
    ],
    “comentario” : “Exemplo de comentário”,
    “endereco” : “Unnamed Road, Pato Branco – PR, brasil”,
    “file”: “image.png”
}
```

Listagem 11 - Representação de um documento point

4.3 DESCRIÇÃO DO SISTEMA

A página inicial, representada na Figura 10, contém alguns textos breves para auxiliar o usuário na navegação. Abaixo do topo da página existe um botão verde que direcionará o usuário para a página que contém o formulário de cadastro de um novo foco do mosquito. Em seguida existe um mapa onde é possível visualizar todos os focos cadastrados até o momento. A região do município é delimitada por uma linha vermelha para melhor visualização.



Ao clicar sobre um foco cadastrado é aberto um *pop-up* com as informações referentes ao foco naquele ponto. O *pop-up* contém o endereço, um comentário da situação e uma imagem do

local. A imagem tem o recurso de *zoom* ao se fazer clique sobre ela. Essa funcionalidade é visualizada na Figura 11.

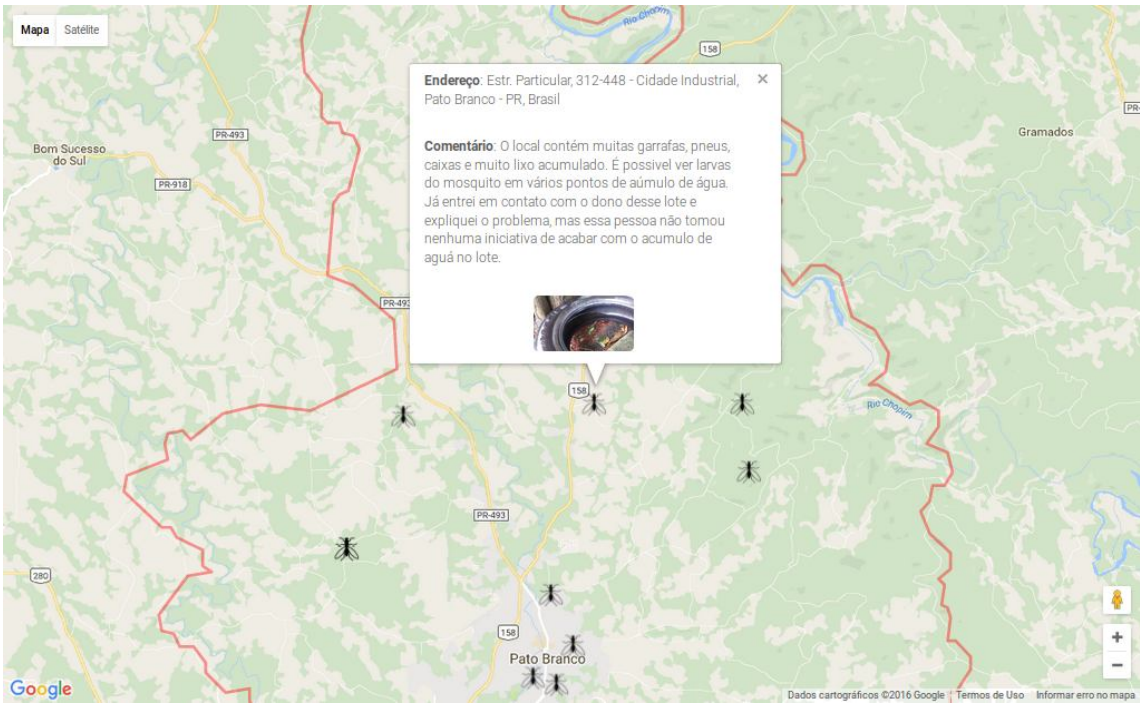


Figura 11: Recurso de pop-up

A tela de cadastro de um novo foco do mosquito (Figura 12) tem um formulário com os campos endereço, comentário e imagem. Existe uma validação para evitar que o formulário seja enviado antes de todos os campos estarem completos.

Há duas formas de selecionar o endereço, clicando sobre alguma região do mapa ou escrevendo diretamente no campo. Quando se faz um clique no mapa, usa-se o recurso *Geo-coder* do Google Maps para obter o endereço do ponto geográfico exato e colocá-lo dentro da caixa de texto. Ao escrever diretamente no campo o Google Maps oferece o recurso *Search-Box* para fornecer sugestões de consulta baseado na entrada de texto.

