UNIVERSIDADE FEDERAL DO VALE DO SÃO FRANCISCO

CURSO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

GABRIEL RAFAEL GOMES

SISTEMA DE INFORMAÇÃO ONLINE PARA LEITURA E ARMAZENAMENTO DE DADOS METEOROLÓGICOS

JUAZEIRO - BA

2018

GABRIEL RAFAEL GOMES

SISTEMA DE INFORMAÇÃO ONLINE PARA LEITURA E ARMAZENAMENTO DE DADOS METEOROLÓGICOS

Trabalho apresentado à Universidade Federal do Vale do São Francisco - Univasf, Campus Juazeiro, como requisito da obtenção do título de Bacharel em Engenharia de Computação.

Orientador: Prof. M. Sc. Fábio Nelson de Sousa Pereira

JUAZEIRO - BA

2018

UNIVERSIDADE FEDERAL DO VALE DO SÃO FRANCISCO
CURSO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

FOLHA DE APROVAÇÃO

GABRIEL RAFAEL GOMES

SISTEMA DE INFORMAÇÃO ONLINE PARA LEITURA E ARMAZENAMENTO DE DADOS METEOROLÓGICOS

Trabalho apresentado à Universidade Federal do Vale do São Francisco - Univasf, Campus Juazeiro, como requisito da obtenção do título de Bacharel em Engenharia de Computação.

Orientador: Prof. M. Sc. Fábio Nelson de Sousa Pereira

Aprovado em: _____ de _____ de 2017

Banca Examinadora

Fábio Nelson de Sousa Pereira, Mestre, Universidade Federal do Vale do São Francisco

---

Jorge Luis Cavalcanti Ramos, Doutor, Universidade Federal do vale do São Francisco

---

Ricardo Argenton Ramos, Doutor, Universidade Federal do Vale do São Francisco

A minha família...

AGRADECIMENTOS

A minha mãe Josiane Gomes, meu pai Weliton de Carvalho e meus irmãos pelo esforço imensurável para que minha formação se concretizasse.

Ao meu primo Manoel Rafael pelo apoio essencial durante a vida acadêmica.

Aos meus amigos e companheiros de turma pela parceria, colaboração, compartilhamento de estresse e de fortes emoções durante o curso. Especialmente Esron Dtamar, Johnathan Alves, Gustavo Marques e Leonardo Cavalcante.

Aos amigos conquistados no decorrer da caminhada, João Bastos, José Matias, Delmiro Daladier, Daniel Simião, Hallan Ferreira, Victor Silva, Antônio Noronha, Marlon Rocha e o pessoal restante do grupo do "AP"pela partilha da tarimba, casos e contos em mesas de bar.

A Ana Letícia Menezes pelo companheirismo, paciência e pelo abraço caloroso.

A meu tio Antônio Marcos pelo exemplo integridade e competência no aspecto pessoal e profissional.

A meu primo Marco Antônio pela inspiração e minha tia Neiva Gomes por toda educação e amor dados à mim.

Ao meu orientador, professor Fábio Nelson, pelos ensinamentos, dedicação e disposição.

Aos professores Rômulo Câmara e Jairson Rodrigues pelas oportunidades de trabalhos extracurriculares que me enriqueceram muito como pessoa e profissional.

A todos os demais professores que repassaram para mim um pouco de seu conhecimento e experiência, contribuindo de alguma forma para a realização deste trabalho e para minha formação.

A minha madrinha Lúcia Ricarte, Joelma Duarte e Lurdinéia Guimarães por todo carinho e apoio de sempre.

A Miguel Pérez Pasalodos por compartilhar seu conhecimento através de *software open-source* e me aliviar de uma grande dor de cabeça.

Por fim, agradeço a todos aqueles que auxiliaram de alguma maneira a existência e o desenvolvimento de meu ser.

Se pude enxergar a tão grande distância, foi subindo nos ombros de gigantes.

Isaac Newton
Carta à Robert Hooke, 1676

RESUMO

Na região do Submédio do Vale do São Francisco, a organização social Biofábrica Moscamed Brasil, ou simplesmente Moscamed, sediada na cidade de Juazeiro-BA, é responsável pelo controle biológico da mosca-das-frutas e do mosquito-da-dengue. Esta é uma das maiores regiões produtoras de frutas do mundo. Entretanto, diversas culturas são atacadas pela praga da mosca-das-frutas ocasionando perdas consideráveis na produção. Por outro lado, a Moscamed atua também no combate ao mosquito *Aedes Aegypti*, que é vetor de diversas doenças de grande importância quando se diz respeito à saúde pública. A Moscamed emprega um dos métodos mais eficazes no controle de ambos os insetos, a Técnica do Inseto Estéril (TIE). Porém, os ciclos de vida das duas espécies e consequentemente o êxito da técnica citada estão fortemente relacionados com as alterações climáticas do ambiente. Deste modo, visando auxiliar a tomada de decisões no que diz respeito aos processos relacionado à TIE e os demais processos da organização, o presente trabalho tem o intuito de desenvolver um sistema de informação *web*/aplicativo *Android* que fornece um mecanismo de obtenção e armazenamento e uma interface de visualização dos dados das variáveis climáticas provenientes de uma estação meteorológica *Vantage Vue* ^{TM}.

Palavras-chave: *Vantage Vue, Android, Ionic, Weather Link IP*, meteorologia.

ABSTRACT

In the sub-region of the São Francisco Valley, the social organization Biofábrica Moscamed Brasil, or simply Moscamed, based in the city of Juazeiro-BA, is responsible for the biological control of the medfly and the dengue mosquito. This is one of the largest fruit producing regions in the world. However, several crops are attacked by the medfly pests, causing considerable losses in production. On the other hand, the Aedes Aegypti mosquito, which is a vector of several diseases of great importance when it comes to public health. Moscamed employs one of the most effective methods in controlling both insects, the Sterile Insect Technique (TIE). However, the life cycles of both species and consequently the success of the cited technique are strongly related to the environmental changes of the environment. In this way, in order to assist in the decision making process related to TIE and other processes of the organization, the present work aims to develop an information system that provides a retrieval and storage mechanism and a visualization interface of weather data from a Vantage Vue^{TM} weather station. The work was done using web technologies, framework for multiplatform applications and non relational database. The resulting system consists of a web / Android application as the user interface, an application that retrieves weather station information and an API to manage the data.

Key-words: *Vantage Vue, Android, Ionic, Weather Link IP, meteorology.*

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE CÓDIGOS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application programming interface* - tradução: Interface de programação de aplicativos
APP	*Application* - tradução: Aplicação
ART	*Android runtime* - tradução: Tempo de execução *Android*
CECOMP	Colegiado de engenharia de computação
CSS	*Cascading style sheet* - tradução: Tabela de estilos em cascata
GNU	*Gnu's not unix* - tradução: Gnu não é Unix
HAL	*Hardware abstration layer* - tradução: Camada de abstração de *hardware*
HTML	*Hypertext markup language* - tradução: Linguagem de marcação de hipertexto
HTTP	*Hypertext transfer protocol* - tradução: Protocolo de transferência de hipertexto
IDE	*Integrated development environment* - tradução: Ambiente de desenvolvimento integrado
INMET	Instituto nacional de meteorologia
JSON	*Javascript object notation* - tradução: Notação de objeto Javascript
MIP	Manejo integrado de pragas
MIT	*Massachusetts institute of technology* - tradução: Instituto de tecnologia de Massachusetts
REST	*Representational state transfer* - tradução: Transferência de Estado Representacional
SDK	*Software development kit* - tradução: Kit de desenvolvimento de programa
SGBD	Sistema de gerenciamento de banco de dados
TCC	Trabalho de conclusão de curso
TCP	*Transfer control protocol* - tradução: Protocolo de controle de transmissão

TIE	Técnica do inseto estéril
UR	Umidade relativa
URI	*Uniform resource identifier* - tradução: Identificador de recurso uniforme
XML	*Extensible markup language* - tradução: Linguagem de marcação extensível

SUMÁRIO

1 INTRODUÇÃO

A mosca-das-frutas é uma praga causadora de diversos danos à produção de frutas e hortaliças. No Submédio do Vale do São Francisco são encontradas várias espécies desses insetos. A Organização Moscamed Brasil surge com a intenção de realizar a supressão dessa população através da Técnica do Inseto Estéril - TIE, cuja aplicação é adotada em mais de 28 países (MOSCAMED, 2003b).

Além da mosca-das-frutas, a Moscamed também é responsável pelo controle biológico da espécie de mosquito *Aedes Aegypti*, vetor da doença reemergente mais importante do mundo, a dengue. Quando em 2007 cerca de 70% dos Municípios brasileiros estavam infestados pelo mosquito (BRAGA; VALLE, 2007). Neste caso, o controle ocorre de forma análoga ao das moscas, ambos usufruem da TIE.

A técnica criada por E.F.Knipling consiste em produzir machos estéreis e liberá-los na natureza em grande quantidade. As fêmeas da natureza então copulam com os machos estéreis e colocam ovos não fecundados, fazendo com que a próxima geração tenha sua densidade populacional reduzida (PARANHOS, 2008). O êxito da TIE depende do sucesso dos machos estéreis na competição contra os nativos pelo acasalamento com as fêmeas da mosca-das-frutas ou do mosquito-da-dengue, e da consequente postura dos ovos não fecundados.

Contudo, o ciclo de vida das moscas na natureza é fortemente dependente da temperatura ambiente, além de outros fatores climáticos (RAGA; SOUSA FILHO, 2000). A mosca-da-carambola necessita, por exemplo, de vinte e dois dias com clima favorável (26 ºC e 70% UR) para se desenvolver completamente partindo da fase de ovo até a fase adulta (MALAVASI; ZUCCHI, 2000). De mesmo modo, o ciclo de vida dos mosquitos também é fortemente dependente da temperatura e de outros fatores climáticos, seja em seu desenvolvimento ou em sua sobrevivência (HOPP; FOLEY, 2001; RIBEIRO et al., 2006).

Neste contexto, com a finalidade de auxiliar a organização em suas tomadas de decisão, a proposta deste TCC é composta pelo desenvolvimento de um sistema *Web*/aplicativo *Android open-source* para consulta e armazenamento em banco de dados de informações provenientes de uma estação meteorológica sem fio *Vantage Vue* ^{TM}, instalada na Biofábrica Moscamed Brasil.

1.1 JUSTIFICATIVA

Este TCC é um subprojeto do projeto Sistema de Gestão da Produção de Insetos Modificados para Regiões Endêmicas, conjunto entre a Moscamed e o Colegiado de

Engenharia de Computação da Universidade Federal do Vale do São Francisco - CECOMP.

1.2 OBJETIVOS GERAIS

Desenvolver um sistema capaz de informar, via *internet*, dados meteorológicos obtidos de uma estação *Vantage Vue* ^{TM} e disponibilizar para o usuário dados e gráficos através de uma aplicação *Web* e de um aplicativo para o sistema operacional *Android*.

1.3 OBJETIVOS ESPECÍFICOS

- Modelar graficamente a arquitetura do sistema por meio de diagramas de casos de uso, diagrama de atividades, diagrama de sequência e protótipos das telas de interface com o usuário;
- Definir a melhor forma de estruturar os dados provenientes da estação meteorológica;
- Modelar a API responsável por armazenar e disponibilizar os dados;
- Desenvolver a versão inicial do sistema.

1.4 ORGANIZAÇÃO DO TRABALHO

O trabalho é dividido em três seções, fundamentação teórica, metodologia e conclusão e trabalhos futuros.

A fundamentação teórica é organizada em duas subseções, na primeira é explanado sobre a Moscamed, suas responsabilidades e a técnica de controle biológico empregada. Em seguida é estudada a estação meteorológica em questão e a influência das variáveis climáticas nas espécies controladas pela Moscamed. A segunda parte discorre sobre as bases conceituais das tecnologias utilizadas no processo de desenvolvimento, como a pilha de software do sistema operacional *Android*, desenvolvimento *web* e desenvolvimento de aplicações *cross-platform*.

Na metodologia são descritas a metodologia adotada para a concepção do projeto, os requisitos dos sistema e uma visão macroscópica da arquitetura do projeto. Nesse capítulo, ainda é mostrado o conjunto de tecnologias adotadas para a implementação do sistema.

No capítulo de resultados e discussões são apresentados os passos para a concepção e construção do sistema através de diagramas e modelos gráficos dos seus vários elementos, como por exemplo banco de dados e interface de usuário.

O capítulo conclusão e trabalhos futuros são explanados sucintamente os resultados obtidos com a finalização do projeto. Algumas sugestões que podem ser realizadas a posteriori também são indicadas.

2 REFERENCIAL TEÓRICO

2.1 MOSCAMED, A INSTITUIÇÃO E A TÉCNICA

A Biofábrica Moscamed Brasil é uma organização social situada no Vale do São Francisco, mais precisamente em Juazeiro-BA, responsável pelo controle biológico e monitoramento ambientalmente seguro de pragas em culturas de alto interesse econômico, tais como, manga, uva, melão, maçã, papaia, goiaba e acerola (MOSCAMED, 2010a). Bem como o controle do vetor transmissor de doenças mais importante para a saúde pública atualmente, o mosquito *Aedes Aegypt* (MOSCAMED, 2003b; MOSCAMED, 2010a).

A organização possui atividades em larga escala voltadas para a produção e liberação na natureza de insetos estéreis modificados, via raios-X no caso da mosca-das-frutas e via alteração genética no caso do Mosquito-da-dengue. Possui o planejamento de produção/liberação de 200 (duzentos) milhões de machos estéreis por semana (MOSCAMED, 2003b; MOSCAMED, 2003a). Ademais, realiza capacitação, treinamento e disseminação de informações técnico-científicas de sua área de atuação para a comunidade (MOSCAMED, 2010b).

