Projeto SAMSUNG Centro de Informática UFPE FADE UFPE

Relatório de planejamento e desenvolvimento de projeto - Desafio Shopping

Nome do projeto	Desafio Shopping – Cin/Samsumg
Solicitante	Fabiana Marinho
E-mail do solicitante	fom@cin.ufpe.br
Executor	Michael Lopes Bastos
Número de Revisão	1.0
Data de entrega	28/03/2021

Recife – PE

Sumário

1. Introdução

Este relatório tem como objetivo principal, demonstrar o processo de planejamento de desenvolvimento da solução para o Desafio do Shopping, como parte do requisito para a seleção de Engenheiro de Software Júnior, do projeto Cin-Samsung. Toda a solução aqui explanada, foi desenvolvida com base no documento de desafio, enviado por e-mail ás 22:00 h do dia 23/03/2021.

Neste contexto, a solução proposta pelo desafio, se trata de um sistema que possa ser capaz de registrar pedidos de compra e, a partir deles, gerar uma fatura adequada para os itens que forem solicitados dentro das ordens de compra. Os dados que o sistema irá consumir, serão derivados de arquivos de texto simples, seguindo determinado padrão previamente estipulado pelo documento. A saída do sistema também deve ser em um .txt, contendo as informações finais da fatura (quantidade de ordens, montante de mentidos e total de cada ordem).

Assim, para atender aos requisitos solicitados e ir um pouco mais além do que foi solicitado, foi desenvolvida uma solução com interface gráfica em WPF e outra com interação por meio de Console, ambas com a linguagem C#. Os dois projetos então em um mesmo ambiente, logo, as duas soluções têm acesso ao mesmo back-end, mudando no final, apenas a forma de interação do usuário com a aplicação.

Para melhor detalhar o passo a passo desse desenvolvimento, este relatório será apresentado da seguinte forma: a Seção 2 irá explanar sobre os Requisitos Funcionais de Não-Funcionais do projeto; a Seção 3, irá abordar um pouco sobre as tecnologias utilizadas; a Seção 4 demonstra a arquitetura da solução; a Seção 5 demonstra as principais funções do protótipo do sistema; a Seção 6, fala sobre os planejamento e execução dos casos de teste e; por fim, a Seção 7 traz as considerações final sobre a execução.

2. Requisitos Funcionais e Não-Funcionais

Como já mencionado anteriormente, os principais requisitos funcionais e não funcionais para a elaboração deste projeto foram derivados do arquivo do desafio. A ***Tabela 1*** descreve de forma mais detalhada os Requisitos Funcionais identificados.

RF-ID	Nome	Descrição
RF-01	Registrar pedido de compra	Criar um mecanismos de leitura e organização de pedidos de compra conforme modelo definido em documento

RF-02	Criar sistema de carregamento de Ordens de produto	Implementar mecanismo para realizar a leitura de um arquivo .txt com as informações referentes as ordens de compra
RF-03	Listar produtos com desconto	Listar todos os produtos dentro do conjunto de produtos com desconto
RF-04	Adicionar desconto	Criar mecanismo para fornecer desconto ao cliente em caso do produto comprado atender às regras de determinado desconto
RF-05	Criar sistema de carregamento de produtos com desconto	Implementar mecanismo para realizar a leitura de um arquivo .txt com as informações referentes aos produtos com desconto
RF-06	Listar produtos	Criar mecanismo para listar todos os produtos disponíveis
RF-07	Criar sistema de carregamento de Lista de produtos disponíveis	Implementar mecanismo para realizar a leitura de um arquivo .txt com as informações referentes aos produtos com disponíveis
RF-08	Gerar Fatura	Criar mecanismos de geração automática de fatura
RF-09	Criar sistema para exportação	Implementar mecanismo de exportação, com resultado da fatura final para um arquivo .txt, tendo como base o padrão e as regras definidas no documento do desafio.
RF-10	Listar fatura	Listar todas as informações referentes a fatura que foi gerada

Tabela 1 - Requisitos Funcionais

Além dos requisitos funcionais, os seguintes Requisitos Não-Funcionais foram identificados e listados dentro do escopo do domínio da solução. A ***Tabela 2*** faz um apanho geral de tais exigências.

