UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
CURSO DE ESPECIALIZAÇÃO EM TECNOLOGIA JAVA

ROBINSON DONIZETE DA SILVEIRA SANTOS

DESENVOLVIMENTO DE UM SISTEMA WEB PARA MONITORAMENTO DE SENSORES

MONOGRAFIA DE ESPECIALIZAÇÃO

PATO BRANCO
2015

ROBINSON DONIZETE DA SILVEIRA SANTOS

DESENVOLVIMENTO DE UM SISTEMA WEB PARA MONITORAMENTO DE SENSORES

Trabalho de Conclusão de Curso, apresentado ao III Curso de Especialização em Tecnologia Java, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, como requisito parcial para obtenção do título de Especialista.

Orientador: Prof. Dr. Fábio Favarim

PATO BRANCO
2015

DESENVOLVIMENTO DE UM SISTEMA WEB PARA MONITORAMENTO DE SENSORES

Por

Robinson Donizete Da Silveira Santos

Esta monografia foi apresentada às 16h00 do dia 30 de novembro de 2015 como requisito parcial para a obtenção do título de ESPECIALISTA, no III Curso de Especialização em Tecnologia Java, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. O acadêmico foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

Prof. Dr. Fábio Favarim
Orientador
UTFPR – Campus Pato Branco

Prof. Msc. Vinicius Pegorini
Banca
UTFPR – Campus Pato Branco

Prof. Dr. Pablo Gauterio Cavalcanti
Banca
UTFPR – Campus Pato Branco

Prof. Msc. Robison Cris Brito
Coordenador do curso de Especialização
UTFPR – Campus Pato Branco

Aos amigos(as), familiares, professores(as) e todos aqueles(as) que de uma forma ou outra me ajudaram a trilhar o caminho do conhecimento.

AGRADECIMENTOS

Em especial aos professores Fábio Favarim e Robison Cris Brito pela paciência, confiança e auxilio.

As pessoas que são loucas o suficiente
para achar que podem mudar o mundo
são aquelas que o mudam.

Comercial “Pense diferente” da Apple, 1997

RESUMO

SILVEIRA SANTOS, Robinson Donizete da Silveira Santos. Desenvolvimento de um sistema web para monitoramento de sensores. 2015. 51 f. Monografia (Trabalho de especialização) – Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2015.

Existe um grande e crescente número de dispositivos inteligentes conectados à Internet, fornecendo dados para as mais variadas aplicações. Isso deve ao que se conhece por Internet das Coisas, a qual permite que qualquer dispositivo possa ser conectado à Internet. O uso desses dispositivos reflete a grande tendência do mundo moderno, coletar, armazenar e processar a maior quantidade possível de dados afim de produzir informação relevante. Esse contexto cria um momento de oportunidade em um mercado de biliões de dólares. Cientes disso *startups* e grandes empresas estão desenvolvendo plataformas para coleta, armazenamento e analise de dados. Esse trabalho se insere neste contexto fornecendo uma alternativa, um sistema *web* desenvolvido na plataforma Java com o auxilio do Play Framework, o qual permite a coleta, armazenamento e visualização de dados de sensores.

Palavras-chave: Dispositivos inteligentes, Sistema *web*, Plataforma Java, Play Framework.

ABSTRACT

SILVEIRA SANTOS, Robinson Donizete da Silveira Santos. Development of a web system for sensor monitoring. 2015. 51 f. Monografia (Trabalho de especialização) – Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2015.

There is a large and growing number of intelligent devices connected to the Internet, providing data for the most varied applications. This due to what is known as the Internet of Things, which allows any device to be connected to the Internet. The use of these devices reflects a major trend of the modern world, collect, store and process the greatest possible amount of data in order to produce relevant information. This context creates a moment of opportunity in a billion dollar market. Aware that startups and large companies are developing platforms for collection, storage and data analysis. This work is inserted in this context providing an alternative, a web system developed on the Java platform with the Play Framework aid, which allows the collection, storage and sensor data visualization.

Keywords: Smart devices, Web System, Java Plataform, Play Framework.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTAGENS DE CÓDIGOS

LISTA DE SIGLAS

API	*Application Programming Interface*
BSD	*Berkeley Software Distribution*
CSS	*Cascading Style Sheets*
DER	*Diagrama Entidade-Relacionamento*
EE	*Enterprise Edition*
EPL	*Eclipse Public License*
GPL	*GNU General Public License*
HTTP	*Hypertext Transfer Protocol*
IDC	*Internacional Data Corporation*
IDE	*Integrated Development Environment*
I/O	*Input/Output*
JSON	*Javascript Object Notation*
MIT	*Massachusetts Institute of Technology*
MVC	*Model-View-Controller*
REST	*Representational State Transfer*
SSL	*Secure Sockets Layer*
TCP	*Transmission Control Protocol*
UDP	*User Datagram Protocol*
UIT	*União Internacional de Telecomunicações*
UUID	*Universally Unique Identifier*
XML	*Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

As considerações inicias, os objetivos e a justificativa para elaboração do trabalho são apresentadas nesse capítulo. Os capítulos subsequentes são apresentados ao final do mesmo.

1.1 CONSIDERAÇÕES INICIAIS

Atualmente é inegável a grandeza do mercado de dispositivos conectados a Internet, seu estrondoso crescimento e as infinitas possibilidades de uso desses *devices*.

> De acordo com o IDC (International Data Corporation), no ano de 2014 somente o mercado brasileiro de aparelhos conectados na internet movimentou em torno de 2 bilhões de dólares. Ainda segundo a IDC esses dispositivos movimentarão cerca de US$ 1,7 trilhão no mundo até o ano de 2020, sendo que mais de 29,5 milhões de dispositivos estarão conectados até o final desta década (CANALTECH, 2015).

A redução no tamanho dos dispositivos eletroeletrônicos, assim como a redução dos seus custos de produção tem possibilitado o uso desses dispositivos em diversos ambientes e aplicações. Além disso, a possibilidade desses dispositivos se comunicarem via rede, e consequentemente conectados a Internet, é denominada de "Internet das Coisas" (do inglês *Internet of Things -IoT*) (DUQUENNOY et al., 2009).

Esse cenário criou a oportunidade e pequenas e grandes empresas iniciaram o desenvolvimento de serviços *web* e/ou plataformas para o gerenciamento dos dados gerados por estes dispositivos. Essas plataformas permitem o envio, a coleta, o armazenamento e a análise dos dados coletados.

Este trabalho propõe uma alternativa para o gerenciamento dos dados de dispositivos inteligentes, por meio do desenvolvimento de um aplicativo *web* construído utilizando a plataforma Java e os recursos fornecidos pelo Play Framework.

1.2 OBJETIVOS

1.2.1 Objetivo Geral

Desenvolver um protótipo de um sistema *web* para monitoramento de sensores que permita a coleta, o envio, a persistência e a visualização dos dados coletados.

1.2.2 Objetivos Específicos

- Permitir o envio dos dados de sensores através de requisições HTTP;
- Permitir a visualização dos dados coletados por meio de gráficos;
- Explorar as funcionalidades fornecidas pelo Play Framework.

1.3 JUSTIFICATIVA

A ideia de Internet das Coisas é permitir que qualquer dispositivo físico se torne um dispositivo virtual, isto é, que possa ser acessado e/ou controlado via Internet. Esse trabalho visa consolidar os conceitos vistos nas disciplinas da especialização em Java com tecnologias de desenvolvimento emergentes.

O desenvolvimento de um produto para Internet precisa atender aos novos requisitos de software. As ferramentas devem propiciar um ambiente de alta produtividade, flexibilidade e que respeite os padrões da *web*. Essas ferramentas que auxiliam o programador no desenvolvimento são conhecidas como *framework*.

O Play Framework (BOAGLIO, 2014) é um dos muitos *frameworks* disponíveis para o desenvolvimento de aplicações para Internet que utilizam a linguagem e a plataforma Java, no entanto, ele apresenta certas características que o fazem distinguir dos demais. Possui uma arquitetura simples e leve de modo que não implementa o padrão Java EE, não mantêm estado em requisições, servidor HTTP integrado, API de serviços REST, suporte a I/O assíncrono, cache integrado, persistência de dados e sistema de *build* próprio, dentre outras.

Esse trabalho busca explorar esse *framework* no desenvolvimento da aplicação para coleta, armazenamento e visualização dos dados oriundos de sensores.

1.4 ESTRUTURA DO TRABALHO

A estrutura do trabalho está organizada em capítulos. Este é o primeiro e fornece a visão geral do texto no sentido de sua estrutura. O segundo capítulo descreve o embasamento técnico para o trabalho através do referencial teórico, na sequência o terceiro capítulo apresenta as ferramentas e métodos utilizados no desenvolvimento. O quarto capítulo de nome Resultados, apresenta o sistema *web* em si, descreve o processo de modelagem e fornece uma visão geral das principais funcionalidades desenvolvidas, detalhando quando necessário aspectos técnicos de sua implementação. Por fim, o quinto capítulo conclui o trabalho, apresentando os resultados obtidos, a análise se os objetivos foram alcançados e as considerações sobre as possibilidades de continuidade do projeto em trabalhos futuros.

2 REFERENCIAL TEÓRICO

O embasamento técnico e teórico para o desenvolvimento do proposto trabalho é apresentado nesta seção.

2.1 SENSORES INTELIGENTES

Podemos definir como sensor todo dispositivo que permite mensurar de forma digital ou analógica um estímulo físico ou químico.

> Transdutor é o nome dado para um sensor ou atuador, que por sua vez são dispositivos de detecção e atuação em um determinado processo, respectivamente. Esses dispositivos são protagonistas no cenário da automação industrial, desempenhando um papel de suma importância também em outras áreas, como biomédica e monitoramento ambiental, por exemplo.
>
> Com o advento dos microcontroladores e a grande disponibilidade de ferramentas e recursos para o processamento de sistemas digitais, foi possível introduzir uma elevada capacidade de computação aos transdutores (Chong & Kumar, 2003). Dessa afirmação nasce o termo transdutor inteligente, que é a integração de um sensor analógico ou digital ou um atuador, uma unidade de processamento e uma interface de rede (Elmenreich &Pitzek, 2003).
>
> Para um exemplo de sensoriamento, um transdutor inteligente transforma os sinais brutos deste sensor em uma representação digital padronizada, transmitindo este sinal digital para os seus usuários através de um protocolo de comunicação digital padronizado. Para o caso de um atuador, o transdutor inteligente obedece a comandos padronizados e transforma-os em sinais de controle para o atuador (Elmenreich & Pitzek, 2003).
>
> O conceito de redes de sensores inteligentes não está somente ligado à troca de informações de um transdutor para outro, mas também com o compartilhamento e disponibilização das informações muitas vezes em tempo real. O funcionamento dos sensores inteligentes em rede através de interfaces padronizadas torna possível o compartilhamento de informações e recursos de um sistema, a supervisão de um processo completo, além de acesso e monitoramento remoto das variáveis envolvidas.
>
> Com o crescente avanço tecnológico e o surgimento de microcontroladores modernos, que utilizam circuitos integrados facilmente encontrados no mercado, capazes de prover uma comunicação padrão, tornou-se possível a construção de transdutores inteligentes de baixo custo. Dessa forma, os transdutores podem vir em grandes variedades com capacidades diferentes e de diferentes fabricantes. (LEÃO, 2007).

