

**Projeto Aplicado – Relatório Final**

<b>Nome do Aluno</b>	<b>José Rocha Gravito Martins</b>
<b>Título do Trabalho</b>	Modelando um ecossistema de micro serviços bancário
<b>Curso</b>	MBA em Arquitetura de Software
<b>Linha de Especialização</b>	Arquiteturas de sistemas distribuídos escaláveis
<b>Orientador</b>	<b>Antônio George Saraiva</b>
<b>Data</b>	<b>18/07/2019</b>

## INTRODUÇÃO

### 1. Apresentação do Desafio e da Solução:

- a. Setor do mercado e a justificativa de tal seleção;
- b. Características e restrições de escopo do Desafio;
- c. Oportunidade vislumbrada que motivou o desenvolvimento da Solução.

O desafio se dá em uma instituição financeira focada no mercado corporativo de investimentos. Atualmente esta oferece serviços para seus clientes somente de maneira personalizada, com a necessidade do contato presencial de seus clientes com os respectivos gestores de conta. A organização possui uma infra estrutura própria, que contém diversos sistemas legados e que ainda são mantidos, porém, visando uma modernização de seu parque tecnológico, a mesma conta com suporte de serviços da nuvem Azure, facilitando a criação de novos componentes neste contexto. Devido a constante demanda por agilidade e estreitamento nas comunicações com seus clientes, a instituição pretende alcançar um novo nicho de varejo, oferecendo serviços bancários para pessoas físicas através da disponibilização de um novo canal digital.

### 2. Identificação da(s) pessoa(s) envolvida(s) no desafio:

Os impactados pela solução serão, principalmente, os clientes da instituição que contaram com a possibilidade de um novo canal para estreitamento das comunicações e operações diárias. Além do mais, os gerentes de contas também serão impactados, uma vez que, poderão otimizar seu tempo e tornar mais eficiente sua comunicação através de um novo canal. Neste novo cenário profissionais de auditoria poderão contar com uma nova plataforma para extração de dados, incrementando a segurança sobre as operações da instituição junto a órgãos regulamentadores. As possibilidades também se estendem a gerentes de produto que se interessem, eventualmente, em explorar as possibilidades deste novo canal de comunicação na criação e integração de novos produtos bancários.

**Persona:** Pedro (Cliente)



**Pedro**

**Gerente de produção industrial**

**Empresa:** Indústria automotiva

**Idade:** 48 anos

**Educação:** Mestrado

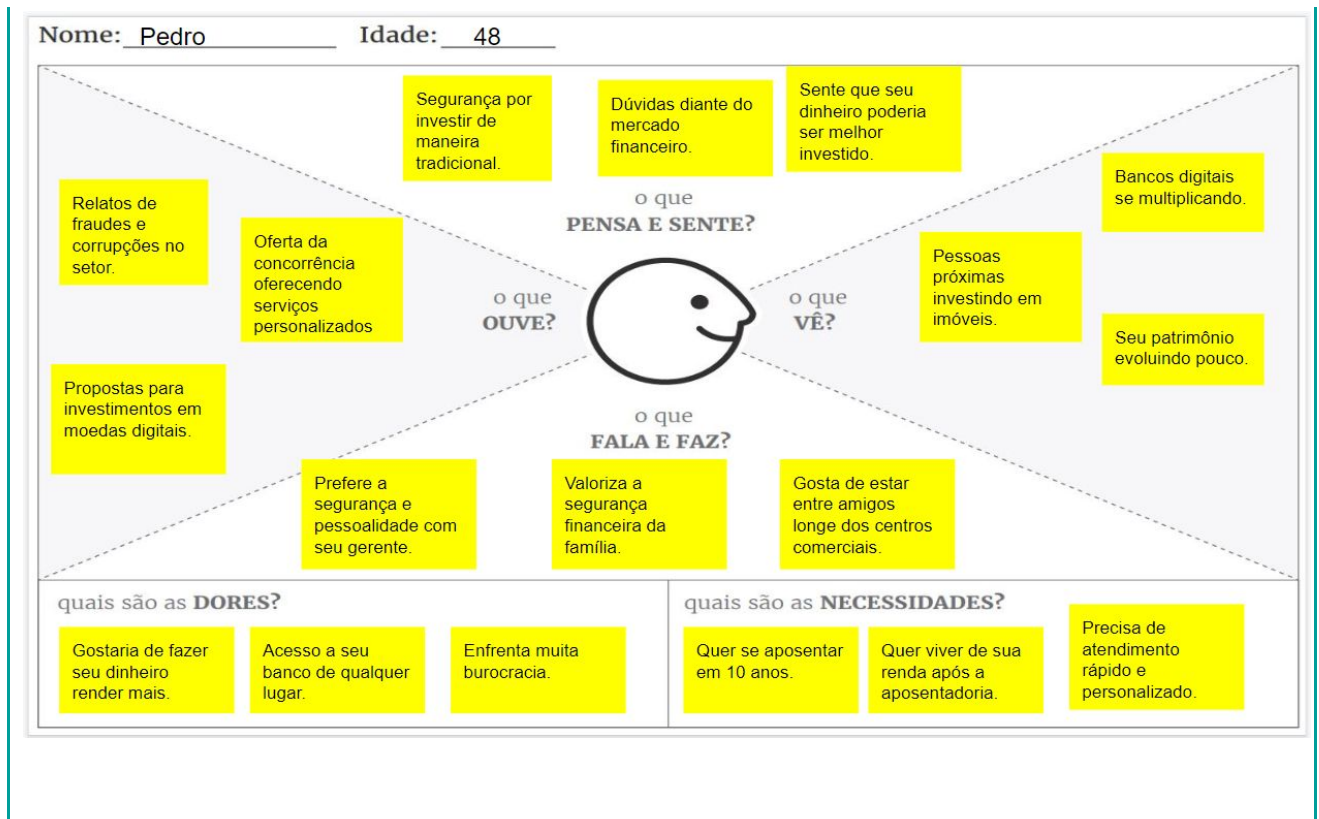
**Mídias:** Lê a jornais e revistas, e recentemente usa o whatsapp para comunicação com amigos e colaboradores.