O controle e monitoramento é realizado através da TIE em ambas as linhas de produção. Concebida em 1937, por Edward F. Knipling, a TIE é considerada um tipo de controle autocida ou genético, visto que a praga é utilizada em seu próprio controle (IMPERATO; RAGA, 2015). Os insetos estéreis são produzidos e liberados em quantidades bem maiores do que as encontradas na natureza, competem pelo acasalamento com os selvagens e acabam copulando com as fêmeas. As fêmeas, por sua vez, no caso das moscas, realizam a postura de ovos não fecundados. Já no caso dos mosquitos, o gene modificado repassado para as próximas gerações elimina ou inviabiliza as fêmeas adultas. As duas maneiras fazem com que a geração posterior de insetos tenha sua população reduzida. A repetição deste processo continuamente garante a manutenção do baixo índice populacional e em alguns casos leva à erradicação (MOSCAMED, 2003b; MOSCAMED, 2003a; MOSCAMED, 2010a).

Segundo Imperato e Raga (2015), até 1970 o controle de insetos era efetuado por meio de métodos baseados na utilização de inseticidas químicos sintéticos. O uso dos inseticidas sofreu diversas restrições durante a década de 70, à partir de então popularizou-se o conceito de Manejo Integrado de Pragas (MIP), onde diferentes ferramentas de controle (como por exemplo produtos químicos, agentes biológicos, ferômonios, dentre outras) são integradas de maneira planejada e coerente. A TIE foi posteriormente incorporada aos programas de MIP em grandes áreas, pois se trata de uma técnica de controle biológico onde as ações não são diretamente executadas por um produtor e sim por uma coordenação

regional gestora do programa. A TIE supre, em sua maioria, a necessidade de tratamento com inseticidas no combate às pragas, promovendo a redução das despesas de produção e o aumento consequente da qualidade final e da segurança alimentar do produto agrícola (MOSCAMED, 2003a).

Cerca de 28 países estão aplicando a TIE em larga escala para supressão populacional e erradicação em locais como a Ásia, ilhas do Pacífico, Oceania, África, Europa e alguns países das Américas e Caribe. No Brasil, 11 estados são beneficiados pela TIE por intermédio da coordenação da Moscamed juntamente ao Ministério da Agricultura, Pecuária e Abastecimento e das Agências estaduais de Defesa Agropecuária. São os estados da Bahia, Pernambuco, Ceará, Rio Grande do Norte, Piauí, Paraíba, Sergipe, Minas Gerais, Espírito Santo, Rio Grande do Sul e Santa Catarina (MOSCAMED, 2003b; MOSCAMED, 2003a).

2.2 A ESTAÇÃO METEOROLÓGICA E A INFLUÊNCIA DAS CONDIÇÕES CLIMÁTICAS NAS ESPÉCIES CONTROLADAS PELA MOSCAMED

As estações meteorológicas surgiram com a finalidade de suprir a necessidade de medir variáveis climáticas. O estudo dessas variáveis é de grande valia para as atividades humanas, pois implica de forma direta ou indireta no atual modelo de desenvolvimento, visto que uma política de desenvolvimento sustentável necessita de monitoramento sobre os recursos naturais finitos fortemente relacionados ao clima. No Brasil, inicialmente, a obtenção e avaliação de dados meteorológicos coletados era feita por intermédio do Instituto Nacional de Meteorologia - INMET. Esse processo possuía uma amostragem relativamente pequena e estava sujeita à erros humanos (TORRES et al., 2015).

O crescimento tecnológico proporcionou a possibilidade da criação de estações meteorológicas automáticas que abrem mão da interferência humana, prevenindo assim a ocorrência de erros e além disso incrementam consideravelmente o desempenho, praticidade e confiabilidade das medições realizadas (TORRES et al., 2015).

A estação meteorológica *Vantage Vue*^{TM} (figura 1) é uma estação automática *wireless* composta de dois módulos, sendo o primeiro deles um conjunto de sensores (figura 1a) e o segundo um console (figura 1b). A suíte de sensores é alimentada por energia solar, inclui uma bateria reserva e contém os seguintes sensores: coletor de chuva, sensor de umidade/temperatura, anemômetro e sensor de direção do vento. Após a coleta de dados pelos sensores, os mesmos são atualizados e enviados via rádio de baixa potência para o console (DAVIS INSTRUMENTS, 2017; DAVIS INSTRUMENTS, 2009a).

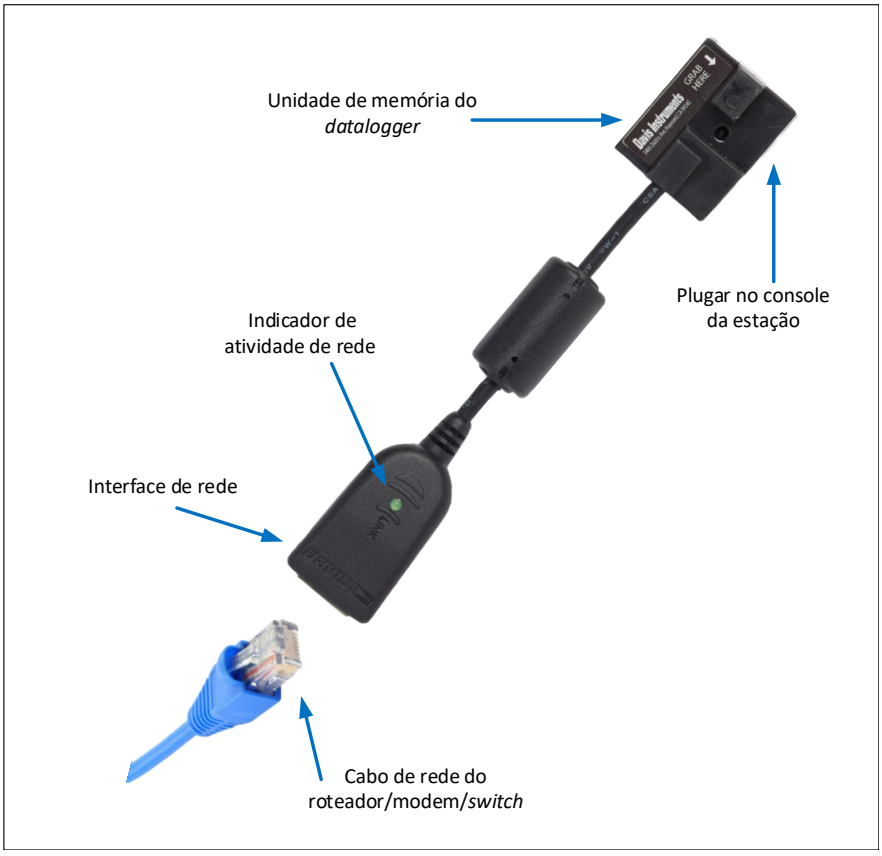
Figura 1 – A estação meteorológica *Vantage Vue*™

(a) Suíte de sensores **(b)** Console

Fonte: DAVIS INSTRUMENTS (2017)

Por sua vez, o console *Vantage Vue*™ é responsável pela exibição e registro dos dados de sua estação. Possui também a opção de interação com computador via módulo *ethernet* vendido separadamente, *WeatherLinkIP*® (figura 2) (DAVIS INSTRUMENTS, 2017; DAVIS INSTRUMENTS, 2009b).

Figura 2 – O *datalogger WeatherLinkIP*®



Fonte: O autor

O módulo *WeatherLinkIP*® consiste de um *datalogger* que conecta o console da estação à internet. O módulo transfere os dados climáticos obtidos do console para o computador, com a finalidade gerar uma base de dados permanente. As principais variáveis exportadas e arquivadas pelo *datalogger* são a hora/data da medição, temperatura interna e externa, direção e velocidade do vento, precipitação e a umidade interna e externa (DAVIS INSTRUMENTS, 2008).

O tamanho da memória não volátil do módulo é de 128KB. A memória é preenchida de acordo com o intervalo (em minutos) de atualização definido pelo usuário. Portanto, o período de tempo ao qual as amostras se referem pode ter uma extensão maior ou menor. Por exemplo, caso o usuário escolha a taxa de atualização de um e um minuto, o módulo terá espaço suficiente para armazenar dados de cerca de quarenta e duas horas. Por outro lado, se a frequência de atualização for de trinta minutos, o armazenamento terá dados de cinquenta e três dias (DAVIS INSTRUMENTS, 2008).

2.2.1 A influência do clima na mosca-das-frutas

A mosca-das-frutas é um inseto que sofre diversas alterações em seu ciclo de vida de acordo com as mudanças climáticas. Por volta de 250 (duzentas e cinquenta) espécies dessas pragas atacam variedades de plantas frutíferas de grande interesse econômico. São insetos da ordem *Diptera* e família *Tephritidae*. Estão presentes em todo território brasileiro, porém existem dois gêneros com importância maior, sendo uma originária das Américas Central e do Sul e outra cuja a introdução no país ocorreu no início do século XX, os gêneros *Anastrepha* e *Ceratitis*, respectivamente. Na Região do Vale do Submédio São Francisco há 11 (onze) espécies de *Anastrepha* já identificadas (PARANHOS, 2008).

Os ciclos de vida de ambos os gêneros supracitados são semelhantes, diferenciando-se apenas na questão do tempo de desenvolvimento das fases. O ciclo se inicia com a oviposição realizada pela fêmea nos frutos, após a eclosão a larva penetra no endocarpo e depois deixam o fruto para empupar no solo e por fim emergirem como adultos (PARANHOS, 2008).

Segundo Calore et al. (2013), em um estudo relacionado aos insetos de goiaba, a sazonalidade dos elementos climáticos/meteorológicos têm influência direta ou indireta sobre esses insetos. Diretamente, os elementos do clima podem atuar na mortalidade ou no desenvolvimento dos insetos, alterando a oviposição, alimentação, crescimento, desenvolvimento e migração. Os aspectos climáticos podem atuar de forma indireta pela influência na atividade dos inimigos naturais dos insetos e pela variação na qualidade dos recursos disponíveis devido as mudanças fisiológicas e bioquímicas na planta hospedeira.

Ainda de acordo com Calore et al. (2013), as variáveis meteorológicas que estão mais relacionadas com a dinâmica populacional de insetos e diversos ecossistemas são a temperatura, a umidade relativa, a precipitação pluviométrica e a velocidade do vento.

Em algumas espécies a evaporação, a insolação e o fotoperíodo também são importantes.

Em relação a mosca-das-frutas têm-se a temperatura do ar como principal influenciador no desenvolvimento de suas fases de ovo, larva e adulta, enquanto a temperatura do solo é predominante na fase de pupa (GARCIA; CORSEUIL, 1998). Por exemplo, em situações de temperaturas superiores a 35ºC ou inferiores a 10ºC não há desenvolvimento de nenhuma das fases do ciclo de vida da mosca-da-fruta da espécie *Anastrepha fraterculus* (ARAUJO et al., 2008). Além disso, em regiões semiáridas a precipitação, as temperaturas elevadas e a disponibilidade de hospedeiros regem os picos populacionais da mosca-das-frutas da espécie citada, pois a precipitação ocasiona o aumento da umidade no solo, viabilizando o desenvolvimento da fase de pupa, que em sua maioria ocorre numa camada contemplada pelos dez primeiros centímetros do solo. Essa camada é altamente vulnerável ao dessecamento e portanto pode causar a inviabilidade das pupas e dificultar a emergência dos adultos devido ao solo ressecado (CALORE et al., 2013; ARAUJO et al., 2008).

2.2.2 A influência do clima no mosquito-da-dengue

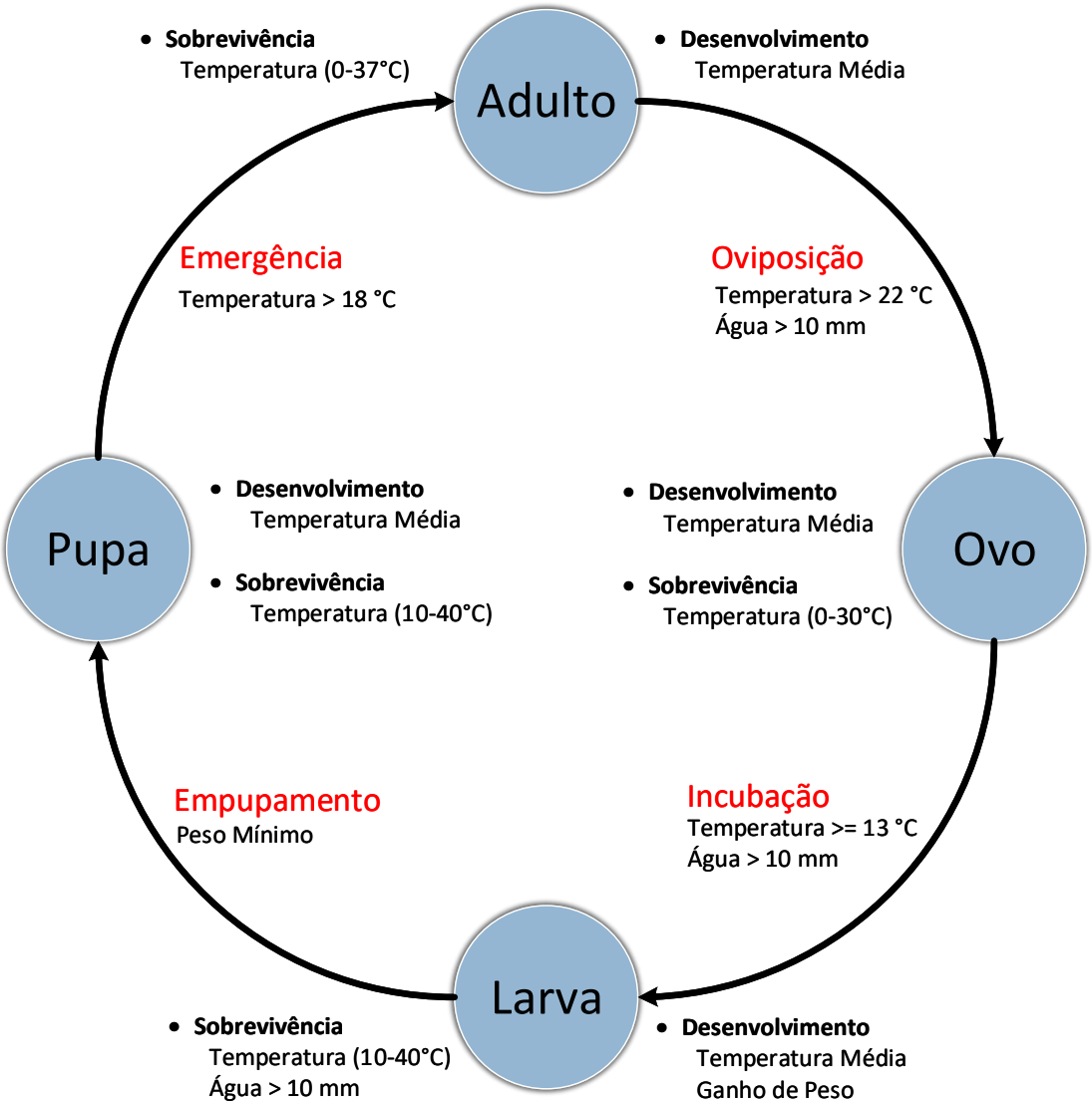
De forma análoga à mosca-das-frutas, os mosquitos *Aedes Aegypti* também sofrem variações em seu ciclo vital e em seu comportamento mediante modificações climáticas. De acordo com Hopp e Foley (2001), o aumento da temperatura global e outras mudanças climáticas podem modificar a área geográfica de presença desse inseto. Como por exemplo na Colômbia, onde os insetos eram limitados pela altitude de 1500 metros devido ao intenso frio e hoje em dia são encontrados em níveis acima de 2200 metros em razão da elevação da temperatura.