2.2 PLAY FRAMEWORK

Play Framework é um dos muitos *frameworks* disponíveis para o desenvolvimento de aplicações para Internet que utilizam a linguagem e a plataforma Java. O projeto teve início no ano de 2007 pela empresa Zenexity (hoje Zengularity) e no ano de 2009 tornou-se um projeto de código fonte aberto, atualmente é mantido pelas empresas Zengularity (http://www.zengularity.com) e Typesafe (http://www.typesafe.com), possui uma grande e ativa comunidade de usuários e é utilizada por empresas como Linkedin e The Guardian.

2.2.1 COMPONENTES DA ARQUITETURA

O Play foi projetado não para ser apenas um *framework*, mas sim uma solução completa para o desenvolvimento *web*, que envolve servidor HTTP, *cache*, persistência de dados, sistema de *build* e vários outros componentes. A grande premissa do projeto é tornar simples e altamente produtivo o desenvolvimento de aplicações para Internet utilizando a plataforma Java. A Figura 1 mostra a pilha de componentes do Play Framework.

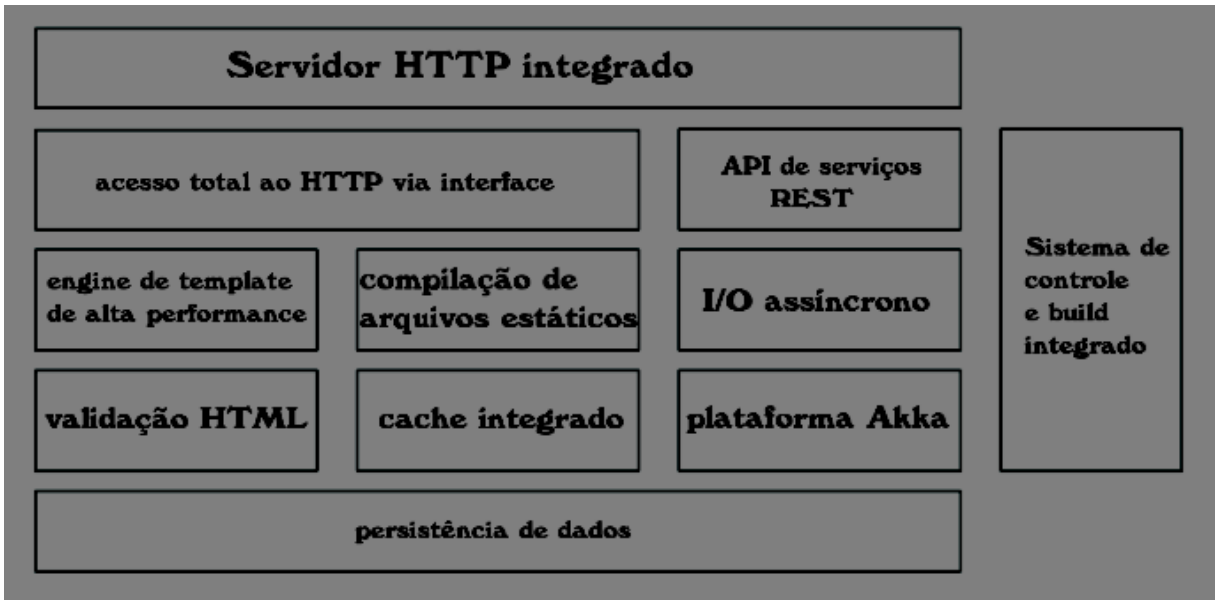

Figura 1: Pilha de componentes do Play Framework
Fonte: (CASA DO CÓDIGO, 2015).

O *framework* possui o servidor Netty como um de seus componentes integrados, sendo assim, não existe inicialmente a necessidade de configurar um servidor *web* externo.

2.2.2 ESTRUTURA DE UM PROJETO

O Play Framework esta disponível para download em formato ZIP no endereço http://www.playframework.com/download. Após descompactar, os seguintes arquivos serão listados, como mostra a Figura 2. Nesse momento o arquivo executável *activator* é o mais importante para o desenvolvedor, esse *script* é o ponto de entrada para iniciar o desenvolvimento com o Play Framework.

Figura 2: Estrutura de arquivos do Play Framework

Iniciar um novo projeto com o Play é algo simples. Após digitar *activator new* "nome do projeto" a ferramenta realiza a busca de *templates* disponíveis no repositório *online.* Esses *templates* são modelos de projetos que permitem o programador iniciar o desenvolvimento de sistemas utilizando as linguagens Java ou Scala, e caso necessário suporte a plataforma akka, sem a necessidade de configurações adicionais. A plataforma akka é um conjunto de ferramentas utilizadas na construção de sistemas distribuídos de alta concorrência, resiliência e escalabilidade que se comunicam por meio de troca de mensagens.

Na sequência o desenvolvedor seleciona um desses *templates,* e então o *activator* cria toda a estrutura inicial do projeto e realiza todas as configurações necessárias. A Figura 3 demonstra o processo de criação de um novo projeto de nome NovoProjeto com base no *template play-java.*

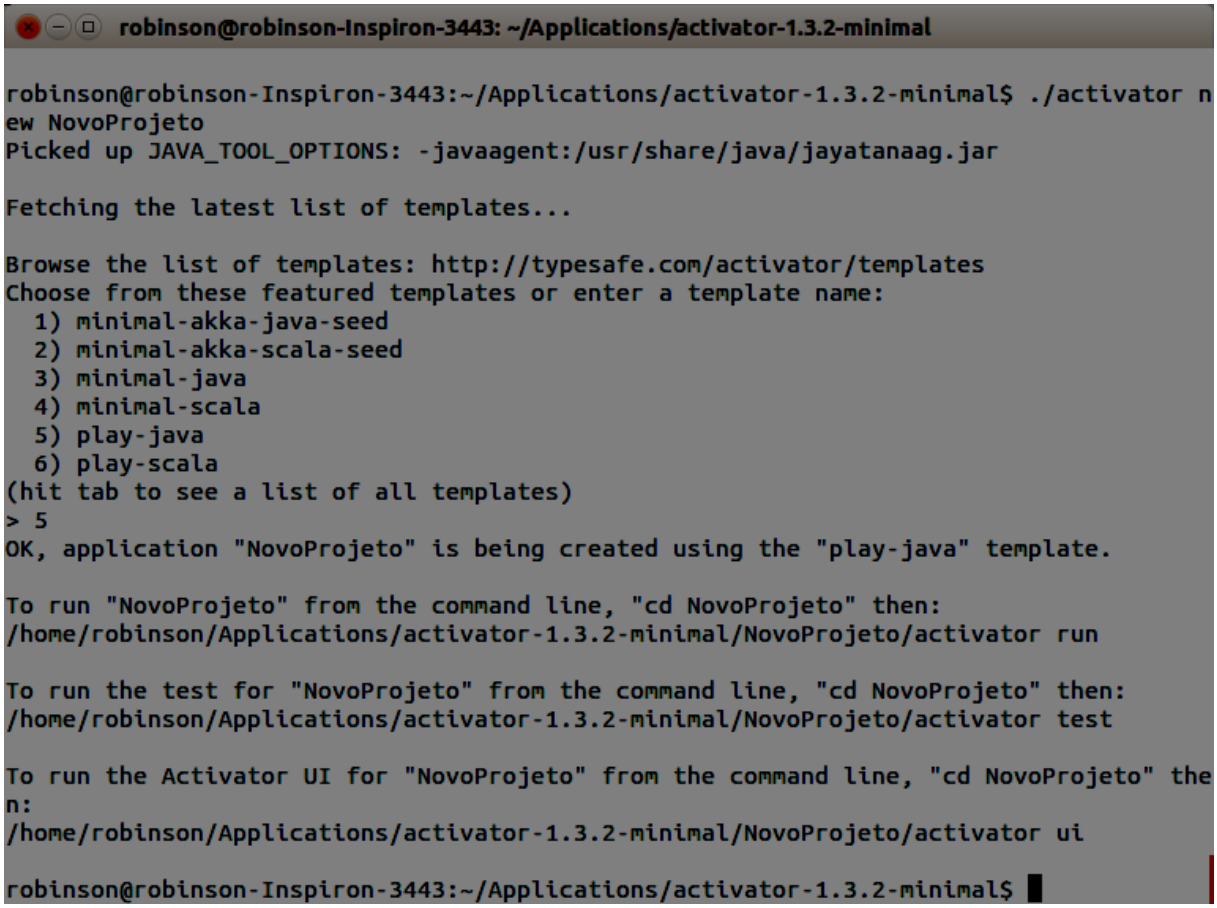

Figura 3: Criando um novo projeto no Play Framework com o *activator*

Ao criar um novo projeto não existe a necessidade do desenvolvedor configurar um único arquivo, tudo é baseado em convenções (*Convention Over Configuration*). Ao finalizar o processo de criação, um novo diretório com o nome do projeto será criado bem como toda estrutura necessária. A Figura 4, mostra a estrutura de diretórios e arquivos gerados para o projeto NovoProjeto.

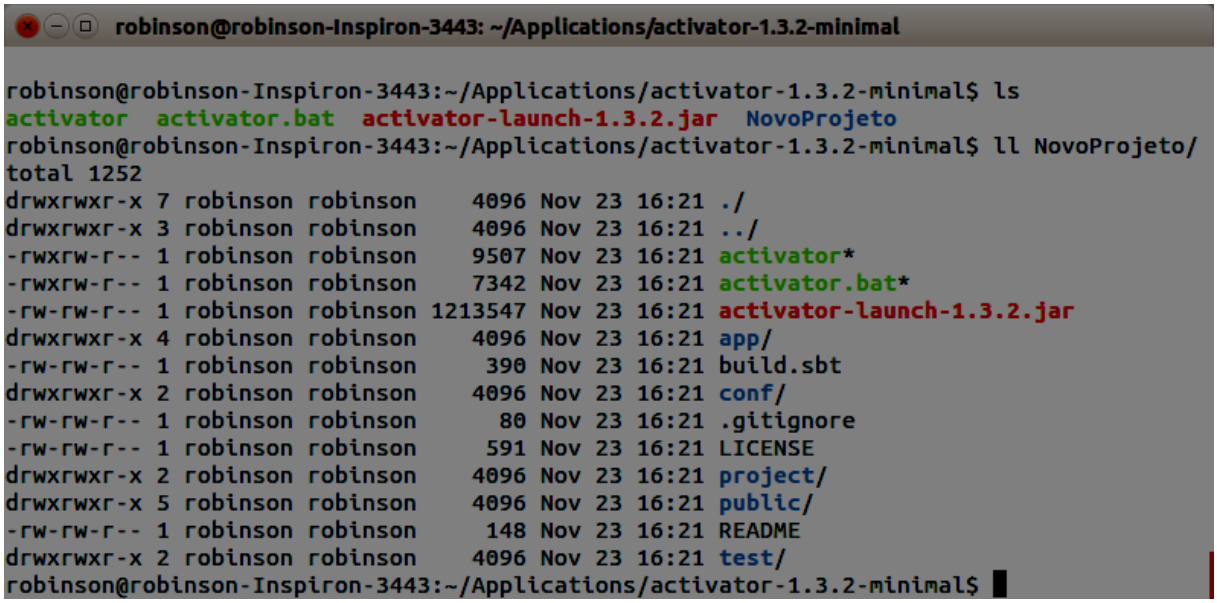

Figura 4: Estrutura de diretórios e arquivos criados para um novo projeto

Os artefatos de *software* (bibliotecas) necessários para o projeto são gerenciados pela ferramenta SBT embutida no *framework*. A adição de dependências, como por exemplo: Biblioteca para geração de PDF é feita no aquivo de texto build.sbt através de uma sintaxe simples. A sintaxe do aquivo é vista na Figura 5.