**Objetivos:** Deseja manter seu capital e fazer com ele renda um pouco por já estar se planejando para sua aposentadoria. Quer uma orientação para fazer seus investimentos de forma conservadora porém, um pouco mais arrojado do que métodos tradicionais como a poupança.

**Desafios:** Não conhece do mercado financeiro e não quer correr riscos com seu capital. Quer tentar entender melhor de investimentos para tentar potencializar um pouco mais seus investimentos.

**Como minha empresa pode ajudá-lo:** A empresa pode gerir os recursos do cliente de forma responsável, fornecer produtos de forma personalizada além de permitir um acompanhamento rápido do progresso de seus investimentos.

**Mapa de empatia:**



### 3. Construção da proposta de solução:

- Requisitos da construção do protótipo/MVP, com descrição do experimento e as métricas de validação;
- Indicadores econômico-financeiros do projeto.

Com o intuito de testar a hipótese de geração de valor da solução, o produto mínimo viável será a disponibilização de serviços para a abertura de contas através de uma infra estrutura de nuvem bem como uma “landing page” disponível na internet informando sobre o início das operações por canal digital para os potenciais clientes.

Com este protótipo será possível acompanhar como a aplicação se comporta de forma descentralizada, fora da infraestrutura local e medir a performance em comparação aos produtos legados disponíveis atualmente bem como a quantidade de clientes interessados.

As principais métricas a serem observadas serão a quantidade de requisições por segundo, o tempo de indisponibilidade dos serviços e se a adesão será suficiente para que a operação seja sustentável.

O modelo de nuvem favorece o uso do protótipo por não obrigar o investimento em infraestrutura além do necessário para conclusão do experimento.

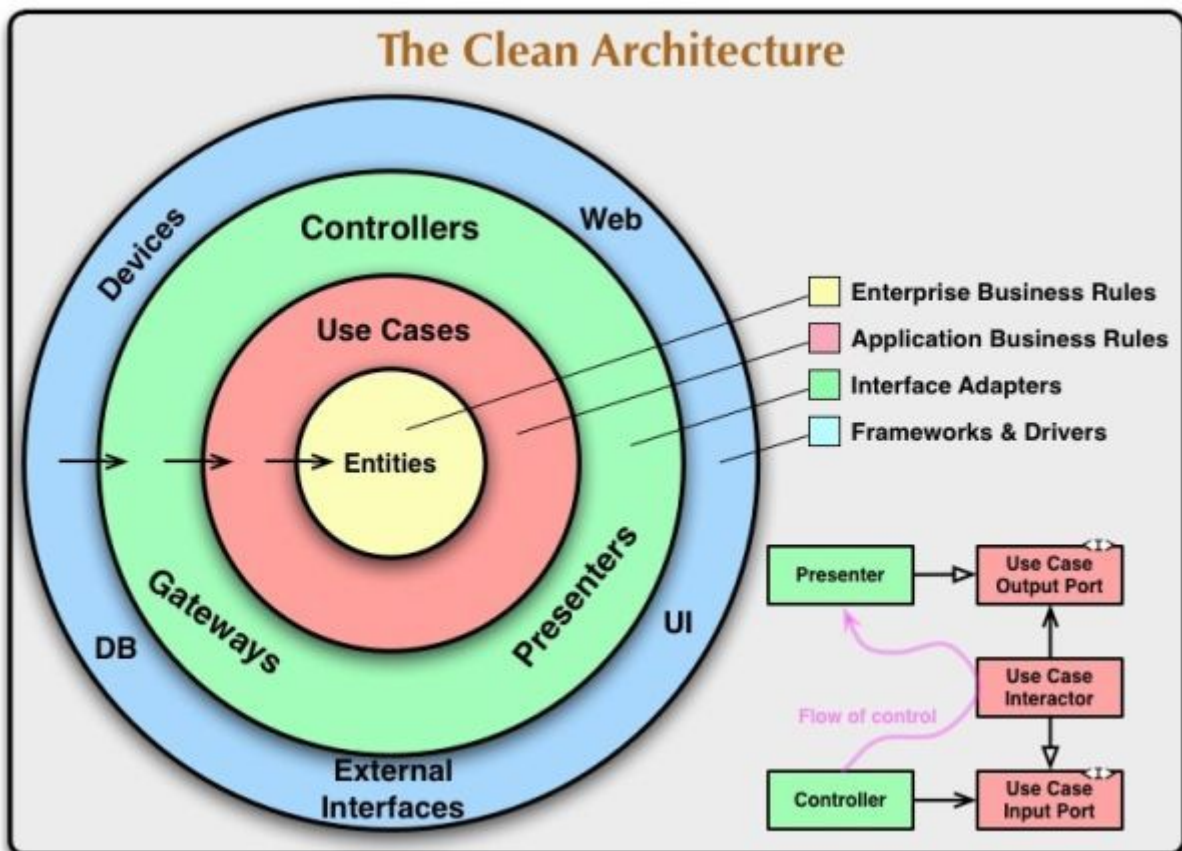
## **DESCRIÇÃO DO DESENVOLVIMENTO DO TRABALHO**

### **4. Definição da arquitetura dos serviços:**

- a. Modelagem da arquitetura dos serviços;
- b. Modelagem da infra-estrutura.

