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

MOISÉS MEIRELLES FILHO

SISTEMA WEB PARA GERENCIAMENTO DE VÔOS DE AERONAVES NÃO TRIPULADAS

MONOGRAFIA DE ESPECIALIZAÇÃO

PATO BRANCO
2017

MOISÉS MEIRELLES FILHO

ISTEMA WEB PARA GERENCIAMENTO DE VÔOS DE AERONAVES NÃO TRIPULADAS

Trabalho de Conclusão de Curso, apresentado ao 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. Robison Cris Brito

PATO BRANCO

2017

MINISTÉRIO DA EDUCAÇÃO
Universidade Tecnológica Federal do Paraná
Câmpus Pato Branco
Departamento Acadêmico de Informática
Curso de Especialização em Tecnologia Java

TERMO DE APROVAÇÃO

SISTEMA WEB PARA GERENCIAMENTO DE VÔOS DE AERONAVES NÃO TRIPULADAS

por

MOISÉS MEIRELLES FILHO

Este trabalho de conclusão de curso foi apresentado em 01 de março de 2018, como requisito parcial para a obtenção do título de Especialista em Tecnologia Java. Após a apresentação o candidato foi arguido pela banca examinadora composta pelos professores Robison Cris Brito (Orientador), Andreia Scariot Beulke e Beatriz Terezinha Borsoi, membros da banca. Em seguida foi realizada a deliberação pela banca examinadora que considerou o trabalho aprovado.

_____
Robison Cris Brito
Prof. Orientador (UTFPR)

_____
Andreia Scariot Beulke
Banca (UTFPR)

_____
Beatriz Terezinha Borsoi
Banca (UTFPR)

_____
Robison Cris Brito
Coordenador da IV Especialização em Tecnologia Java

A Folha de Aprovação assinada encontra-se na Coordenação do Curso.

AGRADECIMENTOS

Agradeço a todos os gigantes que emprestaram seus ombros para que este trabalho fosse realizado.

Ao meu orientador, Prof. Robison Cris Brito, pelo apoio e força que meu durante todo o desenvolvimento, por me desculpar pelos atrasos e pelo carisma, muito obrigado.

Agradeço a todos os professores que me proporcionaram aprendizado, repassando seus conhecimentos durante toda a especialização.

Agradeço a meus pais, que sempre me apoiaram e me incentivaram a iniciar e também concluir a especialização.

A minha esposa Gabriela Frigotto Zorzan Meirelles, por compartilhar todos os momentos comigo, por me apoiar, suportar meus momentos de ausências e principalmente me incentivar a continuar.

RESUMO

MEIRELLES, Moisés. Sistema web para gerenciamento de vôos de aeronaves não tripuladas. 2017. 47f. Monografia (Trabalho de especialização) – Especialização em Tecnologia Java – Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2017.

Drones ou veículos aéreos não tripulados tem deixado de ser apenas um passatempo de aeromodelismo para ganhar um espaço no mercado, atuando como uma ferramenta importante de captura de dados e auxílio a tarefas cotidianas como gerenciamento agrícola e sistemas de *delivery*, pelos avanços tecnológicos e aperfeiçoamento do hardware que disponibilizam. Assim, o objetivo deste trabalho consiste no desenvolvimento de um sistema de gerenciamento de vôos de aeronaves não tripuladas com objetivo de plantio agrícola, possibilitando o gerenciamento de áreas e as ações realizadas pelos Drones, chamadas missões, utilizando tecnologias de geolocalização. Por meio do sistema é possível armazenar esses dados e o histórico de missões. Para o desenvolvimento do sistema foram utilizadas diversas tecnologias, dentre elas a Linguagem Java que permitiu o desenvolvimento do servidor de gerenciamento. Destacam-se, também, a utilização do *framework* Angular para o desenvolvimento do lado do cliente e da plataforma Node.js para o desenvolvimento do *middleware* responsável pela integração e comunicação com as aeronaves via rede local.

Palavras-chave: Drones. Sistema *web*. Linguagem Java. Node.js. Angular 4.

ABSTRACT

MEIRELLES, Moisés. Web system to manager flights of unmanned aerial vehicles. 2017. 47f. Monografia (Trabalho de especialização) – Especialização em Tecnologia Java – Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2017.

Drones or unmanned aerial vehicles are anymore just a hobby, they are gaining the market space by acting as an important tool to data capture and to help in everyday tasks, such as management of agricultural areas and delivery in ecommerce. This is possible due to the technological advancements and improvement in hardware and software. Thus, the main goal of this work is the development of a flight management system for drones aiming to help agricultural planting, allowing the management of areas and of the actions performed by drones, called missions, using technologies as global positioning system. The software developed allows to store the collected data and mission's history. The software integrates several technologies; among them are the Java Language that allowed the development of the management server; Angular framework for client-side development; and Node.js platform for the development of the middleware, that is responsible for integration and communication with the drone via local network.

Key-words: Unmanned aerial vehicles. *Web system*. Java Language. Node.js. Angular 5.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTAGENS DE CÓDIGO

LISTA DE SIGLAS

API	*Applications Programming Interface*
CSS	*Cascade Style Sheet*
ES6	ECMAScript 6
HTML	*HyperText Markup Language*
HTTP	*Hypertext Transfer Protocol*
JSON	*JavaScript Object Notation*
POJO	*Plain Old Java Objects*
REST	*Representational State Transfer*
RPAS	Remotely Piloted Aircraft Systems
UAV	*Unmanned Aerial Vehicle*
VANT	*Veiculo Aéreo Não Tripulado*

SUMÁRIO

1 INTRODUÇÃO

Nas últimas décadas, sistemas de aeronaves não tripulados ou drone, quadricóptero, Veiculo Aéreo Não Tripulado (VANT), Remotely Piloted Aircraft Systems (RPAS) ou Unmanned Aerial Vehicle (UAV) surgiram em um número crescente de aplicações, principalmente para militares, mas, também, civis. Para os menos familiarizados, drones, generalizando todas as outras denominações para veículos aéreos não tripulados, são veículos aéreos não tripulados, podendo ser controlados por um controle remoto ou por meio de um terminal.

Bastianelli et al. (2012) destacam que existe uma demanda muito grande para uso de drones, seja no meio militar, para reconhecimento e lugares, ou no meio civil em aplicações como monitoração ambiental, mapeamento e agricultura de precisão. As aplicações para um drone hoje são vastas e pouco exploradas a fundo, de modo que ainda existem muitas dificuldades do mapeamento, controle e precisão. Esses aspectos têm motivado diversos estudos científicos para desenvolver soluções melhores em termos de controle dos drones. O fato de ser um equipamento que precisa ser manuseado com cuidado, dificultou por anos sua homologação para uso comercial pela Força Aérea Brasileira (FORÇA AÉREA BRASILEIRA, 2015).

De acordo com Neto et al. (2005), o conceito de agricultura de precisão está normalmente associado à utilização de equipamento de alta tecnologia para avaliar, ou monitorizar, as condições em determinada parcela de terreno, aplicando depois os diversos fatores de produção (sementes, fertilizantes, água, etc.) em conformidade com as condições levantadas. Uma das aplicações de drones estudadas hoje é sua utilização no meio agrícola, são diversos processos ainda não automatizados. Esses processos podem ser adaptados para a utilização de veículos não tripulados com auxílio das técnicas de programação para mapeamento de terrenos, aprendizagem de máquina para decisões de plantio e otimização dos processos de resolução de problemas das lavouras como o alto custo para utilização de máquinas e armazenagem de produtos e insumos.

Drones se destacam na área agrícola como já apresentado por George et al. (2013) que registram que drones fornecem melhores plataformas para avaliar a produção, apresentando vantagens em relação às técnicas atualmente utilizadas via

meio terrestre tradicional, permitindo a leitura de dados em qualquer ambiente e tipo de terreno.

Os próprios drones ainda possuem suas limitações, como a precisão de controle, a capacidade de se adaptar a falhas, segurança e autonomia de vôo. Esses problemas poderão ser solucionados nos próximos anos pelo avanço da tecnologia e descobertas de aprendizagem de máquina, permitindo decisões mais precisas para os dispositivos.

Levando em consideração essas limitações, destaca-se a importância de formas de gerenciar o vôo e supervisionar o funcionamento adequado dos drones para obter um melhor aproveitamento dos seus recursos. E, ainda, a captura desses dados para, em conjunto da tecnologia, obter análises e resultados relevantes para alcançar os objetivos de otimização em resultados no meio agrícola.

Nesse cenário, esse trabalho visa desenvolver um sistema *web* para gerenciamento de missões realizadas por aeronaves não tripuladas. O sistema gerenciará o funcionamento do drone em campo, bem como o cadastro de dados relevantes para uma missão, desde a área até pontos de recarga. O trabalho apresenta também os conceitos técnicos de testes unitários e técnicas avançadas de *deploy* seguindo padrões do mercado bem estabelecidos.

2 MATERIAIS E PROCEDIMENTOS

Esse capítulo tem por objetivo apresentar os materiais utilizados na modelagem e na implementação do aplicativo desenvolvido.

2.1 MATERIAIS

O Quadro 1 apresenta as ferramentas e as tecnologias utilizadas na modelagem e no desenvolvimento do aplicativo. Após o quadro está a descrição das principais tecnologias utilizadas.

Ferramenta / Tecnologia	**Versão**	**Disponível em**	**Aplicação**

Node.js	8.9.0	https://nodejs.org	*Server Framework* Javascript.
Angular	5.0	https://angular.io/	*Framework* Typescript.
Sketchboard.io	1.0	https://sketchboard.me	Modelagem do sistema: diagrama de casos de uso.
Material Design	4.0.0	https://material.angular.io/	*Framework front-end.*
Intellij IDEA	2017.2.5	https://www.jetbrains.com/idea/	Ambiente de desenvolvimento do sistema.
VSCode	1.20.1	https://code.visualstudio.com/	Editor de texto *open-source* para desenvolvimento em qualquer linguagem. Utilizado para desenvolvimento do front-end em Angular.
PostgreSQL	9.6.6	https://www.postgresql.org/	Banco de dados.
PgAdmin III	1.22.2	https://www.pgadmin.org/	Software para administração do banco de dados PostgreSQL.
Java	8	https://java.com/en/download/	Linguagem de programação.
Spring Boot	1.5.4	https://projects.spring.io/spring-boot/	Uso de convenções sobre configurações.
Spring Data	2.0.0	https://projects.spring.io/spring-data- jpa/	*Framework* de implementação de repositórios.
Spring Security	4.2.3	https://projects.spring.io/spring-security/	*Framework* de autenticação e segurança da aplicação.
JUnit	4.0.0	http://junit.org/	Framework *open-source* para implementação de testes unitários..
Balsamiq	Build 1233	https://balsamiq.cloud	Modelagem de protótipos.

Quadro 1 - Tecnologias e ferramentas utilizadas na modelagem e na implementação do aplicativo

2.1.1 Node.js

O Node.js é uma plataforma *open-source* de desenvolvimento que utiliza a linguagem de programação JavaScript e o motor V8 JavaScript Engine para possibilitar o desenvolvimento de aplicações *server-side*.