A área em que se pode adicionar um novo ponto no mapa esta restrita ao município de Pato Branco. Um clique fora da área delimitada não adicionará nenhum valor de endereço no formulário, impedindo assim que ocorra a submissão e a gravação de registros que não estejam dentro do município.

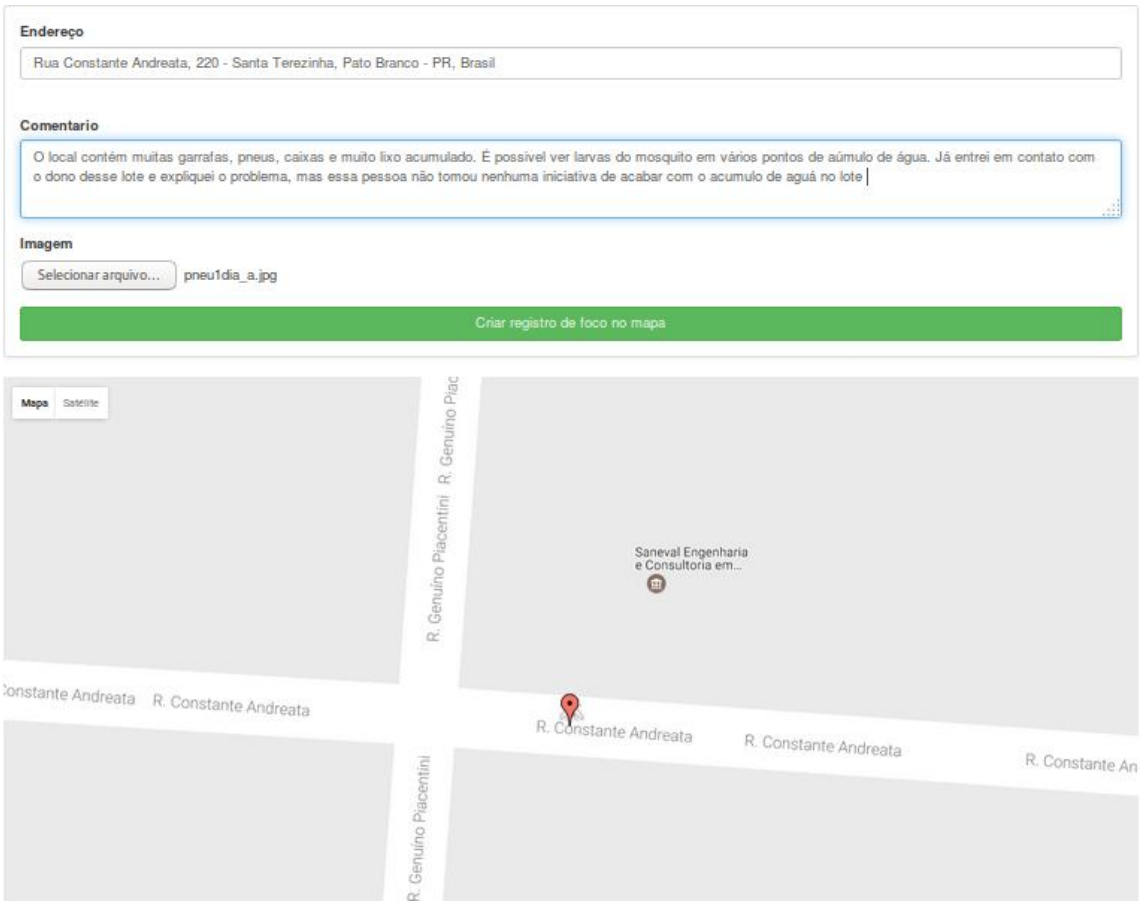


Figura 12: Tela de cadastro

4.4 IMPLEMENTAÇÃO DO SISTEMA

A implementação do sistema começa na criação do servidor Express. É feita a importação dos módulos que serão usados, como o Express e o Mongoose. Em seguida é feita a conexão com o banco de dados e definido o diretório de arquivos estáticos que será útil para que as páginas *html* carregarem *scripts* e imagens.

O Express permite fazer chamadas de funções *middleware* através da função *app.use*(). Funções *middleware* são aquelas que tem acesso ao objeto de requisição e resposta HTTP, e podem executar algum código para alterar algum desses objetos. Como exemplo se pode citar o trecho *app.use(Express.static(__dirname + '/public'))* que define que os arquivos estáticos que serão servidos nas respostas do servidor estão dentro da pasta /*public*.

Outro *middleware* usado é o *body-parser*. Ele extrai o corpo das requisições e cria um objeto *JSON* a partir dos valores dos campos enviados de um formulário. Usando o objeto *req.body* pode-se ter acesso a todos os valores. O *body-parser* tem algumas funções de configuração, entre elas:

- *bodyParser.json*() faz o parse de requisições em JSON.
- *bodyParser.urlencoded*() faz parse dados codificados para URL (*url encode*).
- *bodyParser.text*() permite ao *bodyParser* buscar por texto puro **.**

O servidor é iniciado através da linha de comando, chamando o serviço do Node.js e em seguida o nome do arquivo inicial do projeto, como descrito na listagem 12.