RNF-ID	Nome	Descrição
RNF-01	Modularidade	A solução deve ser modular, capaz de se adequar a novos contextos sem necessidade de grandes mudanças na estrutura do código.
RNF-02	Desempenho	Boa performance de execução por meio da

		implementação eficiente dos seus algoritmos
RNF-03	Escalabilidade	O sistema deve ser capaz de se adaptar a novas necessidades, com fácil expansão e suporte a um número maior de informações.
RNF-04	Linguagem	O sistema deve ser implementado utilizando C#, Java ou C++
RNF-05	Aplicação de console	O sistema deve suportar a entrada e leitura de informações via console.
RNF-06	As entradas devem ser lidas pelo console, através de arquivos .txt	O caminho de cada arquivo de entrada deverá ser imputado no sistema via janela de console.
RNF-07	O fatura final deve estar em formato .txt	O caminho da fatura final, deve ser especificado pela janela de console e deve ser no formato .txt
RNF-08	Devem ser implementados testes de unidade	Criação de testes unitários para validar as funcionalidades do sistema
RNF-09	Expor o funcionamento interno	Criar debugs internos para exibir como saída, o fluxo de funcionamento do sistema.
RNF-10	Criação de um arquivo README	Criar arquivo de texto explicando como executar a aplicação
RNF-11	O código final deve estar em um BitBucket git repositor compartilhado com selecaodesafiocinsamsung@gmail.com	Enviar para um repositório online de versionamento de código, a versão final da aplicação.

Tabela 2 - Requisitos Não-Funcionais

Com base nesse conjunto inicial de informações e definições, foram definidas as tecnologias a serem utilizadas e deu-se início a fase de criação da arquitetura e do fluxo das classes do sistema, que serão explanadas com maiores detalhes nas próximas Seções.

3. Tecnologias Utilizadas

Tomando como base os Requisitos Funcionais e não Funcionais que foram levantados, o desenvolvimento teve como tecnologias principais o WPF com a linguagem XAML e C# para criação de uma interface gráfica simples e do back-end, JUnit para execução de testes de unidade, Photoshop, para ajuste e edição das imagens utilizadas na interface, Visual Studio 2019 como IDE para desenvolvimento e, o GitHub, como ferramenta de versionamento. Para construção da arquitetura foi utilizada a versão gratuita do Visual-Paradigm e, para gestão das atividades de execução e planejamento, o Trello foi a ferramenta escolhida.

4. Arquitetura da solução

A arquitetura da solução foi construída tendo em vista criar um sistema modular, preparado para se relacionar com diferentes tipos de interface e base de dados. Através de uma composição MVC (model, view, controller) as principais camadas do sistema ficam modularizadas, tornando o fluxo entre os componentes mais fluído, otimizado e organizado. Isso melhora a performance do sistema, torna ele mais legível, escalável e mais fácil de manter, tendo em vista a dinâmica bem definida de seus componentes. Na **Erro! Fonte de referência não encontrada.** é representada a arquitetura geral da a plicação, demonstrando a interação principal entre seus módulos.