Lopes (2014) destaca que o Node é *single threaded*. Embora isso possa parecer uma desvantagem, ao desenvolver com Node.js isso simplifica extremamente a construção da aplicação. E por Node.js utilizar uma abordagem não obstrutiva, essa diferença será imperceptível na maioria dos casos.

O Node.js utiliza o V8 Javascript Engine, que é o interpretador de JavaScript *open source* implementado pelo Google em C++ e utilizado pelo Chrome (NODEJS, 2017).

2.1.2 Angular 5

Angular é um *framework* front-end mantido e atualizado pela Google, com objetivo de simplificar o desenvolvimento de aplicações *web*. O Angular implementa o conceito de diretivas e *Two-way Data Binding*, suporta o desenvolvimento de módulos e tem fácil integração com diversos *frameworks* e ferramentas JavaScript.

Angular 5 apresenta a linguagem TypeScript e implementa funcionalidades do ECMAScript 6 (ES6) com tipagens nas variáveis e uma sintaxe mais clara, aproximando-se das linguagens C# e Java (ANGULAR, 2017).

2.1.3 Sketchboard.io

O Sketchboard.io é uma ferramenta *online* para criação de diagramas de sistemas, oferecendo uma gama de componentes visuais para organização de modelos de *design* de software, capaz de salvar os trabalhos em nuvem (SKETCHBOARD, 2017).

2.1.4 Spring (Boot, Data, Security)

O Spring é um framework *open-source* para a plataforma Java criado por Rod Johnson, possui uma arquitetura baseada em interfaces e *Plain Old Java Objects* (POJO), oferecendo características como mecanismos de segurança e controle de transações. Também facilita testes unitários e surge como uma alternativa para outros *frameworks* complexos.

A biblioteca Spring Framework fornece um modelo de desenvolvimento e configuração de aplicativos baseados em Java. O foco do Spring é montar a estrutura para os aplicativos de maneira que as equipes fiquem centradas nas regras de negócio do sistema (SPRING FRAMEWORK, 2017).

2.1.5 Bootstrap

Bootstrap é um *framework open-source* lançado pela empresa Twitter em 2011 para auxiliar no desenvolvimento de *websites* e aplicações *web*. Atua no *front-end* (camada de visualização) por meio de um conjunto de componentes *HyperText Markup Language* (HTML), *Cascade Style Sheet* (CSS) e JavaScript pela biblioteca JQuery. Um de seus recursos mais populares é o de *responsive design (BOOTSTRAP, 2017).*

2.1.6 Junit

O JUnit é um *framework open-source*, que facilita a criação de código para a automação de testes com apresentação dos resultados. Com ele, pode ser verificado se cada método de uma classe funciona da forma esperada, exibindo possíveis erros ou falhas podendo ser utilizado tanto para a execução de baterias de testes como para extensão (JUNIT, 2017).

2.1.7 Balsamiq

Balsamiq é uma ferramenta *online* de *wireframing,* utilizada para a criação de protótipos visuais das telas de sistemas. A ferramenta oferece uma gama de componentes visuais para elaboração do modelo de design visual do software, capaz de salvar os trabalhos em nuvem (BALSAMIQ, 2017).

3 RESULTADOS

Este capítulo apresenta o resultado da realização do trabalho. O capítulo inicia com a descrição do escopo do sistema, seguido da modelagem, apresentação e implementação do sistema desenvolvido.

3.1 ESCOPO DO SISTEMA

O sistema tem o objetivo de gerenciar missões de vôo realizadas por drones conectados a uma rede local e posteriormente possibilitando a gestão de avaliação e análise das áreas cadastradas. O cadastro de áreas é realizado por uma interface de mapa com as missões sendo realizadas por meio de dados de geolocalização, que em conjunto com o hardware do drone permite enviar essas informações para o módulo de GPS da aeronave.

O sistema realiza as missões de plantio nas áreas cadastradas, apresentando dados calculados de tempo e área, quantidade de recargas necessárias para uma missão, bem como o acompanhamento em tempo real dessas missões pela interface do sistema. Todas as missões realizadas são armazenadas, gerando e disponibilizando um histórico da área.

É possível realizar a customização de configurações padrões do sistema, como as métricas utilizadas para cálculos de distância e tempo. O sistema também permite o cadastro de culturas, que são utilizadas na descrição da área para especificar a cultura realizada na missão do plantio.

O sistema possibilita, ainda, o armazenamento das informações de drones conectados, bem como o cadastro de pontos de recarga com os dados de geolocalização associados.

3.2 MODELAGEM DO SISTEMA

A Figura 1 apresenta uma visão geral do sistema para facilitar o entendimento da comunicação entre as partes desenvolvidas.

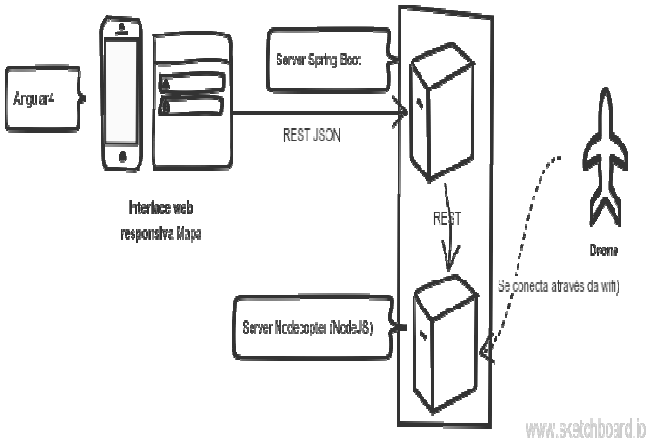

Figura 1 - Visão geral do sistema

A implementação do sistema foi realizada utilizando a linguagem Java (JAVA, 2017) com o banco de dados PostgreSQL (POSTGRESQL, 2017) e camada de aplicação com a utilização do *framework* Spring (SPRING FRAMEWORK, 2017). Para desenvolver a interface foi utilizado o *framework* Angular 5 responsável pela comunicação com o back-end e apresentação dos dados ao usuário.

Os dados foram obtidos a partir da conexão com um servidor de controle implementado utilizando o *framework* Node.js (NODEJS, 2017), que realiza a leitura dos dados das aeronaves conectadas. O Node.js é um *framework* desenvolvido para implementação de sistemas *web*, baseado em JavaScript (MOZILLA, 2017).

3.2.1 Diagrama de Casos de Uso

A Figura 2 apresenta o diagrama de casos de uso do sistema com as principais funcionalidades e os três atores, sendo estes respectivamente: administrador, fazendeiro e sistema.

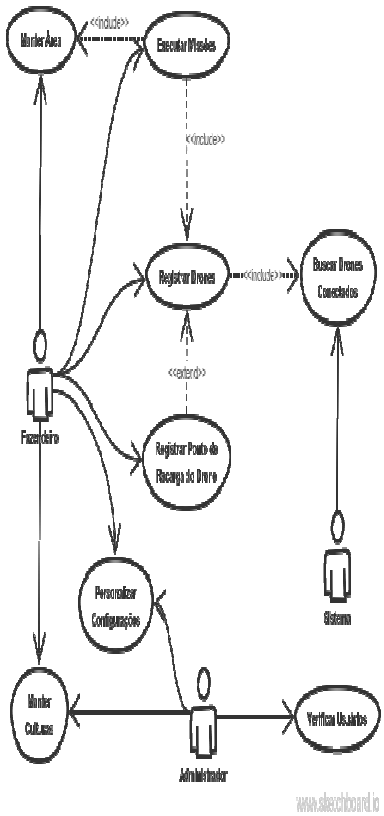

Figura 2 - Diagrama de casos de uso

O fazendeiro é o usuário principal do sistema e, atua diretamente com a maior parte das ações relacionadas às análises e missões realizadas pelos drones, bem como a personalização de configurações e cultura relacionadas ao seu perfil de permissões de acesso. O administrador é responsável pela manutenção geral do sistema personalizando configurações e culturas e supervisão dos usuários que se conectam no sistema. O sistema é responsável pela conexão com os drones utilizando o *middleware* de rede para conexão com os dispositivos.

3.2.2 Protótipos

A Figura 3 apresenta o protótipo criado para apresentar as funcionalidades de cadastro de área dentro do sistema.

Figura 3 - Protótipo da tela de cadastro de área

A Figura 4 apresenta o protótipo das listagens de histórico de missões.

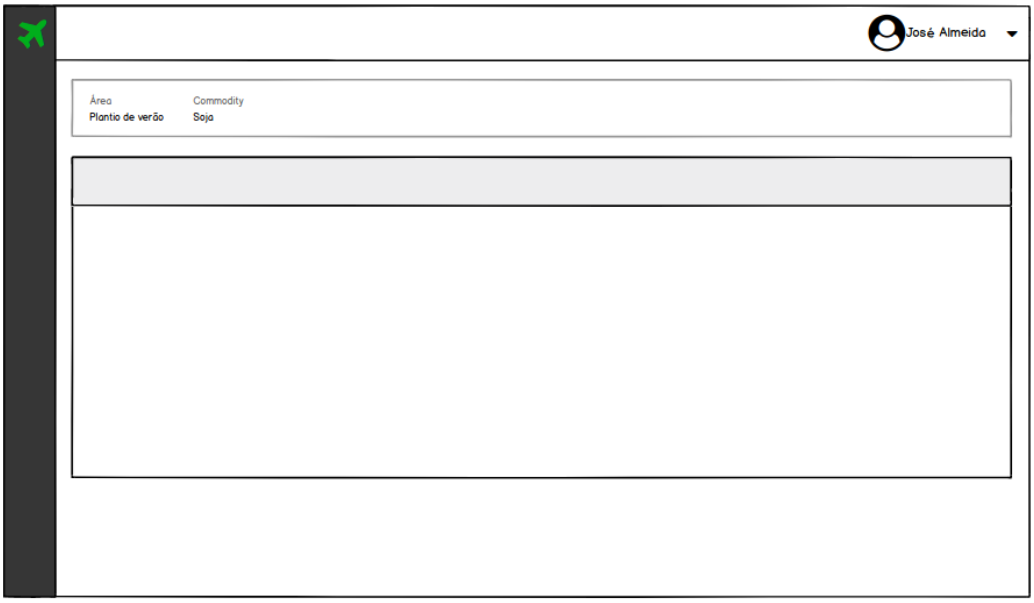

Figura 4 - Protótipo das listagens de histórico de missões

O cadastro de área e realizações de missões é apresento na Figura 5.