A temperatura e a precipitação pluviométrica, apresentam grande significância em relação ao desenvolvimento e sobrevivência dessa espécie, como pode ser observado na figura 3 (HOPP; FOLEY, 2001; RIBEIRO et al., 2006).

Ainda na figura 3, em algumas fases estão destacadas as faixas de valores das variáveis que são cruciais para o desenvolvimento e/ou sobrevivência do inseto, em outras etapas destaca-se apenas a dependência de determinados fatores. Na incubação por exemplo (transição entre a fase de ovo e a fase de larva), necessita-se da temperatura igual ou superior a 13ºC e uma lâmina d'água de pelo menos 10 milímetros de espessura. Em outro caso, na fase de pupa o desenvolvimento do inseto é sensível à temperatura média diária. As faixas de temperaturas aqui ilustradas são referentes à taxa de sobrevivência igual a 1 (um), como pode ser visualizado na figura 4.

De acordo com Depradine e Lovell (2004), uma elevada pressão de vapor de água facilita a atividade dos mosquitos, tornando suas atividades de alimentação ou ataque mais frequentes, provocando a propagação acentuada da doença.

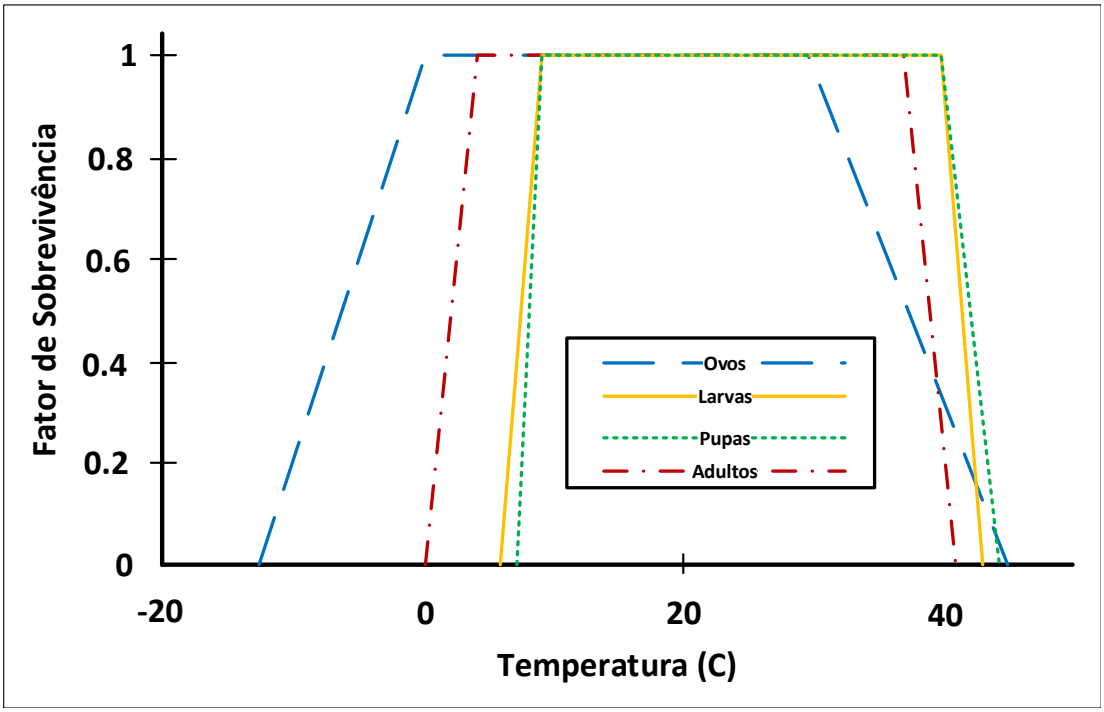
Figura 3 – O Ciclo de Vida do Mosquito *Aedes Aegypti* e suas dependências climáticas diárias



Fonte: Hopp e Foley (2001) (Adaptado)

Além do ciclo de vida do mosquito, a dinâmica de casos de doenças relacionadas ao vetor também flutua com as condições do clima. A flutuação está diretamente ligada ao aumento de temperatura, pluviosidade e umidade do ar, pois o conjunto desses fatores favorecem o crescimento da quantidade de criadouros disponíveis e o desenvolvimento do vetor (RIBEIRO et al., 2006).

Figura 4 – Dependência da temperatura na sobrevivência dos estágios de vida do mosquito *Aedes Aegypti*



Fonte: Hopp e Foley (2001)(Adaptado)

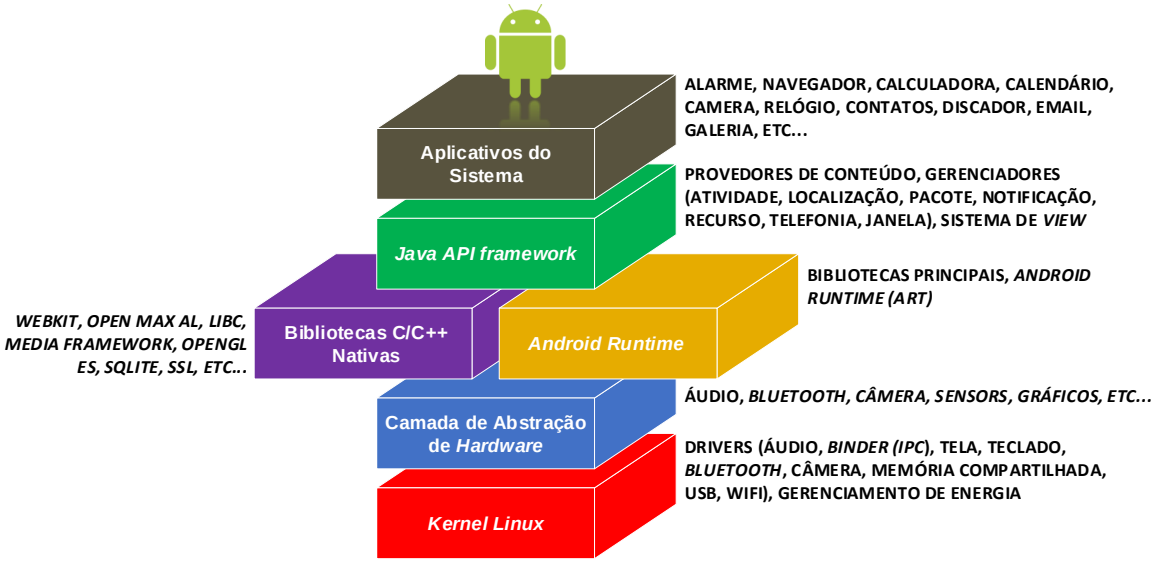
2.3 SISTEMA OPERACIONAL ANDROID

O *Android* é a plataforma de desenvolvimento *mobile* mais popular do mundo, está presente em mais de 190 países e possui centenas de milhões de dispositivos ativos. A plataforma conta com a contribuição da comunidade *open-source* Linux e mais de 300 parceiros em *hardware*, *software* e operadoras. A plataforma permite a criação de aplicativos por meio da linguagem de programação Java. A Google, dententora da plataforma, disponibiliza o Kit de Desenvolvimento de Software (*Software Development Kit - SDK*) e o Ambiente de Desenvolvimento Integrado (IDE) para a criação de aplicativos nativos para *Android* (ANDROID, 2017b).

2.3.1 Características e Arquitetura

A pilha de software da plataforma *Android* possui código aberto e é baseada em Linux, conforme ilustrada pela figura 5. A escolha do Linux como base da pilha é justificada pela utilização de recursos de segurança, outras atividades de gerenciamento de *hardware* já disponíveis nesse sistema operacional e pela facilidade proporcionada aos fabricantes no desenvolvimento de *drivers* de *hardware* para um *kernel* conhecido (ANDROID, 2017b; GANDHEWAR; SHEIKH, 2010).

Figura 5 – A pilha de software do Android



Fonte: ANDROID (2017a) (Traduzido)

A segunda camada, abstração de *hardware* (*Hardware Abstration Layer - HAL*), é responsável por fornecer para a interface de programação de aplicação (*Application Programming Interface - API*) Java o acesso aos componentes de *hardware*, tais como o módulo de câmera ou *bluetooth*. O *Android Runtime - ART* é capaz de executar para cada aplicativo uma máquina virtual própria que requer poucos recursos, possibilitando a execução de vários aplicativos simultaneamente. Os dois últimos componentes e alguns serviços são implementados em código nativo que utiliza bibliotecas em C/C++, porém essas funcionalidades podem ser acessadas através de APIs Java (ANDROID, 2017b).

O sistema operacional fornece seus recursos por intermédio de APIs programadas em linguagem Java. As APIs compõem os blocos necessários para a criação dos aplicativos, como o sistema de visualização e os gerenciadores de recursos, notificações e atividades (ANDROID, 2017b).

Na camada mais alta têm-se os aplicativos padrão do sistema que podem ser utilizados para prover funcionalidade à outros aplicativos de terceiros, eliminando a necessidade de desenvolver novamente uma funcionalidade, por exemplo o envio de uma mensagem de texto ou a captura de uma fotografia (ANDROID, 2017b; SAHA, 2008).

2.4 APLICAÇÕES WEB

O grande sucesso da *Web* como arquitetura para desenvolvimento de aplicações pessoais, sociais e de negócios tem impulsionado os desenvolvedores à criarem novas aplicações ou portar *softwares* existentes para a *Web* (FRATERNALI; PAOLINI, 1998). A numerosidade das aplicações desenvolvidas fez nascer a necessidade de estabelecer padrões/princípios para o desenvolvimento de aplicações *Web* (PRESSMAN, 2000).

A engenharia de software de desenvolvimento *Web* possui pontos semelhantes e diferentes em relação à de *software* convencionais. É semelhante pelo fato de buscar a realização de uma aplicação correta e completa, mediante os requisitos propostos, e difere porque deve considerar a arquitetura *Web* para sua execução (CONTE; MENDES; TRAVASSOS, 2005).

As aplicações *Web* podem ser divididas em duas categorias, aplicações hipermídia *Web* e aplicações de *software Web*. A primeira é uma aplicação não-convencional onde as informações são disseminadas através de nós, *links*, âncoras, estruturas de acesso e é disponibilizada na *Web*. As tecnologias utilizadas comumente para desenvolver essas aplicações são o *HyperText Markup Language - HTML*, JavaScript e pacotes de multimídia. As aplicações de *software Web* são aplicações nos modelos tradicionais que dependem da arquitetura *Web* para seu funcionamento, tais como sistemas legados de banco de dados, bases de conhecimento, *e-commerce*, entre outros (MENDES; MOSLEY; COUNSELL, 2005).

A elaboração de aplicações *Web* deve considerar três dimensões em sua concepção, são elas:

- Estrutural: diz respeito a organização das informações e seus relacionamentos semânticos.
- Navegacional: como o nome já diz, representa a forma como as informações organizadas pela camada Estrutural serão acessadas.
- Apresentação: é a forma como as informações e a aparência dadas à ela serão expostas ao usuário.

A qualidade da navegação e da apresentação são tão importantes quanto a qualidade da informação em si (FRATERNALI; PAOLINI, 1998). O principal objetivo da engenharia de software *Web* é que haja o desenvolvimento correto da aplicação em termos de estrutura, funcionalidades, aspectos navegacionais e interação com o usuário (PASTOR, 2004).

2.5 APLICAÇÕES MULTIPLATAFORMAS

O desenvolvimento de aplicativos multiplataformas consiste em gerar aplicações para diversos sistemas operacionais e para *Web* à partir de um único código fonte. Essa forma de desenvolver ganha mais popularidade à cada dia, principalmente devido ao fato de algumas ferramentas permitirem a utilização de linguagens de programação bastante conhecidas pela comunidade, tais como o HTML, JavaScript e o *Cascading Style Sheet (CSS)*. O código nativo de cada sistema operacional é mascarado por chamadas à APIs, realizadas apenas quando se deseja operar sobre um dispositivo, como câmera ou giroscópio (RAJ; TOLETY, 2012; PALMIERI; SINGH; CICCHETTI, 2012; DALMASSO et al., 2013).

Devido ao fato de se possuir apenas uma base de código, a manutenibilidade do software se torna bem menos custosa do que prover soluções nativas específicas para cada plataforma (RAJ; TOLETY, 2012). Palmieri, Singh e Cicchetti (2012) destacam alguns pontos positivos relacionados ao desenvolvimento de aplicações com esse tipo de ferramenta:

- Redução das habilidades requeridas aos programadores;
- Redução do tamanho do código;
- Redução do tempo de desenvolvimento e custos de manutenção a longo prazo;
- Redução de conhecimento necessário sobre às APIs nativas;
- Maior facilidade de desenvolvimento;
- Incremento da participação no mercado.