Com o intuito de criar uma solução que permita uma evolução ágil, entrega e integração contínua com menor impacto e que seja criada nativamente para a nuvem, a arquitetura foi baseada em microsserviços. Nesse modelo cada serviço agrupa e encapsula funcionalidades que são inter relacionadas por responsabilidade. Desta forma condicionamos as mudanças a serem executadas em seus contextos próprios e minimizamos o impacto gerado por alguma eventual alteração nos requisitos.

Visando uma arquitetura limpa, independente de framework, independente de interface de usuário e facilmente testável, a estrutura interna dos serviços será codificada baseada no paradigma do “Clean Architecture”, modelo criado por Robert Cecil Martin (“Uncle Bob”) e meados de 2012. Este modelo foi escolhido por se encaixar perfeitamente na nossa necessidade de criar serviços distribuídos, independentes de tecnologia e interface.



Acima o diagrama mostra as quatro camadas dessa arquitetura e sua principal regra: A de regra de dependência (representada pelas setas tracejadas). Essa regra indica que a dependência só pode acontecer de fora pra dentro, ou seja, camadas mais internas não devem ter qualquer dependência com camadas mais externas. As responsabilidades de cada camada são:

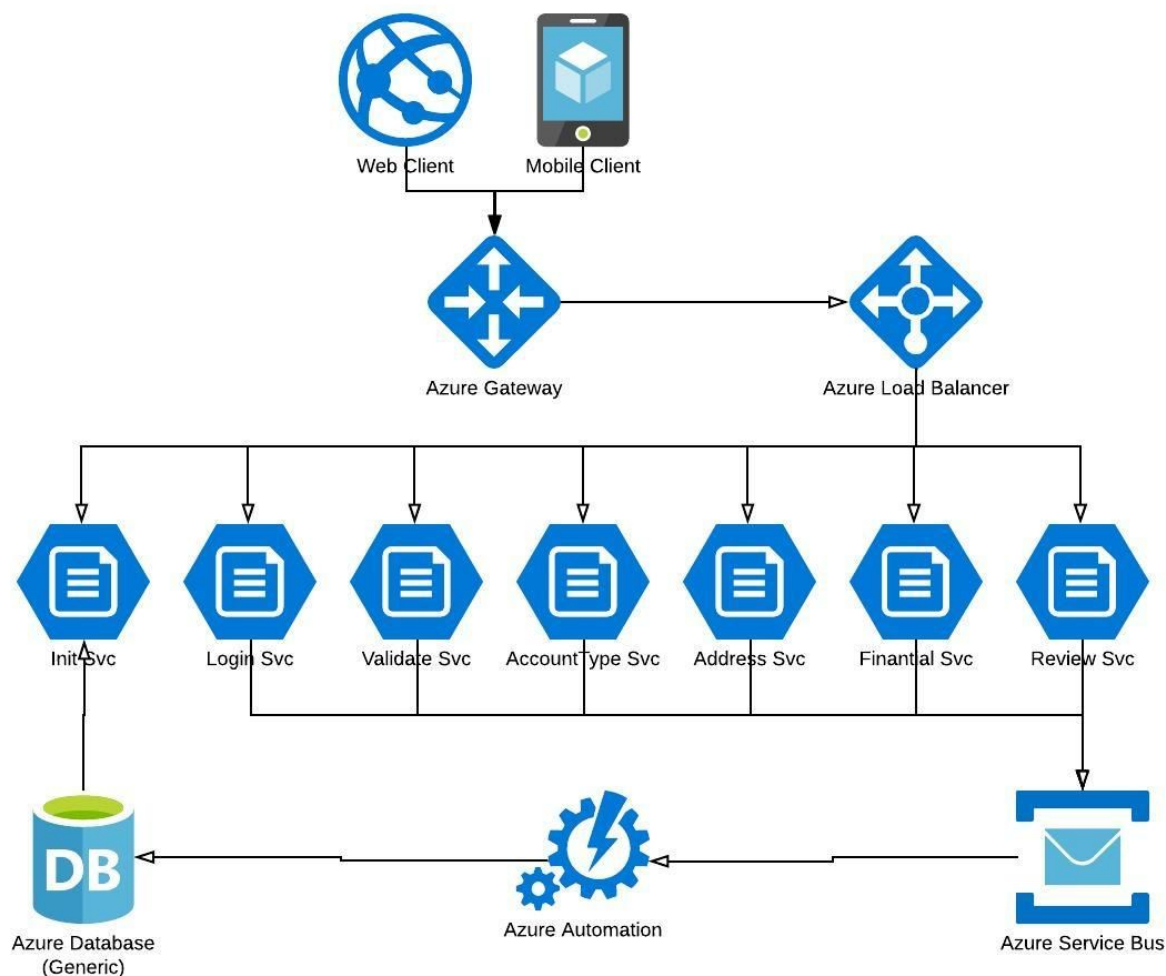
**Entities:** Camada que encapsula as entidades e regras de negócio de maneira abstrata.

**Use cases:** Esta camada contém as regras de negócio mais específicas ao que a aplicação se dispõe a entregar.

**Interface Adapters:** Tem como responsabilidade converter dados da maneira mais conveniente para as camadas Entities e Use Cases, esta camada tem esse nome por fazer extenso uso do “pattern” Adapter (GoF).

