**Relatório de Projeto LP2 – 2019.2**

**Psquiza**

**Design geral:**

O design geral do nosso projeto foi escolhido para possibilitar uma melhor integração entre as diferentes partes do sistema, criando camadas de abstração que diminuem o acoplamento, através do uso de um *Controller* geral , chamado de Psquiza, que delega atividades para cada controladora mais abaixo e de uma *Facade*, responsável por invocar as funcionalidades principais do programa.. As camadas menores também possuem um gerenciador próprio, para aumentar o nível de abstração do sistema.

Para tratamento de erros (*exceptions*) e validação das entradas, criou-se uma classe de validação, para evitar repetição e extensão de código. Logo, temos *exceptions* para atributos inválidos, tanto de criação como de atualização de dados.

As controladoras de mais baixo nível e as entidades estão encapsuladas em *packages* separadas, para facilitar a compreensão e divisão dos módulos de desenvolvimento. As próximas seções detalham a implementação em cada caso:

**- Caso 1 (Holliver Costa):**

O primeiro caso solicita a criação de uma entidade que represente a pesquisa. Com isso, para representar essa entidade, criou-se uma classe chamada pesquisa, que contém código, descrição e campo de interesse. E para gerenciar a pesquisa, foi criado um *controller* pesquisa, no qual, contém os métodos de pesquisa. Além disso, a mesma é guardada no mapa de pesquisas, onde a chave é o código gerado no cadastro e armazenado no objeto.

**- Caso 2 (Pedro Henrick):**

O segundo caso, ele solicita a criação de uma nova entidade chamada Pesquisador, com isso se foi criado uma classe com o nome Pesquisador, responsável por guardar os atributos de nome, função, e-mail, biografia e a URL de sua foto, também foi criada uma classe ControllerPesquisador, classe que era responsável por controlar os pesquisadores e seus métodos , cadastrados no sistema, bem como os ainda não cadastrados. No ControllerPesquisa foi criado um mapa de pesquisadores no qual é responsável por guardar os pesquisadores já cadastrados no sistema, a chave de cada pesquisador é o seu próprio e-mail. Com isso, finalizando o teste case 2.

**- Caso 3 (Danilo Medeiros):**

Para o caso 3 foram criadas duas novas entidades: Problema e Objetivo. A classe Problema representa um desafio geral a ser resolvido, contendo como atributos sua descrição, sua viabilidade e um código identificador. Já a entidade Objetivo representa uma finalidade a ser alcançada. Seus atributos são sua descrição, seu tipo (GERAL ou ESPECIFICO), sua viabilidade, sua aderência e seu código identificador. A fim de gerenciar as classes criadas, foi desenvolvido um controlador único, que comanda o cadastro, a remoção e a exibição tanto de um problema como de um objetivo. O *controller* possui dois mapas, um para objetivos e outro para problemas, onde as chaves são códigos gerados automaticamente pelo *controller*.

**- Caso 4 (Caio Medeiros):**

O caso 4 pede que seja criado uma entidade que represente a Atividade Metodológica. Cada Atividade planejada deverá ter uma descrição, uma duração planejada, um conjunto de resultados esperados e um risco associado, podendo este ser categorizado em 3 tipos: “BAIXO”, “MEDIO” e “ALTO”. Além do risco, é necessário também uma descrição do nível de risco da atividade. Para

Para organizar as Atividades, utilizou-se uma estrutura de dados de mapa: HashMap, chamado de “atividades”. Cada atividade criada tem um id composto pela letra “A” e um número, iniciado em 1 e este id será a chave da atividade no mapa criado. A ordem de criação é permanente, não sendo alterada caso uma Atividade seja apagada.

Para organizar os resultados esperados(itens) criou-se uma lista de itens dentro de cada atividade. Estes itens não podem ser repetidos e sua ordem é determinada pela ordem de cadastro.

Além da classe para a entidade Atividade, também foi criada a classe do Item, que possui apenas uma descrição e um status, podendo este ser “REALIZADO” ou “PENDENTE”.

**- Caso 5 (Danilo Medeiros):**

O caso 5 solicita que seja possível associar problemas e objetivos à pesquisas existentes no sistema. Para isso, foram adicionados na classe Pesquisa dois novos atributos: String contendo o id do problema associado e uma lista de Strings contendo os objetivos associados. Foi necessário a implementação de um *controller* geral para viabilizar a comunicação entre o *controller* de problemas e objetivos e o *controller* de pesquisa. Assim, foram adicionadas novas funcionalidades no *controller* de pesquisas: ‘associarObjetivo’, ‘associarProblema’, ‘desassociarObjetivo’, ‘desassociarProblema’. Foi necessário também a adição de um método de listagem das pesquisas baseadas em um critério. O *controller* de problemas e objetivos sofreu uma alteração que foi o acréscimo de uma funcionalidade para checar a existência de um código (problema ou objetivo).