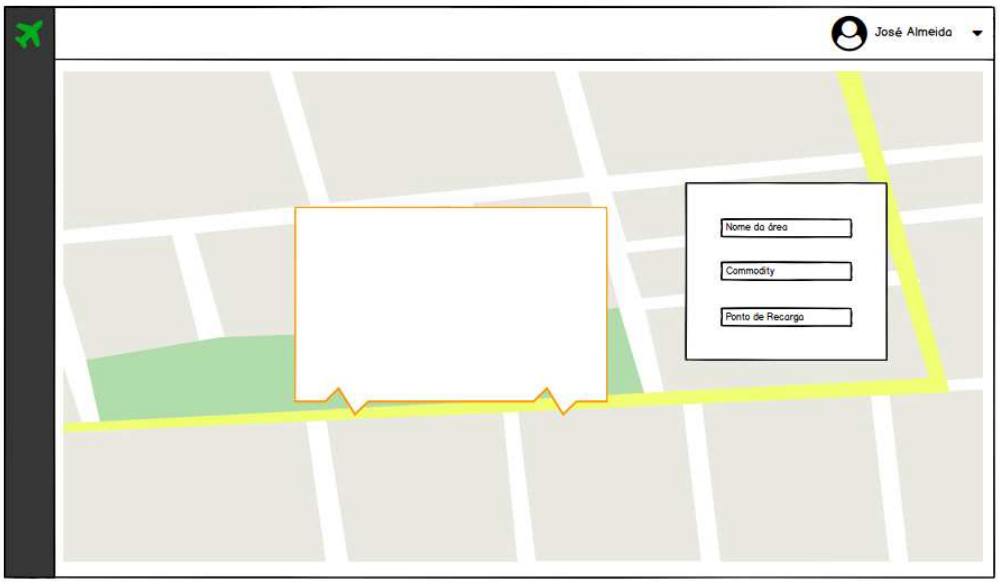

Figura 5 - Protótipo do cadastro de área

A Figura 6 apresenta a visão de detalhes de uma área, sendo possível visualizar suas informações e iniciar uma missão.

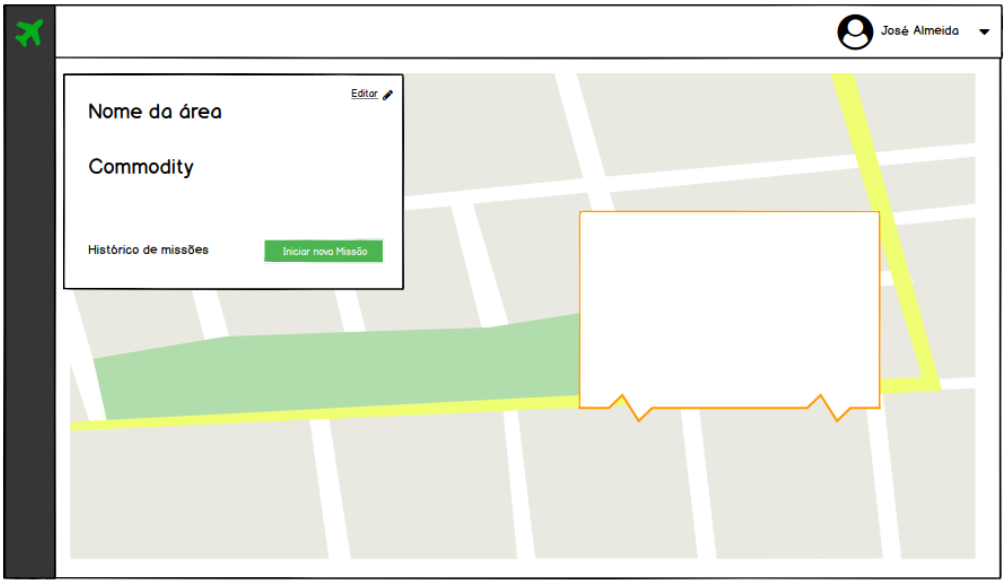

Figura 6 - Protótipo da visão de detalhes de uma área e início de uma missão

3.3 APRESENTAÇÃO DO SISTEMA

A apresentação do sistema desenvolvido é realizada com imagens das telas e

explicações das principais funcionalidades do sistema, que pode ser acessado localmente pela porta 4200.

A tela de acesso e autenticação é apresentada na Figura 7.

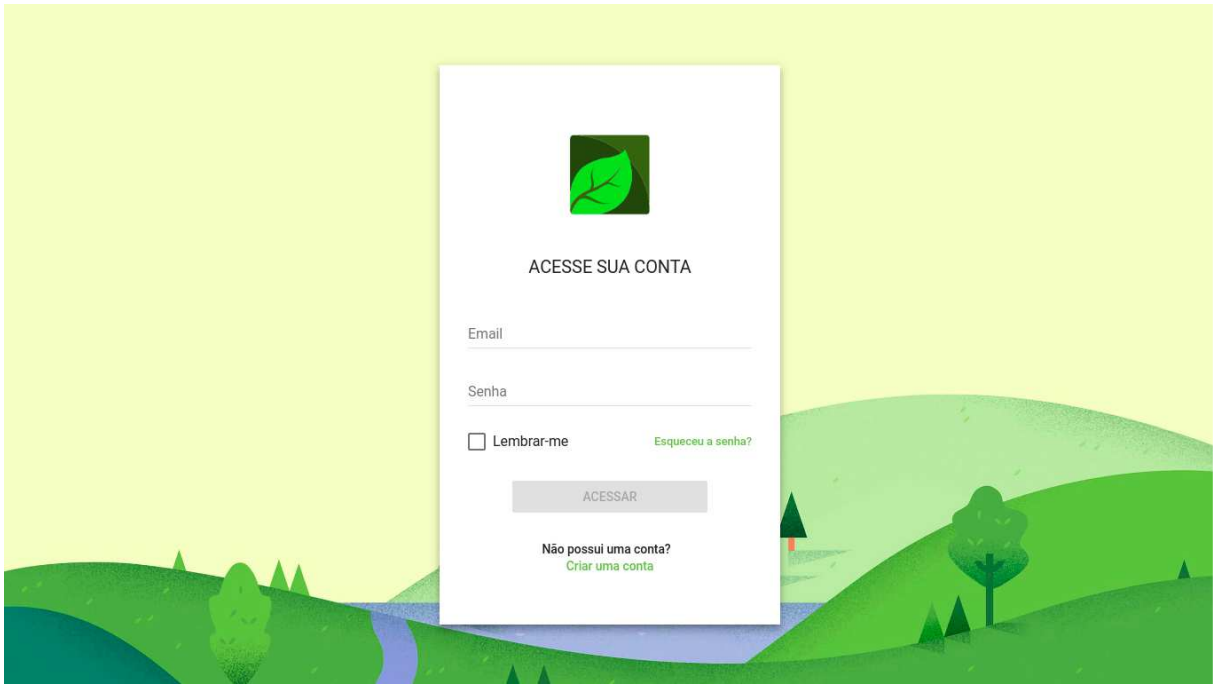

Figura 7 - Tela de autenticação no sistema

Ao se autenticar, o usuário terá acesso às funcionalidades do sistema, podendo efetuar os cadastros, configurar e administrar suas informações para posteriormente efetuar as missões de plantio. A Figura 8 apresenta a tela de *commodities*, equivalente aos cadastros de culturas que o sistema possui.

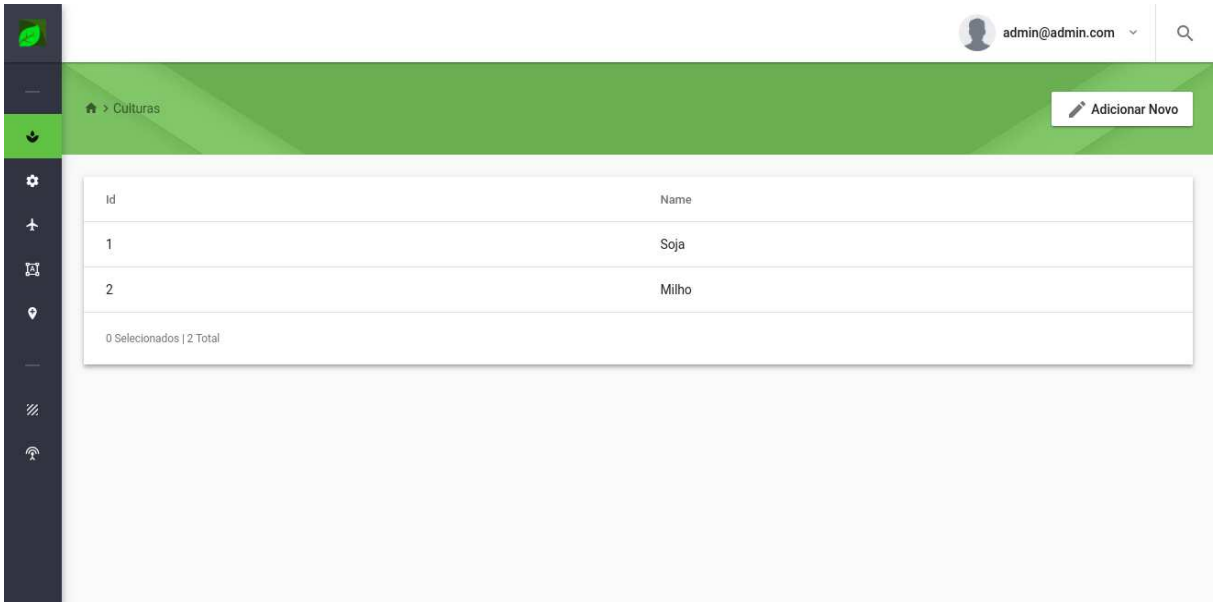

Figura 8
- Tela de culturas

Todas as telas do sistema são acessadas pelo menu lateral, permitindo que o usuário acesse o conteúdo da tela, que é dinamicamente alterado pelo *framework* Angular através do gerenciamento de rotas realizado pelo *framework*. A Figura 9 apresenta os menus que podem ser acessados pelo usuário autenticado.

A Figura 9 apresenta um formulário de cadastro padrão do sistema, referente ao cadastro de novas culturas. Todos os cadastros possuem validações, e permitem a inserção somente se todos os dados obrigatórios forem informados corretamente.

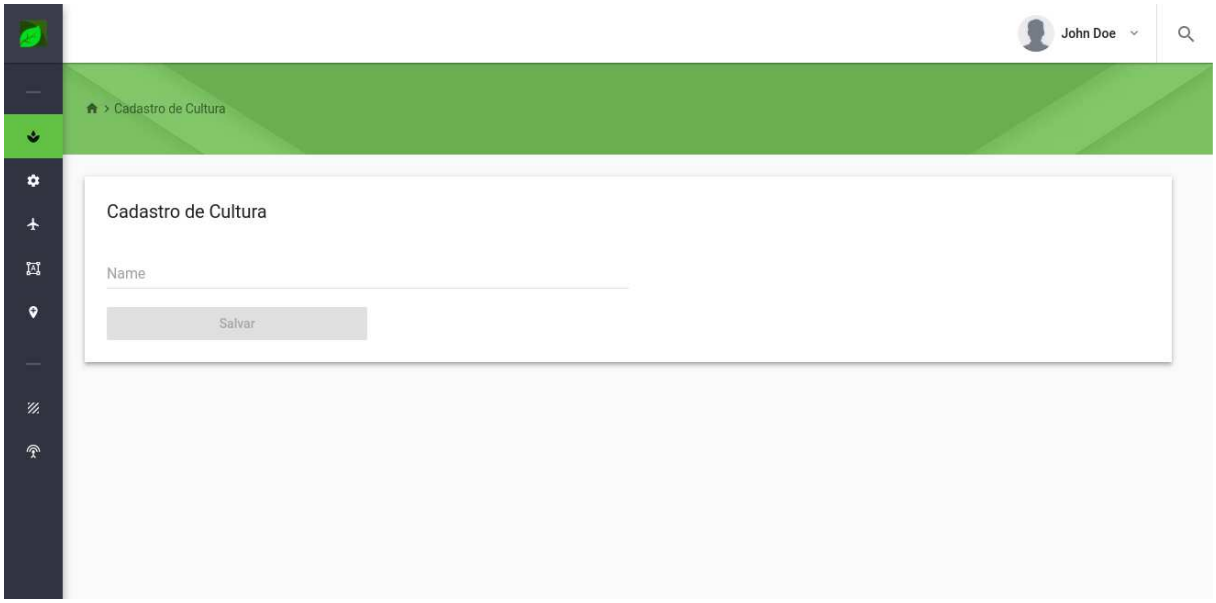

Figura 9 - Tela de cadastro de cultura

A Figura 10 apresenta a tela de configurações do sistema, onde é possível personalizar as unidades de medidas padrão do sistema, bem como as limitações que serão utilizadas para as missões dos drones.