**Frameworks:** Essa camada tem a responsabilidade de encapsular ferramentas como banco de dados, comunicação com outros serviços, interface de usuários e etc.

## Diagrama de infra estrutura.



A infraestrutura foi criada usando serviços Azure disponíveis para a instituição, as setas no diagrama representam o fluxo da informação na infraestrutura. Os clientes, que podem ser móveis ou web, fazem requisições para um gateway que, por sua vez, redireciona a requisição para o serviço correspondente. Antes de chegar ao serviço a requisição é interceptada por um load balancer que controla qual instância de serviço será acionada. Como o processo de abertura de contas é baseado em passos de cadastro e verificação, cada passo do processo foi separado em um serviço que pode ser escalado horizontalmente conforme a demanda. As requisições de escrita Não são enviadas para o banco diretamente, mas sim, para uma fila, que conta com um serviço de automação para sincronismo com o banco de dados permitindo uma escrita assíncrona e escalável.

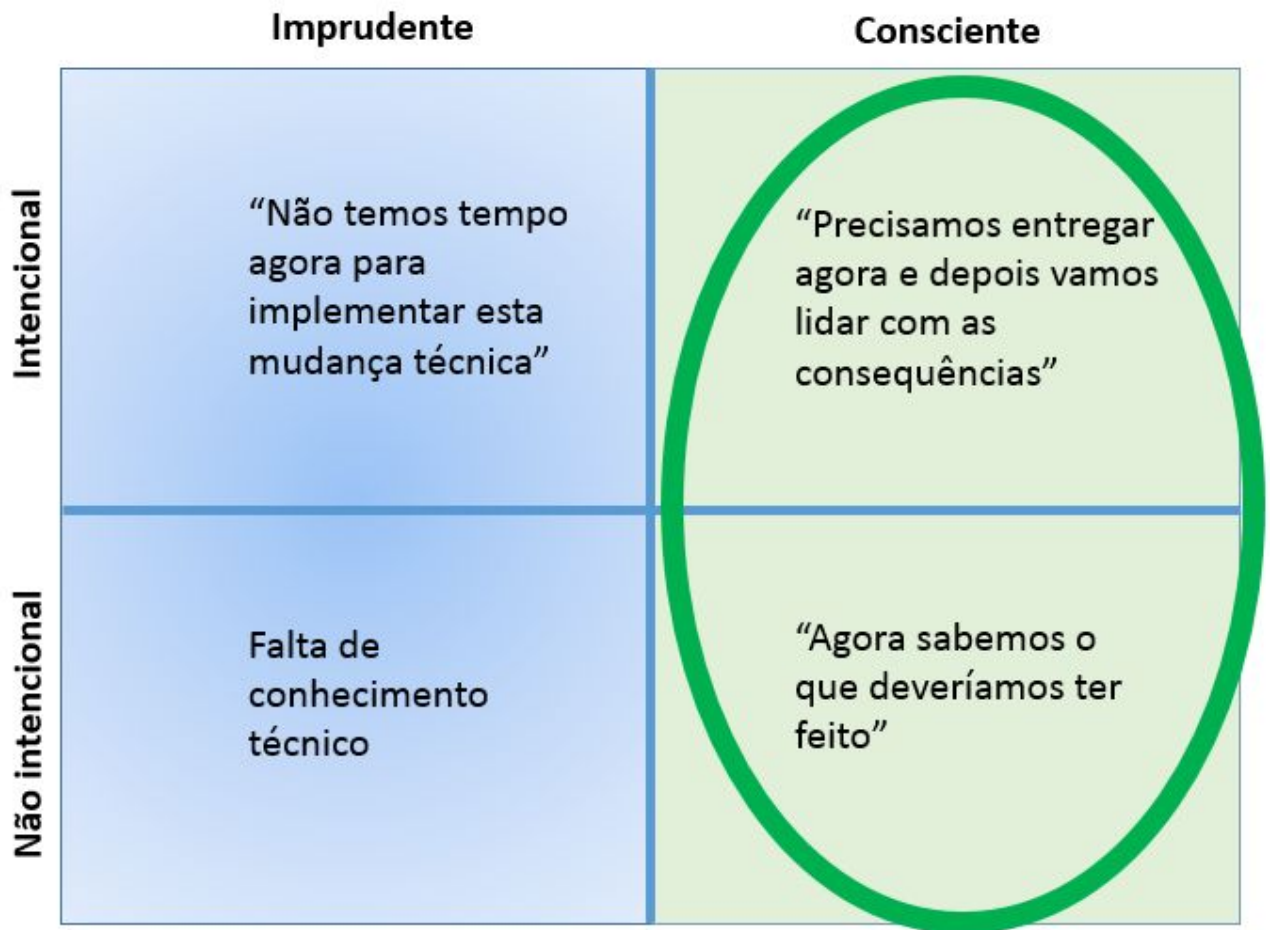
## 5. Definição de padrões de projeto e gestão de débito técnico:

- a. Apresentação da descrição aplicada de padrões de projeto utilizados;
- b. Apresentação do PoC de cenários relevantes.

A arquitetura foi modelada para uso principalmente do padrão de projeto conhecido como “Event Sourcing”, descrito por Martin Fowler neste artigo (<https://martinfowler.com/eaDev/EventSourcing.html>). Com este padrão não só o estado atual do cadastro é importante, para uma eventual continuação por exemplo, mas também a forma de como ele chegou naquele estado. Funciona assim, cada requisição gera um evento que é capturado e registrado em um tópico ou fila mantendo o histórico de como o objeto chegou aquele estado, posteriormente um agente de sincronismo faz a escrita dos dados de forma sequencial no banco de dados que armazena o estado consolidado do objeto para consulta. Este padrão oferece um “tradeoff” de consistência e atualização dos dados em prol da escalabilidade, uma vez que todo o processo ocorre de forma assíncrona permitindo que a carga seja controlada separadamente conforme a demanda de cada componente.

