**Bootcamp: Arquitetura de Software**

**Aluno(a): Leandro Rodrigues Pedroso**

**Relatório de Entrega do Desafio Final**

Introdução

O desafio final é uma atividade realizada pelo(a) aluno(a), **individualmente**, visando **levar à prática o conteúdo do bootcamp na totalidade**, utilizando as principais ferramentas, conteúdos, dinâmicas e modalidades orientadas pelo coordenador do curso.

O desafio final é **composto por um enunciado**, com viés de atividade prática (laboratório guiado, desafio a resolver, case a analisar etc.) com os principais tópicos vistos ao longo dos quatro módulos do bootcamp.

Quando necessário, o enunciado do desafio final pode ser acompanhado de uma videoaula gravada de orientação, onde o professor explica qual é a proposta do desafio final do bootcamp em questão, detalhando a dinâmica e elencando ferramentas/pré-requisitos para a resolução do enunciado.

Os principais passos, as escolhas e decisões tomadas para a resolução do enunciado do desafio final, bem como o resultado final, devem ser registrados pelo(a) aluno(a) neste documento, observando-se as **instruções de preenchimento**. Importante também observar os critérios no **protocolo avaliativo** da entrega.

Expirado o prazo de entrega, será disponibilizada uma **videoaula gravada contendo a resolução** da prática/problema/case proposto no enunciado do desafio final, bem como as devidas explicações acerca das decisões tomadas/ações executadas pelo professor para chegar no resultado.

**Protocolo Avaliativo**

O valor total do desafio final é de **95 pontos**. Ele será avaliado pelo professor do módulo em questão, com base no relatório do desafio final enviado pelo(a) aluno(a) e seguindo-se o protocolo avaliativo abaixo:

1. Todos os itens solicitados pelo(a) professor(a) no enunciado do desafio final foram entregues?

Satisfatório - 19 pontos (50% a 100% dos itens)

Parcialmente Satisfatório - 12 pontos (entre 25% e 50% dos itens)

Parcialmente Insatisfatório - 5 pontos (1% a 25% dos itens)

Insatisfatório/Não entregue - 0 pontos (0% dos itens)

1. Os desafios/requisitos/funcionalidades foram resolvidos/considerados/implementados corretamente pelo(a) aluno(a)?

Satisfatório - 19 pontos (50% a 100% dos itens)

Parcialmente Satisfatório - 12 pontos (entre 25% e 50% dos itens)

Parcialmente Insatisfatório - 5 pontos (1% a 25% dos itens)

Insatisfatório/Não entregue - 0 pontos (0% dos itens)

1. O(a) aluno(a) utilizou corretamente as ferramentas, conceitos e tecnologias apresentadas ao longo do bootcamp?

Satisfatório - 19 pontos (50% a 100% dos itens)

Parcialmente Satisfatório - 12 pontos (entre 25% e 50% dos itens)

Parcialmente Insatisfatório - 5 pontos (1% a 25% dos itens)

Insatisfatório/Não entregue - 0 pontos (0% dos itens)

1. O(a) aluno(a) forneceu comentários detalhados e documentação completa, facilitando a compreensão da sua proposta de resolução do desafio?

Satisfatório - 19 pontos (50% a 100% dos itens)

Parcialmente Satisfatório - 12 pontos (entre 25% e 50% dos itens)

Parcialmente Insatisfatório - 5 pontos (1% a 25% dos itens)

Insatisfatório/Não entregue - 0 pontos (0% dos itens)

1. O(a) aluno(a) entregou a solução com qualidade e zelo, demonstrando dedicação e empenho na solução do desafio?

Satisfatório - 19 pontos (50% a 100% dos itens)

Parcialmente Satisfatório - 12 pontos (entre 25% e 50% dos itens)

Parcialmente Insatisfatório - 5 pontos (1% a 25% dos itens)

Insatisfatório/Não entregue - 0 pontos (0% dos itens)

**Instruções de Preenchimento do Relatório do Desafio Final**

O preenchimento do relatório deve ser realizado com base nas diretrizes abaixo, para assegurar uma avaliação justa e detalhada de cada item solicitado.

1. **Entrega Completa dos Itens Solicitados**

Verifique se todos os itens mencionados no enunciado do desafio final foram devidamente contemplados na sua entrega, pois a entrega deles é fundamental para a avaliação do desafio final na totalidade.

1. **Resolução dos Desafios e Implementação dos Requisitos**

Avalie se você abordou corretamente o(s) desafio(s) proposto(s), considerando e implementando todas as funcionalidades e requisitos apresentados no enunciado. O objetivo é verificar se a solução atende às expectativas técnicas e funcionais solicitadas.

1. **Utilização das Ferramentas, Conceitos e Tecnologias**

Certifique-se de que você aplicou corretamente as ferramentas, conceitos e tecnologias discutidas ao longo do bootcamp. A adequação e o uso eficaz dos recursos disponíveis são elementos essenciais para a solução do desafio.