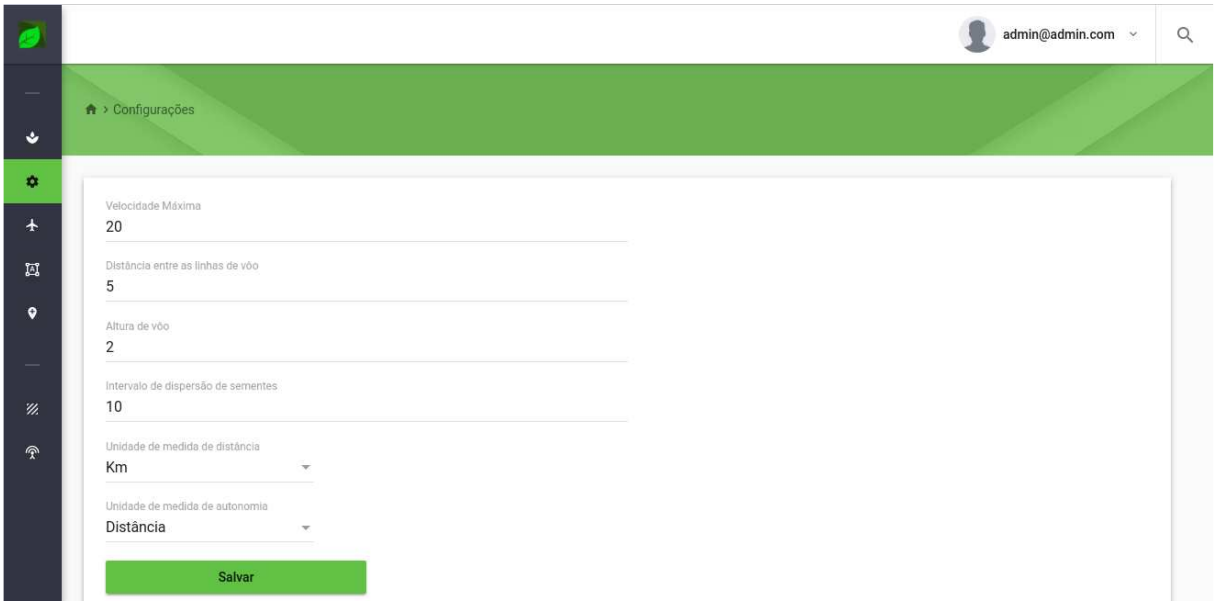

Figura 10 - Tela de configurações do sistema

Ainda através do menu lateral é possível acessa a tela de cadastro de pontos de recarga. Acessando a ação de adicionar novo, é redirecionado para a tela de

cadastro de pontos de recarga, os quais representam um ponto do qual o drone pode autonomamente se direcionar para ser carregado. A Figura 11 apresenta essa tela, que é composta de um componente de mapa integrado ao Google Maps, que ao realizar a ação de clique, mostra uma *popup* de cadastro do novo ponto. O nome do ponto, a informação de se o drone deve se redirecionar de forma autônoma ou não e os dados geográficos capturados do mapa são armazenados no banco de dados.

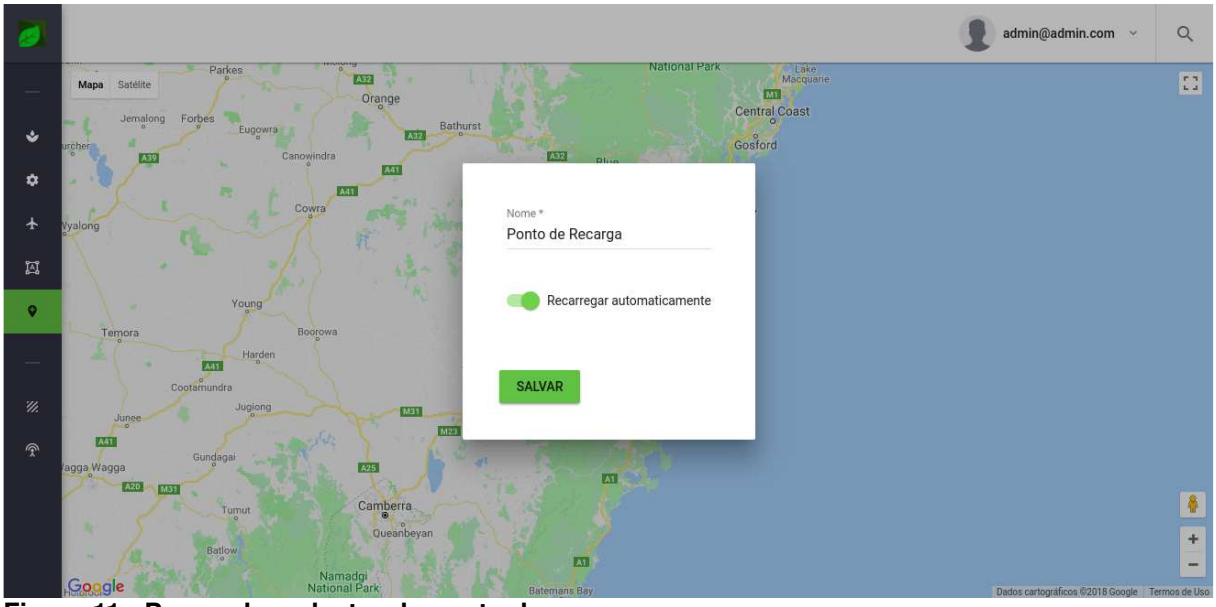

Figura 11 - Popup de cadastro de ponto de recarga

O sistema permite também o cadastro de drones através do menu de cadastro de drones. A figura 12 mostra o menu de acesso lateral para a tela, a qual se caracteriza em um cadastro simples informando o nome e modelo do drone.

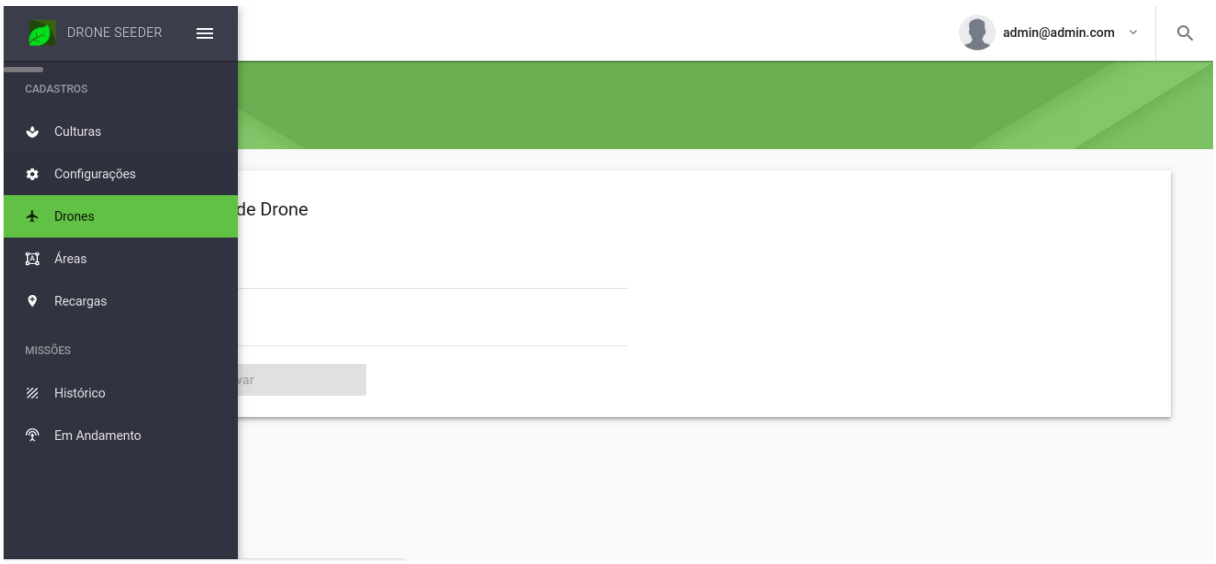

Figura 12 - Menu lateral para acesso a tela de cadastro de drones

Uma vez que o sistema possui pontos de recarga cadastrados, é possível efetuar o cadastro de áreas, que serão essenciais para a execução das missões através do sistema. A tela de cadastro de áreas possui um componente de mapa integrado ao Google Maps, utilizando de uma ferramenta de desenho de polígonos, é possível delimitar o desenho correspondente a área no mapa. Uma vez que o polígono é finalizado, o sistema apresenta uma *popup* de cadastro da área, como apresenta a Figura 13, na qual é necessário informar o título da área, a cultura plantada na área e qual é o ponto de recarga padrão. Esses dados serão armazenados associado aos pontos geográficos correspondentes ao polígono desenhado no mapa.