Sabe-se que ao longo do desenvolvimento da solução mudanças na arquitetura podem acontecer como por exemplo, a necessidade do uso de um cache para que os dados consolidados sejam mais rapidamente acessados, ou que um micro serviço ainda pode ser dividido em sua modelagem. Essas e outras demandas serão consideradas débitos técnicos e para serem melhores gerenciadas, precisam ser classificadas. Para isso foi usado o seguinte quadrante:





Cada débito pode se encaixado em um dos quadrantes acima que significam:

Imprudente e Intencional: *“Sabe-se do problema, não será resolvido”*

Imprudente e Não Intencional: *“Trabalhar com uma nova linguagem de programação”.*

Consciente e Intencional: *“Dado nosso deadline, vamos entregar com o problema e depois corrigir o problema”.*

Consciente e Não Intencional: *“Agora que entregamos o projeto, deveríamos ter trabalhado utilizando Métodos Ágeis”.*

#### 6. Definição de processamento distribuído:

- Apresentação das definições de processamento distribuído ou paralelo;
- Apresentação do PoC para validação da arquitetura.

Segundo Tanenbaum, um sistema distribuído é “um conjunto de computadores independentes entre si que se apresenta a seus usuários como um sistema único e coerente” ou seja, uma coleção de computadores autônomos interconectados por uma rede, com software projetado para produzir uma aplicação integrada. Logo este projeto pode ser considerado uma aplicação distribuída ao considerando-se este conceito, uma vez que os componentes que compõem a aplicação como um todo podem estar distribuídos ao longo de uma rede ou mesmo geograficamente separados.

Um pequeno serviço REST deverá ser criado para validar tanto a arquitetura do software usando o “Clean Architecture”, quanto a linguagem de programação a ser escolhida. Será avaliado como este serviço se comporta quanto a quantidade de requisições, se a linguagem é compatível com a nuvem e se é viável de se manter perante um requisito de mudança.

#### 7. Definição de requisitos não funcionais:

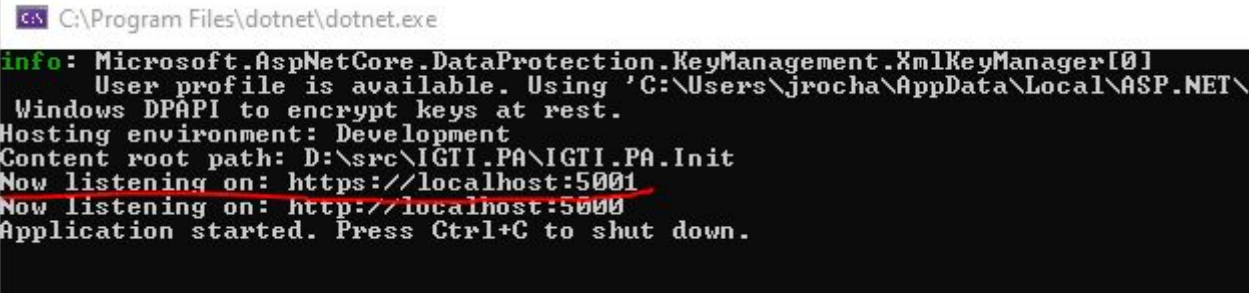
- a. Descrição dos principais requisitos não funcionais da solução;
- b. Apresentação do PoC para validação de RnFs críticos.

Requisitos não funcionais não devem ser negligenciados porém podem elevar ao extremo o custo se não forem elencados por priorização uma vez que, na visão de todo gestor, todos os requisitos não funcionais são importantes. Como o sistema tem como usuário o cliente final e o principal objetivo é a geração de “leads” para o negócio, os requisitos não funcionais mais salientados foram o da categoria de performance, confiabilidade e segurança. Sendo eles:

- (1) Performance: Cada requisição devem ter no máximo 300ms de timeout.
- (2) Performance: Os serviços devem responder em média a 4 requisições por seg.
- (3) Disponibilidade: Cada serviço deve estar disponível, 80% do tempo.
- (4) Segurança: Todo tráfego deve ser feito por conexão criptografada.
- (5) Manutenção: O desenvolvimento deve ser feito em linguagem que demande menor curva de aprendizado para o time.

