## Composição do projeto

O projeto startup-lanche-loja é composto pelos seguintes módulos:

|  |  |
| --- | --- |
| Projeto/ Sub módulo | Descrição |
| Startup-lanche-loja | Projeto agregador de todos os módulos |
| Build | Projeto agregador dos sub módulos e dependências. Tem a finalidade de gerar o arquivo jar executável do projeto. |
| Startup-lanche-application | Projeto agregador dos submodulos que compõem as regras da aplicação e a api REST (não implementada) que funcionariam como base para o módulo front-end. |
| Startup-lanche-commons | Projeto que contém as classes e interfaces utilizadas para transferir informações entre as camadas. |
| Startup-lanche-rest | Sub módulo destinado a conter os recursos expostos via api REST (não implementada). Módulo responsável pelo startup do projeto, devendo portanto ter em seu pom todas as dependências dos demais módulos “abaixo” dele (core e demais). |
| Startup-lanche-service | Sub módulo que contém a interface de acesso ao core, que possui as implementações das regras de negócio. Deve ser importado pelo projeto “startup-lanche-rest”, a fim de injetar a implementação do core, e pelo próprio core, para realizar a implementação das funções definidas no contrato. |
| Startup-lanche-core-application | Sub módulo que possui as implementações de regras de negócio, bem como implementa o contrato utilizado pela api REST. |
| Startup-lanche-adapter-services | Sub módulo destinado a guardar todos os contratos para informações que o core precise consultar ou inserir em recursos. Não representa exatamente a implementação do pattern adapter, mas o termo adapter se encaixa bem na funcionalidade atribuída ao módulo |
| Startap-lanche-data-adapter | Sub módulo que representa a implementação do contrato de adapter de dados contido no projeto “startup-lanche-adapter-services” |
| Startup-lanche-loja-front | Sub-modulo do nível raiz do projeto destinado a conter a implementação das interfaces de usuário para o projeto. |

## Modo de execução

O projeto pode ser executado através do arquivo jar que é gerado na pasta target do projeto build após a execução do build do projeto.

Etapas para a execução

1. Em uma estação com o apache maven instalado navegar via cmd até a pasta raiz na qual o projeto foi clonado
2. Navegar até o nível zero do projeto (pasta startup-lanche-loja)
3. Executar o comando mvn clean install –e
4. Navegar até a pasta build/target
5. Executar o comando “java –jar build.version.jar”, onde deve-se substituir o termo version pela versão do jar gerada após o build.

Justificando a solução

A primeira preocupação foi com a elaboração do modelo dos dados. Como representar a estrutura de lanches pedidos e ingredientes de forma que atendessem as regras definidas para calculo de preço de lanches e pedidos padrão e customizados. Então foi decidido estruturar a aplicação em “Lanches” que possuem itens compostos por “Ingredientes” e “quantidade de cada ingrediente”. Os ingredientes possuem a classificação definida com base em uma enumeração, “TipoIngrediente” . Pedidos existem a partir do momento que um cliente escolhe um ou mais lanches, dessa forma possuindo itens compostos por “lanches” e “quantidade de lanches”.

O valor do lanche, assim como o valor do pedido são elementos intrínsecos aos seus respectivos objetos, “lanche” e “pedido”, porém suas regras são estabelecidas pelo negócio e podem incluir elementos que os objetos de transferência não conhecem nem precisam conhecer. Devido a essa característica foi criado um contrato através do qual os preços são calculados em nível de cada lanche, contrato este que deve ser informado no momento no qual se deseja recuperar o valor do preço de um determinado pedido.

Definido o modelo “ideal” a preocupação seguinte foi em como registrar em memória os dados para composição de lanches e pedidos de forma eu pudesse compartilhar este contexto com um projeto externo de testes automatizados, o qual segundo o especificado no enunciado do teste, seria realizado com massa de dados controlada, devido a constante variação de preços dos ingredientes. Inicialmente seria utilizado o banco de dados em memória H2, porém a situação seria bem parecida com utilizar uma lista de objetos , opção escolhida. Não importaria ai o meio no qual os dados são persistidos, mas sim a relação de acesso do “core-application” com este módulo. Desta forma optei por implementar o repositório no mesmo projeto das regras de negocio e utilizei contrato de service apenas para receber as listas da passa de dados. O que em um projeto de testes automatizados (que não foi feito- já estava tarde rsrs) teria a dependência maven deste ponto específico trocada para que o serviço principal utilizasse a massa de dados definida para os testes.

## Pontos de melhoria

Apesar do ter sido encaminhado, existem pontos que poderiam estar mais organizados, como por exemplo, os seguintes:

1. Organização e centralização das dependências do maven no projeto pai, para evitar a necessidade de configurar as versões de cada dependência no projeto filho. Isso possibilita a existência de uma série de conflitos como a sobrescrita de versão de dependências de forma indevida, etc.
2. Realização de Testes unitários.
3. Utilização de um sub módulo para a implementação do próprio repositório, não apenas para a massa de dados.

## Projeto de testes

Não pude implementar o projeto de teste no tempo que utilizei para construir o projeto principal, mas a sua composição basicamente ficaria dividida em 3 partes:

1. Uma camada central que possuiria a implementação dos cenários de testes ou bdds para avaliação da solução e que possuiria acesso aos contratos das duas demais camadas;
2. Uma camada responsável por enviar informações para o projeto principal (na minha concepção para o testes seria construída em contato com a api REST do projeto principal).
3. Uma camada responsável por persistir e consultar as informações nos repositórios de dados do projeto principal, ou mais especificamente para este projeto, por substituir o “data-adapter” utilizado pelo ”core-application” como base de dados.

O projeto “core-application” então ficou responsável por resgatar a massa de dados através do “data adapter” e do repositório, prover a lógica de cálculo para o preço de um lanche e prover a implementações do serviço utilizado pela camada REST, ILojaService.

Obs1: O desenvolvimento do projeto foi iniciado no dia 01/08/2017 ás 21:30h e terminado aproximadamente as às 03:30h do dia 02/08/2017. Devido ao tempo, apesar de ter sido dada oportunidade de elaborar o projeto no fim de semana, o mesmo não se encontra concluído o foco para o seu desenvolvimento foi elaborar a solução de forma mais simples, atendendo as regras implementadas, e com estrutura passível de inclusão de um projeto de testes automatizados.

Obs2: Não foram escritos testes unitários para o projeto devido a duas questões:

1. Não estão especificados no enunciado do projeto
2. Mantive o foco na estrutura do projeto para que ficasse o mais orientado a objetos possível e passível de ser associado a um projeto de testes automatizados.