Aplicações *cross-platform* podem ser classificadas em quatro tipos de abordagem, *Web*, Híbrida, Interpretada e Compilação Cruzada. A abordagem *Web* classifica aplicações cuja execução é feita no navegador do dispositivo móvel e os dados são providos por um servidor, neste caso nenhuma instalação é realizada no armazenamento do aparelho. A metodologia Híbrida requer instalação e também usa o navegador do aparelho para renderizar e mostrar as telas da aplicação, porém existe a possibilidade de efetuar chamadas às APIs JavaScript com a finalidade de operar sobre o *hardware* do aparelho (RAJ; TOLETY, 2012; DALMASSO et al., 2013).

As aplicações interpretadas, como o próprio nome já diz, têm seu código interpretado em tempo de execução por um interpretador. Assim como as aplicações Híbridas, possuem uma camada de abstração que provê o acesso ao *hardware* via APIs. No caso da Compilação Cruzada, o código fonte é convertido para os códigos binários nativos de cada plataforma e o *hardware* é acessado pelas APIs nativas de cada sistema (RAJ; TOLETY, 2012).

Existem prós e contras em cada tipo de abordagem, portanto deve-se adotar a que mais se adequa ao projeto a ser desenvolvido. Por exemplo, a abordagem *Web* não possui acesso aos sensores do aparelho *mobile*, já a abordagem Híbrida possui performance inferior em relação as aplicações nativas, pois são executadas no navegador. Enquanto isso, na metodologia de Compilação Cruzada, códigos específicos de cada plataforma não podem ser reutilizados, como interface, acesso à localização e notificações (RAJ; TOLETY, 2012).

A seleção da metodologia ideal depende majoritariamente dos requerimentos da aplicação. A tabela 1 mostra em uma escala de valores, quais abordagens são ideais para cada tipo de aplicação: 1 – Não preferível, 2 – Preferível, mas não é a metodologia ideal e 3 – Metodologia perfeita (RAJ; TOLETY, 2012).

Tabela 1 – Tipos de aplicações e abordagens preferenciais.

Código da Aplicação	**Web**	**Híbrida**	**Interpretada / Compilação Cruzada**
Aplicações baseadas em dados providos por um servidor	3	2	1
Aplicações independentes	1	2	3
Aplicações baseadas em sensores e processamento de dados no dispositivo	1	2	3
Aplicações baseadas em sensores e processamento de dados no servidor	1	3	2
Aplicações Cliente-Servidor	1	3	2

Fonte: Raj e Tolety (2012) (Traduzido)

2.6 API RESTFUL

O estilo de arquitetura REST (*Representational State Transfer*) consiste da utilização dos verbos do protocolo HTTP para representar ações no acesso à recursos identificados por um URI (*Uniform Resource Identifier*). As comunicações realizadas com o estilo REST são *stateless*, ou seja, os dados de uma requisição antiga não são repassados para uma requisição posterior. Portanto, é necessário o envio de todas as informações a cada nova requisição. As respostas enviadas pela API ao cliente possuem o formato JSON (*Javascript Object Notation*) ou XML (*Extensible Markup Language*) independentemente da representação original do recurso (ARCURI, 2017; FIELDING; TAYLOR, 2000).

Os verbos HTTP utilizados comumente nesse tipo de arquitetura são o GET, POST, PUT e DELETE. As aplicações dos verbos estão exemplificadas na tabela 2 abaixo.

Tabela 2 – Métodos HTTP.

Método HTTP	**Ação**	**Notas**
GET	Ler	Recupera um recurso. Exemplo: GET /livros/1 (recupera o recurso de identificador 1) GET /livros (recupera vários recursos)
POST	Criar	Cria um recurso com os parâmetros repassados. Exemplo: POST /livros “nomeLivro” : “nome”
PUT	Atualizar	Atualiza ou cria (caso não exista) um recurso identificado, com os valores repassados. Exemplo: PUT /livros/1 “nomeLivro” : “novoNome”
DELETE	Deletar	Deleta um recurso identificado. Exemplo: DELETE /livros/1 (deleta o recurso de identificador 1)

Fonte: Hsieh, Hsieh e Cheng (2016) (Traduzido)

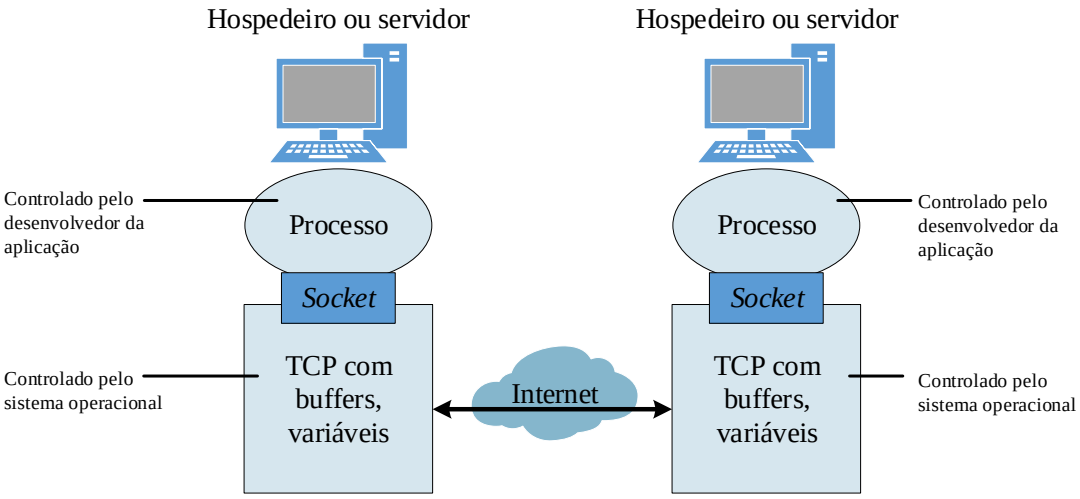
2.7 SOCKET TCP

O protocolo TCP (*Transmission Control Protocol*) trabalha entregando os pacotes recebidos aos respectivos protocolos de aplicação atribuídos às portas de destino. Por exemplo, ao receber um pacote destinado à porta 80, os dados são entregues ao protocolo HTTP (TORRES, 2001).

Porém, quando mais de uma aplicação de mesmo tipo estão comunicando-se com a rede é necessário o uso de um *socket* para definir uma conexão diferente para cada aplicação dentro de uma porta. Os *sockets* gerados quando o transmissor e o receptor criam uma conexão, são definidos em duas classes, ativo e passivo. O primeiro é aquele que envia dados, enquanto o segundo é aquele que recebe dados (TORRES, 2001).

As aplicações comunicam-se escrevendo e lendo de seus respectivos *sockets*, ou seja, um *socket* é a interface entre a camada de aplicação e a de transporte dentro de uma máquina como ilustra a figura 6 (KUROSE; ROSS, 2005).

Figura 6 – Processos de aplicação, *sockets* e protocolo de transporte subjacente



Fonte: Kurose e Ross (2005)

3 MATERIAIS E MÉTODOS

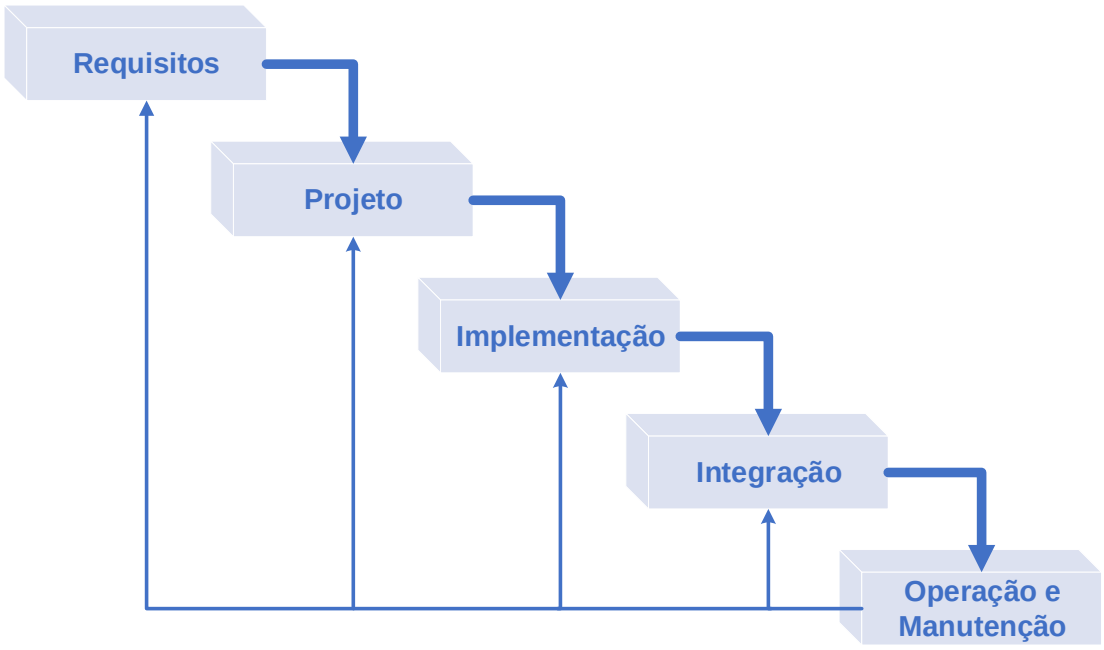
Este capítulo tem como finalidade descrever o projeto de *software* através da metodologia utilizada, de maneira a elucidar as várias facetas e componentes do sistema como um todo. O capítulo está dividido nas seguintes partes: Projeto de software e Tecnologias utilizadas.

3.1 PROJETO DE SOFTWARE

O projeto foi construído baseando-se em uma adaptação da metodologia Cascata em conjunto com a metodologia Iterativa e Incremental.

A metodologia Cascata foi cunhada por Royce (1970). É composta por cinco etapas, onde a saída de cada etapa se torna a entrada da etapa posterior. Para isso, é necessário que a etapa anterior seja concluída e esteja correta para que se possa prosseguir com o desenvolvimento. As etapas do modelo são mostradas pela figura 7.

Figura 7 – Modelo de Desenvolvimento de *Software* Cascata



Fonte: Royce (1970) (Traduzido)

A primeira etapa busca compreender os requisitos do sistema, que normalmente baseiam-se nas funcionalidades que precisam ser fornecidas, nas limitações e nos objetivos do software. Em seguida há o projeto do sistema, composto do modelo de dados, arquitetura

do *software*, interfaces de usuário e demais detalhes (CASA DA CONSULTORIA, 2017; MEDEIROS, 2017a).

As etapas de implementação e testes dizem respeito à programação do *software* em si e ao correto funcionamento das lógicas internas e funcionalidades externas do sistema, respectivamente. Por último têm-se a etapa de operação/manutenção, esta etapa resume-se a implantação, correção de erros não identificados durante a verificação, melhorias funcionais e suporte ao projeto (CASA DA CONSULTORIA, 2017; MEDEIROS, 2017a).

A adaptação buscou simplificar o método cascata para adequá-lo à realidade do projeto. Pois devido à existência de um único projetista / desenvolvedor e ao tempo disponível para a execução do trabalho, algumas restrições do método não foram levadas em consideração, como por exemplo a rigidez sequencial na transição entre as etapas e o estabelecimento de requisitos explícitos no início do projeto.

Para solucionar as restrições citadas anteriormente utilizou-se o modelo Iterativo e Incremental. Este método prioriza o desenvolvimento/entrega de *software* funcional em pequenos pedaços. A cada nova iteração o sistema recebe novas funcionalidades. Deste modo, o sistema é construído de forma incremental (BERNARDO, 2015).

Portanto, o sistema foi subdividido em diversas partes de acordo com os requisitos elencados, onde em cada subdivisão era escolhido um requisito para ser projetado, implementado, verificado e incrementado ao sistema.

3.1.1 Requisitos do sistema

Os requisitos de um sistema nada mais são do que um conjunto de descrições à respeito do que o sistema deve fazer, quais funcionalidades/serviços são oferecidos e quais são as restrições ou qual é o escopo do mesmo. Os requisitos de um sistema podem ser classificados como funcionais e não funcionais (SOMMERVILLE, 2011).

Os requisitos ou características funcionais explicitam os serviços que o sistema deve oferecer, como ele deve reagir mediante determinadas entradas e qual seu comportamento em situações especificas (SOMMERVILLE, 2011; FILHO, 2003).

As características não funcionais são restrições aos serviços ou funcionalidades fornecidas pelo sistema. Ou seja, os requisitos não funcionais qualificam o requisitos funcionais. A qualificação pode ser referente ao *software*, como a performance, usabilidade, confiabilidade, tolerância à falha, dentre outros. Ou pode ser referente ao processo de desenvolvimento, como tempos de entrega, método de desenvolvimento, e assim por diante (MEDEIROS, 2017b).

A seguir são listados os requisitos elencados para este projeto.

3.1.1.1 Requisitos funcionais

- O sistema deve permitir o cadastro de novos usuários através da interface.
- O sistema deve permitir o *login* dos usuários através da interface.
- O sistema deve permitir ao usuário visualizar as informações mais recentes sobre o clima.
- O sistema deve permitir ao usuário visualizar o histórico das variáveis climáticas mediante escolha do período a qual se refere.
- O histórico deve ser exibido na forma de gráficos, permitindo a seleção das variáveis mostradas.
- O sistema deve coletar os dados da estação meteorológica periodicamente e salvá-los na base de dados.

3.1.1.2 Requisitos não funcionais

- O sistema deve ser de natureza multiplataforma, permitindo a consulta de dados tanto através da *Web* quanto através de aplicativo *Android*.
- O sistema deve permitir a consulta de dados via *internet* a qualquer momento, desde que haja conexão.
- Todas as consultas do sistema só podem ser efetuadas por usuários autenticados.
- O sistema deve possuir uma interface com boa experiência de usuário.

À partir dos requisitos elencados foram elaborados os casos de uso discutidos na subseção 4.2.1.

3.1.2 Arquitetura do sistema

Inicialmente buscou-se compreender a arquitetura do sistema como um todo, levando em consideração o objetivo geral do projeto. Diante disso, identificou-se a necessidade de projetar uma arquitetura composta de um módulo responsável por obter os dados da estação meteorológica, um módulo com o papel de gerenciar os dados obtidos e por fim, um módulo para exibir as informações, ver figura 8.