```
nodejs index.js
```

Listagem 12 - Inicialização servidor Node.js

A função *app.listen*() do Express é usada para determinar uma porta para a aplicação e esperar por requisições. No caso dessa aplicação ela foi definida com a porta 9000. O código do servidor Express é demonstrado na listagem 13.

```
var Express = require('Express');
var mongoose = require('mongoose');
var bodyParser = require('body-parser');
var app = Express();

mongoose.connect('mongodb://localhost/focosDeDengue');

app.use(Express.static(__dirname + '/public'));
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({extended: true}));
app.use(bodyParser.text());
app.use(bodyParser.json({type: 'application/vnd.api+json'}));

app.set('views', __dirname + '/public');

require('./app/route.js')(app)

app.listen(9000);
```

Listagem 13 - Servidor Express

O Express também se encarrega de definir as rotas na aplicação. As rotas mais importantes são */get-all-points* e */save-point,* descritas na listagem 14.

A rota /*get-all-points* usa o método HTTP *GET para* retornar um *array* com todos os documentos da coleção *points* através de uma consulta no banco de dados com o método *find*() do modelo definido. O método *exec()* executa a consulta e tem como parâmetro uma função de *callback*. Caso não tenha ocorrido nenhum erro, o *array* com os resultados é retornado como um JSON.

A rota /*save-point* recebe requisições do tipo *POST* e é utilizada para salvar um novo registro no banco de dados. Usando o esquema definido no Mongoose e o corpo da requisição *POST* com os dados enviados, é criada uma instância do modelo *point*. A função *save()* persiste o registro no banco de dados. Se não ocorreu nenhum erro o registro que acabou de ser salvo é retornado.