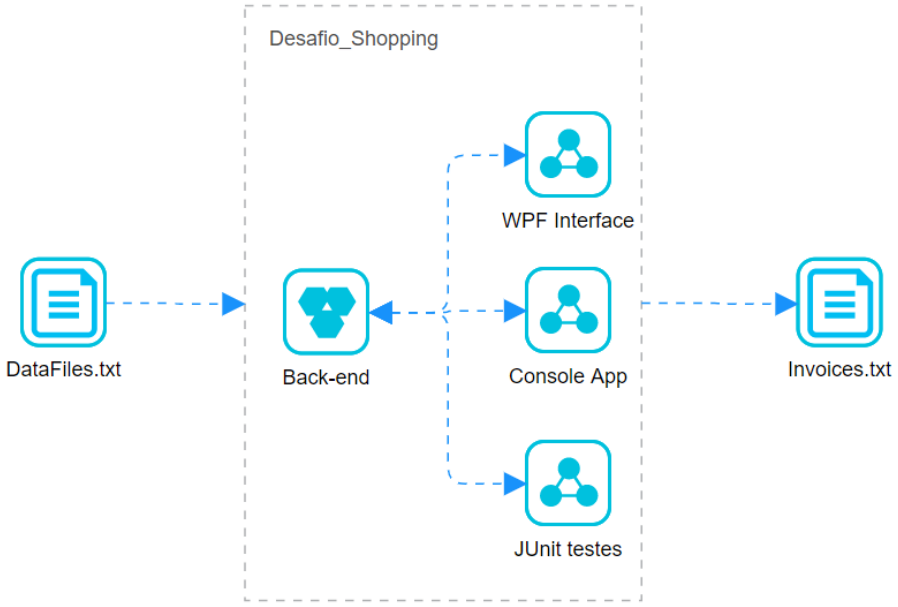

Figura 1 - Arquitetura Geral do Sistema

No passo inicial representado na arquitetura, o sistema faz o carregamento de todos os arquivos de texto com as informações dos produtos, descontos e ordens de compra e, a partir daí, essas entradas são processadas pelo back-end e repassadas para as duas interfaces e para o módulo de testes de unidade (se acionado), que faz a validação dos principais métodos da aplicação. Como etapa final, o sistema emite um arquivo .txt com as informações formuladas da fatura, de acordo com o padrão estipulado no documento de desafio. Como a própria arquitetura já mostra, diferentes tipos de interface e base de dados podem ser integradas ao mesmo núcleo de sistema, o que sugere a possibilidade de implementação de uma segunda fonte de dados e armazenamento. Como sugestão, foi construído um modelo de arquitetura de dados para implementação futura de algum modelo de armazenamento de dados NoSQL, baseado em documentos. Tendo em vista que não existe um padrão bem difundido da representação gráfica desse tipo de armazenamento, foi utilizado o estudo de (LIMA, 2016), como base teoria para esta elaboração. O resultado é demonstrado na **Figura *2* –Sugestão de Modelo de projeto lógico para banco de dados NoSQL baseado em documentosFigura *2***.

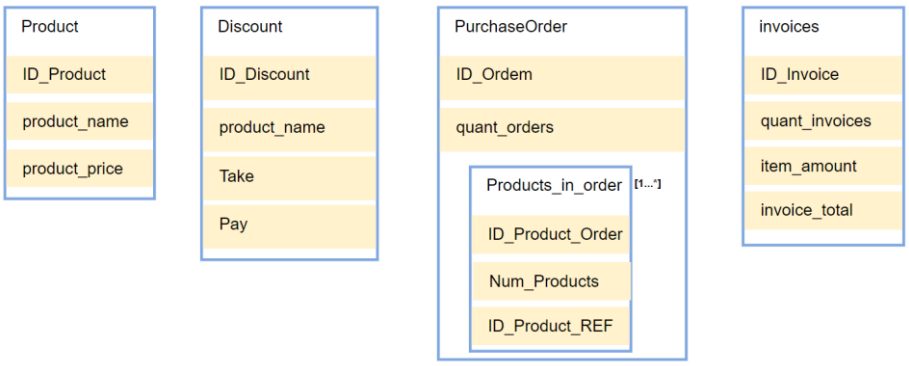

Figura 2 –Sugestão de Modelo de projeto lógico para banco de dados NoSQL baseado em documentos

Em relação a estrutura interna do projeto, visando uma construção mais assertiva e a modularização do sistema, foi desenvolvido um Diagrama UML de Classes. Nessa representação, é descrita de forma visual as principais classes, interfaces e pacotes do sistema, assim como suas respectivas associações, hierarquias, agregações e cardinalidades. O Diagrama foi feito visando também, deixar claro alguns dos padrões de projetos utilizados, como MVC e FACHADA. Outros padrões como Singleton, foram utilizados aplicado dentro das classes DAO (Data Access Object), da própria FACHADA e das View's do WPF, para garantir a existência de apenas uma instância para cada uma dessas classes. Na Representação do diagrama, também está representado os diretórios auxiliares, referentes ao diretório de arquivos e ao diretório de imagens, utilizados pelo Model e pela View, respectivamente. O Diagrama final e suas interações é detalhado na **Figura *3***.