Para satisfazer a necessidade de operação da arquitetura da estação *Vantage Vue™*, aderiu-se o *socket TCP* como forma de captura dos dados pelo módulo responsável. Como o objetivo geral do projeto especifica que o sistema deve operar via *internet*, o módulo de gerenciamento de dados foi projetado como uma API RESTful. O fato de ser adotada uma

Figura 8 – Módulos do sistema

Fonte: O Autor

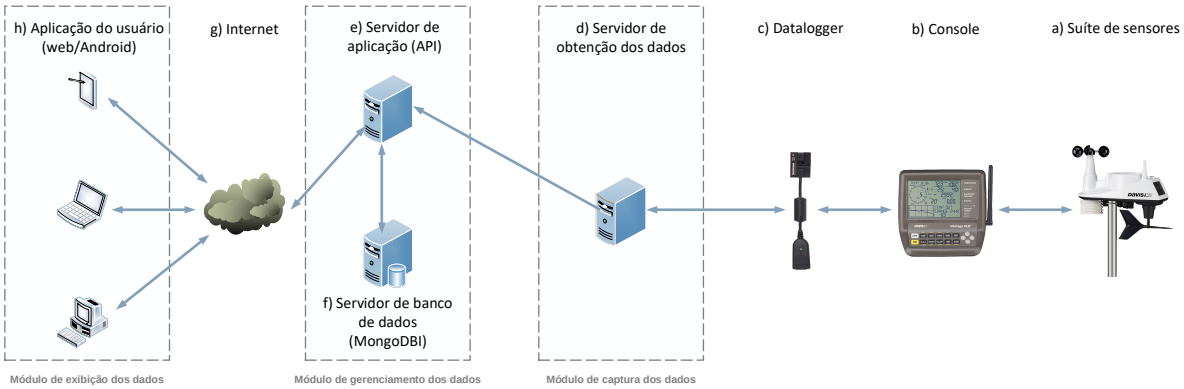
API para o sistema, resolve o problema de intercomunicação entre os módulos e a base de dados, pois atua como uma camada de abstração que gerencia toda a comunicação. Desta forma, o módulo de captura dos dados e o módulo de exibição podem se comunicar com o módulo de gerenciamento através de requisições HTTP, explanadas na subseção 4.2.5.

Além do mais, ambos os módulos de captura e de gerenciamento dos dados foram implementados utilizando a linguagem *Javascript* no ambiente de execução *Node.js*.

Ainda para satisfazer o objetivo geral do sistema, onde é previsto a visualização das informações por meio de uma aplicação *Web* e também por meio de um aplicativo *Android*, adotou-se o *framework Ionic* para o desenvolvimento da interface de usuário.

A figura 9 demonstra o panorama geral do projeto e o contexto onde cada componente está inserido.

Figura 9 – Visão geral do sistema



Fonte: O autor

A suíte de sensores da estação meteorológica (a) transmite as informações captadas por ondas de rádio para o seu console (b) que as exibe localmente em sua tela. A API (d) requisita para o *datalogger* as informações contidas no console. O *datalogger* (c) captura essas informações e responde a requisição. A API então processa as informações colhidas, salva os dados no banco e disponibiliza-os através da *internet* (e) para a aplicação cliente (f).

3.2 TECNOLOGIAS UTILIZADAS

3.2.1 *Ionic Framework*

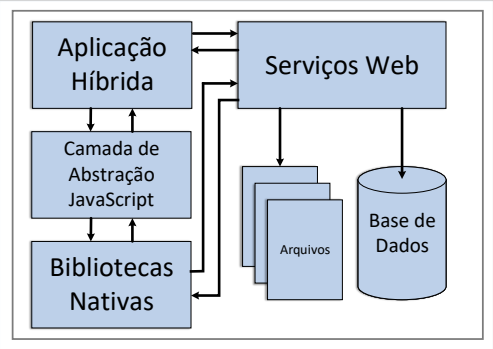
O *Ionic Framework* é um SDK que permite o desenvolvimento de aplicações multi-plataformas (*Web, Android, iOS*) através de tecnologias *Web* bem estabelecidas (HTML, CSS e JavaScript). O *framework* é suportado pela tecnologia Angular (IONIC FRAMEWORK, 2017).

Angular é uma plataforma, mantida pela Google, que torna fácil a criação de aplicações *Web* cliente através de combinações entre *templates* declarativos, injeção de dependências e ferramentas ponta-a-ponta (GOOGLE ANGULAR, 2017).

No caso do *Ionic Framework*, o Angular é responsável pelo pilar base da arquitetura, a API de componentes. Os componentes são os blocos de construção da interface de usuário do aplicativo móvel (IONIC FRAMEWORK, 2017).

A escolha desse *framework* deve-se ao fato dele ser completamente gratuito, de código aberto (licenciado sob a licença MIT) e seguir o modelo de arquitetura *cross-platform* híbrida (figura 10), possibilitando a utilização de tecnologias *Web* para construção de aplicativos e também a distribuição/instalação do *APP* via o *Market Place* da plataforma Android (IONIC FRAMEWORK, 2017). O Ionic será empregado no desenvolvimento da interface de usuário do sistema, ou seja, o *front-end* da aplicação.

Figura 10 – Modelo de aplicação *cross-platform* híbrida



Fonte: Raj e Tolety (2012) (Traduzido)

3.2.2 Node.js

O Node.js é um ambiente de tempo de execução JavaScript assíncrono baseado em eventos, construído sobre o motor JavaScript V8 da Google rodando do lado do servidor. Surgiu com o propósito de possibilitar a criação de aplicações de rede escaláveis. O paradigma baseado em eventos permite múltiplas conexões de diversos clientes ao servidor de forma eficiente, escalável e oferecendo muito mais controle para o desenvolvedor sobre as atividades de sua aplicação (TILKOV; VINOSKI, 2010).

O *Node.js* será utilizado no escopo deste projeto para criação de um serviço responsável pela obtenção dos dados da estação meteorológica, empregando uma comunicação serial através de um *socket TCP*, e pela inserção posterior desses no banco de dados. Haverá também outro serviço para a recuperação dos dados à serem usados na aplicação cliente, ou seja, o serviço terá o papel de fornecer uma API que por meio de requisições HTTP insira e recupere informações da base de dados.

A motivação da escolha dessa tecnologia é baseada nas duas características seguintes, o código do software é aberto e o fato da tecnologia também usar JavaScript torna natural a integração com aplicações *cross-platform* que utilizam da mesma linguagem de programação. Além de facilitar a comunicação com a base de dados.

3.2.3 MongoDB

O MongoDB é um banco de dados não relacional baseado em documentos, que armazena os dados em um formato semelhante ao JSON. O software é grátis e de código aberto, licenciado sob a *GNU Affero General Public License*. É adotado por grandes empresas como *Adobe, Twitter, GitHub e Amazon*. A adoção desse banco de dados é justificada pela sua consistência, velocidade, escalabilidade e facilidade de utilização (MONGODB, Inc., 2017). Além disso, os dados a serem trabalhados são do tipo chave-valor simples e não requerem uma base de dados relacional.

O banco de dados será usado majoritariamente para guardar as informações das variáveis climáticas provenientes das leituras da estação meteorológica, com a finalidade de manter um histórico das variações. Pois, o armazenamento disponível na estação é limitado e os dados são substituídos por novos à medida que as novas leituras ocorrem.

4 RESULTADOS E DISCUSSÕES

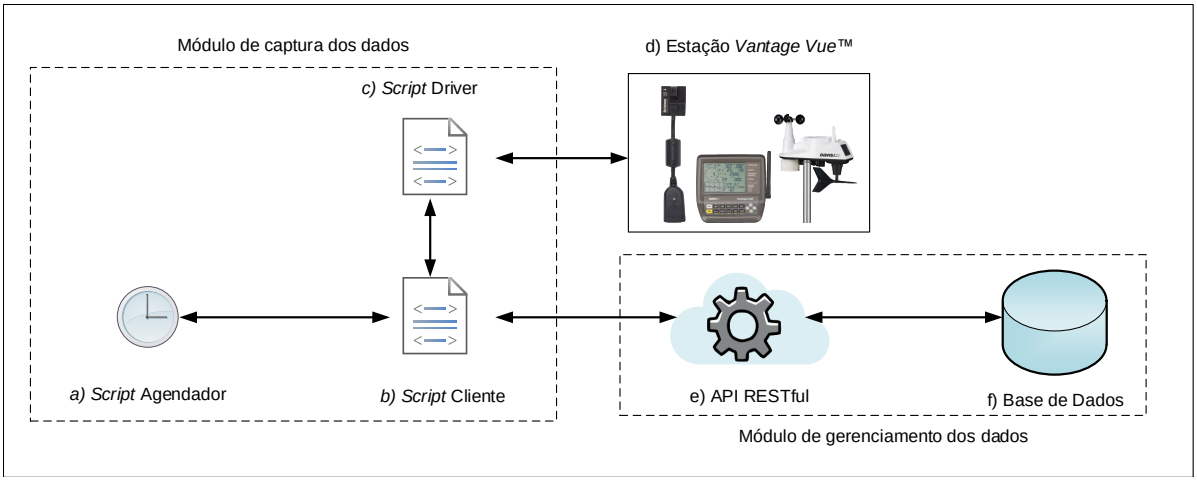
Neste capítulo são apresentados os resultados obtidos com o projeto da arquitetura dos módulos do sistema e o seu desenvolvimento, juntamente com a documentação, os diagramas de engenharia de *software* e o protótipo composto pela interface de usuário e a parte lógica do sistema.

4.1 MÓDULOS DO SISTEMA

4.1.1 Módulo de captura dos dados

O módulo de captura dos dados têm seu funcionamento ilustrado na figura 11. Como dito anteriormente, periodicamente este módulo obtém informações da estação meteorológica *Vantage Vue™* e os armazena na base de dados por meio do módulo de gerenciamento de dados. Para cumprir sua função, o módulo foi dividido em três *scripts* escritos em *Javascript* e executados em *Node.js*.

Figura 11 – Módulos de captura e gerenciamento dos dados



Fonte: O Autor

O primeiro *script*, Script Agendador (a), é responsável por executar, a cada intervalo pré-determinado de tempo, uma *thread* composta pelo Script Cliente (b). O Script Driver é um programa que se conecta à estação meteorológica utilizando *socket TCP*, envia um comando de *LOOP* (ver anexo A), recebe a resposta do comando contendo os dados e converte as unidades de medida. O Script Cliente possui o papel de criar uma instância do Script Driver (c), e após o recebimento dos dados, salvá-los na base de dados por intermédio de uma requisição HTTP para a API RESTful.

O agendador do Node.js possui a mesma sintaxe do GNU crontab (MELLOR, 2014; ALECRIM, 2005; MERENCIA, 2017) onde cada asterisco representa uma medida de tempo de acordo com o seguinte padrão:

[minutos] [horas] [dias do mês] [mês] [dias da semana]

Código 1 – Configuração do intervalo de execução no Script Agendador

```
1 cron.schedule('*/15 * * * *', function(){
2     const { fork } = require('child_process');
3     const forked = fork('client.js');
4 });
```

Fonte: O autor

No exemplo acima o Script Cliente é executado por uma *thread* a cada 15 (quinze) minutos.

4.1.2 Módulo de gerenciamento dos dados

O módulo de gerenciamento dos dados é formado pela API do sistema e pela base de dados. A API possui rotas relacionadas ao gerenciamento dos dados climáticos e aos dados dos usuários, como pode ser observado na subseção 4.2.5. Também foi construída utilizando-se da linguagem *Javascript* e do ambiente de execução *Node.js*, incluindo o modelo de documentos dos usuários e dos dados climáticos.

O servidor de banco de dados executa uma base não relacional baseada em documentos chamada MongoDB (ver subseções 3.2.3 e 4.2.6).

4.1.3 Módulo de exibição dos dados

O módulo de exibição dos dados é constituído apenas da aplicação *Ionic*, também escrita em *Javascript*, porém com a adição de HTML e de CSS para construção das páginas. Neste caso, o *Javascript* é empregado para dar funcionalidades às páginas renderizadas pelo HTML juntamente com CSS.

A aplicação Ionic pode ser compilada tanto para *Web*, quanto para as principais plataformas móveis, devido a isso foi adotada tal tecnologia no projeto. O funcionamento da aplicação está detalhado na subseção 4.2.4.

4.2 DIAGRAMAS DE ENGENHARIA DE SOFTWARE

A etapa inicial no desenvolvimento de *software* é compreender os relacionamentos entre o *software* em projeto e o seu ambiente exterior. Esta etapa é essencial porque mostra

como se dará a estrutura do sistema para se comunicar com o seu ambiente e quais serão os limites estabelecidos para o mesmo. A definição do sistema informa quais serão as funcionalidades implementadas e os recursos disponíveis (SOMMERVILLE, 2011).

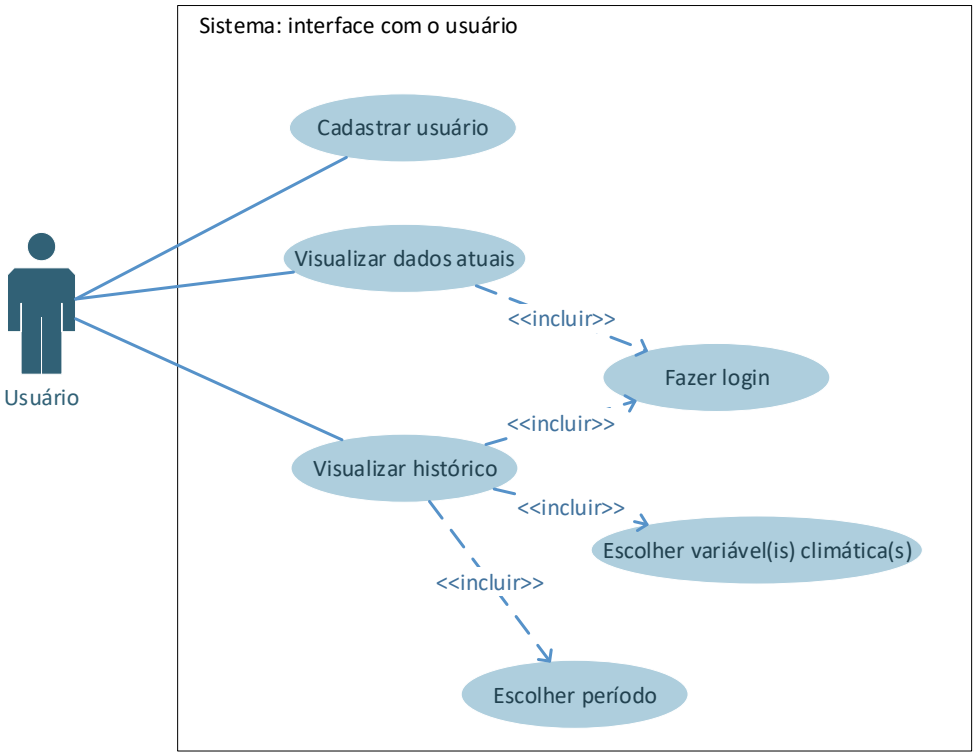
Os diagramas abordados serão o de caso de uso, de sequência e de atividade. Esses três diagramas proporcionam uma visão geral de cada etapa do funcionamento do sistema. Além disso, será discutido também um modelo de documento referente ao banco de dados adotado.