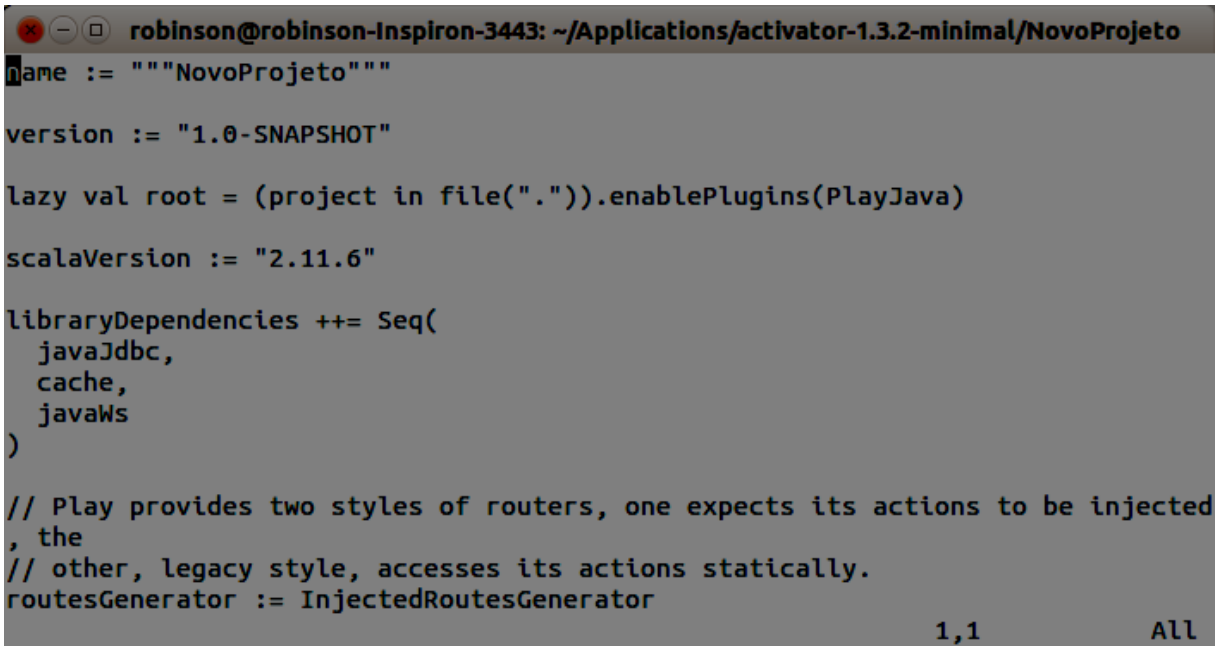

Figura 5: Exemplo da sintaxe do arquivo build.sbt

Havendo a necessidade de adicionar uma nova biblioteca basta inserir o nome da biblioteca em libraryDependencies. É possível utilizar bibliotecas dos repositórios SBT e Maven.

O diretório *public* mantém todos os arquivos estáticos do projeto, como: CSS, Javascript e HTML. A Figura 6 demonstra essa estrutura.

Figura 6: Diretório *public*

A Figura 7 destaca a estrutura no padrão MVC (*Model-View-Controller*) criada pelo Play Framework ao gerar o projeto NovoProjeto.

Figura 7: Estrutura MVC gerada pelo Play Framework

2.2.3 O MODELO DE APLICAÇÃO MVC

O padrão de arquitetura MVC (*Model-View-Controller*) permite que o desenvolvedor separe o código fonte de sua aplicação em três camadas distintas *Model*, *View* e *Controller* de modo que o processo de construção do *software*, adição de novas funcionalidades e a manutenção sejam executadas de forma organizada e separada. Esse padrão foi testado ano pós ano em diferentes plataformas e linguagens de programação e por diferentes gerações de programadores de *software*. O Play Framework utiliza essa arquitetura. A Figura 8 exemplifica o padrão MVC (*Model-View-Controller*) aplicado no tratamento de requisições HTTP.

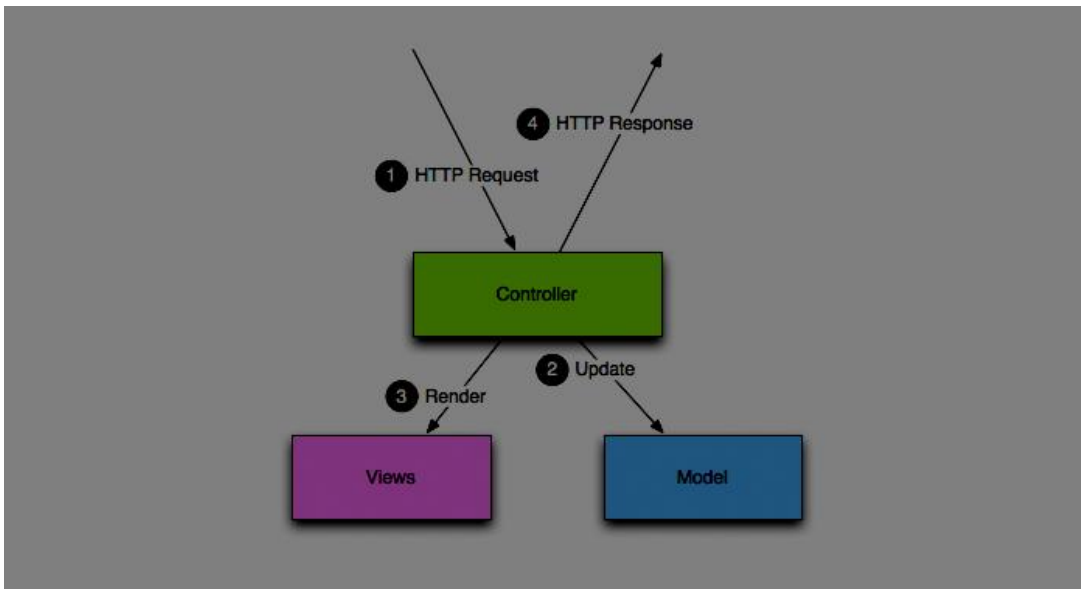

Figura 8: Padrão MVC aplicado no tratamento de requisições HTTP
Fonte: (PLAY DOCUMENTATION, 2015).

Play Framework estrutura as três camadas do modelo MVC (*Model-View-Controller*) em diretórios dentro no diretório app da aplicação.

2.2.3.1 app/controllers

Um *controller* é uma classe Java escrita com código procedural que contém métodos públicos e estáticos. Cada método público estático também é conhecido como *action* e é invocado quando uma requisição HTTP é recebida pelo *framework.* O método *action* extrai dados como cabeçalho, corpo da mensagem e tipo de requisição, para então ler ou atualizar os objetos do modelo e na sequência enviar para o requisitante uma resposta HTTP.

2.2.3.2 app/models

Diferentemente do *controller* a classe *model* utiliza todo poder da orientação a objetos disponível na linguagem Java. A classe reflete os atributos e comportamentos de um objeto real, por meio de sua estrutura de dados e métodos de manipulação. Uma de suas responsabilidade é a persistência em banco de dados, podendo para isso utilizar anotações JPA ou mesmo instruções SQL.

2.2.3.3 app/views

Em uma aplicação Play o *controller* ao responder uma requisição HTTP obtêm os dados do *model*, (que pode ser via acesso a banco de dados) e na sequência aplica um *template* (modelo) nesses dados, retornando para o usuário uma *view*. Esse objeto *view* pode ser uma página web ou mesmo um arquivo JSON ou XML.

2.2.4 CICLO DE VIDA DE UMA REQUISIÇÃO HTTP

Aplicações desenvolvidas no Play Framework são *stateless*, ou seja, não mantém estado entre uma requisição e outra, todo fluxo de informação é orientado a requisição/resposta.

Ao receber uma requisição o Play Framework utiliza o componente Router para encontrar a rota mais especifica habilitada para manipular a requisição, caso encontre, o método do *controller* que trata essa rota é invocado, nesse momento a lógica da aplicação é executada, dados podem ser acessados no banco de dados pelo *model* por meio do *controller* e uma *view* pode ser devolvida pelo *controller* ao retornar a resposta HTTP.

A Figura 9 explica através do diagrama o caminho de uma requisição HTTP.