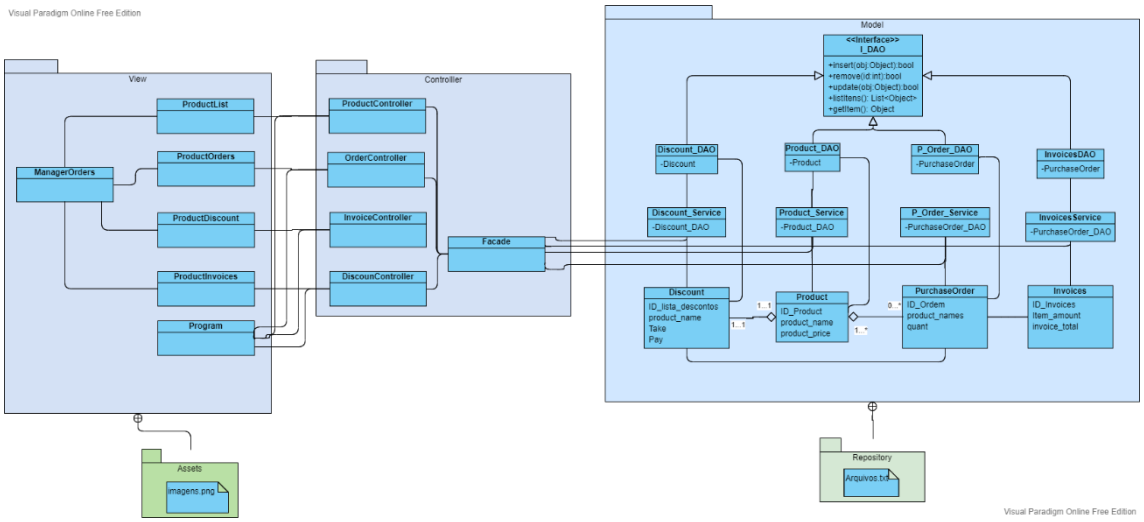

Figura 3 - Diagrama UML de Classes simplificado da solução

É importante destacar que, devido ao breve tempo para elaboração do planejamento e execução do projeto, detalhes sobres alguns atributos e métodos acabaram ficando de fora do diagrama, buscando simplificar seu processo de desenvolvimento.

5. Funções do protótipo

Tendo em vista incrementar o desenvolvimento do desafio e deixar mais eficiente a visualização das informações processadas pela aplicação, além do módulo de console, foi desenvolvida uma interface gráfica simplificada, com um usuário previamente definido e ações para quatro principais funções, são elas: Carregar produtos (*List All Products*), Carregar Produtos com Desconto (*Products with Discounts*), Carregar Ordens (*Orders Products*) de produto e, gerar Fatura final (*Invoices Products*). A tela de Login e de Home, podem ser vistas com detalhes na **Figura *4*** e as demais telas referente as funções da aplicação, estão representadas na **Figura *5***.