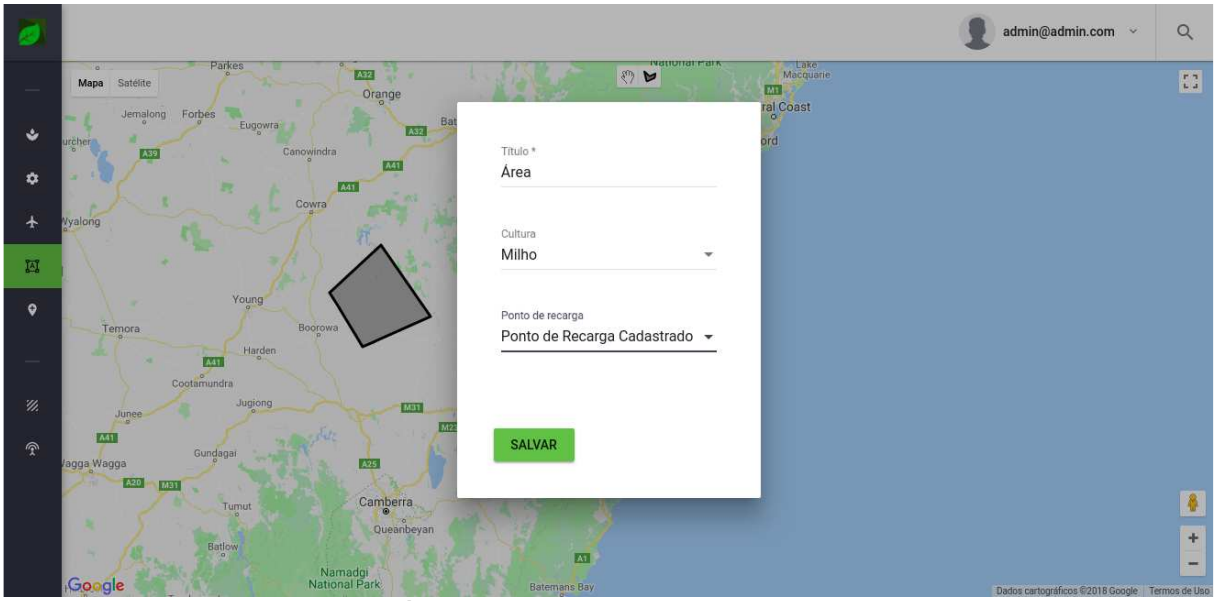

Figura 13 - Tela de cadastro de área

Com uma área cadastrada, é possível acessar o registro recém incluído através da listagem, como apresenta a Figura 14.

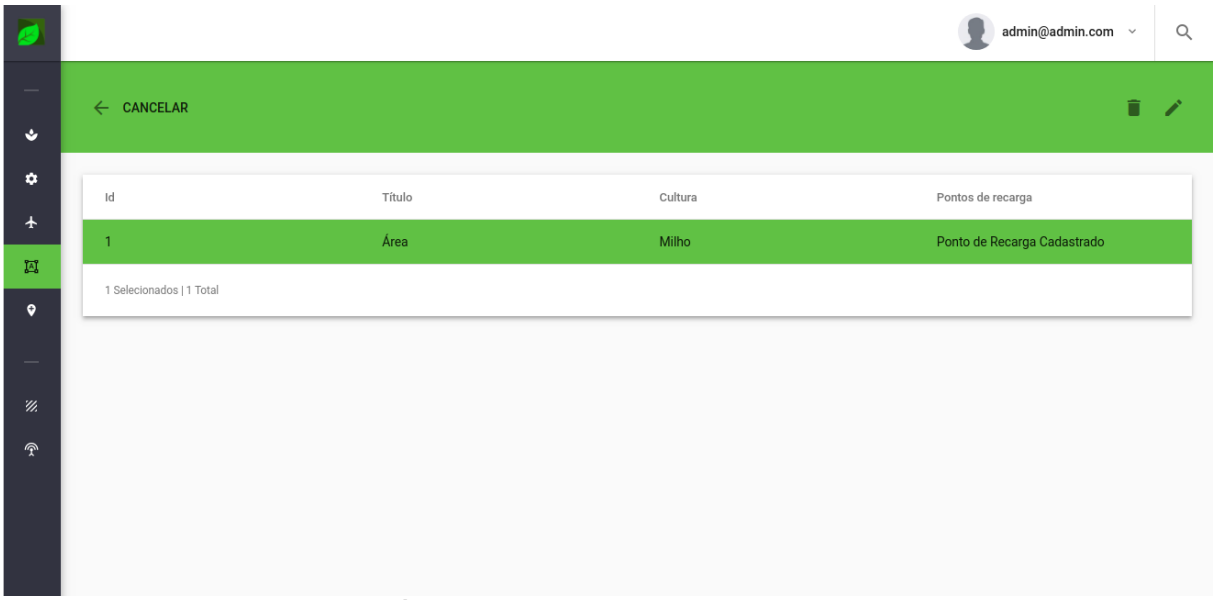

Figura 14 - Tela de listagem de áreas com um registro selecionado

O sistema não permite efetuar edição de áreas, o acesso ao comando habilitado como edição apenas redireciona para uma tela de visualização de detalhes do registro e habilita a opção de execução de uma missão para a

respectiva área, como apresentado na Figura 15.

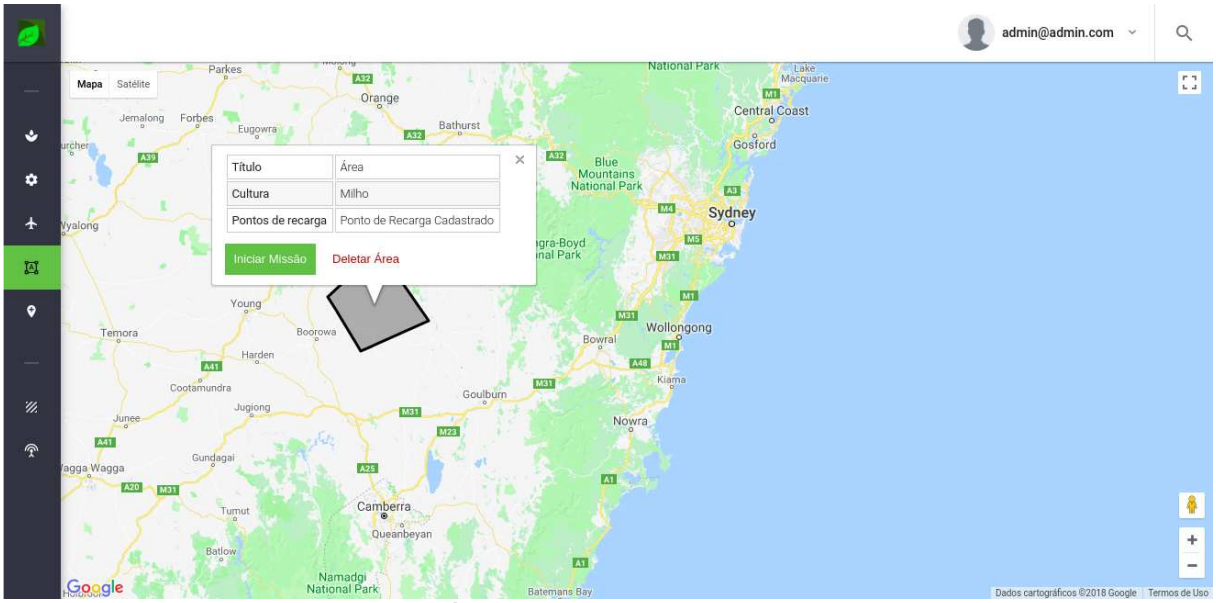

Figura 15 - Tela de visualização de área habilitada para executar uma missão

No momento que o usuário executa uma missão, é solicitado que seja informado um título e selecione qual drone será utilizado para a execução da nova missão. A Figura 16 apresenta a tela de cadastro de missão.

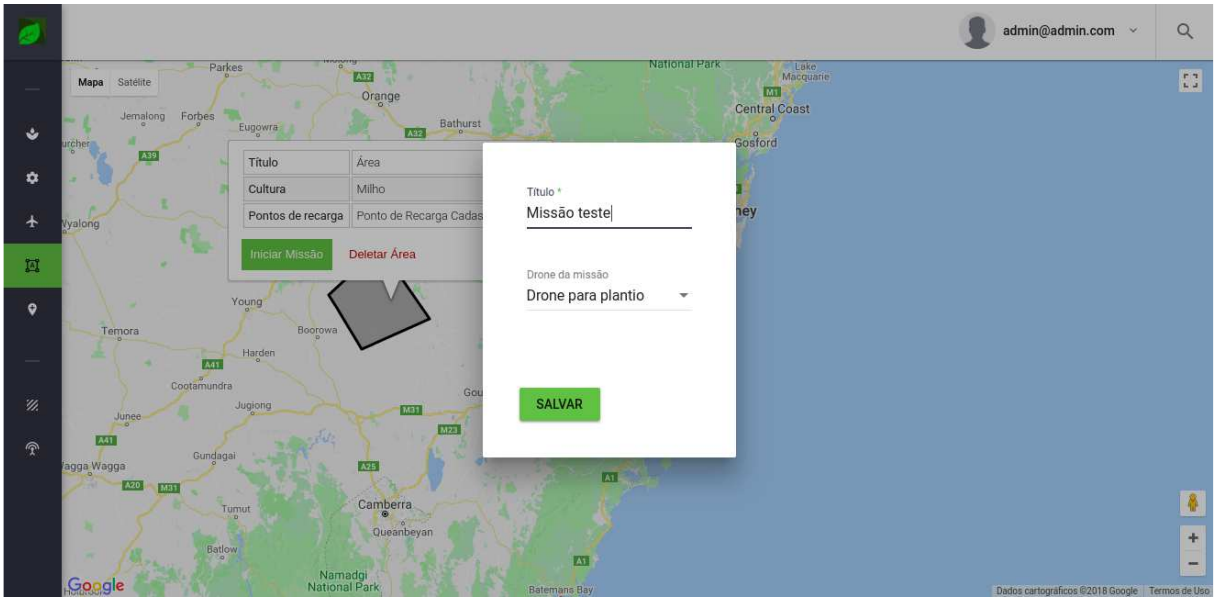

Figura 16 - Tela de execução de missão

Ao confirmar a execução de missão, um novo registro é criado e apresentado na listagem de missões em andamento. Esse registro é mantido e atualizado por uma rotina no sistema que verifica a conclusão das missões, as quais passam a ser apresentadas no histórico de missões apresentando uma data de término e concluindo assim o fluxo de funcionamento do sistema. A Figura 17 apresenta a listagem de missões em andamento.