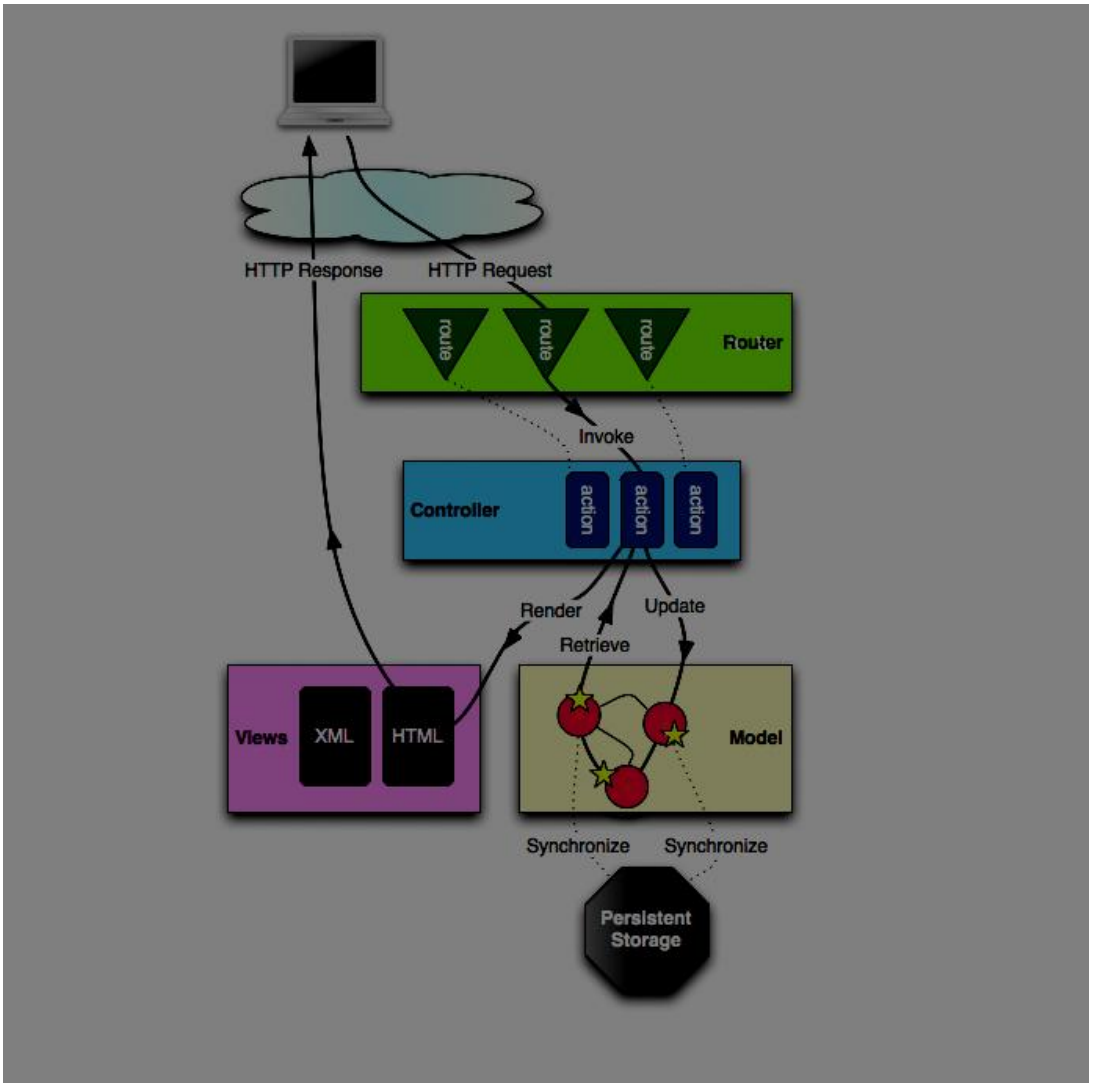

Figura 9: Caminho percorrido por uma requisição HTTP
Fonte: (PLAY DOCUMENTATION, 2015).

3 MATERIAIS E MÉTODO

Este capítulo apresenta as ferramentas utilizadas e método aplicados no desenvolvimento deste trabalho.

3.1 MATERIAIS

Os materiais utilizados neste trabalho estão brevemente descritos a seguir:

- Java (http://www.java.com) é uma linguagem de programação interpretada orientada a objetos e uma plataforma computacional. A linguagem Java não compila para código nativo e sim para um *bytecode* que posteriormente é executado por uma máquina virtual. A linguagem possui como principais características: portabilidade, segurança e confiabilidade. O projeto mantem o código fonte sob a licença GPL (*GNU General Public License*).
- Play Framework (http://www.playframework.com) é um *framework* para desenvolvimento de aplicações *web* que fornece suporte as linguagens Java e Scala, utiliza o modelo MVC (*Model-View-Controller*) como padrão de arquitetura, faz uso da ferramenta SBT para a geração de *build* e *deploy* e tem seu código fonte liberado por meio da licença Apache 2.
- SBT (http://www.scala-sbt.org) é uma ferramenta de *build* similar ao Maven e ao Ant, fornece suporte a projetos Java e Scala, realiza o gerenciamento de dependências através do Ivy, fornece compilação contínua, teste e implantação e seu código fonte possui licença BSD.
- Netty (http://www.netty.io) é um *framework* para desenvolvimento de aplicações de rede não bloqueantes, tais como servidores de protocolo e clientes (TCP, UDP), possui arquitetura assíncrona dirigida a eventos, tem como características: Facilidade de uso, performance e segurança. Netty é licenciado sob Apache License 2.0.
- DIA (http://www.dia-installer.de) é uma ferramenta utilizada para criar diagramas e fluxogramas, possui interface e funcionalidades similares ao Microsoft Visio. O código fonte é disponibilizado sob licença GPL.
- PostgreSQL (http://www.postgresql.org) é um sistema de gerenciamento de banco de dados *objeto-relacional*, extremamente robusto, confiável e flexível. Alguns dos recursos incluem: Consultas complexas, integridade transacional, controle de concorrência multi-versão, suporte a SSL, *triggers*, *views* e *stored precedures* em várias linguagens. O código fonte esta sob licença BSD.
- Twitter Bootstrap (http://www.getbootstrap.com) é um *framework* para desenvolvimento de interfaces para projetos *web*, ou seja, basicamente uma coleção de arquivos de CSS (*Cascanding Style Sheets*) e Javascript que fornece o design a tipologia e o comportamento para componentes como botões, formulários e menus. O código fonte possui licença MIT.

- Eclipse (http://www.eclipse.org) é um ambiente de desenvolvimento integrado (IDE) desenvolvido para linguagem Java, mas com suporte a inúmeras outras através do uso de *plugins.* O software possui licença EPL (*Eclipse Pubic License*) .
- Python (http://www.python.org) é uma linguagem de programação de alto nível, interpretada, totalmente orientada a objetos, com suporte ao paradigma funcional, sem tipos primitivos. Possui como principais características: Tipagem dinâmica e forte, curva de aprendizagem extremamente baixa e alta produtividade. O código fonte do projeto é licenciado por meio da Python License (Python-2.0).
- NVD3 (http://www.nvd3.org) é uma biblioteca javascript que permite desenhar gráficos de duas dimensões no navegador *web*. O projeto é mantido pela empresa Novus Partners (www.novus.com) e possui licença Apache License 2.0.
- Scala (http://www.scala-lang.org/) é uma linguagem de programação, orientada a objetos e funcional, roda sob a plataforma Java e mantém interoperabilidade com a linguagem Java. Possui código fonte com licença BSD.

3.2 MÉTODO

Os procedimentos para definir o sistema estão baseados nas fases propostas por Pressman (2002) para o modelo sequencial linear que são análise, projeto, codificação e testes. A seguir as fases definidas para este trabalho:

a) Levantamento de requisitos – A percepção do aumento significativo do número de dispositivos inteligentes conectados a Internet e o novo e crescente interesse de pessoas e empresas no gerenciamento dos dados gerados por esses aparelhos embasou os requisitos funcionais e não funcionais necessários na decisão do problema a ser resolvido. A análise de sistemas semelhantes permitiu a definição do modelo de interface de usuário e as funcionalidades primárias.

b) Análise – Foi realizado o estudo das tecnologias necessárias e das plataformas de gerenciamento de dados de sensores já existentes.

c) Projeto – Fez necessário a aplicação dos conceitos e metodologias do modelo da programação orientada a objetos e a decomposição do problema, descrevendo as principais funcionalidades do sistema através de diagramas de caso e de entidades e relacionamentos.

d) Implementação do sistema – Foi realizado a codificação das classes do tipo *model* que definem os objetos do sistema, das classes do tipo *controller* que possuem a tarefa do tratamento das requisições HTTP's oriundas do usuário e das views do sistema responsáveis por devolver conteúdo HTML/Javascript/CSS ou JSON para o usuário.

e) Realização dos testes - Todos os testes realizados foram informais e cobrem os seguintes procedimentos: Cadastro de novo usuário, cadastro de novo dispositivo, envio de registro, visualização dos registros enviados e gráfico gerado. O envio dos dados foi realizado utilizando um pequeno programa escrito na linguagem Python (Listagem 1), em que esse programa enviava várias requisições do tipo POST para a plataforma, contendo o identificador do dispositivo e o valor, de modo que simulasse um dispositivo real.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-


import random
import requests
import json


def randomize():
    return round(random.uniform(20, 30), 2)


class Temperature(object):
    def __init__(self):
        pass
```

```
    def get_value(self):
        return randomize()


def test():
    t = Temperature()


    url = "http://localhost:9000/records"
    data = {'uuid': 'eb137f38-37e4-481f-8505-b5c51773659f',
            'value': t.get_value(), 'createdAt': '2015-02-11 20:31 UTC'}


    headers = {'Content-type': 'application/json', 'Accept': 'text/plain'}
    r = requests.post(url, data=json.dumps(data), headers=headers)
    print r