4.2.1 Casos de uso

As interações entre alguns elementos do sistema serão relatadas pelos diagramas de caso de uso seguintes, utilizando de uma abordagem abstrata e sem muitos detalhas de implementação, pois destinam-se apenas à oferecer uma compreensão do que o sistema faz, ou seja, define os requisitos funcionais do sistema (SOMMERVILLE, 2011).

A figura 12 mostra o caso de uso referente à utilização da aplicação cliente pelo usuário final, enquanto a figura 13 demonstra as interações do sistema do ponto de vista da API.

Figura 12 – Diagrama de caso de uso referente ao usuário final



Fonte: O autor

- **Cadastrar usuário**
  - Atores
    - Usuário
  - Fluxo de eventos primário
    1. o usuário deve se cadastrar informando seu nome, *e-mail* e senha;
    2. caso o usuário não seja autenticado, o sistema informa a respeito de credenciais inválidas e encerra o caso de uso;
    3. a API armazena os dados do usuário;
    4. o usuário é liberado para realizar o *login*.
  - Fluxo alternativo
    - o usuário desiste de se cadastrar e cancela o caso de uso clicando no botão voltar.
- **Visualizar dados atuais**
  - Atores
    - Usuário
  - Pré-condições
    - o usuário deve estar autenticado
  - Fluxo de eventos primário
    1. o usuário deve efetuar o *login* informando o *e-mail* e a senha;
    2. caso o usuário não seja autenticado, o sistema informa a respeito de credenciais inválidas e encerra o caso de uso;
    3. a API autentica o usuário;
    4. o usuário é liberado para visualizar os dados atuais dos sensores da estação;
    5. após a visualização o usuário pode finalizar o caso de uso ou efetuar uma nova consulta se desejar.
  - Fluxo alternativo
    - o usuário desiste de visualizar os dados atuais e cancela o caso de uso clicando no botão voltar.
- **Visualizar histórico**
  - Atores
    - Usuário
  - Pré-condições

  - o usuário deve estar autenticado

– Fluxo de eventos primário

1. o usuário deve efetuar o *login* informando o *username* e a senha;
2. caso o usuário não seja autenticado, o sistema informa a respeito de credenciais inválidas e encerra o caso de uso;
3. a API autentica o usuário;
4. o usuário é liberado qual período cujo histórico será exibido;
5. o usuário seleciona as variáveis a serem exibidas no gráfico;
6. após a visualização do histórico o usuário pode finalizar o caso de uso se desejar.

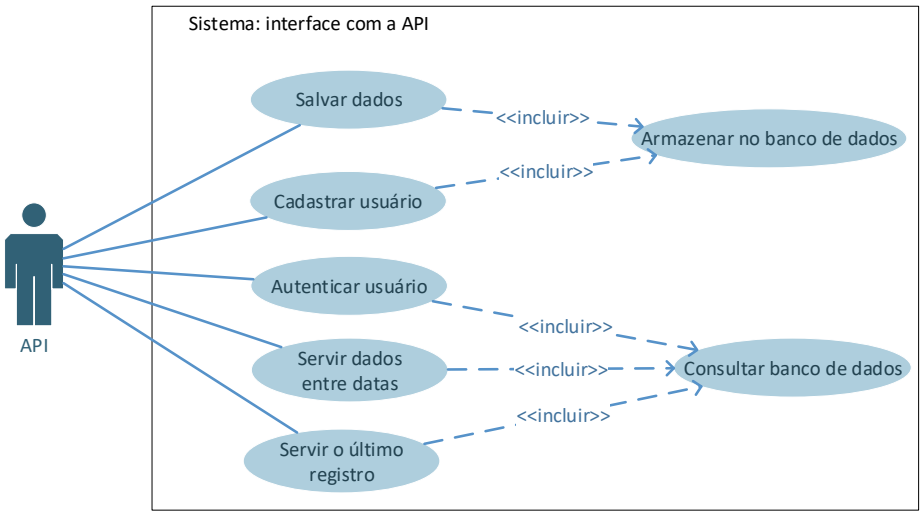
– Fluxo alternativo

1. após a escolha do período de exibição do histórico o usuário pode voltar para a tela anterior e um novo período;
2. o histórico é exibido para o usuário;
3. após a visualização do histórico o usuário pode finalizar o caso de uso ou efetuar uma nova consulta se desejar.

– Fluxo alternativo

1. o usuário desiste de visualizar o histórico e cancela o caso de uso clicando no botão voltar.

Figura 13 – Diagrama de caso de uso do ponto de vista da API



Fonte: O Autor

- **Salvar dados**
  - Atores
    - API
  - Pré-condições
    - o Módulo de captura dos dados realiza uma requisição HTTP solicitando o armazenamento dos dados
  - Fluxo principal
    1. a API recebe a requisição;
    2. a API requisita ao banco de dados a inserção dos dados;
    3. a API responde a requisição HTTP informando o *status* da operação (bem sucedida ou não).
  - Fluxo alternativo
    1. a API não consegue se conectar ao banco de dados, o caso de uso é cancelado.
  - Pós-condições
    - novas informações estão disponíveis no banco de dados para serem consultadas.

- **Cadastrar usuário**
  - Atores
    - API
  - Pré-condições
    - a aplicação de usuário realiza uma requisição HTTP solicitando o armazenamento dos dados
  - Fluxo principal
    1. a API recebe a requisição;
    2. a API requisita ao banco de dados a inserção dos dados do usuário;
    3. o banco de dados verifica a existência de algum registro contendo o mesmo *e-mail*;
    4. a API responde a requisição HTTP informando o *status* da operação (bem sucedida ou não).
  - Fluxo alternativo
    1. a API não consegue se conectar ao banco de dados, o caso de uso é cancelado.
  - Pós-condições

    - o usuário está apto a realizar seu *login*.

- **Autenticar usuário**
  - Atores
    - API
  - Pré-condições
    - a aplicação de usuário realiza uma requisição HTTP solicitando a checagem das credenciais do usuário
  - Fluxo principal
    1. a API recebe a requisição;
    2. a API requisita ao banco de dados a verificação dos dados;
    3. a API responde à requisição de *login* enviando o *token* de acesso do cliente.
  - Fluxo alternativo
    1. a API não consegue se conectar ao banco de dados, então o caso de uso é encerrado.
  - Fluxo alternativo
    1. a API conecta-se ao banco de dados;
    2. a o banco de dados verifica que as credenciais do usuário são inválidas;
    3. a API responde à requisição informando erro.

- **Servir dados entre datas**
  - Atores
    - API
  - Pré-condições
    - verificar qual o período dos dados requeridos pela aplicação cliente através dos parâmetros da requisição HTTP
  - Fluxo principal
    1. a API recebe a requisição;
    2. a API requisita ao banco de dados os dados que foram obtidos no intervalo entre as datas;
    3. a API envia os dados para a aplicação e encerra o caso de uso.
  - Fluxo alternativo
    1. a API não consegue se conectar ao banco de dados, então o caso de uso é encerrado.

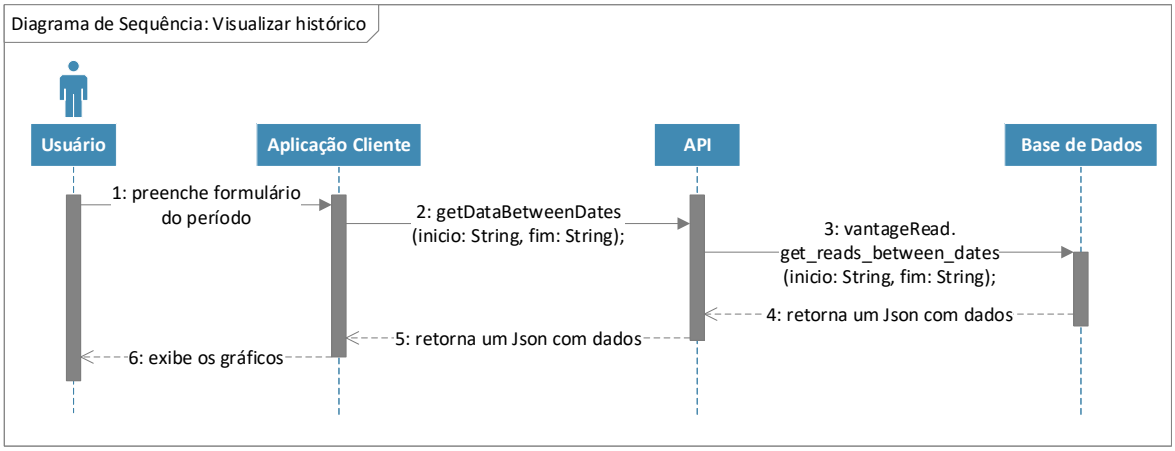
  - Fluxo alternativo
    1. não há dados referente ao período solicitado;
    2. a API responde à requisição com um conjunto de dados vazio;
    3. o caso de uso é encerrado.
  - Pós-condições
    * o usuário pode visualizar as informações consultadas.

- **Servir o último registro**
  - Atores
    * API
  - Pré-condições
    * a aplicação de usuário realiza uma requisição HTTP solicitando os dados mais atuais
  - Fluxo principal
    1. a API recebe a requisição;
    2. a API requisita ao banco de dados o registro mais recente;
    3. a API envia os dados para a aplicação e encerra o caso de uso.
  - Fluxo alternativo
    1. a API não consegue se conectar ao banco de dados, então o caso de uso é encerrado.
  - Pós-condições
    * o usuário pode visualizar as informações consultadas.

4.2.2 Diagrama de sequência

O diagrama de sequência apresenta a colaboração dinâmica entre as entidades do projeto. Diante dele é possível verificar as mensagens trocadas entre os objetos no decorrer do tempo (SAMPAIO; ROCHA NETO, 2005). A sequência de interações entre as entidades componentes do sistema no que tange a consulta de dados do histórico estão representadas pelo diagrama de sequência da figura 14.

Figura 14 – Diagrama de sequência do sistema no contexto de visualização dos dados



Fonte: O Autor

1. O Usuário deseja ver o histórico das variáveis climáticas, então através da interface de usuário escolhe o período ao qual o histórico se refere;
2. A aplicação solicita à API através de uma requisição HTTP contendo o momento de início e o momento do fim do período em seus parâmetros;
3. A API recebe a solicitação e se comunica com a base de dados, então requere as informações quem possuem a data de leitura no intervalo escolhido;
4. A base de dados retorna os dados em formato Json para a API;
5. A API responde à requisição retornando os dados, também em formato Json, para a aplicação cliente;
6. A aplicação cliente renderiza os gráficos utilizando o conjunto de dados obtidos.

O diagrama de sequência para o caso de uso de visualização dos dados atuais possui a lógica de funcionamento idêntica ao diagrama da figura 14, diferenciando-se nos fatos de que o usuário não necessita preencher nenhum formulário, os métodos executados pela aplicação cliente e pela API são diferentes, e a aplicação não gera gráficos para este cenário, como pode ser observado na figura 17e. Os respectivos métodos utilizados pela aplicação cliente e pela API neste caso são mostrados abaixo.

Código 2 – Método executado para a requisição dos dados atuais pela aplicação cliente

```
1 getRead();
```

Fonte: O autor

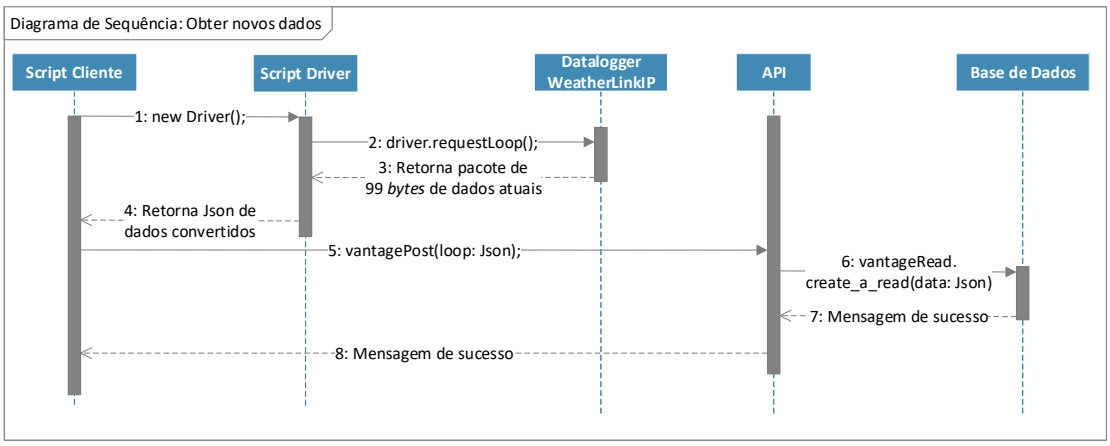
Código 3 – Método executado pela API para a busca dos dados atuais

```
1 vantageRead.getLastRead();
```

Fonte: O autor

A figura 15 demonstra o caso de obtenção dos dados da estação por parte do sistema, onde a API recupera dos dados provenientes da estação e os insere no banco de dados para futuras consultas.