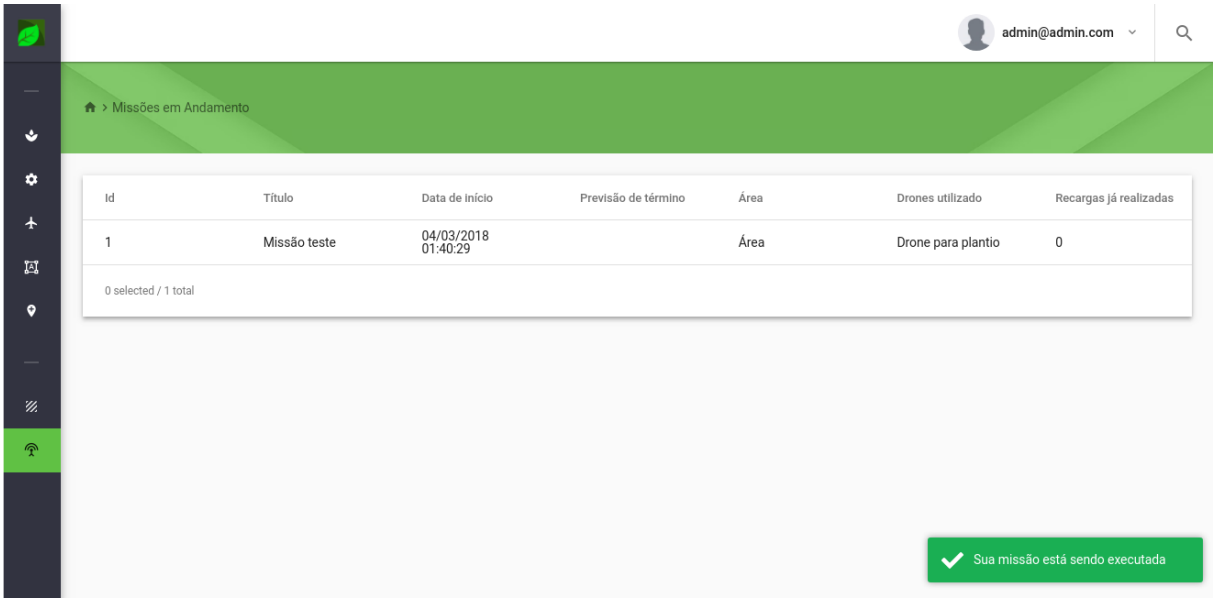

Figura 17 - Tela de listagem de missões em andamento

3.4 IMPLEMENTAÇÃO DO SISTEMA

A seguir são apresentados alguns códigos que exemplificam o desenvolvimento de um sistema utilizando as tecnologias apresentadas no Quadro 1. O desenvolvimento foi separado em três etapas:

a) *Front-end*: apresentação do *framework* Angular 5 e do código do programa responsável pela interface de usuário. O sistema pode executar localmente pela porta 4200 ou ter o *deploy* realizado para um servidor remoto.

b) *Back-end*: apresentação do servidor utilizando *framework* Spring e da linguagem Java. O servidor é responsável pelo armazenamento de dados e autenticação no sistema, se comunicando por meio dos protocolos *Representational State Transfer (REST)* e autenticação via *token* em mensagens no formato *JavaScript Object Notation* (JSON).

c) NodeJS*: API* responsável pela comunicação e execução das tarefas dos drones, enviando e respondendo mensagens pelos protocolos *REST* e autenticação via *token* em mensagens formato JSON.

O desenvolvimento deste projeto foi executado em sistema operacional

Debian GNU/Linux.

A Figura 11 mostra a estrutura utilizada para a implementação do projeto *Applications Programming Interface* (API) REST com Spring.

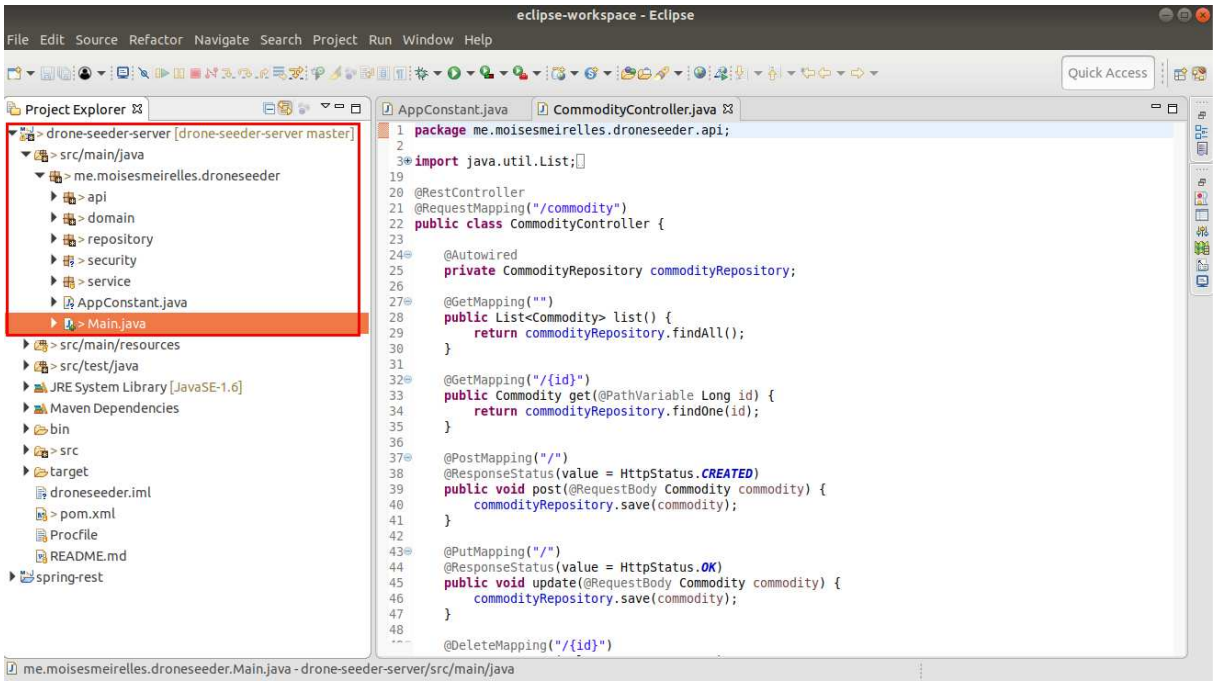

Figura 18 - Estrutura do projeto back-end

A Figura 19 mostra a estrutura utilizada para a implementação do *front-end* utilizando Angular 5, onde as pastas são organizadas com os componentes de suas respectivas áreas de negócio: *field, commodity, drone, recharge, login, mission, settings* e *shared* (em português respectivamente: área, cultura, drone, recarga, login, missão, configurações e compartilhados).

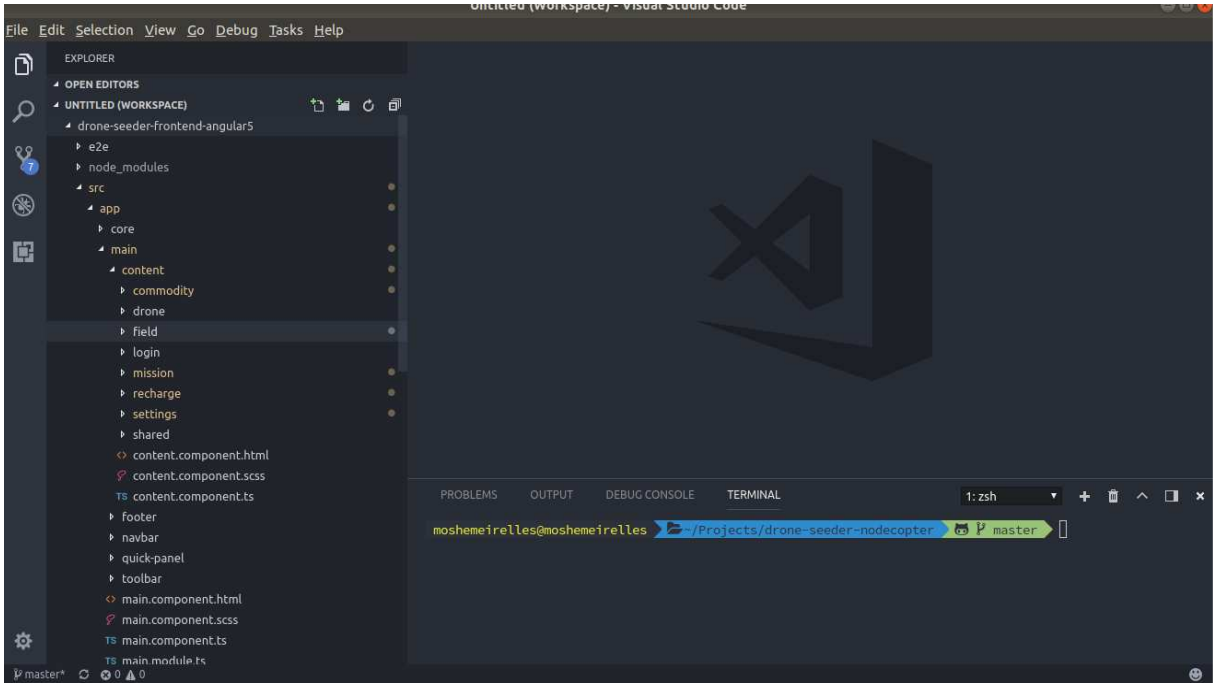

Figura 19 - Estrutura do projeto front-end

A pasta *shared* representa a organização de componentes que podem ser reutilizados por outras telas, utilizando das vantagens e facilidades de componentização proporcionadas pelo *framework* Angular. Um exemplo é o componente *selected-bar* que se trata da barra de ações apresentada ao selecionar um registro nas telas de listagem do sistema. É possível estender esse componente como apresenta o trecho de código retirado da tela de listagem de áreas apresentado na Listagem 1.

```
<!-- SELECTED BAR -->
 <fuse-selected-bar
  class="mat-accent-600-bg h-100"
  *ngIf="selected.length"
  [@slideInTop]
  (onDeselectAll)="deselectAll($event)"
  (onDeleteSelected)="deleteArea($event)"
  (onEditSelected)="editArea($event)"></fuse-selected-bar>
 <!-- / SELECTED BAR -->
```

Listagem 1 - Utilização do componente *selected-bar*

Utilizando da estratégia de componentes reutilizáveis, é possível criar componentes agnósticos que expõem métodos padrões que são chamados ao executarem determinadas ações. Dessa forma, os métodos podem ser sobrescritos pelas telas ou componentes que os utilizarem. A Listagem 2 apresenta o código utilizando no componente *selected-bar* para expor e utilizar os métodos ao clicar no botão de excluir que o componente possui.

```
@Output() onDeselectAll = new EventEmitter();
@Output() onDeleteSelected = new EventEmitter();
@Output() onEditSelected = new EventEmitter();



deleteSelected() {
    this.confirmDialogRef = this.dialog.open(FuseConfirmDialogComponent, {
        disableClose: false
    });

    this.confirmDialogRef.componentInstance.confirmMessage = 'Você tem certeza
que deseja deletar os registros selecionados?';

    this.confirmDialogRef.afterClosed().subscribe(result => {
        if (result) {
            this.onDeleteSelected.emit();
            this.confirmDialogRef = null;
        }
    });
}
```

Listagem 2 - Método de deletar no componente selected-bar

Com a utilização do *framework* Spring Boot, a inicialização do sistema é realizada pela classe principal Main.java, que contém o método *main*, como apresentado na Listagem 3.

```
package me.moisesmeirelles.droneseeder;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Main {
	public static void main(String[] args) throws Exception {
		SpringApplication.run(Main.class, args);
	}
}
```

Listagem 3 - Classe Main.java com o método main

O usuário realiza a autenticação de uma rota específica informando um usuário e uma senha válidos. O servidor retorna seu *token* para a aplicação cliente que e salva localmente essa informação em um *sessionStorage*. Dessa forma, sempre que necessário realizar uma nova requisição, o *token* é transmitido pelo *header* da requisição, evitando assim que o servidor solicite o usuário e a senha novamente para cada requisição.

