

Relatório do Projeto eDOE

Laboratório de Programação 2

Anderson Filipe Clemente Silva
Bruno Roberto Silva de Siqueira
João Pedro Santino Espíndula
Vítor Braga Diniz

13 de dezembro de 2018

Sumário

1	Design Geral	3
2	Caso de Uso 1	3
3	Caso de Uso 2	4
4	Caso de Uso 3	4
5	Caso de Uso 4	5
6	Caso de Uso 5	5
7	Caso de Uso 6	5
8	Caso de Uso 7	5
9	Link para repositório no GitHub	6

1 Design Geral

O design do projeto foi arquitetado para minimizar o acoplamento do sistema de forma que foram utilizados vários níveis de abstração. No nível mais alto, há o Mediator, que tem a função de criar, controlar e integrar os controladores de itens necessitados e doados e o controlador de usuário. No nível intermediário estão os controladores: *DoadosController* e *NecessitadosController*, ambos são classes filhas de *ItemController* – pois são controladores de itens – *UsuarioController*, que é o controlador de usuários, e *doacaoController*, que é o controlador das doações. Já no nível mais básico, temos as classes *Item* e *Usuário*, que são abstrações de um item e de uma entidade que doará ou receberá um item no sistema. Além disso, foi implementada uma classe extra chamada *Validador*, cuja responsabilidade é verificar se o sistema apresenta erros e lançar exceções.

2 Caso de Uso 1

No primeiro caso, deve ser criado um CRUD (*Create*, *Read*, *Update* e *Delete*) de usuários, os quais podem ser doadores ou receptores de itens. Para isso, a equipe de desenvolvimento implementou duas classes: “*UsuarioController*” – que tem como funções principais criar e administrar conjuntos de usuários – e “*Usuario*” – que é a abstração de um doador ou receptor no sistema.

Em “*UsuarioController*”, foi criado um mapa (*LinkedHashMap*, uma vez que a ordem de cadastro importa) de usuários, cujo identificador único é o CPF ou CNPJ e, também, um set de classes, tal que as classes são a categoria a qual um usuário pode pertencer, como igreja, ONG, sociedade, pessoa física. Além disso, foram implementados os métodos *adicionaDoador()* para cadastrar usuário doador, *atualizaUsuario()* e *removeUsuario()* para atualizar os atributos de um usuário qualquer e remover um usuário, respectivamente.

Já “*Usuario*” possui id, nome, e-mail, telefone, categoria e uma variável indicadora se o usuário é doador ou receptor. Há, também, um conjunto de métodos *get* – para informar às classes externas os valores dos atributos – e *set* – para alterar os atributos.

3 Caso de Uso 2

O caso 2 pede para que os doadores possam inserir os itens a serem doados no sistema. Dessa forma, foram criadas a classe *Item* (que é a abstração de um item no sistema), a classe abstrata *ItemController*, a qual tem como objetivo gerenciar todos os itens do sistema e a classe *DoadosController*, que gerencia os itens doados.

Na classe *ItemController*, foram implementados 2 atributos, um contador, que contém o identificador (ID) único do item, e um mapa de mapa – o qual relaciona um Usuário a um outro mapa que, por sua vez, relaciona o id do item ao próprio item –, uma vez que é necessário que o item esteja externamente relacionado ao seu id único e ao seu respectivo usuário. Foram implementados métodos para cadastro, atualização e remoção de itens. Aquele gera o id único incrementando 1 para o próximo item a ser cadastrado e grava os valores necessários no mapa. Além disso, foi incorporado o método *listaTodos()* que lista todos os itens cadastrados em uma ordem específica solicitada pelo usuário.

Na classe *DoadosController*, que estende de *ItemController* – afinal, é, também, um controlador de itens, mas específico àqueles doados –, foram implementados um *set* de descrições, indentificando quais as descrições de itens já existentes no sistema, e métodos para cadastrar e exibir item.

Já na classe *Item*, foram implementados os seguintes atributos necessários para descrever um item com todas as suas propriedades: ID, descrição, quantidade, tags e usuário. Além disso, há um conjunto de métodos *get* e *set* para transmissão de dados para outras classes.

4 Caso de Uso 3

O caso 3 demanda uma ferramenta de pesquisa de itens a serem doados, além da listagem de todos os descritores e dos itens cadastrados em ordem alfabética e em ordem especificada pelo usuário, respectivamente. Com isso, foram implementados os métodos *listaDescritorDeItensParaDoacao()*, *listaItensParaDoacao()* e *pesquisaItemParaDoacaoPorDescricao()* na classe *DoadosController*, a fim de listar os descritores e itens cadastrados no sistema nas ordens mencionadas e para pesquisar itens para a doação utilizando a descrição como parâmetro de busca.

5 Caso de Uso 4

No caso 4, os usuários do sistema que são receptores devem poder indicar os itens que estão precisando receber (itens necessários). Para gerenciar esses itens, a equipe criou uma segunda classe filha de `ItemController`, chamada `NecessitadosController`, cuja função é gerenciar itens necessários por usuários receptores. Nessa classe, foi implementado o método `listaTodos()`, que tem como função listar todos os itens necessários cadastrados no sistema.

6 Caso de Uso 5

O caso 5 traz a funcionalidade mais importante deste sistema: encontrar *matches* entre itens a serem doados e itens necessários. Para isso, foi criada uma nova classe: a `ItemMatcher`, a qual cria um item base – que será a base de comparação – e possui o método `match` que realiza a comparação entre todos os itens e retorna uma *string* com a pontuação dos matches. Tal pontuação é calculada nos métodos `match` e `matchTags` implementados na classe `Item`.

7 Caso de Uso 6

Conhecendo os possíveis *matches* entre itens necessários e para doação, um usuário receptor pode solicitar uma doação de um item. Para isso, foi implementado um controlador de doações, o `DoacaoController`, que tem como responsabilidade realizar todas as doações que ocorrem no sistema e registrá-las em uma lista.

8 Caso de Uso 7

Conforme pede o caso 7, o `edoe.com` deve armazenar em disco todos os dados necessários para manter o seu estado mesmo que o programa seja fechado. Dessa forma, foi implementada uma classe, chamada "Persistência", que tem como responsabilidade salvar todas as doações em um arquivo, a fim de manter o estado da informação mesmo com o programa desligado, e iniciar o programa carregando todas as doações já feitas previamente.

9 Link para repositório no GitHub

<https://github.com/pedroespindula/eDoe-LP2.git>