if __name__ == '__main__':
    for i in xrange(100):
        test()
```

Listagem 1: Programa Python para simular o envio de dados para o sistema

4 RESULTADOS

Este capítulo apresenta o produto deste trabalho, um protótipo de um sistema *web* para monitoramento de sensores que permite a coleta, o armazenamento e a visualização dos dados coletados. A ênfase é apresentar a macro ideia do projeto, descrever sobre o processo de modelagem do sistema, fornecer uma visão ampla das funcionalidades desenvolvidas e por fim detalhar aspectos técnicos de sua implementação.

4.1 ESCOPO DO SISTEMA

O sistema permite a coleta, o armazenamento e a visualização de dados de sensores, fornecendo funcionalidades como: cadastro de dispositivos (sensores), uso de requisição HTTP na coleta de dados e a visualização gráfica das informações coletadas.

Após realizar um cadastro o usuário ficará apto a acessar sua *dashboard*, permitindo o cadastro de dispositivos e a visualização dos dados coletados. O envio dos dados para plataforma é realizado por meio do uso de uma API. A visualização dos dados é realizada por meio de relatório em lista, assim como por meio de gráfico. O acesso do usuário é realizado através do uso de um navegador de Internet.

O sistema não será responsável pela aquisição dos dados dos sensores e sim somente por fornecer um meio para coleta (API), persistência e visualização dos dados.

4.2 MODELAGEM DO SISTEMA

Requisitos funcionais:

a) Permitir ao usuário cadastrar novos dispositivos
b) Permitir ao usuário excluir dispositivos
c) Permitir ao usuário o envio/coleta de dados dos sensores para o sistema
d) Armazenar os dados em banco de dados
e) Permitir ao usuário visualizar os dados persistidos por meio de relatório e gráfico.

Requisitos não funcionais

a) A plataforma deve ser web
b) Obrigatório o uso de usuário e senha para acessar o sistema

A Figura 10 demostra o diagrama de casos de uso que representa os requisitos funcionais do sistema.

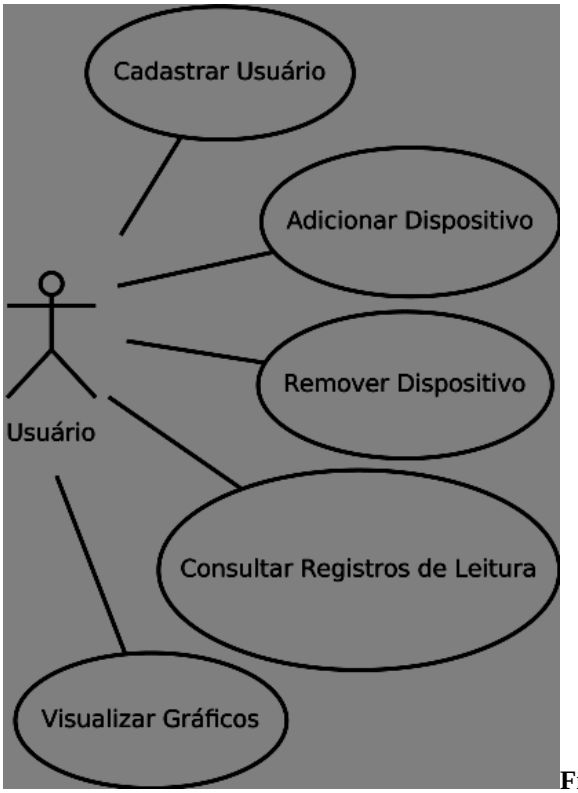

Figura 10: Diagrama de casos de uso

A Figura 11 apresenta o diagrama de entidade e relacionamento (DER) que representa tabelas do banco de dados do sistema e a relação entre essas tabelas. O sistema ao todo faz uso de quatro tabelas sendo: *account*, *password_reset*, *device* e *record*.

A tabela *account* armazena os dados de cadastro de todos os usuários do sistema, contem informações de usuário para *login*, senha com criptografia do tipo BCrypt com salto, data e hora que o cadastro do usuário foi realizado e o e-mail para recuperação do acesso ao sistema em caso de perda da senha.

O sistema utiliza também a tabela auxiliar *password_reset* responsável por armazenar as solicitações de usuário que perderam suas senhas. No momento da solicitação da redefinição de senha o sistema grava nessa tabela quem é o usuário solicitante, data e hora da solicitação, data e hora em que a solicitação expira (12 horas após a solicitação) e um *token* que nada mais é do que uma sequência de caractere gerada de forma aleatória, utilizada como uma forma de segurança no momento da redefinição da senha.

Todo novo dispositivo que o usuário adiciona ao sistema tem suas informações persistidas na tabela *device*. Essa entidade armazena uma referência ao dono do dispositivo, o

nome do dispositivo, o nome da unidade de medida e um identificador do tipo sequência de caractere que é utilizado no momento do envio dos registros pela plataforma.

Por fim a tabela *record* armazena todos os registros gerados por cada dispositivo, contendo os campos: valor, data de criação do registro e a referência a que dispositivo pertence o registro.

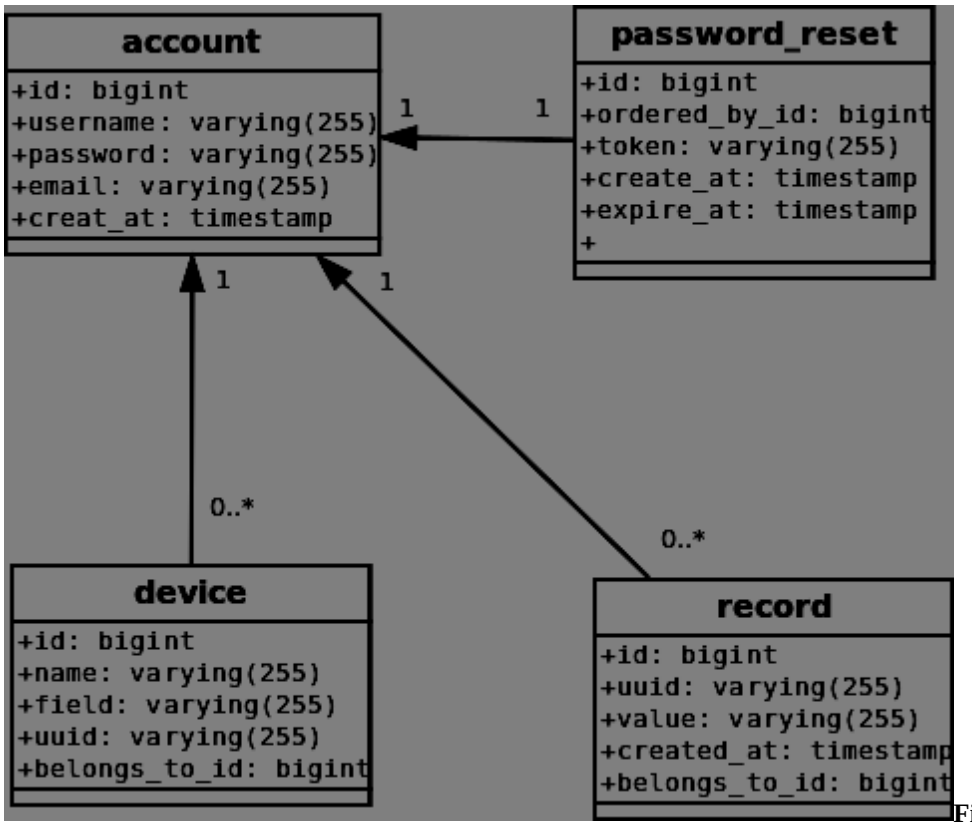

Figura 11: Diagrama Entidade-Relacionamento DER

Essa seção faz uso de quadros para apresentar uma visão mais detalhada de cada entidade do banco de dados apresentada na Figura 11.

O Quadro 1 apresenta os campos presentes na tabela *account*.

Nome do campo	**Descrição**	**Tipo**	**Obrigatório**
id	Chave primária da tabela	Inteiro longo	Inserido pelo banco de dados
username	Único, utilizado no *login* do sistema	Texto	Sim
password	Utilizado no *login* do sistema	Texto Criptografado com BCrypt com salto	Sim
e-mail	Utilizado para redefinição de senha do sistema	Texto	Sim
create_at	Data e Hora da criação do registro	Data/Hora	Inserido pelo sistema

Quadro 1: Tabela *account*

Os detalhes sobre os campos da tabela *device* são apresentados pelo Quadro 2.

Nome do Campo	Descrição	Tipo	Obrigatório
id	Chave primária da tabela	Inteiro longo	Inserido pelo banco de dados
name	Nome do dispositivo	Texto	Sim

field	Nome da unidade de medida	Texto	Sim
uuid	Sequência de caractere que identifica o dispositivo	Texto	Inserido pelo sistema
belongs_to_id	Chave estrangeira para tabela *account*. Informa quem é o dono do dispositivo	Inteiro longo	Sim

Quadro 2: Tabela *device*

O Quadro 3 mostra todos os campos que a tabela *record* possui.