```
var Point = require('./mapPointModel.js');

app.get('/get-all-points', function(req, res){
    var q = Point.find({});
    q.exec(function(err, points){
        if(err){
            res.send(err);
        }
        res.json(points)
    });
});

app.post('/save-point', function(req, res){
    var point = new Point(req.body);
    point.save(function(err){
        if(err){
            res.send(err);
        }
        res.json(req.body);
});

app.get('/form', function (req, res)
{
       res.render('form.html');
});
```

Listagem 14 - Rotas importantes da aplicação

A rota */form* serve para renderizar a página HTML que contém o formulário e o mapa. Essa página HTML está relacionada a um módulo AngularJS primário, que fará a ligação com outros módulos e serviços para fornecer as regras de negócio. Para criar esse relacionamento da *view* com o módulo primário foi usado a diretiva *ng-app (L*istagem 15*)* em um elemento *<html>*, para determinar que o AngularJS deve usar ele como elemento raiz da aplicação. O valor da diretiva *ng-app* é o nome definido na criação do modulo AngularJS, sendo esse modulo criado com a função angular.*module()*.

```
<html class=”no-js” ng-app=”aedesApp”>
```

Listagem 15 - Uso da diretiva ng-app

No arquivo JavaScript onde está o módulo primário (Listagem 16) é definido quais serviços e módulos estarão disponíveis para fornecer as funcionalidades do sistema. O módulo que conterá os métodos necessários para a criação do mapa e submissão do formulário foi definido como *mapController*.

```
var app = angular.module(‘aedesApp’, [‘mapController’]);
```

Listagem 16 - Modulo primário AngularJS

No *mapController*, é registrado um *listener* da biblioteca Google Maps (Listagem 17) que fará uma chamada ao método atualizar() assim que o carregamento da janela estiver acontecendo. Essa função tem como objetivo instanciar e adicionar todos os pontos salvos no mapa. A função *atualizar*() recebe as coordenadas para centralizar o mapa em uma região.

```
google.maps.event.addDomListener(window, ‘load’, atualizar(-26.173, -52.663));
```

Listagem 17 - Listener biblioteca Google Maps

4.4.1 Função Atualizar

A função atualizar tem partes de código importantes para o fluxo do sistema, por isso será descrita separadamente. Logo no início do método é feita uma requisição GET para a rota */get-all-points* para pegar todos os pontos que estão no banco de dados. Um *array* é definido para estruturar os pontos que serão adicionados ao mapa. O restante da função atualizar estará dentro do *successCallback* da requisição, por este ser o escopo dos documentos retornados do banco de dados. Caso a consulta tenho tido algum problema, o *errorCalback* será executado. Esse processo é descrito na Listagem 18.

```
$http({
    method: 'GET',
    url: '/get-all-points'
}).then(function successCallback(response) {
    ...
}, function errorCallback(response) {
    console.error(response);
});
```

Listagem 18 - Requisição de todos os pontos do mapa

Caso a consulta tenha tido êxito é realizada uma iteração no *array* de documentos listagem 19. Cada registro será usado para criar um objeto com as informações necessárias para construir os pontos com o Google Maps API. As coordenadas geográficas dos documentos serão passadas como parâmetros para a construção de um objeto *LatLng*, que é um ponto de coordenadas geográficas da API do Google Maps.

Para a funcionalidade de exibição de informações em uma janela, quando houver o clique nos pontos, é usado o objeto *InfoWindow* do Google Maps. O *infoWindow* recebe como parâmetro uma *string* contendo uma estrutura HTML com texto e imagem proveniente do documento.

```
for(var i = 0; i < response.data.length; i++) {
    var point = response.data[i];
    var  contentString ='<p><b>Endereço</b>: ' + point.endereco ...
    locations.push({
        latlon: new google.maps.LatLng(point.latlong[1], point.latlong[0]),
        message: new google.maps.InfoWindow({
        content: contentString,
        maxWidth: 320
    }),
});
}
```

Listagem 19 - Conversão dos registros do MongoDB para pontos do Google Maps

Em seguida é feita uma verificação analisando se o objeto *Map* do Google Maps já foi instanciado. O objeto *Map* cria um novo mapa dentro de uma <*div*> HTML e recebe como parâmetro a própria <*div*>. Também é necessário definir algumas opções, como nível de *zoom* e o centro do mapa, de acordo com as coordenadas recebidas. A Listagem 20 descreve esse processo.