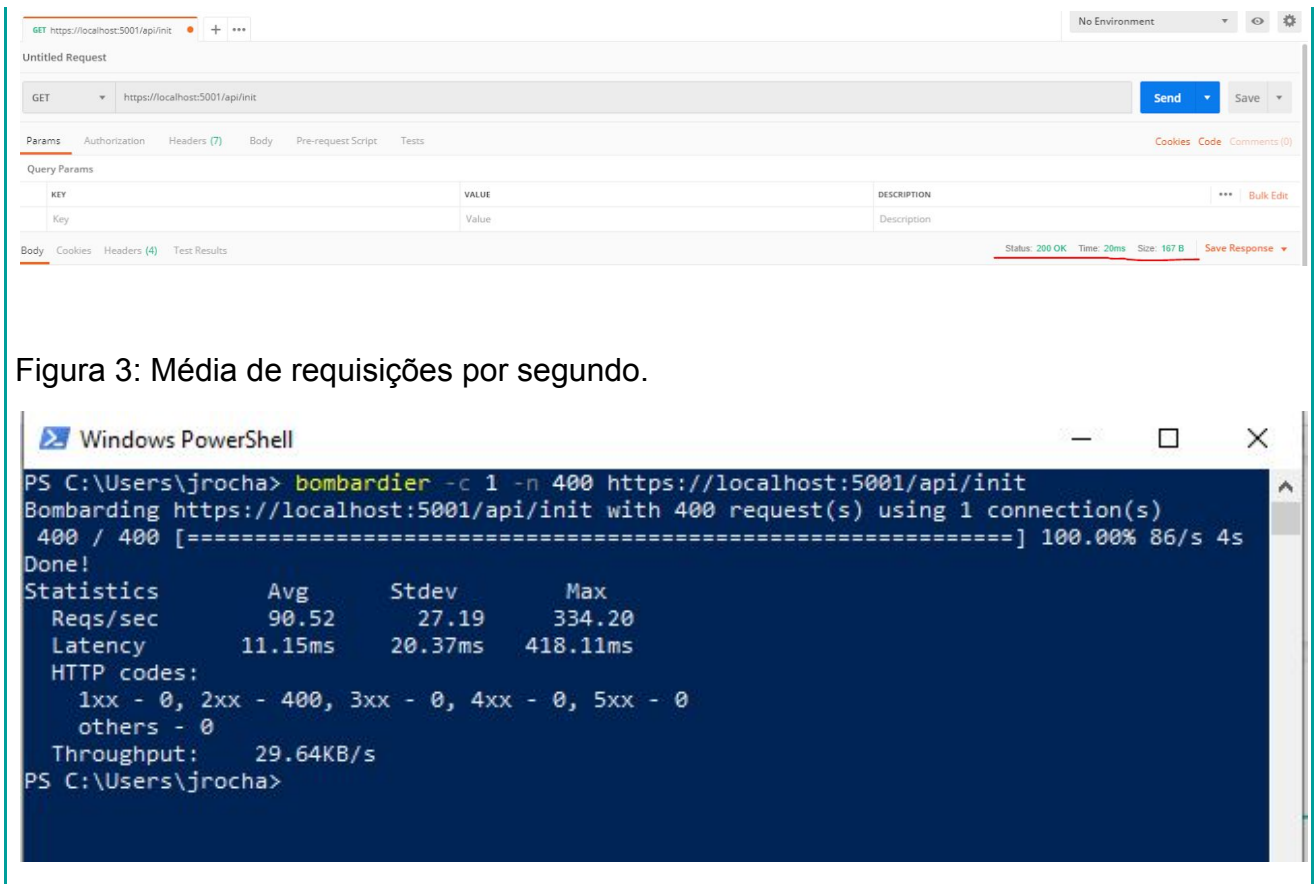
Para validar os requisitos não funcionais uma API com o framework ASP.NET e linguagem C# foi criada, considerando-se que esta é a linguagem de proficiência do time. Neste framework com poucas linhas de código foi possível aceitar conexões seguras conforme a figura 1. Através da ferramenta “postman” foi possível validar que a POC atende ao RnF1 conforme a figura 2. Para validar o RnF2 a ferramenta Bombardier (<https://github.com/codesenberg/bombardier>), que dispara multiplas requisições para um alvo, foi usada no envio de 400 requisições tendo como resultado uma média de 90 requisições por segundo, que mesmo em ambiente de desenvolvimento está bem acima do requisito, conforme figura 3. Para o RnF3, o requisito será atingido na infraestrutura em nuvem que irá permitir a redundância de servidores de aplicação controlados por um componente de “load balancer”.

Figura 1: Aplicação aceitando requisições seguras.



```
C:\Program Files\dotnet\dotnet.exe
info: Microsoft.AspNetCore.DataProtection.KeyManagement.XmlKeyManager[0]
      User profile is available. Using 'C:\Users\jrocha\AppData\Local\ASP.NET\
      Windows DPAPI to encrypt keys at rest.
Hosting environment: Development
Content root path: D:\src\IGTI.PA\IGTI.PA.Init
Now listening on: https://localhost:5001
Now listening on: http://localhost:5000
Application started. Press Ctrl+C to shut down.
```

Figura 2: POC respondendo a 20ms



## 8. Definição de APIs:

- Modelagem das APIs;
- Definição da linguagem, frameworks e soluções;
- Apresentação do PoC da solução.

Como resultado do processo de desenho da solução algumas APIs foram elencadas para compor o projeto, sendo elas:

**POST Register:** Endpoint que irá receber o CPF dos interessados, o código de status http da resposta servirá para verificar se o usuário está iniciando ou continuando um cadastro, (200) para quem está continuando e (201) para quem já iniciou o cadastro. No caso de uma continuação o cliente o cliente irá receber via sms ou email um código de autenticação que deve ser validado.

**POST Login:** Neste endpoint o usuário irá informar seus dados para contato como nome, e-mail e celular. A partir disso, uma mensagem contendo um código de autenticação será disparada para estes contatos, para que o cliente possa prosseguir com o cadastro.

POST Validate: Aqui o usuário irá enviar o código de autenticação recebido, caso seja validado ele receberá um token que será usado para as próximas operações.

POST AccountType: Neste endpoint o usuário poderá escolher qual tipo de conta ele deseja abrir, Corrente, Poupança ou Investimento.

POST Address: Permitirá o registro do endereço residencial do cliente.

POST Financial: Os dados financeiros como patrimônio e conta corrente atual serão adicionados por este endpoint.

