Universidade Federal de Juiz de Fora

DCC025-2020.3-A - ORIENTAÇÃO A OBJETOS - DEPTO DE CIENCIA DA COMPUTACAO /ICE

Professor:

Trabalho final

Arthur Soares Mesquita - 201469023C
Lucas Esteves dos Reis Salgado - 202065502AB
Lucas Santiago Nepomuceno - 201969031C
Tawan Victor Batista de Oliveira - 201469013B

Campus Juiz de Fora
2021

SUMÁRIO

1. Motivação

Sabemos como o restaurante universitário é uma ferramenta de extrema importância para manutenção da universidade e seu processo de inclusão social. Assim sendo, enxergamos as universidades federais como ferramentas para inclusão tanto social como tecnológica da população de nosso país.

Partindo de um ponto de vista social, é notória a segregação ao acesso à tecnologia e a informação em nosso país. Porém, devido a pandemia e a adoção do Ensino Remeto Emergencial pela UFJF, ficou evidenciado as limitações e obstáculos que as instituições públicas e sociais desse país têm de superar para atender a população com a totalidade de suas funções.

Contudo, é uma responsabilidade muito grande o processo de retomada das atividades presenciais nas universidades, pois são ambientes vulneráveis a ataques biológicos, em virtude da alta circulação de pessoas de diversas localidades, assim colocando toda população em vulnerabilidade.

Visto isso, e considerando o cenário real das universidade que apresenta-se burocrático e em alguns casos, inacessível. Considerando os diversos fatores sociais, econômicos e ambientais que impactam a vida de discentes, de docentes e de funcionários desses espaços, pensamos em desenvolver um protótipo para um sistema que possibilite a utilização do restaurante universitário de uma forma segura e inclusiva.

2. Definição do domínio da aplicação

O objetivo do trabalho é criar um sistema que torne possível a utilização do Restaurante Universitário da Universidade Federal de Juiz de Fora (RU UFJF) por meio do agendamento e encomenda de refeições e pelo monitoramento de sua capacidade orgânica, ou seja, sua capacidade segura para realidade atual.

- Agendamento de horário

Esse sistema será capaz de realizar o agendamento prévio de refeições, onde cada usuário cadastrado poderá acesso o aplicativo e após registrar o registro de seus dados poderá agendar um horário e dia para sua refeição no restaurante, que possue sua lotação monitorada constantemente.

Esse sistema será capaz de realizar o agendamento prévio de horário por meio de e-tickets[1] onde cada usuário cadastrado poderá acessar o aplicativo e então cadastrar seus horários de aula ou serviços dentro da universidade, e assim agendar um horário para sua refeição utilizando do sistema de prioridade, para que o sistema retorne da melhor forma o uso do RU ao seu horário de utilização da universidade.

(1) Lucas cadastrou seus horários no sistema; de segunda a quinta ele registrou aulas de 10:00 às 12:00 e de 14:00 às 16:00, logo, Lucas tem prioridade a utilização do RU, podendo marcar com até uma semana de antecedência, de segunda até quinta, para refeição de ALMOÇO, durante o período de 12:00 às 14:00.

(¹) e-tickets: Para evitar a circulação de dinheiro físico, tendo em vista que é um objeto de alto risco biológico, e também a utilização descontrolada do espaço físico.

- Monitoramento do RU

Todos os usuários poderão também monitorar a disponibilidade do restaurante, tendo acesso ao total de pessoas que agendaram sua refeição para determinado horário, bem como ao percentual de locação do local. Caso a lotação ultrapasse os limites sugeridos pelos órgãos reguladores, os agendamentos ficam indisponíveis no horário em questão.

- Cardápio

No sistema a ser implementado, haverá uma aba denominada cardápio onde os usuários poderão ter acesso ao cardápio semanal de refeições disponibilizado pelo RU. Selecionado o dia da semana e o turno (café da manhã, almoço e janta), o usuário terá acesso aos detalhes da refeição ofertada.

- Encomendar Refeição

O usuário também terá a possibilidade de encomendar um marmitex dentro do sistema. Na aba referente ao agendamento de refeições, ele poderá optar pela encomenda ao invés do agendamento e, selecionando o tipo da refeição e seu turno, o usuário poderá realizar essa encomenda e retirar a marmita próximo ao RU, apresentando seu e-ticket.

3. Organização

Este trabalho utiliza o padrão arquitetural Model-View-Controller (MVC), logo essa aplicação se divide em três camadas: a visão (view), o modelo (model), e o controlador (controller), desse modo, utilizando dos preceitos e boas práticas da POO.

A camada model é responsável pela modelagem do sistema, é a camada mais básica, responsável pelo comportamento das demais - ou seja - ela está ligada a manipulação de dados. Sendo assim, as classes AgendarRefeicao, Cardapio, Contato, Discente, Docente, Tae, Horarios, MonitorarLotacao, Propriedade, SemanaHorarios e Usuario e as classes interfaces Eticket, Refeições e Usuarios estão presentes nesta camada.

View é a parte de apresentação de dados ao usuário, toda a interface, informação, não importando sua fonte de origem. As interfaces gráficas MostrarCardapio, TelaAgendarRefeicao, TelaCadastraHorarios, TelaCadastro, TelaCardapio, TelaContato, TelaLogin, TelaMonitoramento, TelaPerfil e TelaPrincipal estão presentes nessa camada.