Nome do Campo	Descrição	Tipo	Obrigatório
id	Chave primária da tabela	Inteiro longo	Inserido pelo banco de dados
uuid	Sequência de caractere que identifica o dispositivo	Texto	Sim
value	Valor da unidade de medida lida	Texto	Sim
create_at	Data e Hora da criação do registro	Data/Hora	Inserido pelo sistema
belongs_to_id	Chave estrangeira para tabela *account*. Informa quem é o dono do registro	Inteiro longo	Sim

Quadro 3: Tabela *record*

No Quadro 4 apresenta os campos da tabela *password_reset*.

Nome do Campo	Descrição	Tipo	Obrigatório
id	Chave primária da tabela	Inteiro longo	Inserido pelo banco de dados
ordered_by_id	Chave estrangeira para tabela *account*. Informa quem	Inteiro longo	Sim

	solicitou a redefinição de senha		
token	Sequência de caractere, utilizada como uma forma de segurança	Texto	Sim
create_at	Data e Hora da criação do registro	Data/Hora	Inserido pelo sistema
expire_at	Data e Hora que a solicitação de redefinição de senha não será mais válida	Data/Hora	Inserido pelo sistema

Quadro 4: Tabela *password_reset*

Certamente quadros com descrições das operações de casos de uso permitem um melhor entendimento da dinâmica dos *softwares.* O Quadro 5 mostra as operações do caso de uso cadastrar usuário.

Identificador do requisito: Cadastrar usuário
Descrição: O usuário deseja se cadastrar no sistema
Evento Iniciador: Tela de cadastro
Atores: Usuário
Pré-condição: Nenhuma
Sequência de Eventos: 1 – O usuário preenche o formulário solicitado pelo sistema 2 – O sistema válida as informações, se corretas realiza o *insert* do registro no banco de dados. 3 – O sistema mostra uma mensagem de sucesso ou falha para o usuário
Pós-Condição: Os dados inseridos pelo usuário devem ser validados
Requisitos não funcionais:
Extensões: Nenhuma

Quadro 5: Caso de uso cadastrar usuário

As operações para o caso de uso logar no sistema são apresentadas no Quadro 6.

Identificador do requisito: Logar no sistema
Descrição: O usuário deseja logar no sistema
Evento Iniciador: Tela de login do sistema
Atores: Usuário
Pré-condição: O usuário deve previamente ter realizado o cadastro
Sequência de Eventos: 1 – O usuário informa os dados de *login* 2 – O sistema válida as informações, se corretas realiza o *login* do usuário no sistema. 3 – O sistema mostra a tela para usuários logados
Pós-Condição: O sistema deve manter a seção do usuário
Requisitos não funcionais:
Extensões: Nenhuma

Quadro 6: Caso de uso logar no sistema

O Quadro 7 expõem as operações de caso de uso cadastrar dispositivos

Identificador do requisito: Cadastrar dispositivo
Descrição: O usuário deseja cadastrar um novo dispositivo
Evento Iniciador: Tela de cadastro de dispositivo
Atores: Usuário
Pré-condição: O usuário deve estar logado no sistema
Sequência de Eventos: 1 – O usuário preenche o formulário solicitado pelo sistema 2 – O sistema válida as informações, se corretas realiza o *insert* do registro no banco de dados. 3 – O sistema mostra uma mensagem de sucesso ou falha para o usuário
Pós-Condição: Os dados inseridos pelo usuário devem ser validados
Requisitos não funcionais:

Extensões: Nenhuma

Quadro 7: Caso de uso cadastrar dispositivo

A dinâmica das operações do caso de uso enviar dados para o sistema são demonstradas no Quadro 8.

Identificador do requisito: Enviar dados para o sistema

Descrição: O usuário deseja enviar dados do dispositivo para o sistema

Evento Iniciador: Tela de envio de dados

Atores: Usuário

Pré-condição: O usuário deve estar logado no sistema

Sequência de Eventos:
1 – O usuário preenche o formulário solicitado pelo sistema
2 – O sistema válida as informações, se corretas realiza o insert do registro no banco de dados.
3 – O sistema mostra uma mensagem de sucesso ou falha para o usuário

Pós-Condição: Os dados inseridos pelo usuário devem ser validados

Requisitos não funcionais:

Extensões: Nenhuma

Quadro 8: Caso de uso enviar dados para o sistema

O caso de uso consultar registro de leitura com suas operações é demonstrado no Quadro 9.

Identificador do requisito: Consultar registro de leitura

Descrição: O usuário deseja consultar os registros de leituras para um determinado dispositivo

Evento Iniciador: Tela de consulta de registros

Atores: Usuário

Pré-condição: Nenhuma

Sequência de Eventos:
1 – O usuário acessa o menu "*My Devices*"

2 – O usuário seleciona o dispositivo requerido

3 – O sistema mostra os dados solicitados

Pós-Condição: Nenhuma

Requisitos não funcionais:

Extensões: Nenhuma

Quadro 9: Caso de uso consultar registro de leitura

Por fim o Quadro 10 permite o entendimento das operações do caso de uso visualizar gráfico de leituras.

Identificador do requisito: Visualizar gráfico de leituras

Descrição: O usuário deseja visualizar os registros de leituras na forma de gráfico

Evento Iniciador: Tela de visualização gráfica dos registros de leitura

Atores: Usuário

Pré-condição: Nenhuma

Sequência de Eventos:

1 – O usuário acessa o menu "*My Devices*"

2 – O usuário seleciona o dispositivo requerido

3 – O sistema mostra o gráfico dos registros

Pós-Condição: Nenhuma

Requisitos não funcionais:

Extensões: Nenhuma

Quadro 10: Caso de uso visualizar gráfico de leituras

4.3 APRESENTAÇÃO DO SISTEMA

A plataforma somente permite que usuários cadastrados possam utilizar seus recursos, sendo assim, ao acessar pela primeira vez o sistema o usuário precisa preencher um cadastro, fornecendo os dados: nome de usuário, e-mail e senha. O nome de usuário e e-mail devem ser únicos para toda plataforma, caso contrário o sistema mostrará uma mensagem de erro e

solicitará a alteração das informações incorretas. O e-mail é utilizado pela plataforma em casos onde o usuário perdeu sua senha e solicita a redefinição. Nesse momento o sistema envia para o e-mail cadastrado pelo usuário uma URL, válida por doze horas, que permite a redefinição da senha, por segurança o campo senha é salvo no banco de dados utilizando criptografia BCrypt com salto. A Figura 12 mostra a tela de cadastro.

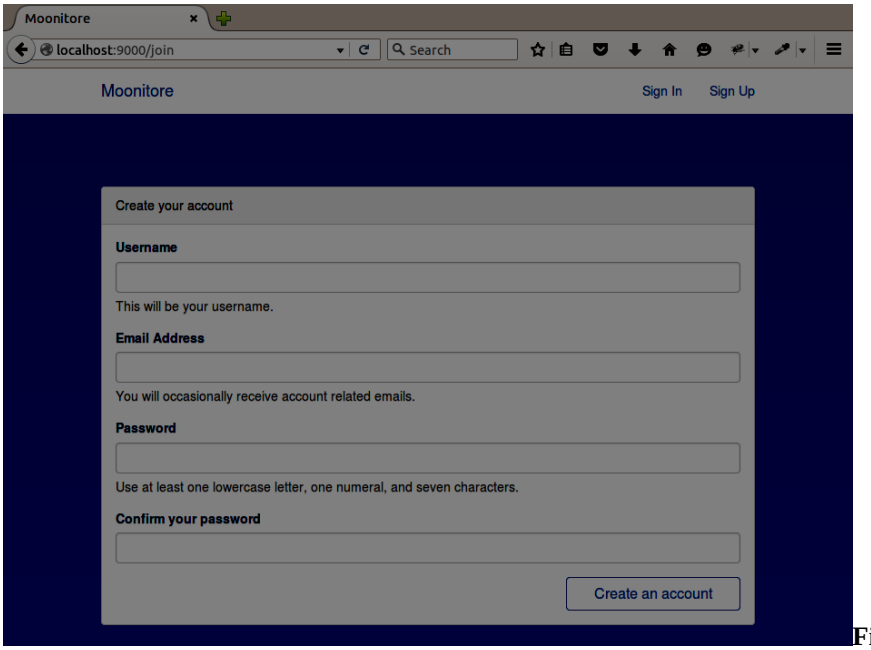

Figura 12: Tela de cadastro

Caso o usuário tenha realizado com sucesso o cadastro, será redirecionado para a tela de *login*, conforme mostra a Figura 13. Essa tela possibilita a entrada no sistema utilizando tanto o nome de usuário quanto o e-mail.

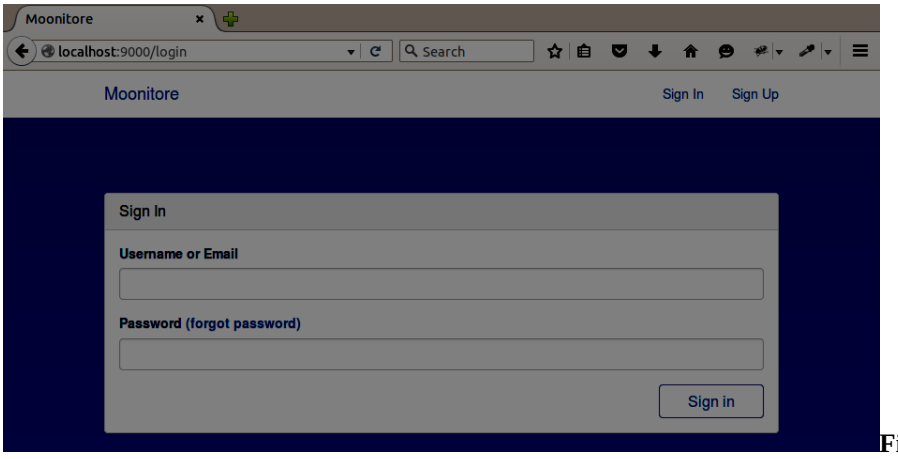

Figura 13: Tela de *login*

Estando logado no sistema o usuário pode adicionar um novo dispositivo selecionando o menu *Devices* e clicando no botão *New Device*, a tela apresentada na Figura 14 será renderizada. Os dados nome e campo são obrigatórios.

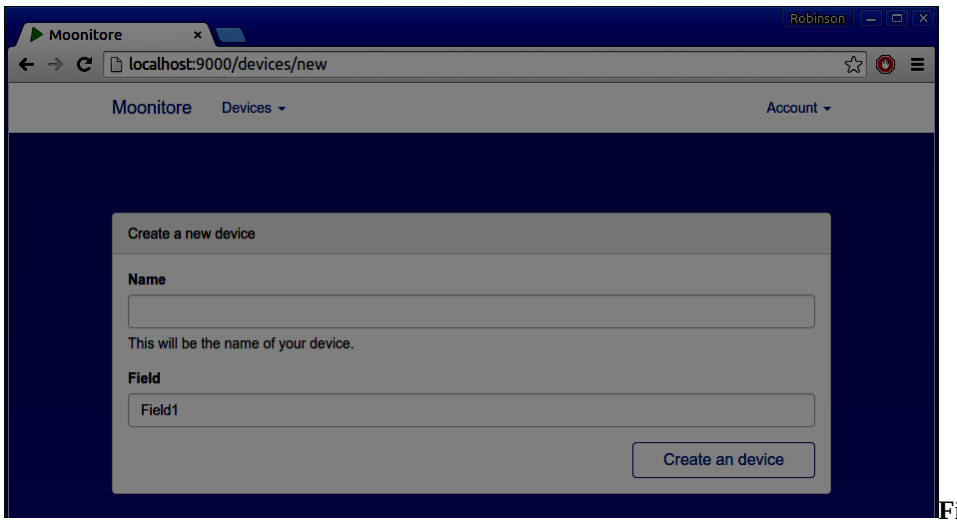

Figura 14: Cadastro de novo dispositivo

Para listar todos os dispositivos cadastrados, o usuário pode selecionar no menu *Devices* a opção *My Devices*. Informações como nome do dispositivo e seu identificador (UUID) serão mostrados, como visto na Figura 15. Para cada registro na tela é adicionado um botão de ação que permite acessar informações detalhadas de cada dispositivo.

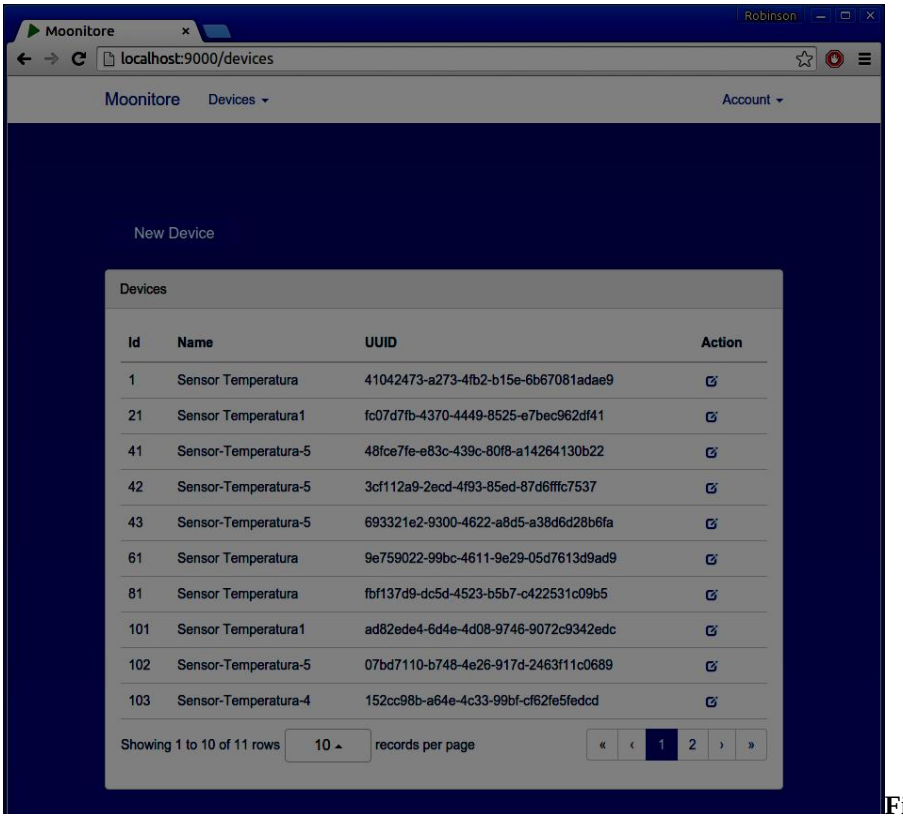

Figura 15: Lista de dispositivos

Na tela de detalhes do dispositivo, como pode ser visto na Figura 16, o usuário pode alterar os dados de nome e campo do dispositivo, também existe a opção de excluir o dispositivo, e uma vez escolhida essa opção o dispositivo e todos os registros a ele associados serão excluídos do banco de dados. Essa ação não poderá ser revertida.

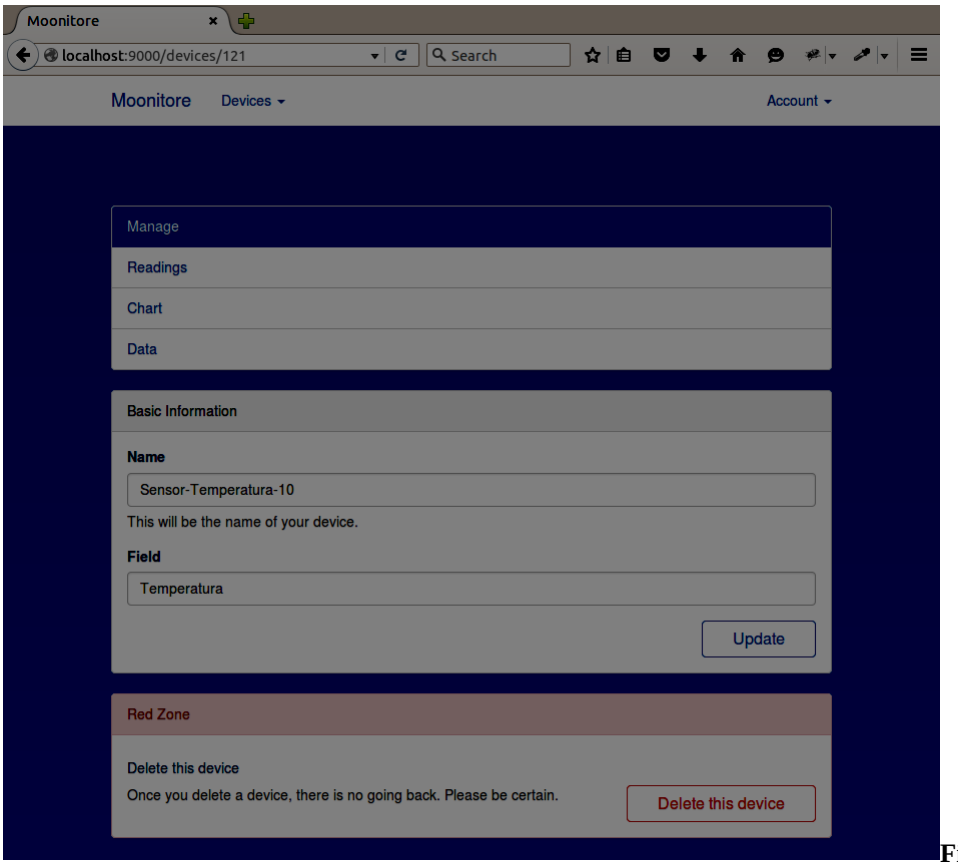

Figura 16: Detalhes do dispositivo

Todas as leituras realizadas por um dispositivo específico podem ser visualizadas acessando o menu *Devices*, selecionando o dispositivo em questão e após a tela de detalhe do dispositivo ser carregada, selecionar a opção *readings*. O registro de leitura apresenta os campos valor e data de criação.

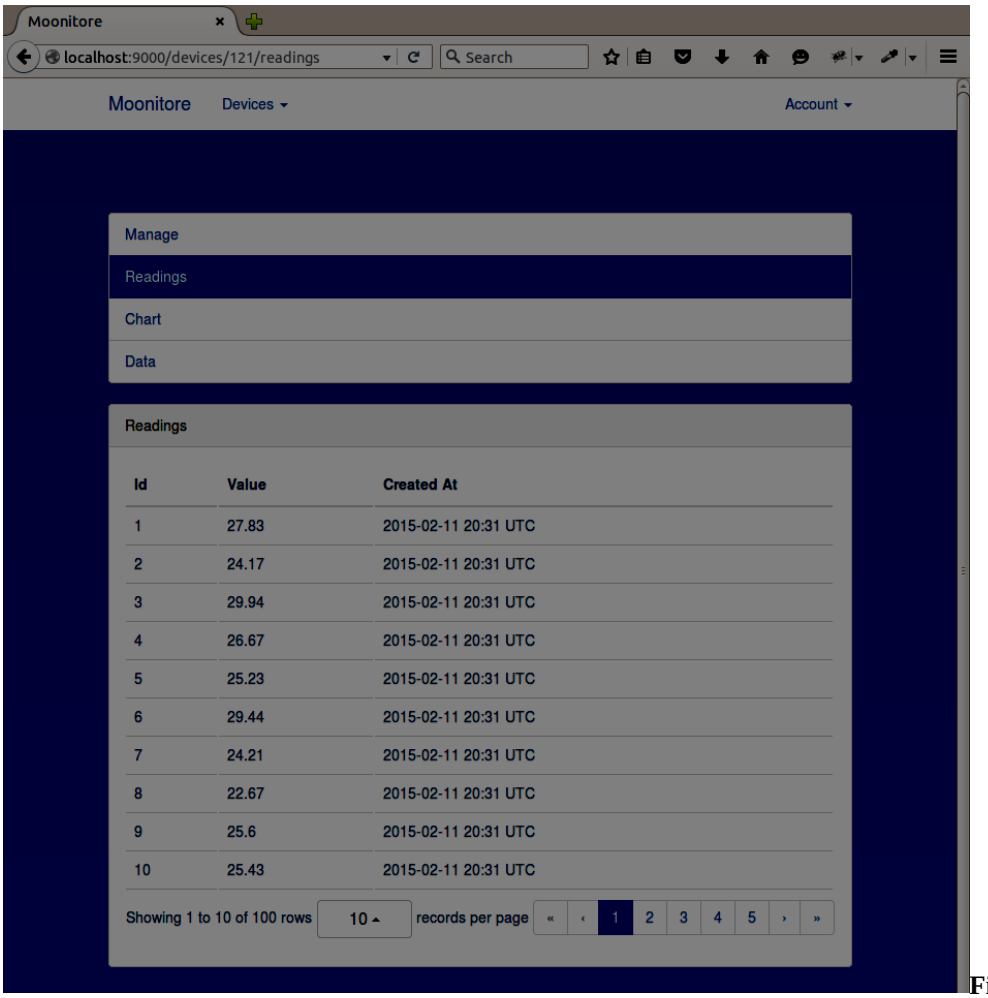

Figura 17: Leituras do dispositivo

Outra opção para análise dos dados coletados é a visualização por meio de gráfico. A Figura 18 demonstra a tela de visualização gráfica dos registros. O gráfico possui opção de zoom acessível pelos botões mais e menos.

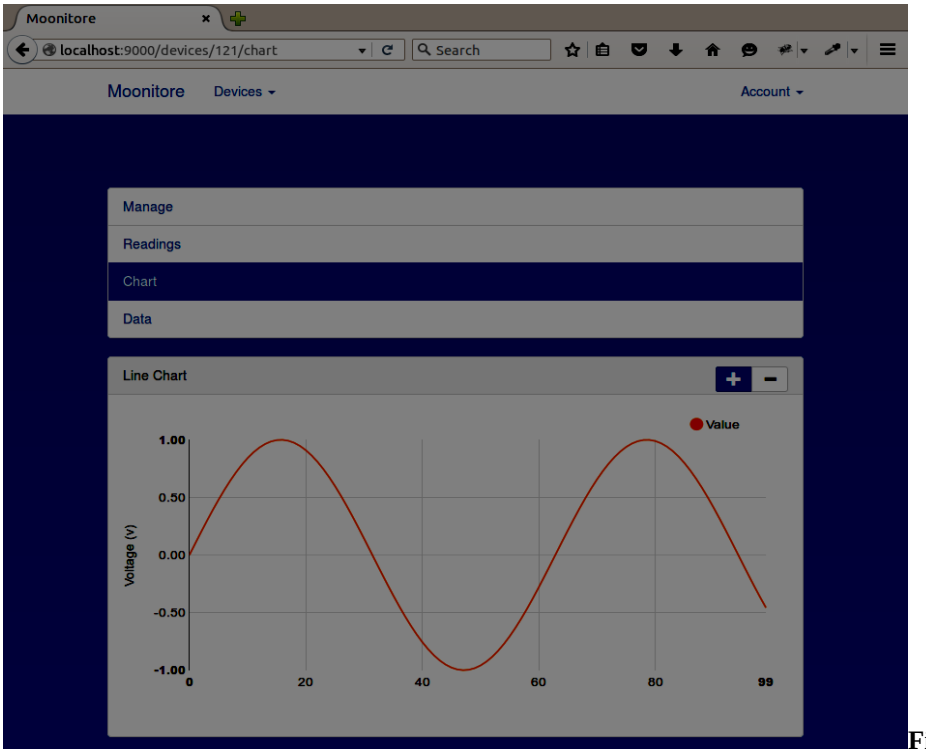

Figura 18: Gráfico de leituras

A plataforma permite que o usuário realize teste e envio de registro por meio da tela de envio de dados (Figura 19), para isso é necessário preencher os dados de identificador e campo e pressionar o botão envio de requisição POST.

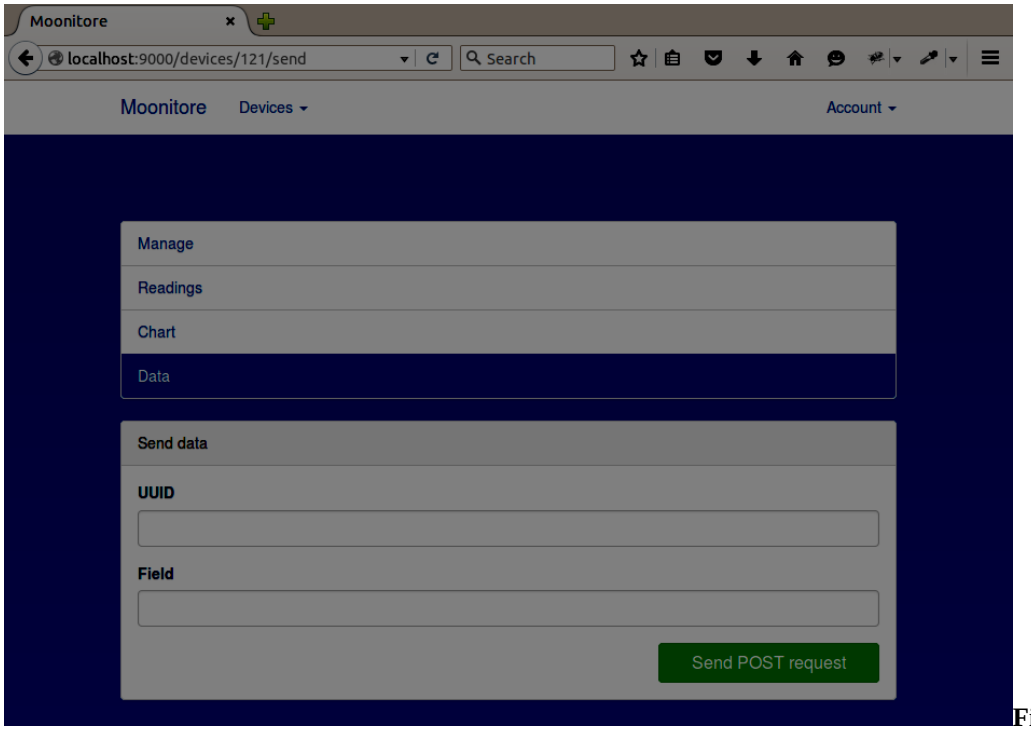

Figura 19: Envio de registro

4.4 IMPLEMENTAÇÃO DO SISTEMA

Esta seção descreve como os recursos foram utilizados no desenvolvimento e apresenta a análise de fragmentos do código fonte das principais funcionalidades presentes no sistema. Destaca-se inicialmente a definição do *Model* e de classes de ajuda no acesso ao banco de dados, em seguida, a construção e o uso da classe *Controller* responsável pela lógica do sistema, mais adiante a apresentação das classes responsáveis pela criação do conteúdo (*View*) para o usuário e por fim a definição das rotas que mapeiam a URL utilizada na requisição para o método adequado.

O grande pilar no desenvolvimento utilizando o Play Framewok é seu modelo baseado no padrão de arquitetura MVC (*Model-View-Controller*), sendo assim, em um típico projeto Play existirá a necessidade da implementação dessas três classes: *Model, View, Controller*.

4.4.1 *Model*

A classe *Device* vista na Listagem 2 representa o objeto *Device*. A listagem mostra o uso da anotação @Entity, quando essa anotação esta presente o Play Framework automaticamente irá iniciar o gerenciador de entidade Ebean, que nada mais é do que um gerenciador objeto relacional (ORM) que implementa o padrão Java Persistence API (JPA) fornecendo a estrutura necessária para persistir os dados, em resumo, realizar as operações de *Create, Retrive, Update* e *Delete* do modelo no banco de dados.