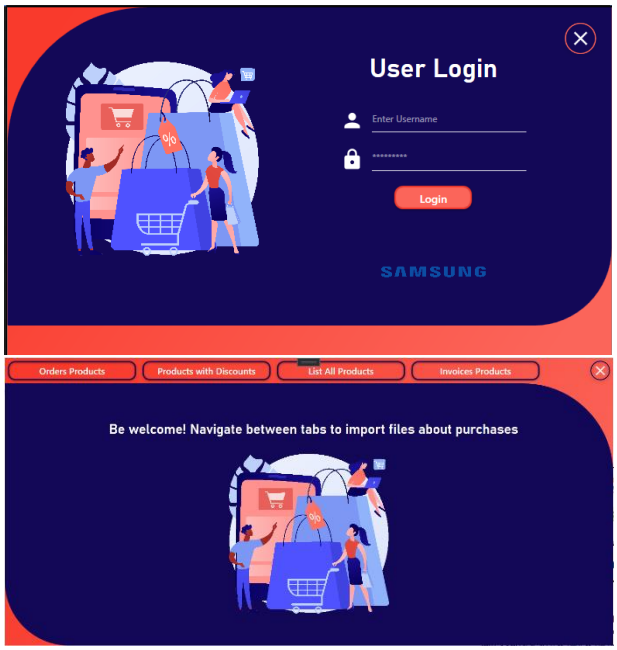

Figura 4 - Tela de Login e Home do protótipo

Figura 5 - Telas das principais funcionalidades da aplicação

Além do módulo de interface com WPF, como já mencionado, também foi criado o sistema baseado na execução dos comandos via Console, por meio da inserção dos dados através de linha de comando. Sua demonstração final é representada na ***Figura 6*** .

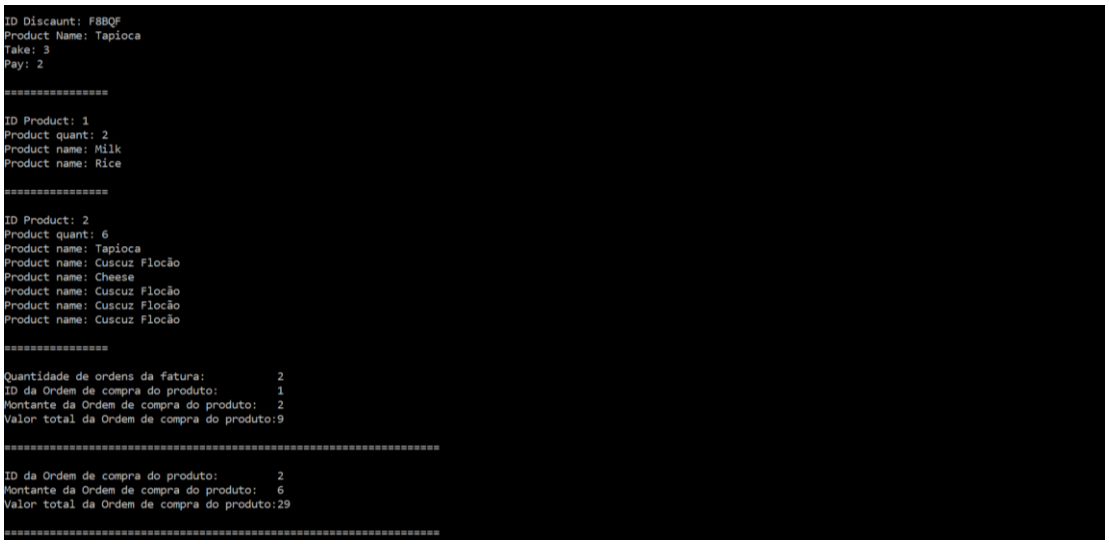

Figura 6- Saída do Console para o resultado da execução

6. Planejamento e execução dos casos de teste

Para a aplicação dos testes, além de testes exploratórios manuais, foram executados um conjunto de 13 testes de unidade nas funções consideradas de maior fluxo e utilização dentro do protótipo. A execução dessa etapa foi baseada no plano de testes relatado na Tabela 3.

Método	Cenário	Resultado esperado
createInvoices()	Erro de carregamento	Levantar Execption
createInvoices()	Sucesso no carregamento	Retornar lista completa
getAllProductsOrders()	Erro de carregamento	Levantar Execption
getAllProductsOrders()	Sucesso no carregamento	Retornar lista completa
getAllProducts()	Erro de carregamento	Levantar Execption
getAllProducts()	Sucesso no carregamento	Retornar lista completa
getAllProductsDiscount()	Erro de carregamento	Levantar Execption
getAllProductsDiscount()	Sucesso no carregamento	Retornar lista completa
invoices()	Retornando valor nulo	Retornar valor Nulo
invoices()	Seguindo o padrão da lista	Lista dentro do padrão desejado
invoices()	Geração correta da lista	Lista fora do padrão
makeDiscount()	Desconto atribuído	Retornar valor positivo com desconto
makeDiscount()	Desconto não atribuído	Retornar mesmo valor do montante sem desconto

