Projeto Avaliativo 2 - DEV-Agro

Peso: 45% da nota do módulo 1

TECHDive



# Sumário

[Sumário](#_xa80pxa96pvy) **1**

[1. Introdução](#_p9e2s15k061r) **1**

[2. Requisitos da Aplicação](#_f0ig6qopez2r) **1**

[3. Entrega](#_8wfii4usjssb) **1**

[4. Critérios de Avaliação](#_q9pybibt78de) **2**

[5. Roteiro da aplicação](#_g4qyulfrxbb8) **3**

# 1. Introdução

Considerando os assuntos estudados em todas as semanas anteriores (tipos de dados, operadores, orientação a objetos, tratamento de exceções, SQL, Maven, Spring, Spring Boot, Spring MVC, Spring Data) tente aplicar todos os conceitos estudados no problema abordado abaixo.

# 2. Requisitos da Aplicação

A aplicação que deverá ser realizada individualmente deve contemplar os seguintes requisitos:

* O sistema deve ser desenvolvido em Java com Spring.
* O sistema deve utilizar banco de dados PostgreSQL.
* O sistema deve seguir o Roteiro da Aplicação.
* O sistema deve ser uma API que se comunica com outros serviços (por exemplo, front-end web ou Postman) através de requisições HTTP.

# 3. Entrega

O código desenvolvido deverá ser submetido no GitHub, e o **link** deverá ser disponibilizado na tarefa **Módulo 1 - Projeto Avaliativo 2**, presente na **semana 12** do AVA até o dia **20/03/2022 às 23h55**.

Entregas realizadas após a data limite sofrerão **decréscimo na nota** de avaliação, sendo considerado **80%** da nota para tarefas submetidas até o dia **27/03/2022 às 23h55**. Não serão avaliados projetos submetidos após a data de término da atividade.

**Importante:** Será considerado como data de entrega a **última atualização** no repositório do projeto no GitHub. Lembre-se de não modificar o código até receber sua nota.

O repositório deverá ser **privado** e deverá adicionar as seguintes pessoas no repositório:

* [fernando.puntel@edu.sc.senai.br](mailto:fernando.puntel@edu.sc.senai.br)
* [jvmdeoliveira@gmail.com](mailto:jvmdeoliveira@gmail.com)

# 4. Critérios de Avaliação

A tabela abaixo apresenta os critérios que serão avaliados durante a correção do projeto. O mesmo possui variação de nota de 0 (zero) a 10 (dez) como nota mínima e máxima, e possui peso de 45% sobre a avaliação do módulo 1.

**Serão desconsiderados e atribuída a nota 0 (zero)** os projetos que apresentarem plágio de soluções encontradas na internet ou de outros colegas. Lembre-se: Você está livre para utilizar outras soluções como base, mas **não é permitida** a cópia.

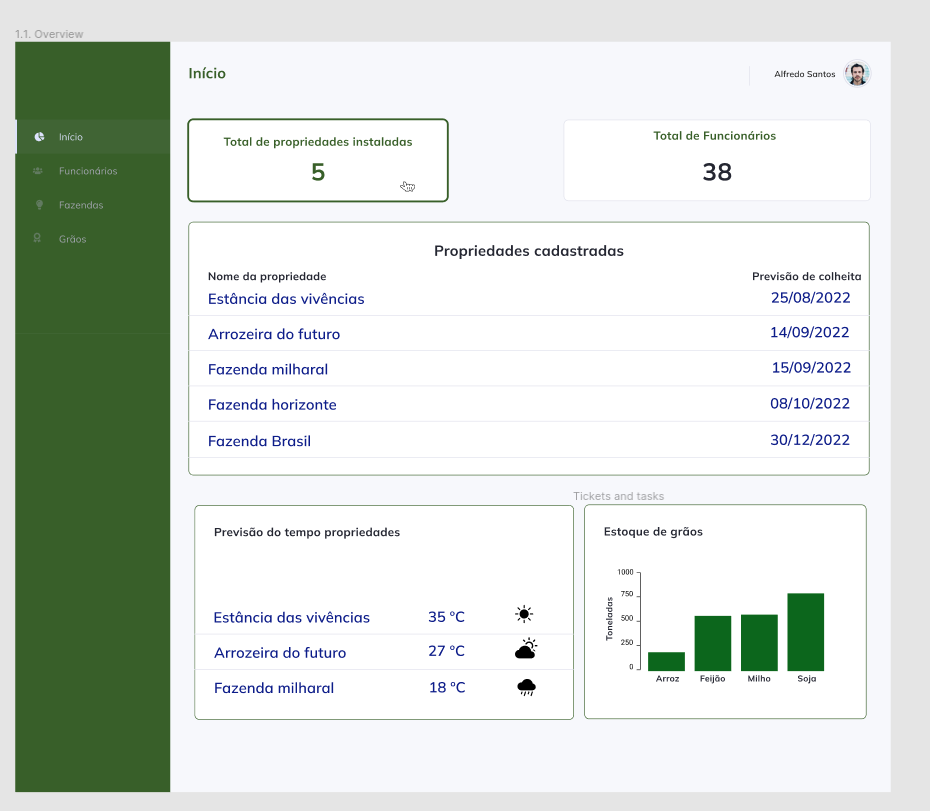
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nº** | **Critério de Avaliação** | **0** | **0,5** | **1** |
| 1 | O aluno implementou as validações necessárias? (peso 1) | O aluno não implementou nenhuma validação. | O aluno implementou algumas validações. | O aluno implementou todas as validações. |
| **Nº** | **Critério de Avaliação** | **0** | **1** | **2** |
| 2 | O aluno implementou todos os models, repositories, controllers, services? | O aluno não implementou as classes. | O aluno implementou de forma parcial as classes, atributos e métodos. | O aluno implementou todas as classes, juntamente com seus atributos e métodos. |
| 3 | O aluno implementou todas as operações de cadastro/registro de dados? | O aluno não implementou as operações. | O aluno implementou de forma parcial as operações. | O aluno implementou todas as operações. |
| 4 | O aluno implementou as associações entre as entidades? | O aluno não implementou as associações. | O aluno implementou as associações de forma parcial. | O aluno implementou todas as associações. |
| **Nº** | **Critério de Avaliação** | **0** | **1,5** | **3** |
| 5 | O aluno implementou todas as operações de retorno de dados? | O aluno não implementou as operações de retorno de dados. | O aluno implementou, de forma parcial, as operações de retorno de dados. | O aluno implementou todas as operações de retorno de dados. |