Figura 15 – Diagrama de sequência do sistema no contexto de obtenção de dados



Fonte: O Autor

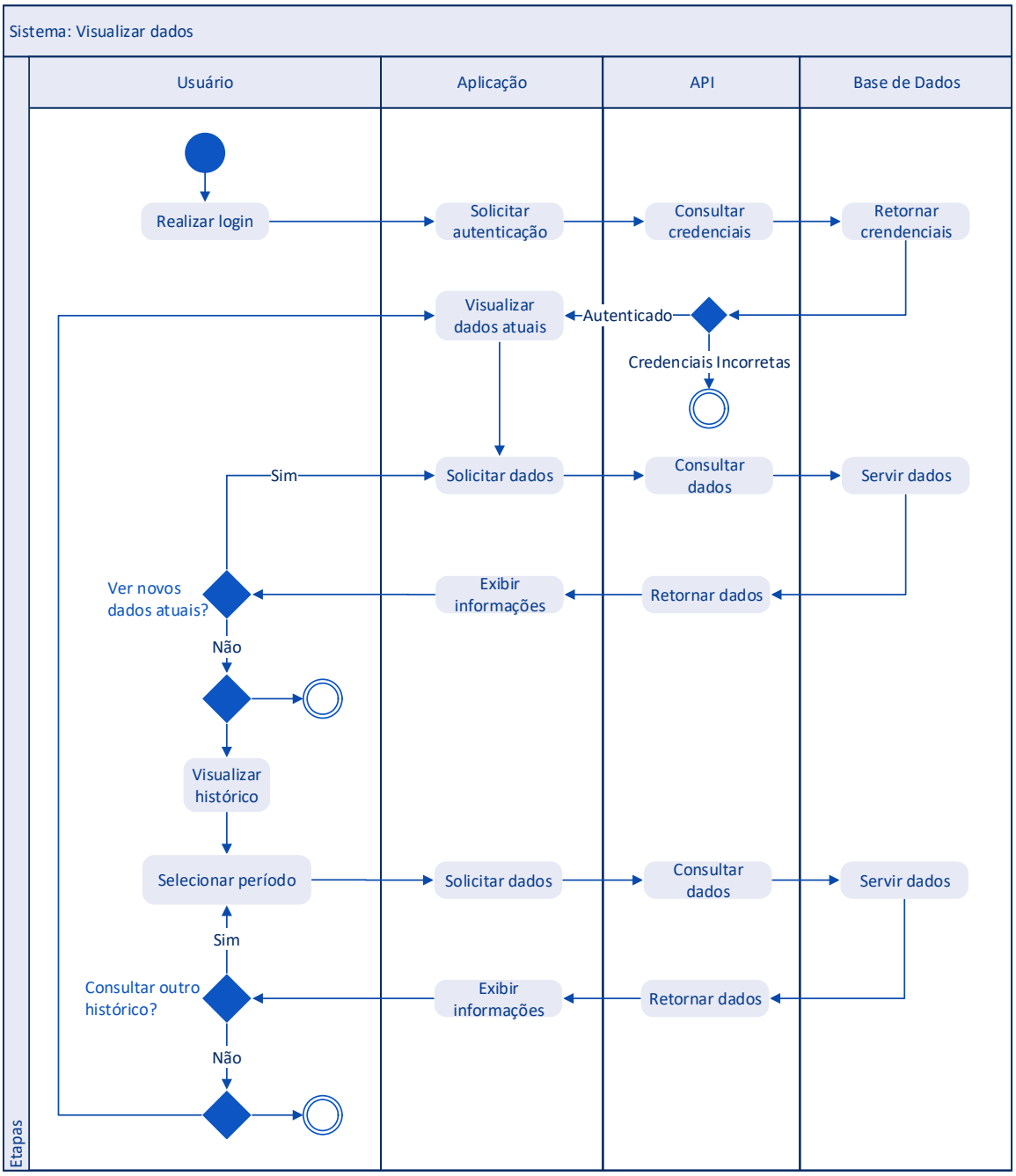
1. O Script Cliente inicia uma nova instância do Script Driver;
2. O Script Driver envia através de uma conexão TCP o comando de LOOP para o *datalogger* da estação;
3. O *datalogger* responde ao comando de LOOP retornando um pacote de 99 *bytes* contendo os dados da última leitura da estação;

4. O Script Driver retorna para o Script Client um Json contendo os dados convertidos para as unidades mais usuais;
5. O Script Cliente envia para a API, através de uma requisição HTTP, os dados coletados;
6. A API solicita à base de dados a inserção das informações;
7. A base de dados confirma a inserção das informações para a API;
8. A API confirma para o Script Cliente o sucesso da operação.

4.2.3 Diagrama de atividades

Os diagramas de atividade são semelhantes a fluxogramas, porque englobam a direção das informações entre as ações em uma atividade. Os estados no diagrama de atividades mudam para uma próxima etapa quando uma ação é executada (VENTURA, 2016). O diagrama de atividades da figura 16 ilustra o fluxo de controle de todo o sistema de consulta, desde a observação dos dados atuais, até a visualização do histórico das variáveis meteorológicas.

Figura 16 – Diagrama de atividades do sistema



Fonte: O Autor

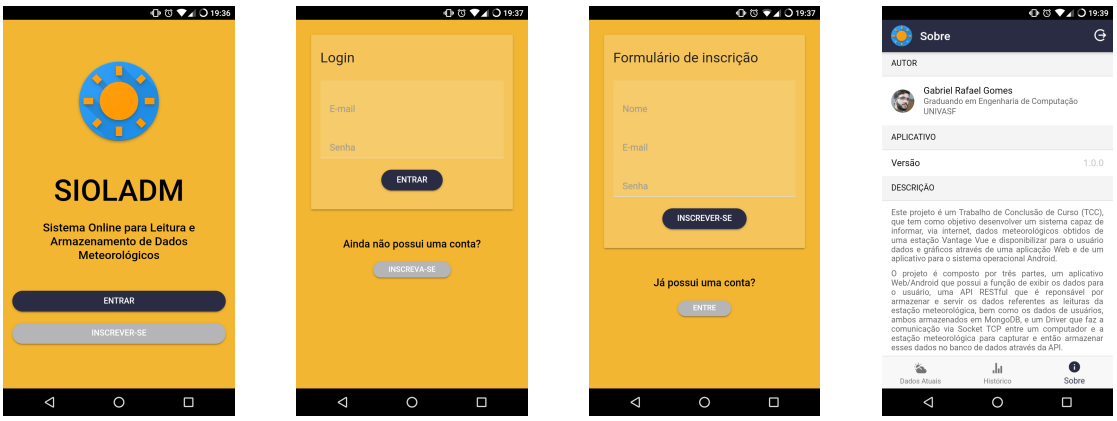
4.2.4 Interface com o usuário

A aplicação cliente ou interface de usuário contém sete telas, sendo: tela de início, tela de *login*, tela de cadastro, tela de dados atuais, tela de seleção de período, tela de histórico e tela de sobre (ver figura 17). A tela inicial a ser exibida assim que o usuário é

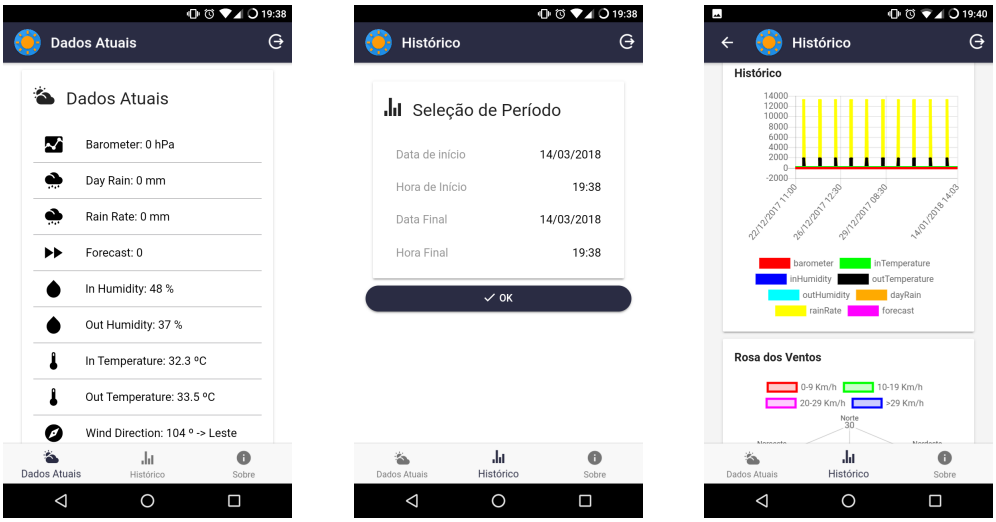
autenticado é a tela de dados atuais, a partir de então o usuário possui a liberdade de permanecer na mesma, consultar o histórico informando os dados necessários para tal ou efetuar o *logoff*.

Figura 17 – Telas da aplicação cliente

(a) Abertura **(b)** *Login* **(c)** Cadastro **(d)** Sobre



(e) Dados atuais **(f)** Seleção de período **(g)** Exibir histórico



Fonte: O Autor

Além disso, será sempre permitido ao usuário retornar para a tela anterior através do botão voltar, com exceção das telas de *login* e cadastro, que só serão acessíveis se o usuário realizar *logoff* ou reinstalar a aplicação. O fluxo de funcionamento da aplicação cliente é representado pelo diagrama de atividades da figura 16 na subseção 4.2.3.

A figura 17b mostra a tela utilizada no caso de uso de autenticação do usuário. A figura 17e mostra a tela de visualização dos dados atuais. As figuras, 17f e 17g correspondem ao caso de uso de visualização de histórico.

4.2.5 A API do sistema

Os URIs da API do sistema estão mostrados na tabela 3 abaixo. A tabela divide-se em três partes, a primeira representa as ações que podem ser executadas por um usuário não autenticado, a segunda evidencia as ações possíveis para um usuário que efetuou *login* e a terceira expõe as ações internas do sistema.

Tabela 3 – API RESTful do sistema.

Ator	Método	URI	Descrição
Usuário não autenticado	POST	/auth/login	Autentica o usuário utilizando os parâmetros passados no corpo da requisição;
	POST	/auth/register	Cria um novo usuário com os parâmetros passados no corpo da requisição;
Usuário autenticado	GET	/read/last	Retorna o último documento da base de dados;
	GET	/read/:readMoment1/:readMoment2	Retorna os dados contidos período especificado pelos parâmetros da requisição;
Sistema	POST	/read	Cria um novo documento no banco com os valores das variáveis passados no corpo da requisição;

Fonte: O Autor

4.2.6 Modelos de dados do sistema

O banco de dados escolhido para a implementação do projeto é não-relacional baseado em documentos. Os documentos podem conter vários pares chave-valor, ou vários pares chave-vetor, ou até vários documentos encadeados. A estrutura do documento é flexível, o que permite alterações e inserções futuras de dados novos em uma base preexistente (MONGODB, Inc., 2017). Um dos modelos de dados adotado para o projeto é mostrado na figura 18 abaixo, onde de um lado têm-se as chaves representando os dados fornecidos pela estação e do outro seus respectivos valores.

Os valores de cada chave adotada para o modelo de documentos são providos pelo *datalogger* da estação. As chaves possuem o valor obtido diretamente dos sensores.

Figura 18 – Modelo de documento das leituras do sistema

```
{
   “_id”: <ID_DA_LEITURA>,
   “barometer”: BAROMETRO,
   “inTemperature”: TEMPERATURA_INTERNA,
   “inHumidity”: UMIDADE_INTERNA,
   “outTemperature”: TEMPERATURA_EXTERNA,
   “outHumidity”: UMIDADE_EXTERNA,
   “windSpeed”: VELOCIDADE_DO_VENTO,
   “windDirection”: DIRECAO_DO_VENTO,
   “dayRain”: PLUVIOSIDADE_DIARIA,
   “rainRate”: PLUVIOSIDADE_MEDIA,
   “forecast”: ICONE_PREVISÃO,
   “readDate”: DATA_HORA_LEITURA
}
```

Fonte: O Autor

O outro modelo de dados do sistema é referente aos dados de usuário, como indicados na figura 19. Os valores desses dados são preenchidos no momento do cadastro do usuário. Neste momento o sistema criptografa a senha enviada, por meio do algortimo *bcrypt*, e salva somente a criptografia no banco.

Figura 19 – Modelo de documento dos usuários do sistema

```
{
   “_id”: <ID_DO_USUARIO>,
   “hash_password”: SENHA_CRIPTOGRAFADA,
   “fullName”: NOME_COMPLETO,
   “email”: EMAIL_USUARIO,
   “created”: DATA_DE_CADASTRO
}
```

Fonte: O Autor

5 CONCLUSÃO E TRABALHOS FUTUROS

REFERÊNCIAS

ALECRIM, E. **Usando cron e crontab para agendar tarefas**. 2005. Disponível em: <https://www.vivaolinux.com.br/artigo/Usando-cron-e-crontab-para-agendar-tarefas>. Acesso em: Março de 2018. Citado na página 38.

ANDROID. **Android Developers**. 2017. Disponível em: <https://source.android.com/setup/>. Acesso em: Fevereiro de 2018. Citado na página 25.

ANDROID. **The Android Source Code**. 2017. Disponível em: <https://developer.android.com>. Acesso em: Agosto de 2017. Citado 2 vezes nas páginas 24 e 25.

ARAUJO, E. L. et al. **Levantamento e flutuação populacional de mocas-das-frutas (Diptera: Tephritidae) em goiaba Psidium guajava L., No município de Russas (CE)**. *Revista Caatinga*, Universidade Federal Rural do Semi-Árido, v. 21, n. 1, 2008. Citado na página 22.

ARCURI, A. **RESTful API Automated Test Case Generation**. In: IEEE. *Software Quality, Reliability and Security (QRS)*. [S.l.], 2017. p. 9–20. Citado na página 28.

BERNARDO, K. **Iterativo e incremental: Suas definições**. 2015. Disponível em: <https://www.culturaagil.com.br/iterativo-e-incremental-suas-definicoes/>. Acesso em: Março de 2018. Citado na página 32.