Tabela 3 - Plano de testes da aplicação

Desta forma, tendo em vista a execução dos casos de teste definidos no planejamento, foi criado o módulo de testes dentro do projeto com JUnit, com uma função de teste para cada

um dos testes levantados dentro do planejamento. O resultado dessa execução e apresentado na **Figura *7***.

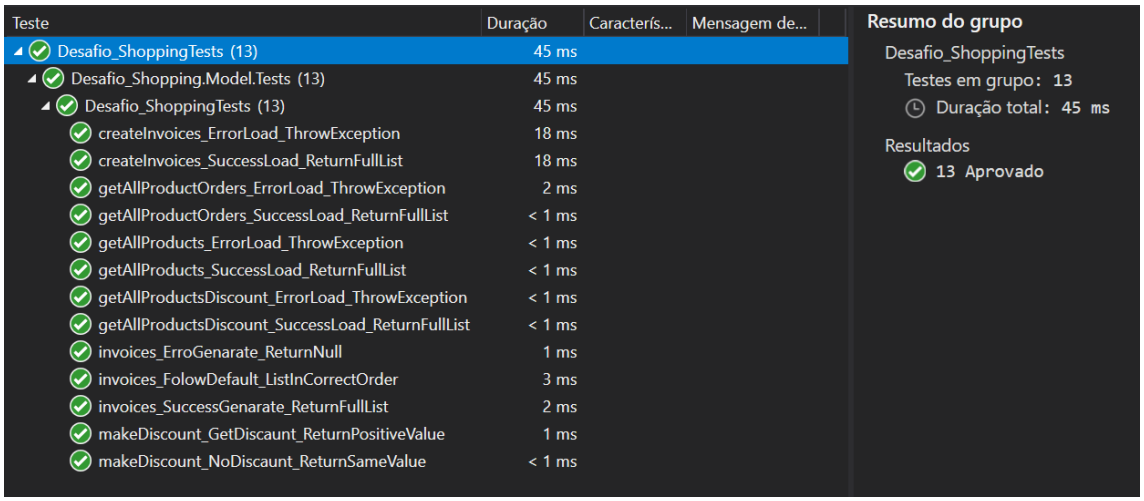

Figura 7 - Resultado da execução dos casos de teste

Como sugestão, em caso de mais tempo para testes, seria recomendado ampliar os testes exploratório dentro do sistema, visando encontrar pontos de falha que não estão cobertos pelos casos de teste. Outro ponto também, seria a realização de uma rodada de testes e validações para a interface, utilizando o feedback dos usuários finais para definição da melhor iteração com o sistema, visando usabilidade, conforto e eficiência funcional do sistema.

7. Considerações finais

O protótipo desenvolvido buscou atender de maneira pontual aos requisitos levantados pelo documento do desafio, tanto de maneira funcional como técnica, buscando uma implementação simples e bem estruturada, através da aplicação de padrões de projeto, conceitos de Orientação a Objetos, Clean code, entre outros. Dessa forma, todos os requisitos solicitados foram atendidos e outros não previstos no documento inicial foram implementados, tornando o que deveria ser um simples sistema de console, em um protótipo funcional. A aplicação final se encontra no GitHub no link: https://github.com/michaellopes16/DesafioShopping03-2021-MichaelLopesBastos.git, justamente com esta documentação e suas instruções de utilização.

Referências

LIMA, C. De. **PROJETO LÓGICO DE BANCOS DE DADOS NOSQL DOCUMENTO A PARTIR DE ESQUEMAS CONCEITUAIS ENTIDADE-RELACIONAMENTO ESTENDIDO (EER) Florianópolis**. *Universidade Federal de Santa Catarina*, [s.l.], 2016.