1. **Qualidade dos Comentários e Documentação**

Verifique se você forneceu os comentários necessários e detalhados acerca dos passos e decisões tomadas na resolução do desafio. Esses comentários servem para facilitar a compreensão da proposta de solução, demonstrando clareza e organização.

1. **Qualidade e Capricho na Entrega Final**

Observe se o seu relatório do desafio final apresenta um nível adequado de qualidade, zelo e capricho, de forma que evidencie sua dedicação, empenho e atenção aos detalhes durante o desenvolvimento da solução.

O preenchimento deve ser realizado com base em evidências concretas encontradas no trabalho entregue, assegurando que todos os aspectos mencionados acima sejam considerados.

**Desenvolvimento da Solução do Desafio Final**

1. **Sobre a tecnologia utilizada**
   1. A primeira definição foi buscar uma tecnologia que fizesse sentido para a proposta da solução que eu tinha em mente criar, alinhando também com a proposta do desafio. A escolha do .NET 8 com linguagem C# foi justamente pensando nisso.  
      Além disso, considerei que linguagens mais robustas como C# e Java proporcionam muito mais segurança para a criação de sistemas incrementais, que tendem a ser mais complexos, levando em consideração que, com a melhora nos últimos anos na plataforma .NET, ela se tornou uma das plataformas mais performáticas do mercado, superando até mesmo, em alguns casos, o Node. Um ponto em particular que vale a pena mencionar é a utilização do *EntityFrameworkCore* como *provider* de comunicação com banco de dados. Além de ser um pacote extremamente produtivo, nas suas últimas atualizações houve uma melhora significativa em sua performance, tornando-se viável para o que foi proposto no desafio e também em muitos projetos para o mercado.
2. **Regras de negócio** 
   1. Ao analisar os requisitos e definir o domínio que iria utilizar, eu parti para a definição das regras de negócio. A validação da formatação das informações do produto e da obrigatoriedade de propriedades no momento da inclusão e atualização do produto foi definida como regra de domínio de aplicação, por se tratar de algo específico do produto. Já a validação se o produto existe, para os processos de obter produto por código, obter produto por nome e excluir produto, foi direcionada para a camada de aplicação, por se tratar de um processo que envolve comunicação diretamente com a infraestrutura por duas vezes.
3. **Arquiteturas de software**
   1. Para a estruturação da solução, optei por utilizar a *Clean Architecture* visando a facilidade da expansão do domínio e na criação de novos, considerando que atualmente todos os softwares já nascem com a premissa de ser incremental. Outro ponto que gostaria de destacar é a flexibilidade para adicionar novas formas de comunicação. Na proposta atual, o caso de uso está sendo chamado por uma API Rest, mas fica flexível para ser chamado por uma apresentação do tipo aplicativo móvel, ou até mesmo da camada de aplicação para dentro ser um pacote, por exemplo, e o melhor de tudo: sem mudar nada no "core" da aplicação. No desenho de arquitetura coloco um diagrama "Arquitetura da aplicação – Proposta futura expansões" para exemplificar os pontos de expansão, que justifica mais ainda a escolha. Vale a pena mencionar que, em nenhum momento na proposta, eu deixei de abordar o conceito de MVC. Porém, acabei reservando o padrão para ser implementado na camada de API Rest, porque na minha opinião, uma arquitetura não invalida a outra, elas se  
      complementam. No meu ponto de vista, para as camadas de apresentação, o padrão MVC fica melhor empregado.
4. **Estrutura de patas e explicações dos componentes**
   1. XPE.SoftwareArch.FinalChallenge.Api:  
      Tratei essa camada como se fosse a camada de apresentação do meu projeto, o que no meu ponto de vista pode ser considerado por se tratar de uma API Rest, aqui nessa camada eu adotei o padrão MCV, conforme foi proposto no desafio, aqui tentei abordar o máximo os níveis de maturidade de Richardson, além de também documentar muito bem a controller utilizando o Swagger .
      1. Models: Estão as minhas entidades necessárias para fazer os processos sobre o produto, a comunicação entre o mundo externo e o sistema, nesse ponto utilizei uma classe base de produto que fosse possível sobrescrever a propriedade Id, o objetivo é que no momento do cadastro não fique exposto no *body* da requisição o código do produto, o que no caso da atualização não é necessário e no caso da atualização o parâmetro código vem via rota, porém para consulta dos produtos é necessário o retorno do código;
      2. Controllers: Fazem a apenas o meio de campo, entre as *view models* e os serviços;
      3. Services: Nesse ponto é a comunicação entre a camada de apresentação com a camada de aplicação, tratando especificamente dessa integração, onde foi destinado ao tratamento de erros que vem da camada de aplicação, domínio e infraestrutura, criei um *helper* com o objetivo de facilitar esse tratamento e deixando explico o que cada tipo de erro;
   2. XPE.SoftwareArch.FinalChallenge.Application  
      Camada responsável por fazer a integração entre a infraestruturas e o domínio sem que ambos sejam visíveis para a camada de aplicação, tem a reponsabilidade de garantir a integridade da entidade de produtos no domínio.
      1. Exceptions: Aqui tem as exceções que pertence a camada de aplicação
      2. ProductUseCases: Os casos de uso para produtos, nesse ponto temos inputs e outputs para garantir a integridade do domínio e a implementação do caso de uso em questão bem com as regras de negócio que seja de responsabilidade desse passo, no caso a validação de produto existente para obter produtos por código ou nome e exclusão.
   3. XPE.SoftwareArch.FinalChallenge.Domain  
      O core da aplicação, camada onde fica as regras do domínio, abstrações para a infraestrutura
      1. Exceptions: Aqui tem as exceções que pertence a camada de domínio, considerando que ela terá também as exceções da camada de infraestrutura.
      2. ProductAggregate: Contém a entidade de produtos e a abstração para o repositório do tipo do produto, aqui nesse ponto garantimos que a tecnologia não influencie na camada de aplicação com o padrão *repository*.
   4. XPE.SoftwareArch.FinalChallenge.Infra  
      Nesse ponto utilizamos a abstração do repositório de produto e temos o forte acoplamento com a tecnologia, que no caso e o *EntityFrameworkCore*, porém, deixando bem flexível a mudança de *provider* de acesso a dados
      1. Migrations: São as configurações padrões do *entityframework core*
      2. ProductData: Contém a configuração para a tabela do banco de dados que será gerada a partir de cada *Migrations*, e também tem a implementação do concreta do repositório
   5. XPE.SoftwareArch.FinalChallenge.Ioc  
      Aqui está a inversão de controle, é a camada que irá garantir que a arquitetura *Clean Architetu*re vai funcionar conforme seu propósito.
5. Entrega do exercício
   1. Desenho arquitetural do software: O desenho da arquitetura está no meu github, [na pasta documentação](a.%09https:/github.com/lrpedroso36/XPE.SoftwareArch.FinalChallenge/tree/main/documenta%C3%A7%C3%A3o), nele tem o desenho da arquitetura atual e uma exemplificação de uma proposta de expansão.
   2. Entrega do Código: [link para acessar o código fonte da solução](https://github.com/lrpedroso36/XPE.SoftwareArch.FinalChallenge)