GET Review: Aqui o usuário poderá verificar quais de seus dados foram consolidados até o momento, além dos termos de uso que ele deverá aceitar.

POST Review: Este endpoint registra que o usuário aceitou os termos de uso e que os dados podem ser enviados para análise. Este endpoint contém um mecanismo importante, pois o termo de uso deve ser lido pelo menos uma vez, pelo verbo GET, para que possa ser aceito, sendo assim, além do token do cliente um código auto gerado de termo, que foi enviado juntamente com o método GET, deve ser verificado aqui para que a operação se concretize.

O experimento foi criado usando a linguagem C# que é um dos requisitos não funcionais do projeto uma vez que é a linguagem de proficiência da equipe. Em conjunto com a linguagem, que tem a Microsoft como mantenedora, para auxiliar no processo de criação foi usado o framework ASP.NET Core que é uma tecnologia relativamente nova lançada pela Microsoft. O uso deste foi justificado por ser open source, multiplataforma e destinado nativamente para a nuvem, além de proporcionar uma curva de aprendizado mais curta para um time que teve contato com tecnologias Microsoft anteriores.

Poucos passos foram necessários para criar o primeira versão de uma API como prova de conceito. São necessários literalmente quatro passos para ter uma aplicação rodando em ambiente local, sendo eles:

1: Fazer o download e instalar o SDK do .NET pelo endereço <https://dotnet.microsoft.com/download>

2: Criar uma pasta com o nome de uma aplicação desejada.

Em seguida acessar a linha de comando do sistema operacional escolhido, navegar até a pasta criada e executar os seguintes comandos:

3: dotnet new webapi

4: dotnet run

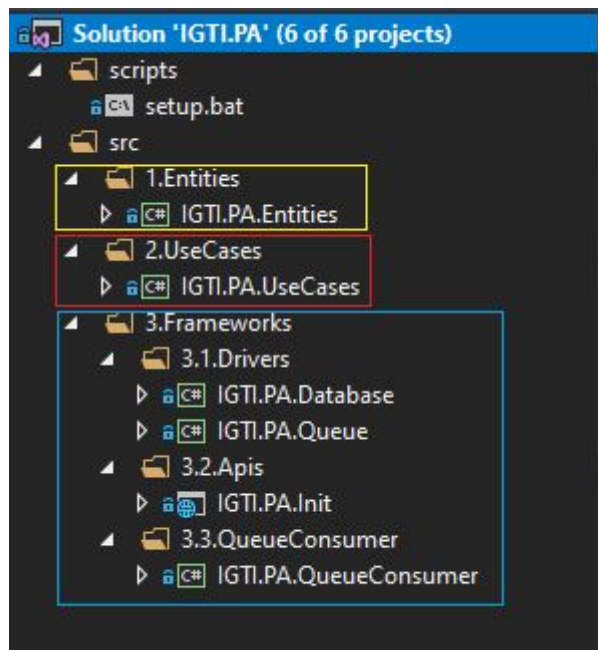
E uma aplicação simples já está respondendo na máquina local.

## DESCRIÇÃO DO TRABALHO FINAL

### 9. Solução arquitetural:

- a. Mapeamento e modelagem;
- b. Provas de conceito.

A solução arquitetural segue o conceito mencionado de “Clean Architecture” com alguns desvios de nomenclatura e camadas uma vez que o framework abstrai boa parte da complexidade de implementação. A imagem abaixo mostra a prova de conceito inicial relacionando alguns pontos da arquitetura.



Onde:

1. Entidades: Biblioteca de classes que representam as capacidades do negócio.
2. UseCases: Biblioteca de classes que representam as capacidades da aplicação.
3. Frameworks: Projetos menos abstratos que contém detalhes de implementação dos bancos de dados, recebimentos de informação por API, regras de comunicação com a filas e do agente de sincronismo com o banco de dados consumidor do service bus.

O código fonte da versão final se encontra no endereço <https://github.com/jrochamartins/igti-ars> hospedado no GitHub. Um ponto interessante é que para simular o ambiente produtivo, o componente Azure Service Bus foi substituído pelo produto “RabbitMQ” rodando em versão local no docker, assim como o componente de banco de dados do Azure foi substituído pelo banco MongoDB hospedado localmente também no docker.

#### 10. Principais requisitos funcionais considerados:

- a. Definição;
- b. Validação.

No desenvolvimento da solução foram considerados os seguintes requisitos funcionais:

- 1: O sistema deve permitir que uma aplicação do cliente inicie o cadastro do cliente pelo seu CPF, caso o cliente já existe este deve poder continuar de onde parou.
- 2: O sistema deve oferecer um endpoint que permita a aplicação do cliente informe nome e informações de contato.
3. O sistema deve oferecer um endpoint que valide o código de acesso recebido pelo cliente. No caso de sucesso deve retornar em qual etapa o cliente se encontra.
4. O sistema deve oferecer um endpoint que permita que o cliente informe qual tipo de conta ele deseja criar.
5. O sistema deve oferecer um endpoint que permita que o cliente informe qual é seu endereço residencial.
6. O sistema deve oferecer um serviço que permita que o cliente informe seus dados financeiros para abertura de conta.
7. O sistema deve oferecer um serviço que mostre o resumo do seu cadastro juntamente com os termos de uso da instituição além de um código identificador do termo no momento de visualização.
8. O sistema deve oferecer um serviço que registre o aceite dos termos de uso da instituição validando seu código de visualização além de enviar os dados para análise interna.



## 11. Principais requisitos não funcionais considerados:

- a. Definição;
- b. Validação.