BRAGA, I. A.; VALLE, D. **Aedes aegypti:** histórico do controle no brasil. *Epidemiologia e Serviços de Saúde,*, Brasília, v. 16, n. 2, p. 113–118, jun. 2007. Citado na página 15.

CALORE, R. A. et al. **Fatores climáticos na dinâmica populacional de Anastrepha spp.(Diptera: Tephritidae) e de Scymnus spp.(Coleoptera: Coccinellidae) em um pomar experimental de goiaba (Psidium guajava L.)**. *Revista Brasileira de Fruticultura*, Sociedade Brasileira de Fruticultura, p. 67–74, 2013. Citado 2 vezes nas páginas 21 e 22.

CASA DA CONSULTORIA. **Modelo Cascata: O que é e como funciona?** 2017. Disponível em: <http://casadaconsultoria.com.br/modelo-cascata/>. Acesso em: Março de 2018. Citado na página 32.

CONTE, T.; MENDES, E.; TRAVASSOS, G. H. **Processos de desenvolvimento para aplicações web:** uma revisão sistemática. In: *Proceedings of the 11th Brazilian Symposium on Multimedia and Web (WebMedia 2005)*. [S.l.: s.n.], 2005. v. 1, p. 107–116. Citado na página 26.

DALMASSO, I. et al. **Survey, comparison and evaluation of cross platform mobile application development tools**. In: IEEE. *Wireless Communications and Mobile Computing Conference (IWCMC), 2013 9th International*. [S.l.], 2013. p. 323–328. Citado na página 27.

DAVIS INSTRUMENTS. **WeatherLinkIP™ for Vantage Pro® and Vantage Pro2™**. 2008. Disponível em: <http://www.davisnet.com/product_documents/weather/spec_sheets/6555_spec_WeatherLinkIP.pdf>. Acesso em: Agosto de 2017. Citado na página 21.

DAVIS INSTRUMENTS. **Vantage Vue ™:** Manual de Instalação do Conjunto de Sensores Integrados. 2009. Disponível em: <http://www.agrosystem.com.br/static/files/produtos/manual_instalacao_iss_vantagevue_color_rgb_q1-1455821923-t.pdf>. Acesso em: Agosto de 2017. Citado na página 19.

DAVIS INSTRUMENTS. **Vantage Vue ™:** Manual do Console. 2009. Disponível em: <http://www.agrosystem.com.br/static/files/produtos/manual_instalacao_iss_vantagevue_color_rgb_q1-1455821923-t.pdf>. Acesso em: Agosto de 2017. Citado na página 20.

DAVIS INSTRUMENTS. **Vantage Pro ™, Vantage Pro2 ™ and Vantage Vue ™ Serial Communication Reference Manual**. 2013. Disponível em: <http://www.davisnet.com/support/weather/download/VantageSerialProtocolDocs_v261.pdf>. Acesso em: Outubro de 2017. Citado na página 58.

DAVIS INSTRUMENTS. **Davis Instruments**. 2017. Disponível em: <http://www.davisnet.com/solution/vantage-vue/>. Acesso em: Agosto de 2017. Citado 2 vezes nas páginas 19 e 20.

DEPRADINE, C.; LOVELL, E. **Climatological variables and the incidence of Dengue fever in Barbados**. *International journal of environmental health research*, Taylor & Francis, v. 14, n. 6, p. 429–441, 2004. Citado na página 22.

FIELDING, R. T.; TAYLOR, R. N. ***Architectural styles and the design of network-based software architectures***. [S.l.]: University of California, Irvine Doctoral dissertation, 2000. Citado na página 28.

FILHO, W. de P. P. *Engenharia de Software: fundamentos, métodos e padrões*. [S.l.]: LTC, 2003. v. 2. Citado na página 32.

FRATERNALI, P.; PAOLINI, P. **A conceptual model and a tool environment for developing more scalable, dynamic, and customizable web applications**. *Advances in Database Technology—EDBT'98*, Springer, p. 419–435, 1998. Citado na página 26.

GANDHEWAR, N.; SHEIKH, R. **Google Android:** an emerging software platform for mobile devices. *International Journal on Computer Science and Engineering*, v. 1, n. 1, p. 12–17, 2010. Citado na página 24.

GARCIA, F. R. M.; CORSEUIL, E. **Influência de fatores climáticos sobre moscas-das-frutas (Diptera: Tephritidae) em pomares de pessegueiro em Porto Alegre, Rio Grande do Sul.** *Revista da FZVA*, v. 5, n. 1, 1998. Citado na página 22.

GOOGLE ANGULAR. **Angular**. 2017. Disponível em: <https://angular.io>. Acesso em: Agosto de 2017. Citado na página 35.

HOPP, M. J.; FOLEY, J. A. **Global-scale relationships between climate and the dengue fever vector, Aedes aegypti**. *Climatic change*, Springer, v. 48, n. 2, p. 441–463, 2001. Citado 4 vezes nas páginas 15, 22, 23 e 24.

HSIEH, C.-Y.; HSIEH, H.-A.; CHENG, Y. C. **A method for web application data migration based on RESTful API: A case study of ezScrum**. In: IEEE. *Applied System Innovation (ICASI), 2016 International Conference on*. [S.l.], 2016. p. 1–4. Citado na página 29.

IMPERATO, R.; RAGA, A. **Técnica do Inseto Estéril**. *Agência Paulista de Tecnologia dos Agronegócios*, Campinas, p. 1–16, out. 2015. Citado na página 18.

IONIC FRAMEWORK. **Ionic Framework**. 2017. Disponível em: <https://ionicframework.com>. Acesso em: Agosto de 2017. Citado na página 35.

KUROSE, J.; ROSS, K. **Redes de Computadores e a Internet:** Uma abordagem top-down. São Paulo: Pearson Addison Wesley, 2005. Citado 2 vezes nas páginas 29 e 30.

MALAVASI, A.; ZUCCHI, R. A. **Moscas-das-frutas de importância econômica no Brasil**. Holos, 2000. Citado na página 15.

MEDEIROS, H. **Introdução ao Modelo Cascata**. 2017. Disponível em: <https://www.devmedia.com.br/introducao-ao-modelo-cascata/29843>. Acesso em: Março de 2018. Citado na página 32.

MEDEIROS, H. **Introdução a Requisitos de Software**. 2017. Disponível em: <https://www.devmedia.com.br/introducao-a-requisitos-de-software/29580>. Acesso em: Março de 2018. Citado na página 32.

MELLOR, D. **Crontab Files**. 2014. Disponível em: <https://www.gnu.org/software/mcron/manual/html_node/Crontab-file.html>. Acesso em: Março de 2018. Citado na página 38.

MENDES, E.; MOSLEY, N.; COUNSELL, S. **Investigating Web size metrics for early Web cost estimation**. *Journal of Systems and Software*, Elsevier, v. 77, n. 2, p. 157–172, 2005. Citado na página 26.

MERENCIA, L. **node-cron**. 2017. Disponível em: <https://www.npmjs.com/package/node-cron>. Acesso em: Março de 2018. Citado na página 38.

MONGODB, Inc. **MongoDB**. 2017. Disponível em: <https://www.mongodb.com>. Acesso em: Agosto de 2017. Citado 2 vezes nas páginas 36 e 50.

MOSCAMED. **Folder Ciência, Técnologia e Inovação para a Agricultura no Brasil**. Juazeiro: [s.n.], 2003. Citado 2 vezes nas páginas 18 e 19.

MOSCAMED. **Folder Institucional**. Juazeiro: [s.n.], 2003. Citado 3 vezes nas páginas 15, 18 e 19.

MOSCAMED. **Folder Linhas de Ação**. Juazeiro: [s.n.], 2010. Citado na página 18.

MOSCAMED. **Folder Áreas de Atuação**. Juazeiro: [s.n.], 2010. Citado na página 18.

PALMIERI, M.; SINGH, I.; CICCHETTI, A. **Comparison of cross-platform mobile development tools**. In: IEEE. *Intelligence in Next Generation Networks (ICIN), 2012 16th International Conference on*. [S.l.], 2012. p. 179–186. Citado na página 27.

PARANHOS, B. **Moscas-das-frutas que oferecem riscos à fruticultura brasileira**. In: IN: SIMPÓSIO INTERNACIONAL DE VITIVINICULTURA, 1.; FEIRA NACIONAL DA AGRICULTURA IRRIGADA-FENAGRI, EMBRAPA SEMI-ÁRIDO. Petrolina, 2008. Citado 2 vezes nas páginas 15 e 21.

PASTOR, O. **Fitting the pieces of the Web engineering puzzle**. *Invited Talk, XVIII Simpósio Brasileiro de Engenharia de Software (SBES), Brasília*, 2004. Citado na página 26.

PRESSMAN, R. S. **What a tangled web we weave [web engineering]**. *IEEE Software*, IEEE, v. 17, n. 1, p. 18–21, 2000. Citado na página 26.

RAGA, A.; SOUSA FILHO, M. **Manejo e monitoramento de moscas-das-frutas**. *Reunião Itinerante de Fitossanidade do Instituto Biológico*, v. 3, p. 87–99, 2000. Citado na página 15.

RAJ, C. R.; TOLETY, S. B. **A study on approaches to build cross-platform mobile applications and criteria to select appropriate approach**. In: IEEE. *India Conference (INDICON), 2012 Annual IEEE*. [S.l.], 2012. p. 625–629. Citado 3 vezes nas páginas 27, 28 e 35.

RIBEIRO, A. F. et al. **Associação entre incidência de dengue e variáveis climáticas**. *Revista de Saúde Pública*, SciELO Public Health, v. 40, n. 4, p. 671–676, 2006. Citado 3 vezes nas páginas 15, 22 e 23.

ROYCE, W. W. Managing the development of large systems: Concepts and techniques. In: *9th International Conference on Software Engineering. ACM*. [S.l.: s.n.], 1970. p. 328–38. Citado na página 31.

SAHA, A. K. **A developer's first look at Android**. *Linux For You,(January)*, p. 48–50, 2008. Citado na página 25.

SAMPAIO, M. C.; ROCHA NETO, E. **Diagramas de Sequência**. 2005. Disponível em: <http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/uml/diagramas/interacao/sequencia.htm>. Acesso em: Setembro de 2017. Citado na página 44.

SOMMERVILLE, I. ***Engenharia de Software***. 9. ed. [S.l.: s.n.], 2011. Citado 2 vezes nas páginas 32 e 39.

TILKOV, S.; VINOSKI, S. **Node.js:** using javascript to build high-performance network programs. *IEEE Internet Computing*, IEEE, v. 14, n. 6, p. 80–83, 2010. Citado na página 36.

TORRES, G. ***Redes de Computadores Curso Completo***. 1. ed. Rio de Janeiro: Axcel Books do Brasil Editora Ltda, 2001. Citado na página 29.

TORRES, J. D. et al. **Aquisição de dados meteorológicos através da plataforma Arduino:** construção de baixo custo e análise de dados. *Scientia Plena*, v. 11, n. 2, 2015. Citado na página 19.

VENTURA, P. **Entendendo o Diagrama de Atividades da UML**. 2016. Disponível em: <http://www.ateomomento.com.br/uml-diagrama-de-atividades/>. Acesso em: Setembro de 2017. Citado na página 47.

ANEXO A – COMANDOS SERIAIS DA ESTAÇÃO METEOROLÓGICA *VANTAGE VUE™*

Tabela 4 – Comandos seriais suportados pela estação meteorológica *Vantage Vue™*

Instrução	Descrição
Comandos de teste	
TESTE	Envia a *string* "TEST\n"de volta
WRD	Responde com o tipo de estação meteorológica
RXCHECK	Responde com o diagnóstico do Console
RXTEST	Muda a tela do console de *"Receiving from"* para tela de dados atuais
VER	Responde com a data do *firmware*
RECEIVERS	Responde com a lista das estações que o console "enxerga"
NVER	Responde com a versão do *firmware*
Comandos de dados atuais	
LOOP	Responde com a quantidade de pacotes especificada a cada 2s
LPS	Responde a cada 2s com a quantidade de pacotes diferentes especificada
HILOWS	Responde com todo os dados de *high/low*
PUTRAIN	Seta a quantidade anual de precipitação
PUTET	Seta a quantidade anual de evapotranspiração
Comandos de *download*	
DMP	Faz o *download* de todo o arquivo de memória
DMAFT	Faz o *download* de todo o arquivo de memória após a data especificada
Comandos da EEPROM	
GETEE	Lê toda a memória EEPROM
EEWR	Escreve um *byte* de dados à partir do endereço especificado
EERD	Lê a quantidade de dados especificada iniciando no endereço especificado
EEBWR	Escreve os dados na EEPROM
EEBRD	Lê os dados da EEPROM
Comandos de calibração	
CALED	Envia os dados da temperatura e umidade corrente para atribuir à calibração
CALFIX	Atualiza o *display* quando os números de calibração mudam
BAR	Seta os valores da elevação e o *offset* do barômetro quando a localização é alterada
BARDATA	Mostra os valores atuais da calibração do barômetro

Continua na próxima página

Tabela 4 – Continuação da página anterior

Instrução	Descrição
	Comandos de limpeza
CLRLOG	Limpa todo o arquivo de dados
CLRALM	Limpa todos os limiares dos alarmes
CLRCAL	Limpa todos os *offsets* da calibração da temperatura e da umidade
CLRGRA	Limpa o gráfico do console
CLRVAR	Limpa o valor da precipitação ou da evapotranspiração
CLRHIGHS	Limpa todos os valores de pico diários, mensais ou anuais
CLRLOWS	Limpa todos os valores de mínimos diários, mensais ou anuais
CLRBITS	Limpa os *bits* de alarme ativos
CLRDATA	Limpa todos os dados atuais
	Comandos de configuração
BAUD	Atribui o valor do *baudrate* do console
SETTIME	Define a data e a hora do console
GAIN	Define o ganho do receptor de rádio
GETTIME	Retorna a hora e a data atual do console
SETPER	Define o intervalo de arquivamento
STOP	Desabilita a criação dos registros
START	Habilita a criação dos arquivos
NEWSETUP	Reinicia o console após alguma configuração nova
LAMPS	Liga ou desliga as lâmpadas do console

Fonte – DAVIS INSTRUMENTS (2013) (Traduzido).