INSTITUTO FEDERAL DO NORTE DE MINAS GERAIS
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

FLAVIO RODRIGO SILVEIRA SANTOS,
GUSTAVO HENRIQUE ALVES ROCHA,
NATÃ TEIXEIRA SANTOS DE OLIVEIRA

Relatório de Desenvolvimento – BookARoom

Análise e Projeto de Sistemas

MONTES CLAROS – MG
2024

SUMÁRIO

1.INTRODUÇÃO

O presente relatório tem por objetivo detalhar o desenvolvimento da atividade avaliativa **Projeto com Responsabilidades**, cujo objetivo é a implementação do sistema ***BookARoom* - Agente de Reserva de Salas de Reunião** utilizando os princípios GRASP, bem como a produção dos artefatos necessários para tal.

2.DECISÕES DE PROJETO

2.1.Escolha da MVC (*Model-View-Controller*)

O fato de que o código não poderia ser descartável tornou necessária uma estruturação adequada do projeto. Nesse sentido, visando desenvolver um sistema extensível, ao passo que utilize os princípios GRASP, adotamos a arquitetura MVC (*Model-View-Controller*), a qual promove a separação de responsabilidades, facilitando o desenvolvimento e manutenção do sistema.

Outra grande motivação para a adoção da MVC, visando praticá-la, pois nenhum de nós havia utilizado a MVC anteriormente, foi o fato de que há uma grande probabilidade de o projeto da disciplina ser um projeto WEB, uma vez que a MVC é amplamente utilizada em projetos de tal natureza.

No entanto, o fato da MVC não ser a arquitetura canônica de um projeto *desktop*, o que também não é um fator proibitivo, tornou a busca por documentações e artigos que fornecessem um direcionamento de como utilizá-la para tal aquém do esperado. O que fez com que realizássemos o projeto apenas à luz da ideia conceitual da MVC. Logo, nós mesmos tivemos de realizar a concepção das adaptações necessárias para o empregar da arquitetura na plataforma *desktop*.

3. DESENVOLVIMENTO E APLICAÇÕES DO GRASP

A primeira ponderação foi a respeito de como iríamos persistir os dados, uma vez que não foi requisitada a utilização de um SGBD. Pensando na possibilidade dessa utilização ser futuramente requisitada, para facilitar a refatoração, cada *Model* possui um *ArrayList* da entidade pela qual ele é responsável, assim, sempre que uma entidade é criada, ela é persistida em tal estrutura. Dessa forma, o gerenciamento dos dados é realizado unica e exclusivamente pelos *Models*.

As *Views* são responsáveis apenas por apresentar a interface do usuário, receber sua interação e a apresentação dos dados.

Os *Controllers* são responsáveis por realizar o intermédio entre os *Models* e as *Views*. Eles recebem solicitações das *Views*, utilizam os *Models* para o processamento dos dados e retornam o resultado para as *Views*. Eles controlam o fluxo do sistema.

À luz das implementações supracitadas, é notório que aplicamos o princípio de **alta coesão**, uma vez que cada componente tem uma responsabilidade bem definida e específica.

Ademais, aplicamos o princípio de **baixo acoplamento**. Por exemplo, a *View* não sabe como os dados são manipulados no *Model* e o *Controller* não sabe como os dados são apresentados na *View*. Sempre que uma *View* precisa exibir informações de alguma entidade, em vez de obter os dados diretamente – o que não é de sua responsabilidade – ela necessita de solicitar ao *Controller*, que por sua vez solicita ao *Model*, tais dados.

Outro princípio que aplicamos foi o **polimorfismo**. Por exemplo, cada *View* tem seu modo particular de apresentar a visualização dos dados. A classe genérica e abstrata *View*, por meio de seus métodos abstratos, como o cadastrar e o listar, definem um contrato genérico. As classes que a possuem como superclasse – por exemplo *PredioView* – implementam tais métodos especifica e adequadamente para a sua respectiva entidade. *PredioView* implementa tais métodos adequadamente para a entidade Predio, *SalaReuniaoView* para a entidade SalaReuniao e assim sucessivamente. Desse modo, o mesmo método assume diferentes comportamentos para cada *View*. Assim, quando um *Controller* específico de alguma entidade – por exemplo *CampusController* – chama seu método de listagem, o qual foi herdado da classe genérica e abstrata *Controller*, tal método, após a obter os dados ao chamar o *Model*, chama, fornecendo para ele os dados, o método de listagem de sua *View* específica que foi herdado da classe abstrata *View*, o qual realizará a listagem conforme a implementação realizada naquela *View* específica, entrando assim o polimorfismo em ação, permitindo que a listagem ocorra adequadamente conforme a entidade a qual a *View* específica é referente.

4.CONCLUSÃO

Portanto, o projeto demonstra a importância da utilização dos princípios GRASP para a criação de um sistema robusto e extensível. Foi obtido um sistema com alta coesão e baixo acoplamento devido a separação bem definida das atribuições entre os *Models, Views e Controllers*, além de flexível e reutilizável em virtude do polimorfismo, permitindo que

diferentes implementações pudessem ser facilmente incorporadas sem afetar a estrutura geral do sistema.