**PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS**

**PUC Minas Virtual**

**Pós-graduação *Lato Sensu* em Arquitetura de *Software* Distribuído**

Projeto Integrado

Relatório Técnico

My-Customers - Sistema para Gestão de Clientes em Bancos

Cleidson Ferreira de Jesus

Belo Horizonte

Abril 2022.

# Projeto Integrado – Arquitetura de Software Distribuído

**Sumário**

[Projeto Integrado – Arquitetura de Software Distribuído 2](#_Toc80562699)

[1. Introdução 3](#_Toc80562700)

[2. Cronograma do Trabalho 4](#_Toc80562701)

[3. Especificação Arquitetural da solução 5](#_Toc80562702)

[3.1 Restrições Arquiteturais 5](#_Toc80562703)

[3.2 Requisitos Funcionais 5](#_Toc80562704)

[3.3 Requisitos Não-funcionais 7](#_Toc80562705)

[3.4 Mecanismos Arquiteturais 7](#_Toc80562706)

[4. Modelagem Arquitetural 8](#_Toc80562707)

[4.1 Diagrama de Contexto 8](#_Toc80562708)

[4.2 Diagrama de Container 9](#_Toc80562709)

[4.3 Diagrama de Componentes 9](#_Toc80562710)

[5. Prova de Conceito (PoC) 11](#_Toc80562711)

[5.1 Integrações entre Componentes 11](#_Toc80562712)

[5.2 Código da Aplicação 11](#_Toc80562713)

[6. Avaliação da Arquitetura (ATAM) 13](#_Toc80562714)

[6.1. Análise das abordagens arquiteturais 13](#_Toc80562715)

[6.2. Cenários 13](#_Toc80562716)

[6.3. Evidências da Avaliação 14](#_Toc80562717)

[6.4. Resultados Obtidos 15](#_Toc80562718)

[7. Avaliação Crítica dos Resultados 16](#_Toc80562719)

[8. Conclusão 17](#_Toc80562720)

[Referências 9](#_Toc80562721)

## Introdução

O sistema financeiro nacional tem status constitucional e possui vital importância para o desenvolvimento econômico do país, os bancos promovem a logística complexa e arriscada das intermediações financeiras, atuando em extremos, desde transações milionárias em financiamento de grandes empresas até o cidadão comum, proporcionando acesso a serviços financeiros gratuitos através dos bancos digitais.

No ano de 2020 a pandemia da covid-19 impulsionou o celular a tornar-se o canal favorito dos brasileiros para pagar contas, fazer transferências, contratar crédito dentre outras operações bancárias, no mesmo ano, pela primeira vez as transações realizadas nos apps bancários representaram mais da metade (51%) do total das operações feitas no país, as transações feitas pelo celular chegaram a 52.9 bilhões, ante 37 bilhões no ano anterior, se considerados todos os canais bancários (celular, internet, maquininhas, agências, caixas eletrônicos, correspondentes bancários e *contact centers*) o total das operações chega a 103,5 bilhões, um crescimento de 20%.

Bancos tradicionais possuem um enorme legado tecnológico que não é facilmente escalável e adaptável, tecnologias antigas, acoplamento de sistemas e inexistência de padrões arquiteturais dificultam e podem até inviabilizar o suporte ao crescimento apresentado, além disso, é preciso conviver com *fintechs* que já são constituídas digitalmente e não enfrentam esse tipo de desafio.

A modernização do legado tecnológico é essencial para que os bancos tradicionais se mantenham competitivos no mercado, através da modernização é esperado diminuir o tempo gasto com o processo de *onboarding* de clientes, melhorar a estrutura dos dados facilitando a prestação de contas perante a órgãos reguladores, melhor integração entre sistemas diminuindo o acoplamento e facilitando futuras evoluções e por fim, possibilitar a construção de aplicativos intuitivos que evitem a ida de clientes a agências, gerando assim economia de recursos.

Para que os ganhos apresentados sejam alcançados um dos sistemas que deve ser modernizado é de gerenciamento de pessoas, o objetivo deste trabalho é apresentar o projeto arquitetural para construção dessa aplicação visando alcançar três objetivos:

* Construir um sistema que centralize e gerencie dados de clientes, atendendo a regulamentação imposta por órgãos governamentais.
* Projetar uma arquitetura resiliente, escalável e segura para suportar o crescimento da operação no decorrer dos anos.
* Permitir que as áreas de negócio foquem no que realmente importa fornecendo um sistema simples e eficiente.