A Listagem 4 apresenta o código responsável pela validação do usuário e da senha informada, gerando e retornando o *token* para a aplicação cliente.

```
package me.moisesmeirelles.droneseeder.security.controller;

/* Imports ocultados */

@RestController
@RequestMapping("auth")
public class AuthenticationController {
	@Autowired
	private AuthenticationManager authenticationManager;
	@Autowired
	private TokenUtils tokenUtils;
	@Autowired
	private FarmerUserService userService;
```

```
        @RequestMapping(method = RequestMethod.POST)
        public ResponseEntity<?> authenticationRequest(@RequestBody AuthenticationRequest
authenticationRequest){
                Authentication authentication =
                                this.authenticationManager.authenticate(
                                new UsernamePasswordAuthenticationToken(
                                authenticationRequest.getUsername(),
                                authenticationRequest.getPassword()));
        SecurityContextHolder.getContext()
                                .setAuthentication(authentication);
                UserDetails userDetails =
                                this.userService.loadUserByUsername(
                                        authenticationRequest.getUsername());
                String token = this.tokenUtils
                                .generateToken(userDetails);
                return ResponseEntity.ok(
                                new AuthenticationResponse(token));
        }
```

Listagem 4 - Validação de usuário e retorna *token* de autenticação

A cada requisição do cliente é enviado o *token* para o servidor junto com as requisições *Hypertext Transfer Protocol* (HTTP) com das configurações do *header*. Para facilitar o processo foi interceptado a classe responsável pelas requisições padrões do Angular utilizando uma classe que centraliza as requisições necessárias para a aplicação. Essa implementação é apresentada na Listagem 5.

```
Import {Injectable} from '@angular/core';
import {Http, Headers} from '@angular/http';

@Injectable()
export class HttpClient {

   constructor(private http: Http) {}

   createTokenHeader(headers: Headers) {
```

```
        headers.append('X-Auth-Token', sessionStorage.get('X-Auth-Token'));
    }

    get(url) {
        let headers = new Headers();
        this.createTokenHeader(headers);
        return this.http.get(url, {
            headers: headers
        });
    }

    post(url, data) {
        let headers = new Headers();
        this.createTokenHeader(headers);
        return this.http.post(url, data, {
            headers: headers
        });
    }
}
```

Listagem 5 - Wrapper que intercepta requisições HTTP

O *framework* responsável pelas configurações de acesso ao banco de dados e gerenciamento de dependências é o Maven que é configurado por meio do arquivo *pom.xml*. Neste arquivo estão definidas algumas informações do projeto como apresenta a Listagem 6.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>me.mshmeirelles.droneseeder</groupId>
        <artifactId>droneseeder</artifactId>
        <version>0.0.1</version>
```

```
<name>droneseeder</name>

<parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>1.5.4.RELEASE</version>
</parent>

<dependencies>
        <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-actuator</artifactId>
        </dependency>
        <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-test</artifactId>
                <scope>test</scope>
        </dependency>
        <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-security</artifactId>
        </dependency>
        <dependency>
                <groupId>org.postgresql</groupId>
                <artifactId>postgresql</artifactId>
        </dependency>
        <dependency>
                <groupId>com.h2database</groupId>
                <artifactId>h2</artifactId>
```

```
                <scope>test</scope>
            </dependency>
            <dependency>
                <groupId>org.projectlombok</groupId>
                <artifactId>lombok</artifactId>
                <scope>provided</scope>
            </dependency>
            <!-- Allow for automatic restarts when classpath contents change. -->
            <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-devtools</artifactId>
                <optional>true</optional>
            </dependency>
            <dependency>
                <groupId>io.jsonwebtoken</groupId>
                <artifactId>jjwt</artifactId>
                <version>0.6.0</version>
            </dependency>
        </dependencies>
        <build>
            <plugins>
                <plugin>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-maven-plugin</artifactId>
                </plugin>
            </plugins>
        </build>
</project>
```

Listagem 6 - pom.xml

Outros arquivos de configuração necessários são os arquivos *application.properties, application-dev.properties* e *application-prod.properties*. Nesses arquivos são definidas as propriedades da base de dados e as variáveis de ambiente utilizadas no projeto. O *framework* Spring permite utilizar diferentes perfis, por isso a criação de arquivos diferentes para os ambientes de desenvolvimento e

produção, permitindo que seja possível configurar variáveis locais ou variáveis específicas para o ambiente de produção. As Listagens 7 e 8 apresentam os códigos dos arquivos *application.properties* e *application-prod.properties* respectivamente.

```
spring.profiles.active=prod
```

Listagem 7 - application.properties

```
spring.jpa.hibernate.ddl-auto=create-drop
spring.jpa.show-sql: true
spring.jpa.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
spring.jpa.database=POSTGRESQL
spring.datasource.platform=postgres
spring.datasource.url=jdbc:${DATABASE_URL}
spring.datasource.driverClassName=org.postgresql.Driver
```

Listagem 8 - application-prod.properties

O servidor implementa o mapeamento de entidades relacionais, como na Listagem 9 representando a entidade *Commodity*, que se trata de uma entidade de domínio representando as culturas configuradas no sistema.

```
package me.moisesmeirelles.droneseeder.domain;

import java.io.Serializable;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

import lombok.Data;
import lombok.EqualsAndHashCode;

@Entity
@Data
@EqualsAndHashCode(of = "id")
public class Commodity implements Serializable {
	private static final long serialVersionUID = 1L;
	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private Long id;
	@Column(name = "name", length = 100, nullable = false)
	private String name;
}
```

Listagem 9 - Classe Commodity

São criados também os *controllers* que fornecem a interface de comunicação por meio dos protocolos REST, definem a forma como os dados são enviados e recebidos pelo b*ack-end* e salvando os dados por meio da comunicação com os *Repositories*. Um *controller* é identificado pela anotação @RestController. A Listagem 10 contém o código do *controller* de *commodities*.

```
package me.moisesmeirelles.droneseeder.api;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestController;

import me.moisesmeirelles.droneseeder.domain.Commodity;
import me.moisesmeirelles.droneseeder.repository.CommodityRepository;

@RestController
@RequestMapping("/commodity")
public class CommodityController {

	@Autowired
	private CommodityRepository commodityRepository;

	@GetMapping("")
	public List<Commodity> list() {
		return commodityRepository.findAll();
	}

	@GetMapping("/{id}")
	public Commodity get(@PathVariable Long id) {
		return commodityRepository.findOne(id);
	}

	@PostMapping("/")
```

```
        @ResponseStatus(value = HttpStatus.CREATED)
        public void post(@RequestBody Commodity commodity) {
                commodityRepository.save(commodity);
        }

        @PutMapping("/")
        @ResponseStatus(value = HttpStatus.OK)
        public void update(@RequestBody Commodity commodity) {
                commodityRepository.save(commodity);
        }

        @DeleteMapping("/{id}")
        @ResponseStatus(value = HttpStatus.OK)
        public void delete(@PathVariable Long id) {
                commodityRepository.delete(id);
        }
}
```

Listagem 10 - Commodity Controller

A aplicação *front-end* realiza uma requisição HTTP Get para o servidor b*ack-end* que retorna um objeto no formato JSON. A Listagem 11 apresenta a requisição de *commodities* realizada no componente Commodity da aplicação *front-end* equivalente à tela de listagem de culturas cadastradas.

```
import { Component, OnInit } from '@angular/core';
import { HttpClient } from './http-client';

@Component({
    selector: 'app-commodity',
    templateUrl: './commodity.component.html',
    styleUrls: ['./commodity.component.scss']
})
export class CommodityComponent implements OnInit {

    rows: any[];
    loadingIndicator = true;
    reorderable = true;

    constructor(private http: HttpClient) {}

    ngOnInit() {
        this.http.get('/commodity').subscribe((commodities: any) => {
            this.rows = commodities;
            this.loadingIndicator = false;
        });
    }
}
```

Listagem 11 - Implementação da requisição no arquivo commodity.component.ts

A implementação da interface usando HTML está associada ao componente Commodity, representado pelo arquivo *commodity.component.ts* por meio do funcionamento do *framework* Angular 5, o código para a apresentação dos dados está na Listagem 12.

```
<div id="commodity" class="page-layout simple fullwidth" fusePerfectScrollbar>
    <!-- HEADER -->
    <div class="header mat-accent-bg p-24 h-100" fxLayout="column" fxLayoutAlign="center
center" fxLayout.gt-xs="row" fxLayoutAlign.gt-xs="space-between center">
        <div fxLayout="column" fxLayoutAlign="center center" fxLayout.gt-xs="column"
fxLayoutAlign.gt-xs="center start">
            <div class="black-fg" fxLayout="row" fxLayoutAlign="start center">
                <mat-icon class="secondary-text s-16">home</mat-icon>
                <mat-icon class="secondary-text s-16">chevron_right</mat-icon>
                <span class="secondary-text">Culturas</span>
            </div>
        </div>

        <a mat-raised-button class="reference-button mat-white-bg mt-16 mt-sm-0"
[routerLink]="['/commodity/new']">
            <mat-icon>edit</mat-icon>
            <span>Adicionar Novo</span>
        </a>
    </div>
    <!-- / HEADER -->

    <!-- CONTENT -->
    <div class="content p-24">
        <ngx-datatable class="material" [rows]="rows" [loadingIndicator]="loadingIndicator"
[columnMode]="'force'" [headerHeight]="48" [footerHeight]="56" [rowHeight]="'auto'"
[scrollbarH]="true" [reorderable]="reorderable" [selectionType]="'checkbox'" [limit]="10">

            <ngx-datatable-column [width]="48" [canAutoResize]="false" [sortable]="false">
                <ng-template ngx-datatable-header-template let-value="value" let-
allRowsSelected="allRowsSelected" let-selectFn="selectFn">
                    <mat-checkbox [checked]="allRowsSelected"
(change)="selectFn(!allRowsSelected)"></mat-checkbox>
                </ng-template>
```

```
            <ng-template ngx-datatable-cell-template let-value="value" let-
isSelected="isSelected" let-onCheckboxChangeFn="onCheckboxChangeFn">
                <mat-checkbox [checked]="isSelected"
(change)="onCheckboxChangeFn($event)"></mat-checkbox>
            </ng-template>
        </ngx-datatable-column>

        <ngx-datatable-column name="Id" prop="id"></ngx-datatable-column>

        <ngx-datatable-column name="Name" prop="name"></ngx-datatable-column>
    </ngx-datatable>
  </div>
</div>
```

Listagem 12 - Implementação da interface no arquivo commodity.component.html

O cadastro de cultura é realizado por meio da tela mapeada no sistema de rotas do *framework*, enviando uma requisição HTTP Post para o servidor *back-end,* como apresenta o código na Listagem 13.