Os requisitos não funcionais levados em consideração na elaboração da solução foram enumerado na lista abaixo por ordem de prioridade para o negócio em contraste com os custos gerados na implementação. Normalmente alguns dos requisitos não funcionais existentes na prova de conceito deixaram de existir, porém, alteraram sua relevância e prioridade ao longo do desenvolvimento após um entendimento mais profundo das necessidades da instituição.

1. Segurança: A privacidade do cliente se tornou o requisito mais importante em todo o processo, por envolver o tráfego de dados sensíveis. Para garantir isso o fluxo foi alterado para que nenhum dado do cliente fosse obtido antes do final do cadastro e mesmo assim somente quando fosse estritamente necessários para o acompanhamento.
2. Segurança: Todo tráfego deve ser feito por conexão criptografada. Este objetivo foi atingido fazendo uso da conexão HTTPS.
3. Performance: Os serviços devem responder em média a 4 requisições por segundo. Este requisito levou a otimização dos algoritmos usados internamente no sistema e na pesquisa e comparação de ferramenta, um exemplo foi o uso do MongoDB ao invés do SQL Server.
4. Disponibilidade: Cada serviço deve estar disponível, 80% do tempo. Para atingir este requisito cada serviço foi quebrado para que pudesse ser escalado conforme a demanda, além de permitir a aplicação mais facilitada de meios de redundância.
5. Manutenibilidade: O desenvolvimento deve ser feito em linguagem que demande menor curva de aprendizado para a equipe. Além do uso de framework ASP.NET que já era íntimo a equipe, a quebra da aplicação em contextos menores facilitou a obtenção deste requisito uma vez que foi visto que o entendimento ficou mais claro deixando a aplicação mais enxuta como um todo.



## RESULTADOS

### 12. Descrição e análise dos resultados alcançados:

- a. Resultados positivos encontrados (caso existam). Explique;
- b. Resultados negativos encontrados (caso existam). Explique.

A implementação da solução gerou diversos resultados positivos, mas o primordial deles foi a possibilidade de possuir um caso de referência interna da aplicação de uma arquitetura distribuída em nuvem em um cenário em que a segurança sempre foi fator decisivo na escolha de métodos de desenvolvimento. Usar a nuvem em uma instituição financeira nunca foi bem visto pelas áreas de segurança e compliance. Outro fator que pode ser citado foi que com a terceirização do uso de recursos de TI para a nuvem, foi que a empresa pode economizar alguns custos com a aquisição de servidores em um cenário de baixa previsibilidade e sazonalidade. A modernização da arquitetura do software produzido possibilita um desenvolvimento mais ágil pelo time e bugs neste cenário podem ser corrigidos em tempo recorde em relação aos dispendiosos processos de GMUD. O emprego do componente Azure Service Bus faz com que a operação de abertura de contas se torne bem mais resiliente, otimizando a geração de leads e retorno para o cliente, consequentemente aumentando seu nível de satisfação.

Um ponto negativo gerado foi que agora com o cenário distribuído a equipe de operações e infraestrutura possui um número maior de recursos para monitorar e gerenciar, neste cenário é imprescindível a aplicação de processos automatizados e uma constante conscientização para uma mentalidade orientada ao DevOps. Outro ponto de atenção é que essa modernização também deve ser estender para a área de segurança que agora precisa agilizar mais ainda seus processos de verificação de conformidade de código pois, em um cenário de microsserviços as publicações e testes precisam acontecer de forma ainda mais ágil.

## CONCLUSÃO

### 13. Apresentação da conclusão:

- a. Principais contribuições que seu projeto gera aos envolvidos;
- b. Inovações, particularidades ou vantagens que o projeto/resultado possui em relação a similares;
- c. Limitações do projeto;
- d. Próximos passos necessários para que o projeto evolua/se desenvolva.

A solução traz dentre os benefícios o de proporcionar uma experiência mais resiliente e confiável para o cliente final assim gerando uma melhor conversão para a instituição. O processo de aplicação de micro serviços permite que melhorias sejam distribuídas ainda mais rápido permitindo que hipóteses sejam testadas com mais facilidade.

A inovação aqui foi a surpreendente adequação do “stack” Microsoft no desenvolvimento do projeto, a linguagem C# em conjunto com o novo framework ASP.NET Core supreenderam na performance e produtividade.

Internamente o “Clean Architecture” permite a evolução constante da base código e principalmente, sem depender da tecnologia escolhida, permitido com que os times escolham a melhor forma de trabalho além de introduzir uma curva de aprendizado mais curta para eventuais novos integrantes da equipe.

O projeto se limitou a construção apenas de um “backend” escalável e enxuto sem se preocupar com a construção de um “frontend” específico, de forma que profissionais de “UX” possam aproveitar desse arcabouço para testar suas hipóteses de aplicação da melhor forma possível.

O projeto usa como exemplo um contexto bem definido do processo de abertura de contas na instituição principalmente por ter suas fronteiras muito bem estabelecidas, porém é um caso de uso de referência e pode ter seu conceito expandido para todo o fluxo de negócios da organização. Todas as áreas podem se beneficiar dos conceitos trabalhados aqui como por exemplo, o “frontend” por aplicar os conceitos de “Clean Architecture” na criação de suas visões e criar camadas ainda mais desacopladas e coesas, os processos internos que oferecem serviços bancários podem ser desacoplados

em aplicações menores e ainda mais resilientes com o uso de um service bus. Enfim, existe um longo caminho a ser percorrido, porém as tecnologias aplicadas aqui podem auxiliar para que esse caminho seja ainda mais fácil de ser transposto.