Controller como o nome já sugere, é responsável por controlar todo o fluxo de informação que passa pelo site/sistema. É na controladora que se decide “se”, “o que”, “quando” e “onde” deve funcionar. Define quais informações devem ser geradas, quais regras devem ser acionadas e para onde as informações devem ir, é na controladora que essas operações devem ser executadas. Estão presentes nessa camada as classes HoriosController, MonitoramentoController, RefeicaoController e UsuariosController.

Além das três camadas referentes ao padrão arquitetural MVC, é utilizado mais duas camadas: dao, onde se encontra a classe BancoDeDadosUsuario para verificação e autenticação e utils contendo a classe Arquivo para leitura de arquivos.

4. Aplicação dos conceitos

Neste capítulo é apresentado onde e como os conceitos da orientação a objeto foram explorados.

- **Herança**

  É comum haver similaridades entre diferentes classes. Sendo assim, frequentemente, duas ou mais classes irão compartilhar os mesmos atributos e/ou métodos. A Herança é um princípio de orientação a objetos, que permite que classes compartilhem atributos e métodos, através de "heranças".

  Uma subclasse herda as propriedades de sua classe-mãe (superclasse). Uma subclasse pode herdar a estrutura de dados e os métodos, ou alguns dos métodos, de sua superclasse.

  A aplicação do conceito de herança é importante para modelagem do sistema visando futuramente alguma modificação nas características das classes correspondentes à uma classe abstrata.

  Ela também tem métodos e, às vezes, tipos de dados próprios. A aplicação da herança neste sistema é apresentada a seguir:

  A classe Usuario é uma classe-mãe para as subclasses: Discente, Docente e Tae. As propriedades herdadas são: nome, identificador, email, telefone, senha, confirmarSenha, horariosUsuario.

  A classe Refeicao é uma classe-mãe para as subclasses: EncomendarRefeicao e AgendarRefeicao. As propriedades herdadas são: id, diaSemana, turnoRefeicao, horario.

- **Polimorfismo**

  Polimorfismo é o princípio pelo qual duas ou mais classes derivadas de uma mesma superclasse podem invocar métodos que têm a mesma identificação (assinatura) mas comportamentos distintos, especializados para cada classe derivada, usando para tanto uma referência a um objeto do tipo da superclasse. A aplicação do polimorfismo neste sistema é apresentada a seguir:

As subclasses Discente, Docente e Tae, apesar de serem derivadas da mesma superclasse, se comportam de maneira distintas a partir do atributo tipo.

As subclasses EncomendarRefeicao e AgendarRefeicao, apesar de serem derivadas da mesma superclasse, se comportam de maneira distintas a partir do atributo tipo.

- **Classes Abstratas**

A classe Abstrata é uma classe que não possui objetos instanciados a partir dela. As classes abstratas são criadas quando necessitamos de uma classe que implementa recursos comuns a duas ou mais classes.
Dessa forma, para modelagem de nosso sistema, foi implementado as classes Abstratas: Usuário e Refeição. Assim, podendo trabalhar com herança para uma a instanciação e com a classe mãe abstrata para modelagem de novas características.

- **Interface**

A Interface é uma classe abstrata composta somente por métodos abstratos, logo não pode ser instanciada, ou seja, ela só contém as declarações dos métodos e constantes, nenhuma implementação, somente o 'molde'. Pode-se dizer, a grosso modo, que uma interface é um contrato que quando assumido por uma classe deve ser implementado. A classe Eticket é uma classe Abstrata de Interface, pois o contrato é aplicado diferentemente quando chamado para super classe Usuario, e consequentemente para suas classes filhas..

- **Coleções - Collections**

O pacote java.util é um conjunto de classes e interfaces que implementam Collection, tendo assim que, obrigatoriamente, trabalhar com métodos de adição e remoção de elementos em coleções de dados. Uma coleção (collection) é um objeto que serve para agrupar muitos elementos em uma única unidade, estes elementos precisam ser Objetos. A aplicação de coleções é apresentado a seguir:

A classe BancoDeDadosUsuario armazena os atributos/informações referente aos dados cadastrais dos usuários e suas ações, como encomendas e agendamentos, em ArrayList. A lista usuarios é preenchida com os atributos da classe Usuario. Já as listas refeicoes e encomendas, recebem os atributos da classe Refeicoes. Além dessas listas, ainda estão presentes as listas horarios e mensagem, que recebem respectivamente os atributos das classes SemanaHorarios e Contato, respectivamente.

- **Tratamento de exceções**

Uma exceção, basicamente é uma classe de Java representada na sua forma mais genérica pela classe java.lang.Exception, logo todas as exceções que ocorram ao longo da execução do seu programa podem ser tratadas como objetos do tipo

Exception. A aplicação de exceções é apresentado a seguir: Foi utilizado o tratamento de exceções na verificações na Classe BancoDeDadosUsuario, visto que lá, fazemos as validações dos dados recebidos na view e passados para o controller.

- **Leitura de Arquivo**

  A leitura de arquivos é feita utilizando a API gson, que transforma o objeto Usuario em um json, e assim, esse json é lido toda vez que o programa é iniciado, assim carregando as informações adicionadas anteriormente.