Universidade Federal de Campina Grande
Centro de Ciências e Tecnologia
Departamento de Sistemas e Computação
Disciplina: Laboratório de Programação II
Professora: Raquel Lopes

Relatório de Projeto

Central de Projetos da Computação@UFCG (CPC@UFCG)

Equipe:

Matheus Vasconcelos Figueiredo
Matheus de Souza Coutinho

Campina Grande
2017

Introdução

A Central de Projetos da Computação@UFCG (CPC@UFCG) é um sistema que possui parceria com a unidade acadêmica de sistemas e computação (UASC), ele será utilizado por um usuário administrador e tem como objetivo gerenciar o cadastro dos projetos da unidade e manter a informação atualizada.

Através deste relatório pretendemos informar detalhadamente como a nossa CPC@UFCG foi implementada. O relatório será dividido pelos casos de uso, abordando além do design utilizado no projeto, os principais tópicos estudados nas disciplinas de Programação II e Laboratório de Programação II.

Sobre o design

Para este projeto, foram seguidos dois padrões de design de orientação a objetos: *ModelController* e *Façade* Design Patterns. Os primeiros são oriundos de um design ainda mais complexo, o *ModelViewController* (MVC), cujas ideias centrais são a reusabilidade de código e a separação de conceitos e/ou tarefas.

A utilização desse modelo no nosso projeto permitiu solucionar o problema através da separação das tarefas de acesso aos dados da lógica de negócio, introduzindo um componente entre os dois: o controlador.

Com o padrão *Façade* pudemos simplificar a utilização de um subsistema complexo apenas implementando uma classe que fornece uma interface única e mais razoável, porém se desejarmos acessar as funcionalidades de baixo nível do sistema isso é perfeitamente possível.

Vale ressaltar que o padrão *Façade* não “encapsula” as interfaces do sistema, ele apenas fornece uma interface simplificada para acessar as suas funcionalidades.

Com relação as Exceptions, criamos uma package chamada *exceptions e uma chamada util*, nelas estão todas as classes criadas para o encapsulamento de determinada situação de exceção. Por exemplo, ExcecoesPessoas encapsula todas as exceções relacionadas a Pessoa.

Usamos o conceito de repository, para separar a lógica da manipulação de listas, deixando o sistema mais legível e elegante.

US1

A primeira US tem como objetivo fazer o cadastro de pessoas que participaram ou irão participar de um projeto na unidade. Para essa situação, criamos a classe Pessoa que contém os respectivos atributos indicados e faz a validação dos mesmos, lançando exceção se houver alguma adição fora do padrão. Também adicionamos uma coleção para armazenar as participações de uma pessoa em um projeto e para isso foram gerados métodos de adicionar e remover nessa coleção.

A criação de Pessoas está contida na classe PessoaController, que é quem gerencia as operações envolvendo Pessoa. Foi criada também uma classe para tratar exceções específicas de Pessoa, que é ExcecoesPessoa. Além disso, todas as pessoas do projeto ficam armanezadas no RepositoryPessoa, que também fornece informações sobre as mesmas.

US2

A US2 remete as operações envolvendo projetos. Como existem quatro tipos de projetos, foi criada uma classe pai abstrata chamda Projeto e que possui as filhas PED, PET, Extensao e Monitoria. Cada uma herdando de Projeto, porém com seus respectivos atributos particulares. Os projetos também contém uma coleção de pessoas que são seus participantes.

Para a criação de um determinado projeto, foi feita a classe FactoryProjeto, que possui métodos para a criar todos os tipos de projeto existentes. O tratamento de excessões é feito nos pacotes util e exception. Para a classe PED, foi criada uma lista de categorias que auxilia na sua validação feita pelo pacote util, para que não haja entrada de categorias erradas nesse tipo de Projeto.

O armazenamento dos projetos fica por conta da classe RepositoryProjeto que também tem métodos para fornecer algumas informações. As demais operações são realizadas na classe ProjetoController, que entre outras coisas, adiciona projetos criados no repository.

US3

A US3 implica na implementação das Participações de uma pessoa em um projeto. Assim criamos a classe Participacao que é abstrata e possui quatro classes que herdam dela, que são os tipos de participação.

As participações são armazenadas na classe RepositoryAssociacao que também pode fornercer informações sobre as mesmas. Foi feita uma factory para a criação de novas associações entre pessoas e projetos chamada FactoryAssociacao. A ParticipacaoController gerencia as operações para atribuir um projeto a uma pessoa e uma pessoa a um projeto.

US4

O objetivo desta US é calcular os pontos de participação que uma pessoa já acumulou durante sua permanência em projetos. No nosso projeto, isso é feito dentro da classe ParticipacaoController, no método getPontuacaoPessoa, que procura uma determinada pessoa no repository e através de suas participações retorna sua pontuação.

US5

A responsabilidade dessa US é *calcular o valor da bolsa dos participantes dos projetos*. Então criamos na classe ParticipacaoController os métodos getValorBolsa e retornaValorBolsa. GetValorBolsa retornará o valor total de uma determinada pessoa pesquisada, enquando retornaValorBolsa fará justamente o cálculo do valor da bolsa dessa pessoa.

US6

Nessa US foram implementados recursos para gerenciar a parte financeira dos projetos. Para isso, foram criados métodos na classe ProjetoController que faz as operações relacionadas a demanda dessa US com o auxilio da classe Despesas, que foi uma classe criada para armazenar os valores financeiros recebidos por um projeto.

US7

Implementação da US7 tem como objetivo criar relatórios com os dados do sistema para ter uma persistência em texto dos dados do sistema. Para isso codificamos a classe GerarRelatorio, que lidará com todas as demandas referentes a geração de relatórios do sistema.

US8

O objetivo dessa US é *armazenar o estado atual do sistema para poder recuperar este estado caso o sistema seja desligado. Sendo assim, foram implementados novos métodos nos controllers para iniciação e fechamento do sistema, assim como a classe FileManager para genrenciar a manipulação de arquivos*