```
if (!mapa){
    var mapa = new google.maps.Map(document.getElementById('mapa'), {
        zoom: 11,
        center: {lat: parseFloat(latitude), lng: parseFloat(longitude)}
    });
}
```

Listagem 20 - Instanciação do mapa da biblioteca Google Maps

Com o mapa instanciado é necessário fazer uma iteração no *array* com objetos que foram criados a partir dos registros do banco de dados (Listagem 21). Os pontos no mapa serão criados com o objeto *Marker* do Google Maps. Para instanciar o *Marker* é necessário passar como parâmetro um objeto *LatLng* contendo a posição geográfica, o mapa que está sendo inserido o ponto e o ícone usado para exibir o ponto. Cada um dos objetos *Marker* terão um *listener* no evento de clique para que seja mostrado as informações do ponto.

```
locations.forEach(function(n, i){
    var marker = new google.maps.Marker({
        position: n.latlon,
        map: mapa,
        icon: "mosquito_aedes.png"
    });
    google.maps.event.addListener(marker, 'click', function(e){
        currentSelectedMarker = n;
        n.message.open(mapa, marker);
    });
});
```

Listagem 21 - Criação dos pontos no mapa

A Listagem 22 exibe o registro de um *listener* para que cada clique no mapa adicione uma ícone de referência. Para manter apenas o último clique com marcações, é feita uma verificação para eliminar ícones anteriores. A função *panTo*() tem como objetivo mudar o centro do mapa para o último ponto geográfico marcado com o ícone.