**- Caso 6 (Pedro Henrick):**

O sexto caso, foi criado um mapa com nome de pesquisadores na classe 'Pesquisa', no qual nele são guardados os pesquisadores que foram associados pelo método 'associaPesquisador', método pelo qual tornava um objeto de Pesquisador associado a determinada pesquisa/s. Além disso, na classe Pesquisador, foi criado uma interface de função, no qual essa interface era implementada por uma entidade Professor, Aluno ou SemEspecialidade (classe em que o objeto pesquisador ainda não possuísse atributos de especialização, podendo ser esses atributos para pesquisadores que antes já eram alunos ou professores). Após criar-se essa interface foi possível implementar os métodos 'cadastraEspecialidadeProfessor' e 'cadastraEspecialidadeAluno' de forma correta, e com isso tendo que alterar o método anteriormente feito, 'alteraPesquisador', que é responsável por alterar os atributos de um Pesquisador, adicionando agora as possíveis alterações dos atributos semestre, IEA, formação, unidade e data.

**- Caso 7 (Holliver Costa):**

O sétimo caso solicita que se adicione a funcionalidade de associar e executar atividades de uma pesquisa, e de cadastrar resultados de uma atividade. Com isso, foi-se guardado as associações na classe pesquisa, onde tem um mapa de atividades, e a chave seria o id de atividade, que é único para cada atividade. Assim, para que a associação preservasse o expert da informação, foi criado um *controller* geral, chamado Psquiza, para que as controladoras de atividade e pesquisa se relacionasse, porém, não se conhecessem. Sendo assim, ao executar uma atividade, é exigido o código de pesquisa e o código da atividade, assim, o método foi implementado na classe pesquisa, no qual se tem um mapa das atividades associadas a mesma, e com isso, posso mudar o status da atividade para realizado (por padrão, a atividade se encontra pendente). Por conseguinte, os resultados são guardados em uma lista na classe atividade, enquanto o seu código seria de acordo com a ordem de inserção, e para remover e não prejudicar o código de cada resultado, é necessário que na hora de remover, ao invés de remover da lista, foi adicionado, novamente, uma *string*, mas com o nome removido.

**- Caso 8 (Caio Medeiros):**

O caso 8 solicita a implementação de um sistema de busca por palavra chave. Esta busca deve ser realizada nos atributos das seguintes entidades:

- Pesquisa: Descrição e Campo de interesse

- Pesquisador: Biografia

- Problemas: Descrição

- Objetivo: Descrição

- Atividade: Descrição e Descrição do Risco

A busca se deu usando os ids de cada entidade, em seus respectivos mapas. Criou-se a entidade “Pair”, que consiste em uma tupla de *Strings* e possui os atributos “codigo” e “campo”. Esta tupla recebe o id da entidade (id de Pesquisa, Pesquisador, Objetivo, Problema ou Atividade) e a *String* do campo onde o termo foi encontrado. A partir desta tupla, cria-se uma lista de tuplas para conjunto de pares de cada entidade encontrada. A partir daí, a ordenação é feita em cada lista, comparando-se o id em cada tupla.

**- Caso 9 (Caio Medeiros):**

O caso 9 requer a criação de uma ordem de execução que represente uma lógica na execução das Atividades. Para tanto, criou-se os atributos idPrecedentes(como uma lista de String de ids) e idSubsequente, como uma String, na entidade Ativdade. Por determinação da especificação, uma atividade poderia possuir múltiplas atividades precedentes, mas apenas uma subsequente, por isto uma lista no idPrecedentes.

As ações de varredura na sequência foram feitas criando-se uma lista temporária e preenchendo-a com os ids das atividades encadeadas, seguindo a ordem de seus idSubsequente.

**- Caso 10 (Holliver Costa):**

O décimo caso dá sugestão que a próxima atividade seja executada de uma pesquisa e, para isso, foi passado alguns critérios. Esses critérios são guardados no *controller* de pesquisa e, por padrão, sugere-se a atividade mais antiga, porém, pode-se alterar a estratégia, que muda a forma de sugerir a atividade. Esses critérios podem ser:

a) atividade mais antiga;

b) atividade com menos pendência;

c) atividade com maior risco;

d) atividade com maior duração.

Desse modo, criou-se os métodos de sugestão dentro da pesquisa, uma vez que a pesquisa guarda as atividades, assim, foram feitas 3 classes novas que implementam o *comparable* com a classe atividade para usar nos métodos de ordenação implementado na classe pesquisa. Com isso, cada método ordena de uma forma, enquanto o método, próxima atividade, chama algum desses métodos de acordo com a estratégia armazenada.

**- Caso 11 (Pedro Henrick):**

No Decimo primeiro caso, era necessário o conhecimento da API JAVA.IO, para a criação e exportação de arquivos .txt no sistema. Nesses arquivos é possível visualizar o resumo completo de um de uma instancia de 'Pesquisa', como também o resumo completo dos resultados e itens de uma pesquisa.

**- Caso 12 (Danilo Medeiros):**

Neste caso, era o sistema deve ser capaz de salvar e carregar seu estado a partir de certo ponto. Para isso, foi usada a interface *Serializable*, implementada nos *controllers* específicos (Pesquisa, Pesquisador, Atividades, Problemas e Objetivos) e nas classes que fazem parte do acoplamento. É então gerado um arquivo .ser para cada *controller* com o método salvar. O estado é recuperado pela leitura destes arquivos, e os objetos recuperados são atribuídos a cada *controller* presente no *controller* geral.

O Diagrama das classes está representado no anexo I.

**Uma imagem contendo texto, mapa

Descrição gerada automaticamenteANEXO I – DIAGRAMA DE CLASSES - PSQUIZA**