**Conclusão**A minha proposta de solução foi pensada levando em consideração uma das premissas mais comuns do mercado de trabalho e que foi bastante reforçada no **Módulo 2**, que é o desenvolvimento de softwares incrementais.  
Desde o começo, a minha ideia foi iniciar o projeto já desenhando um esboço das regras de negócio, algo simples, mas que considero essencial para definir as fronteiras da solução e ajudar na hora de tomar algumas decisões.

Eu não quis simplesmente fazer um sistema para atender ao requisito de usar o *MVC*. Eu quis construir um produto que fosse fácil de expandir, desacoplar e escalar. Essa necessidade me levou a trazer também a *Clean Architecture* para dentro da solução. O desenho de arquitetura foi muito importante para validar essa intenção, e ficou claro para mim que uma arquitetura não anula a outra, na verdade, elas podem se complementar e funcionar juntas.

Uma das questões que mais pensei durante o projeto foi, "Será que não estou complicando demais algo que poderia ser mais simples?" Mas sempre me baseei no fato de que, muitas vezes, sistemas são criados sem pensar no futuro, e acabam virando legado ainda durante o desenvolvimento da primeira entrega.

Por isso, decidi trazer uma estrutura mínima de qualidade e arquitetura, mesmo sabendo que era algo mais "pesado" do que o desafio pedia. Quis mostrar que algumas decisões que são tomadas no começo determinam se o projeto vai ser fácil ou difícil de manter, se vai ser barato ou caro para evoluir, e se vai atender bem ou não conforme o sistema cresce.

O projeto finalizado me trouxe uma sensação muito boa, porque consegui mostrar que, mesmo que o desafio tenha proposto algo relativamente simples, dá sim para criar aplicações bem estruturadas desde o início. Todas as funcionalidades ficaram bem descritas, tanto na parte de regra de negócio quanto na parte técnica.  
O resultado foi uma aplicação fácil de ser expandida, corrigida e que está preparada para crescer sem virar um problema.

Esse desafio me ajudou a reforçar ainda mais a importância de pensar na arquitetura desde o início, que sim precisamos ter um mínimo codificado hoje em dia, para atender os requisitos não funcionais, que podem parecer invisíveis, mas estão atrelados diretamente com os requisitos funcionais do sistema, a experiência de utilização também depende uma boa estrutura do projeto. Percebi também o quanto é importante definir bem as regras de negócio e deixa-las bem separadas.

Futuras melhorias são facilmente incorporadas no projeto, desde a criação de novos domínios, integrações com sistemas de terceiros. Mas como melhoria, eu pensaria em criar testes unitários para garantir todas as regras de negócio, criar testes automatizados também, outro ponto é que podemos utilizar outras tecnologias para fazer consulta na base de dados, um repositório somente leitura, utilizando algo ainda mais performático para realizar as consultas e deixar o *EntityFrameworkCore* apenas para as operações de criação, alteração e exclusão