```
google.maps.event.addListener(mapa, 'click', function(e){
    var marker = new google.maps.Marker({
    position: e.latLng,
    animation: google.maps.Animation.BOUNCE,
    map: mapa,
    icon: 'http://maps.google.com/mapfiles/ms/icons/red-dot.png'
});
if(typeof lastMarker !== 'undefined'){
    lastMarker.setMap(null);
}
lastMarker = marker;
mapa.panTo(marker.position);
```

Listagem 22 - Marcação de referência do clique

5 CONCLUSÃO

O trabalho desenvolvido teve como objetivo ser um protótipo de sistema para auxiliar no combate ao mosquito Aedes Aegypti. Caso ele seja disponibilizado para o acesso público pela internet, necessitará da mobilização dos moradores do município de Pato Branco para atuar do processo de controle das doenças transmitidas, mais especificamente a Dengue, Zika e Chikungunya, cadastrando os focos de criadouro do mosquito no sistema.

A integração de tecnologias que tem como característica o uso da linguagem JavaScript forneceu a base para que o protótipo fosse desenvolvido. A conexão entre essas tecnologias foi de fácil entendimento e codificação, e exemplificou o uso da linguagem JavaScript como bloco de construção principal nas aplicações *web*. Isso se deve em grande parte do uso do Node.js, proporcionando a execução da linguagem JavaScript a partir do lado servidor. O Node.js é uma plataforma que contém um grande número de módulos que adicionam funcionalidades que não existe em seu núcleo base, como o módulo Express usado nessa aplicação para gerenciar toda a parte de roteamento e manipulação das requisições.

O AngularJS se mostrou bastante útil para a criação do padrão MVC, e apesar de a curva de aprendizado ter sido maior do que nas outras tecnologias usadas, ele foi crucial na viabilidade do projeto, fornecendo o recurso de *controllers* para gerenciar a lógica da aplicação.

O MongoDB serviu bem ao propósito do sistema. Salvar os pontos geográficos e informações do mesmo em um formato bastante similar ao usado na codificação, no caso, um objeto JavaScript. A uso do MongoDB foi facilitado pelo módulo Mongoose, que forneceu uma forma de padronizar os documentos e simplificar o processo de consultas e persistência de dados.

Os requisitos da parte visual do mapa foram completamente alcançados com o uso do Google Maps JavaScript API. A API se mostrou precisa, estável e com muitas funcionalidades, apesar de poucas delas terem sido aproveitadas neste protótipo. Como ponto negativo se pode citar o fato da API funcionar apenas quando há conexão com a internet, não sendo possível implementar em sistemas *off-line*.

5.1 TRABALHOS FUTUROS

Embora o trabalho tenha alcançado seu objetivo principal, isso é, mostrar os dados georreferenciados salvos no banco de dados como pontos em um mapa, algumas implementa-

ções futuras são necessárias para permitir seu uso como um sistema *web* e criar uma experiência melhor para o usuário.

Uma das melhorias é a criação de uma área onde os administradores do sistema possam ver estatísticas e dados detalhados, alterar o status de um caso reportado, excluir casos falsos reportados, encerrar casos que já foram solucionados, adicionar informações adicionais, delegar a tarefa de visitar determinados pontos aos agentes de saúde e fazer consultas geoespaciais buscando os focos contidos em determinado perímetro.

Para o controle efetivo de usuários do sistema será preciso criar o cadastro de perfil com informações básicas para que cada usuário acesse apenas os recursos e áreas destinados para aquele tipo de perfil. Um sistema de *logs* para todas as atividades que um usuário exerce dentro do sistema também pode ser um recurso interessante.

Como complemento ao sistema *web* pode-se desenvolver um sistema para dispositivos móveis. Nessa plataforma estarão presentes as funcionalidades já desenvolvidas no sistema *web* com a adição de funcionalidades especiais como o uso da câmera do dispositivo para enviar fotos diretamente para o aplicativo, uso do GPS do dispositivo para exibir os focos mais próximos e a opção de sincronizar posteriormente os dados cadastrados caso não exista sinal de internet.

REFERÊNCIAS

NELSON, Michael. **Aedes Aegypti: Biologia y ecologia**. Organizacion Panamericana de la Salud, Washington, 1986.