```
import play.db.ebean.Model;


@Entity
```

```
public class Device extends Model {
        private static final long serialVersionUID = 91143259543604959 30L;

        @Id
        @GeneratedValue(strategy = GenerationType.AUTO)
        public Long id;
}
```

Listagem 2: Classe de domínio (Model)

O Play Framework permite uma maneira simplificada de acesso à consultas ao banco de dados, por meio da classe *Finder* utilizando método estático. Todos os modelos (*Account*, *Device*, *PasswordReset* e *Record*) apresentados nesse trabalho implementam esse *Helper*. O código necessário é apresentado na Listagem 3.

```
public static Model.Finder<Long, Device> find =
        new Model.Finder<Long, Device>(Long.class,Device.class);
```

Listagem 3: *Helper*

Após a definição do método estático *Helper*, todos os métodos de consulta a banco de dados da classe *Model* podem ser acessados de maneira simplificada, como demonstrado na Listagem 4.

```
// Find all devices
List<Device> devices = Device.find.all();

// Find a device by ID
Device anyDevice = Device.find.byId(34L);

// Delete a device by ID
Device.find.ref(1L).delete();

// More complex device query
List<Device> devices = find.where()
   .ilike("name", "%coco%")
   .orderBy("dueDate asc")
   .findPagingList(25)
   .getPage(1);
```

Listagem 4: *Finder*

4.4.2 *Controller*

Basicamente no Play Framework uma classe *Controller* possui todos os métodos que serão invocados para devolver uma resposta após receber uma requisição a partir de uma URL especifica ou caso utilize expressão regular (regex) em um padrão de URL. Essa resposta pode ser uma sequência de caractere, um objeto do tipo JSON ou mesmo uma *View*.

A Listagem 5 mostra a implementação do método *create* que cria a *View* responsável por enviar para o usuário o formulário de cadastro de novos dispositivos. O método show tenta recuperar um objeto *Device* utilizando seu id que foi informado pelo usuário como parâmetro na URL utilizada na requisição.

Por fim a classe *Devices Controller* possui a anotação @Security.Authenticated(Secured.class) para que somente requisições realizadas por usuários autenticados sejam tratadas pela classe, caso contrário o usuário será redirecionado para tela de *login*.

```
package controllers;


@Security.Authenticated(Secured.class)
public class Devices extends Controller {


    public static Result create() {
        return ok(create.render());
    }


    public static Result show(Long id) {
        Device device = Device.find.byId(id);
            if (device == null) {
                return notFound("Not Found, 404");
```

```
        }
    return ok(show.render(device));
  }
```

Listagem 5: Classe de controle (*Controller*) Device

Essa autenticação e autorização básica feita pela aplicação é fornecida pela classe *Secured*. O módulo possui dois métodos *getUsername* e *onUnauthorized* (Listagem 6) que o Play utiliza.

```
package controllers;

import models.Account;
import play.mvc.*;
import play.mvc.Http.*;

public class Secured extends Security.Authenticator {
   @Override
   public String getUsername(Context ctx) {
      return ctx.session().get("username");
   }

   @Override
   public Result onUnauthorized(Context ctx) {
      return redirect(routes.Accounts.login());
   }
}
```

Listagem 6: Classe de controle (*Controller)* Secured

4.4.3 View

Toda página apresentada pelo sistema para o usuário é gerada por uma *View*. A *View* pode utilizar os dados do *Model* para retornar ao usuário uma página HTML com esses dados, por exemplo.

Diferentemente de uma página HTML pura, a *View* permite que o desenvolvedor envie objetos de classes Java para a página que será renderizada para o usuário, nesse momento a *View* irá serializar o objeto de forma transparente para o programador. A Listagem 7 mostra um pequeno trecho do código da *View* que mostra a tela de informações especificas para um dispositivo.

```
@(device: Device)
@main(“Moonitore”) {
   <main class=”bs-docs-masthead” id=”content” role=”main” tabindex=”-1”>
   <div class=”contained”>
    <div class=”row”>
     <div class=”col-md-3”>
      <div class=”panel panel-default”>
       .
       .
       .
      </div>
     </div>
    </div>
   </div>
}
```

Listagem 7: *View Device*

Na Listagem 7, as duas primeiras linhas apresentam códigos na linguagem Scala. Scala é a linguagem utilizada nos *templates* para permitir ao desenvolvedor acesso aos objetos oriundos do *model.* A primeira linha informa que a *view* ao ser renderizada recebe um objeto do tipo *Device*, a segunda linha declara o método *main* com o parâmetro padrão “Moonitore” e seu corpo é definido com código HTML.

4.4.4 Routes

Para que cada requisição feita pelo usuário seja processada e obtenha uma resposta é necessário que em algum ponto do sistema exista o mapeamento das URL para o método responsável por manipular a requisição. Para isso o Play Framework disponibiliza o arquivo *routes*, é nele que todo mapeamento URL/método é feito.

A Listagem 8 fornece a visão de algumas das URL mapeadas para os métodos da classe de controle *Device*.

```
# Device
GET     /devices/new              controllers.Devices.create()
GET     /devices/:id              controllers.Devices.show(id: Long)
GET     /devices                  controllers.Devices.list()
GET     /api/v1/devices           controllers.Devices.listToJson()
POST    /api/v1/devices           controllers.Devices.save()
```

Listagem 8: Arquivo *routes*

4.4.4 Autenticação com criptografia

O sistema desenvolvido implementa a funcionalidade de autenticação de usuário com o uso do método de criptografia BCrypt com salto. Este é um tipo de criptografia *hash.* Algoritmos do tipo *hash* uma vez aplicados tornam impossível obter o dado original. O BCrypt foi apresentado no ano de 1999 por Niels Provos e David Maziéres (PROVOS & MAZIÈRES, 1999) e sua ideia é manter as senhas seguras mesmo com o aumento da velocidade dos *hardwares*, ou seja, ser imune a ataques de força bruta. Isso é possível, porque o algoritmo é adaptativo, permitindo ao usuário a configuração do custo computacional. A Listagem 9 apresenta os métodos *createPasswordBCrypt* e *checkPasswordBcrypt* presentes no arquivo de modelo *Account.java*.

```
public static String createPasswordBCrypt(String passwordRaw) {
        return BCrypt.hashpw(passwordRaw, BCrypt.gensalt());
}

public static Boolean checkPasswordBCrypt(String passwordRaw, String passwordBCrypt) {
                return BCrypt.checkpw(passwordRaw, passwordBCrypt);
}
```

Listagem 9: Uso de criptografia BCrypt com salto

5 CONCLUSÃO

O objetivo deste trabalho foi desenvolver um protótipo de um sistema *web* para monitoramento de sensores, que implementasse as funcionalidades de envio, persistência e a visualização dos dados de sensores, com o intuito de fornecer uma plataforma alternativa para o usuário.

O interesse em desenvolver esse projeto veio da observação do aumento crescente, ano pós ano do número de dispositivos inteligentes conectados a Internet, da enorme quantidade de dados gerados e da fixação da era moderna pela análise desses dados, afim de produzir informação relevante. Certamente, nesse horizonte vislumbra um cenário de oportunidades.

O referencial teórico permitiu o entendimento do assunto que seria abordado, apresentando uma introdução a sensores inteligentes, ao padrão de arquitetura MVC (*Model-View-Controller)* utilizado no desenvolvimento de *softwares* e ao *framework web* Play que foi a principal ferramenta utilizada nesse trabalho.

Play é um *framework* poderoso e alguns pontos no uso dessa ferramenta devem ser apontados: A estrutura do projeto fundamentada no modelo MVC (*Model-View-Controller),* permite que o código fonte e o desenvolvimento em si seja melhor definido e mantido; a simplicidade no uso do ORM (*Object-Relational-Mapping*) Ebean, sem configurações adicionais; a facilidade no o uso e entendimento do arquivo de rotas. Tudo isso somado a ótima documentação, me faz pensar que o Play Framework foi uma escolha sensata.

Certamente o projeto finalizado não possui ambição para ser disponibilizado para produção, devido a pontos que estavam fora do escopo desse trabalho, mas é importante citá-los como uma referência para trabalhos futuros: Em sistemas *web* reais existe a necessidade de assegurar que as informações como credencias do usuário não trafeguem pela interface de rede como texto puro, sendo isso considerado uma grande falha de segurança, para resolver esse problema seria necessário que o sistema possui-se um certificado digital; Uma plataforma real de coleta, armazenamento e análise dos registros de sensores necessita controlar o número de requisições e a carga de processamento utilizada por cada usuário para criar uma forma de cobrança pelo serviço prestado por tipo e uso de recurso; Em um sistema real a complexidade do projeto faz necessário uma equipe multidisciplinar que tenha conhecimento nas áreas de infraestrutura de rede, programação e *design* de *interface.*

Por fim, julgo válido o trabalho desenvolvido, por fornecer recursos básicos de plataformas complexas de gerenciamento de sensores para o usuário final e por permitir ao

autor e as demais pessoas que lerem esse trabalho uma compreensão do que é o Play Framework e como ele pode resolver os problema inerentes a construção de sistemas para Internet.

REFERÊNCIAS

CANALTECH, 2015. **O bilionário mercado da internet das coisas**. Disponível em: <http://corporate.canaltech.com.br/noticia/mercado/o-bilionario-mercado-da-internet-das-coisas-46755/>. Acesso em: 06 out. 2015.

BOAGLIO, Fernando. **Play Framework Java para web sem servelts e com diversão**. São Paulo: Casa do Código, 2014.

DUQUENNOY, S.; GRIMAUD, G.; VANDEWALLE, J. The Web of Things: interconnecting devices with high usability and performance, 2009. In International Conferences on Embedded Software and Systems (ICESS'09).

O'RELLY, Tim. **Web 2.0 Compact Definition: Trying Again**. O'Relly, 10 dez. 2006. Disponível em: <http://radar.oreilly.com/2006/12/web-20-compact-definition-tryi.html>. Acesso em: 29 set 2015.

LEÃO, Erico Meneses. **Uma Arquitetura de Sensores Inteligentes Disparada por Eventos.** Leão, Jul. 2007, Disponível em: <http://www.ufpi.br/subsiteFiles/eml/arquivos/files/artigos/dissertacao_erico.pdf>. Acesso em: 29 out. 2015.

PRESSMAN, Roger. Engenharia de software. McGraw-Hill, 2005.

CHONG, Chee-Yee; KUMAR, Srikanta P. **Sensor networks: Evolution, opportunities, and challenges.** Proceedings of the IEEE, 2003, Vol. 91, pp. 1247–1256.

ELMENREICH, Wilfried; PITZEK, Stefan. **Smart transducers - principles, communications, and configuration.** Proceedings of the 7th IEEE International Conference on Intelligent Engineering Systems (INES), 2003, pp. 510–515.

CASA DO CÓDIGO, 2015. **Play Framework Java para web sem Servlets e com diversão**. Disponível em: <http://www.casadocodigo.com.br/pages/sumario-play-framework-java>. Acesso em: 06 out. 2015.

PLAY DOCUMENTATION, 2015. **The main concepts**. Disponível em: <http://www.playframework.com/documentation/1.0/main>. Acesso em: 10 out. 2015.

PROVOS, Niels; MAZIÈRES, David. **A Future-Adaptable Password Scheme.** 6 Jun. 1999. Provos, Mazières. Disponível em: <https://www.usenix.org/legacy/event/usenix99/provos/provos_html/index.html>. Acesso em: 04 dez. 2015.