```
Import { Component, OnInit } from '@angular/core';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';

import { CommodityService } from '../commodity.service';

@Component({
    selector: 'app-commodity-form',
    templateUrl: './commodity-form.component.html',
    styleUrls: ['./commodity-form.component.scss']
})
export class CommodityFormComponent implements OnInit {

    form: FormGroup;
    formErrors: any;
    formType: string;

    constructor(private formBuilder: FormBuilder,
        private commodityService: CommodityService) {
        this.formErrors = {
            name: {}
        }
    }

    ngOnInit() {
        this.form = this.formBuilder.group({
            name: ['', Validators.required],
        });

        this.form.valueChanges.subscribe(() => {
            this.onFormValuesChanged();
        });

        this.formType = 'new';
    }
```

```
  addCommodity() {
    this.commodityService.saveCommodity(this.form.getRawValue());
  }

  onFormValuesChanged() {
    for (const field in this.formErrors) {
      if (!this.formErrors.hasOwnProperty(field)) {
        continue;
      }

      // Clear previous errors
      this.formErrors[field] = {};

      // Get the control
      const control = this.form.get(field);

      if (control && control.dirty && !control.valid) {
        this.formErrors[field] = control.errors;
      }
    }
  }
}
```

Listagem 13 - Implementação do cadastro no arquivo commodity-form.component.ts

A interface do cadastro de cultura é desenvolvida associada ao componente de formulário da tela de *commodity* realizando as operações e validações necessárias como apresenta a Listagem 14.

```
<div id="commodity-forms" class="page-layout simple fullwidth" fxLayout="column"
fusePerfectScrollbar>
    <!-- HEADER -->
    <div class="header mat-accent-bg p-24 h-100" fxLayout="row" fxLayoutAlign="start
center">
        <div fxLayout="column" fxLayoutAlign="center start">
            <div class="black-fg" fxLayout="row" fxLayoutAlign="start center">
                <mat-icon class="secondary-text s-16">home</mat-icon>
                <mat-icon class="secondary-text s-16">chevron_right</mat-icon>
                <span class="secondary-text">Cadastro de Cultura</span>
            </div>
        </div>
    </div>
    <!-- / HEADER -->

    <!-- CONTENT -->
    <div class="content p-24">
        <div fxLayout="column" fxLayoutAlign="start start" fxLayout.gt-md="row">
            <form class="mat-white-bg mat-elevation-z4 p-24 mr-24 mb-24" fxLayout="column"
fxLayoutAlign="start" fxFlex="1 0 auto" name="form" [formGroup]="form"
(submit)="addCommodity()">
                <div class="h2 mb-24">Cadastro de Cultura</div>

                <div fxLayout="row" fxLayoutAlign="start center" fxFlex="1 0 auto">
                    <mat-form-field fxFlex="50">
                        <input matInput placeholder="Name" formControlName="name">
                        <mat-error *ngIf="formErrors.name.required">
                            Required
                        </mat-error>
                    </mat-form-field>
                </div>

                <button *ngIf="formType === 'new'" mat-raised-button class="mat-accent w-25-p"
[disabled]="form.invalid">Salvar
```

```
      </button>
        </form>
      </div>
    </div>
</div>
```

Listagem 14 - Implementação da interface no arquivo commodity-form.component.html

O sistema efetua a comunicação com uma API desenvolvida com ES6 em NodeJS, que busca os dados atuais dos drones e drones conectados na rede e retorna através de uma interface HTTP. A Figura 20 apresenta a estrutura do projeto da API em NodeJS.

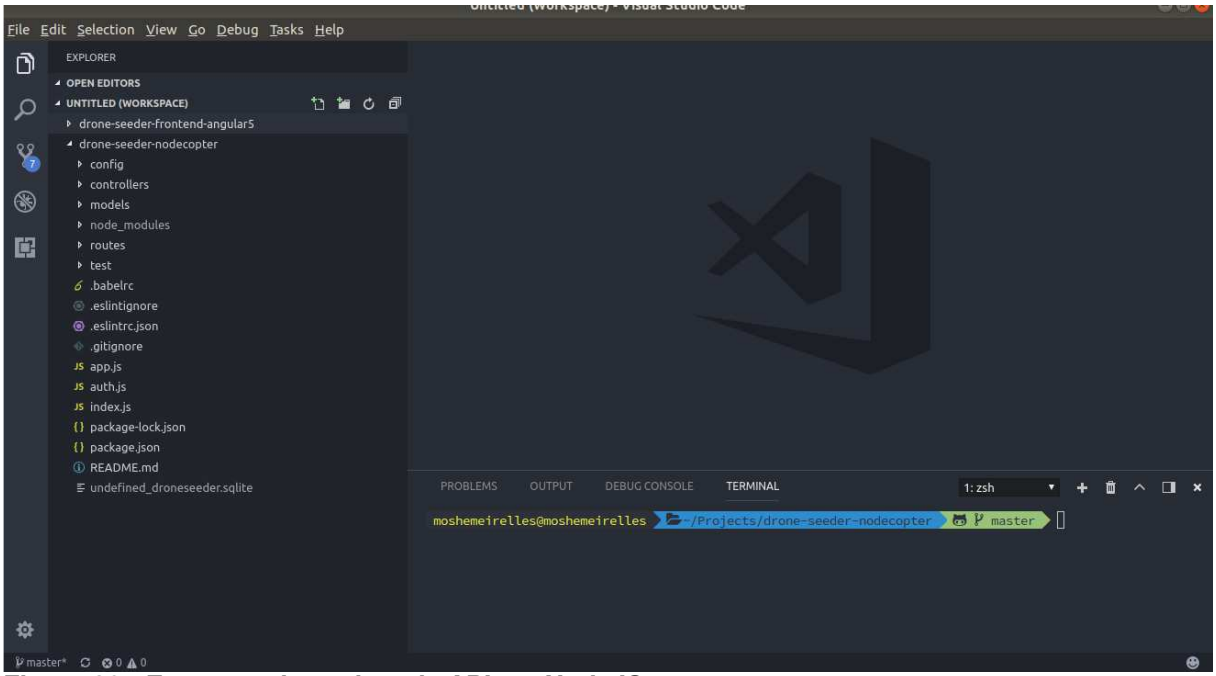

Figura 20 - Estrutura do projeto da API em NodeJS

A API possui a capacidade de retornar dados de um drone conectado, retornado a última latitude e longitude acessa pelo drone. Essa informação é utilizada para validação da conclusão de missões dentro do back-end desenvolvido com Spring. A Listagem 15 apresenta o código do *controller* de vôos de drones da

API em NodeJS que é responsável por retornar a última localização geográfica do drone conectado.

```
import HttpStatus from 'http-status';
import * as arDrone from 'ar-drone';


const defaultResponse = (data, statusCode = HttpStatus.OK) => ({
    data,
    statusCode,
});

const errorResponse = (message, statusCode = HttpStatus.BAD_REQUEST) =>
defaultResponse({
    error: message,
}, statusCode);

class DronesController {

    constructor() {
        this.client = arDrone.createClient();
    }

    getFlightData() {
        return new Promise((resolve, reject) => {
            client.on('navdata', (data) => {
              resolve(data);
            })
        })
    }
}

export default DronesController;
```

Listagem 15 - *Controller* de drones da API em NodeJS

4 CONCLUSÃO

Como forma de exemplificar o uso da *drones* no meio agrícola, o presente trabalho teve como objetivo a implementação de um sistema *web* utilizando o *framework* Spring e a linguagem de programação Java para gerenciamento de aeronaves não tripuladas em missões de plantio.

O trabalho realizado apresenta a forma de uso da comunicação entre sistemas e *drones* utilizando a linguagem Java e o *framework* NodeJS. Os dados das leituras realizadas são armazenados em banco de dados por meio de um sistema *web*. Esses dados são visualizados e manipulados por um sistema *web*, que possui uma interface de usuário.

O Node.js se mostrou um *framework* eficiente e de alto desempenho, disponibilizando o *middleware* de comunicação com os *drones* de maneira eficaz e uma interface de comunicação HTTP de maneira simples atendendo o objetivo de comunicação com as aeronaves

O Spring é um *framework* para desenvolvimento *web* e mostrou-se muito eficiente por ser bem documentado e fornecer os recursos que facilitaram o desenvolvimento do sistema *web*. Entre os destaques do Spring pode-se citar: o módulo de autenticação de usuários, as rotas e os controladores.

Os objetivos deste trabalho foram alcançadas, apesar das dificuldades encontradas pela inserção de novas tecnologias e a curva de aprendizado necessária, bem como a escolha de um tema fora das demandas comuns de tecnologia e mercado.

Como complemento a este trabalho, explorando ainda mais a aplicabilidade dos *drones*, sugere-se expandir a quantidade de dados coletados bem como expandir a comunicação do sistema com as aeronaves aproveitando ainda mais os recursos dos sensores disponibilizados pelo dispositivo. E, ainda, otimizar o sistema aperfeiçoando os módulos que apresentaram neste trabalho, uma demonstração menos completa de todo o potencial a ser desenvolvido.

REFERÊNCIAS

ANGULAR. What is Angular? Disponível em: <https://angular.io/>. Acesso em: 01 dez. 2017.

BALSAMIQ, 2017. About Balsamiq. Disponível em: <https://balsamiq.cloud/>. Acesso em: 05 set. 2017.

BASTIANELLI, G., SALAMON, D., SCHISANO, A., IACOBACCI, A.. **Agent-based simulation of collaborative unmanned satellite vehicles.** In 2012 IEEE First AESS European Conference on Satellite Telecommunications (ESTEL). Institute of Electrical & Electronics Engineers (IEEE), 2012.

FORÇA AÉREA BRASILEIRA. **Comando da aeronáutica publica nova legislação sobre aeronaves remotamente pilotadas**. 2015. Disponível em <http://www.fab.mil.br/noticias/mostra/23937/>. Acessado em 15 de dezembro de 2017.

GEORGE, E. A., TIWARI, G., YADAV, R. N., PETERS, E., SADANA, S. **UAV systems for parameter identification in agriculture**. In 2013 IEEE Global Humanitarian Technology Conference: South Asia Satellite (GHTC-SAS). Institute of Electrical & Electronics Engineers (IEEE), 2013.

LOPES, Cosme. **O que é Node.js e saiba os primeiros passos**. 2014. Disponível em: <https://tableless.com.br/o-que-nodejs-primeiros-passos-com-node-js/>. Acessado em 21 de agosto de 2017.

JAVA, 2017. Java Language. Disponível em: <https://www.java.com/pt_BR/about/whatis_java.jsp>. Acesso em: 06 set. 2017.

JUNIT, 2017. Using Junit. Disponível em: <https://junit.org/junit5/docs/current/user-guide/>. Acesso em: 06 set. 2017.

NETO, M. C., PINTO, P. A., COELHO, J. P. P. **Tecnologias de informação e comunicação e a agricultura**. Porto: Sociedade Portuguesa de Inovação, 2005.

NODEJS, 2017. Getting Started using Node.js. Disponível em: <https://nodejs.org/en/about/>. Acesso em: 28 nov. 2017.

MOZILLA, 2017. JavaScript. Disponível em: <https://developer.mozilla.org/en-US/docs/JavaScript>. Acesso em: 15 nov. 2017.

POSTGRESQL, 2017. Postgresql. Disponível em: <https://nodejs.org/en/about/>. Acesso em: 28 nov. 2017.

SKETCHBOARD, 2017. About. Disponível em: <https://sketchboard.io/>. Acesso em: 29 nov. 2017.

SPRING FRAMEWORK, 2017. About Spring Framework. Disponível em: <https://spring.io/>. Acesso em: 01 dez. 2017.