MINISTÉRIO PÚBLICO DO ESTADO DO PARANÁ. **Plano Nacional de Controle da Dengue**. Disponível em http://www.saude.mppr.mp.br/modules/conteudo/conteudo.php?conteudo=351. Acesso 19 de dezembro de 2016.

BRAGA, Ilma Aparecida; VALLE, Denise. **Aedes aegypti: histórico de controle no Brasil**. Epidemiologia e Serviços de Saúde, Brasília, n. 2, jun. 2007. Disponível em: <http://scielo.iec.pa.gov.br/scielo.php?script=sci_arttext&pid=S1679-49742007000200006>. Acesso em: 29 dez. 2016.

SECRETARIA DE VIGILÂNCIA EM SAÚDE. **Boletim Epidemiológico**. Disponível em <http://portalsaude.saude.gov.br/images/pdf/2016/outubro/18/2016-029-Dengue-publicacao-n-34.pdf>. Acesso 20 de dezembro de 2016.

FUNDAÇÃO NACIONAL DE SAÚDE. **Guia de Vigilância Epidemiológica**. Disponível em <http://www.funasa.gov.br/site/wp-content/uploads/2010/01/Dengue_%20guia_vig_epid.pdf>. Acesso em 20 de dezembro de 2016.

CONSELHO FEDERAL DE FARMÁCIA. **Pharmacia Brasileira**. fev. 2002. Disponível em <http://www.cff.org.br/sistemas/geral/revista/pdf/81/12.pdf>. Acesso em 21 de dezembro de 2016.

SECRETARIA DE VIGILÂNCIA EM SAÚDE. **Febre de Chikungunya: Manejo clínico**. – Brasília: Ministério da Saúde, 2015.

SECRETARIA DE ATENÇÃO BÁSICA. **Zika - Abordagem clínica na atenção básica**. Disponível em <http://arcarea.fiocruz.br/bitstream/icict/15672/1/Zika%20-%20Abordagem%20cl%C3%Ad-nica%20na%20aten%C3%A7%C3%A3o%20b%C3%A1sica.pdf>. Acesso em 21 de dezembro de 2016.

MINISTÉRIO DA SAÚDE. **Tira Dúvidas: Prevenção e combate Dengue, Chikungunya e Zika**. Disponível em <http://combateaedes.saude.gov.br/pt/tira-duvidas>. Acesso em 21 de dezembro de 2016.

Ações do Governo no Combate. **Blog da Saúde**. Disponível em <http://www.blog.saude.gov.-br/index.php/perguntas-e-respostas/50662-acoes-do-governo-no-combate-ao-aedes>. Acesso em 21 de dezembro de 2016.

CASA CIVIL: Presidência da República. **Estratégia de Resposta ao vírus Zika e o combate ao mosquito transmissor**. Brasília, maio 2016. Disponível em: <http://www.casacivil.gov.-br/.arquivos/estrategia-de-resposta-ao-virus-zika.pdf>
Acesso em 22 dez. 2016.

PORTAL DA SAÚDE – Ministério da Saúde. **Ministério da Saúde atualiza casos suspeitos de febre amarela**. Disponível em: <http://portalsaude.saude.gov.br/%20index.php/cidadao/principal/agencia-saude/27343-atualizacao-dos-casos-suspeitos-de-febre-amarela-17012017>. Acesso em: 20 jan. 2017.

HOLMES, Simon. **Getting MEAN with Mongo, Express, Angular and Node**. Shelter Island, NY: Meanning Publications Co., 2013.

MONGODB. **Página *web* da documentação do MongoDB**. Disponível em https://docs.mongodb.com. Acesso em 12 de janeiro de 2017.

NODE.JS. **Página *web* oficial do Node.js**. Disponível em https://nodejs.org/en/. Acesso em 12 de janeiro de 2017.

HAVIV, Amos Q. **MEAN *Web* Development**. Birmingham: Packt Publishing Ltd., 2014.

CANTELON, MIKE et al. **Node.js in Action**. Shelter Island: Meanning Publications CO, 2014.

GOOGLE MAPS APIS. **Página *web* da documentação do Google Maps**. Disponível em <https://developers.google.com/maps/documentation/javascript/?hl=pt-br>. Acesso em 13 de janeiro de 2017.

SVENNBERG, Gabriel. **Beginning Google Maps API 3**. New York, NY: Springer Science+Business Media, 2010.

EXPRESS. **Página *web* do Express**. Disponível em <http://expressjs.com/pt-br/>. Acesso em 13 de janeiro de 2017.

ANGULARJS. **Página *web* do AngularJS**. Disponível em <https://angularjs.org/>. Acesso em 15 de janeiro de 2017.

MONGOOSE. **Página *web* do Mongoose**. Disponível em <http://mongoosejs.com/>. Acesso em 7 de janeiro de 2017.

SECRETARIA DA SAÚDE DA BAHIA. **Aplicativo Caça Mosquito**. Disponível em <http://www.saude.ba.gov.br>. Acesso em 20 de dezembro de 2016.

BRASIL SEM DENGUE. **Página *web* do aplicativo Sem Sengue**. Disponível em <http://www.brasilsemdengue.org>. Acesso em 17 de dezembro de 2016.

NOVETECH. **Página do desenvolvedor do aplicativo Xô Aedes**. Disponível em <http://www.novetech.com.br>. Acesso em 17 de dezembro de 2016.

OBSERVATÓRIO DA DENGUE. **Página *web* do aplicativo**. Disponível em <http://observatoriodadengue.telessaude.ufrn.br/sobre/>. Acesso em 17 de dezembro de 2016.

MOSQUITO ZERO. **Página *web* do aplicativo**. Disponível em <http://mosquitozero.com.br/>. Acesso em 17 de dezembro de 2016.

RPC. **Página *web* do aplicativo Você na RPC**. Disponível em <http://www.rpc.com.br/vc-narpc/>. Acesso em 17 de dezembro de 2016.