# 5. Roteiro da aplicação

Você está prestes a entrar para o time de Desenvolvedores da DevInHouse Agro. Para concretizar a sua contratação, você deverá resolver um desafio utilizando os conceitos ensinados em aula. O time de recrutamento necessita que você crie uma aplicação protótipo para o gerenciamento de propriedades rurais, chamada DEV-Agro.

Neste primeiro projeto você deve desenvolver a API de um sistema de gerenciamento de propriedades rurais chamado DEV-Agro.

Protótipo de tela:



A API deve possibilitar o cadastro de Empresas, Fazendas, Funcionários, Grãos.

**FEITO**

O **cadastro de empresas** deve possibilitar o registro de nome, CNPJ e endereço.

Cada empresa pode ter uma ou mais propriedades rurais associadas (fazendas).

Cada empresa pode ter um ou mais funcionários associados.

Cada empresa pode ter um ou mais grãos associados.

O CNPJ deve ser validado para estar no formato: XX.XXX.XXX/XXXX-XX.

**FEITO**

O **cadastro de fazendas** deve possibilitar o registro de nome, endereço, qual grão é produzido naquela propriedade, qual o estoque inicial daquele grão naquela fazenda (em kg), e qual empresa é proprietária da fazenda.

Cada fazenda pode ter apenas um grão associado.

Cada fazenda pode estar associada a apenas uma empresa.

Cada fazenda deve possuir um atributo para guardar a data da última colheita.

**FEITO**

O **cadastro de grãos** deve possibilitar o registro de nome, empresa e tempo médio de colheita, em quantidade de dias.

**FEITO**

O **cadastro de funcionários** deve possibilitar o registro de nome, sobrenome, CPF, endereço, telefone, sexo, data de nascimento, data de contratação, e qual empresa o emprega.

O CPF deve ser validado para estar no formato: XXX.XXX.XXX-XX.

O telefone deve ser validado para estar no formato: (XX) XXXXXXXXX.

Nesse primeiro momento, **não é necessário fazer o front-end da aplicação**, apenas o back-end, ou seja, a API (em Java com Spring Boot) e o banco de dados (PostgreSQL).

Você poderá testar sua aplicação utilizando ferramentas como Postman, Insomnia ou Curl.

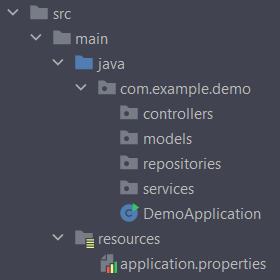
**FEITO**

Será necessário implementar os **endpoints** (controller, service, repository) para cadastro de todas essas entidades (models).

Na aplicação devem existir:

* **Um endpoint** para retornar a lista completa de empresas cadastradas. **FEITO**
* **Um endpoint** que retorna a lista de fazendas de uma empresa. **FEITO**
* **Um endpoint** que retorna a quantidade de fazendas de uma empresa. **FEITO**
* **Um endpoint** que retorna uma lista de fazendas de uma empresa, onde cada elemento da lista deve ter 3 atributos: ID da fazenda, nome da fazenda e a data prevista da próxima colheita (considerando a data da última colheita e o tempo médio de colheita do grão associado a essa fazenda). **FEITO**
* **Um endpoint** para registrar colheita em uma fazenda, que aumenta o estoque de grãos daquela fazenda. **FEITO**
* **Um endpoint** para registrar retirada de grãos de uma fazenda, que diminui o estoque de grãos daquela fazenda. **FEITO**
* **Um endpoint** que retorna a lista de grãos de uma empresa. **FEITO**
* **Um endpoint** que retorna a lista de grãos associados a uma empresa, onde cada elemento da lista deve conter: nome do grão e quantidade em estoque, ordenado de menor para maior quantidade em estoque. **FEITO**
* **Um endpoint** que retorna a lista de funcionários de uma empresa.
* **Um endpoint** que retorna a quantidade de funcionários de uma empresa.

Sugestão de estrutura de diretórios do projeto:



**Observação:** estrutura NÃO obrigatória. Também pode ter outros pacotes além desses.