## Cronograma do Trabalho

**Quadro 1: Cronograma de atividades.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Datas** | | **Atividade / Tarefa** | **Produto / Resultado** |
| **De** | **Até** |
| 07/03/2022 | 07/03/2022 | 1. Desenvolver cronograma do trabalho | Tabela com cronograma do trabalho |
| 08/03/2022 | 09/03/2022 | 1. Redigir introdução do trabalho | Tópico de introdução do trabalho |
| 10/03/2022 | 15/03/2022 | 1. Definir restrições arquiteturais | Lista com restrições arquiteturais identificadas |
| 16/03/2022 | 21/03/2022 | 1. Definir requisitos funcionais | Lista com requisitos funcionais identificados |
| 22/03/2022 | 25/03/2022 | 1. Definir requisitos não-funcionais | Lista com requisitos não-funcionais identificados |
| 28/03/2022 | 30/03/2022 | 1. Definir mecanismos arquiteturais | Lista com mecanismos arquiteturais identificados |
| 31/03/2022 | 04/04/2022 | 1. Elaborar diagrama de contexto | Diagrama de contexto da solução |
| 05/04/2022 | 07/04/2022 | 1. Gravar vídeo de apresentação etapa 1 | Vídeo de apresentação da etapa 1 |
| 08/04/2022 | 11/04/2022 | 1. Construir apresentação em PPT da etapa 1 | Apresentação em Power Point |
| 12/04/2022 | 12/04/2022 | 1. Revisão da etapa 1 | Etapa 1 revisada |
| 13/04/2022 | 13/04/2022 | 1. Publicar no GitHub arquivos produzidos na etapa 1 | Arquivos produzidos disponíveis para acesso público |
| 18/04/2022 | 20/04/2022 | 1. Elaborar diagrama de containers | Diagrama de containers da solução |
| 21/04/2022 | 22/04/2022 | 1. Elaborar diagrama de componentes | Diagrama de componentes da solução |
| 25/04/2022 | 26/04/2022 | 1. Elaborar wireframe da POC | Protótipo do visual do projeto |
| 27/04/2022 | 10/06/2022 | 1. Implementar três requisitos funcionais do total listado | Três requisitos funcionais implementados |
| 13/06/2022 | 13/06/2022 | 1. Revisão da etapa 2 | Etapa 2 revisada |
| 14/06/2022 | 14/06/2022 | 1. Publicar no GitHub arquivos produzidos na etapa 2 | Arquivos produzidos disponíveis para acesso público |
| 20/06/2022 | 23/06/2022 | 1. Redigir tópico Análise das abordagens arquiteturais | Tópico Análise das abordagens arquiteturais |
| 27/06/2022 | 30/06/2022 | 1. Redigir tópico Cenários | Tópico Cenários |
| 04/06/2022 | 11/07/2022 | 1. Redigir tópico Evidências da avaliação | Tópico Evidências da avaliação |
| 12/07/2022 | 18/07/2022 | 1. Redigir tópico Resultados obtidos | Tópico Resultados Obtidos |
| 19/07/2022 | 19/07/2022 | 1. Redigir tópico Avaliação crítica dos resultados | Tópico Avaliação crítica dos resultados |
| 20/06/2022 | 26/07/2022 | 1. Redigir conclusão | Tópico Conclusão |
| 27/07/2022 | 02/08/2022 | 1. Gravar vídeo de apresentação da etapa 3 | Vídeo de apresentação da etapa 3 |
| 03/08/2022 | 03/08/2022 | 1. Revisão da etapa 3 | Etapa 3 revisada |
| 15/08/2022 | 15/08/2022 | 1. Publicar no GitHub arquivos produzidos na etapa 3 | Arquivos produzidos disponíveis para acesso público |

## Especificação Arquitetural da solução

Esta seção apresenta a especificação básica da arquitetura da solução a ser desenvolvida, incluindo diagramas, restrições e requisitos definidos, tal que permitem visualizar a macroarquitetura da solução.

## Restrições Arquiteturais

|  |  |
| --- | --- |
| **ID** | **Descrição** |
| R01 | O back-end deve ser desenvolvido com .Net Core C# |
| R02 | O front-end deve ser desenvolvido com Angular |
| R03 | Pacotes Nuget ou NPM a serem utilizados no projeto devem ser obtidos exclusivamente do repositório interno, a inclusão de novos pacotes no repositório deve ser avaliada pela equipe de segurança |
| R04 | Implementar RabbitMQ como event broker onde necessário a comunicação orientada a eventos |
| R05 | Toda a comunicação de origem externa ao ambiente deve passar por um API Gateway |
| R06 | O registro de log’s deve ser feito utilizando a slack ELK |

## Requisitos Funcionais

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Descrição Resumida** | **Dificuldade (B/M/A)\*** | **Prioridade**  **(B/M/A)\*** |
| RF01 | As implementações efetuadas para este sistema devem atender as orientações estabelecidas pela LGPD. | A | A |
| RF02 | Deve ser possível cadastrar os dados genéricos de clientes | A | A |
| RF03 | Deve ser possível cadastrar os dados específicos de clientes | A | A |
| RF04 | Deve ser possível cadastrar os representantes para Pessoa Física e para Pessoa Jurídica | M | A |
| RF05 | Para clientes menores de idade a inclusão de ao menos um representante é obrigatória | B | M |
| RF06 | Para clientes Pessoa Jurídica é obrigatória a inclusão de ao menos um representante | B | M |
| RF07 | Para clientes analfabetos é obrigatório a inclusão das informações do rogado | B | M |
| RF08 | Deve ser possível cadastrar a composição societária dos clientes Pessoa Jurídica | M | A |
| RF09 | Deve ser possível cadastrar o faturamento dos clientes Pessoa Jurídica | M | A |
| RF10 | O porte dos clientes Pessoa Jurídica deve ser calculado com base no faturamento cadastrado | B | M |
| RF11 | O sistema deve obter e persistir informações de clientes PEP (Politicamente Exposto) | M | A |
| RF12 | O sistema deve obter e persistir a exposição em mídia de clientes | M | M |
| RF13 | O sistema deve obter e persistir o status da situação cadastral de Pessoas Físicas e Jurídicas perante a receita federal | M | M |
| RF14 | O sistema deve iniciar o processo de encerramento de contas para clientes com o status irregular na receita federal | A | A |
| RF15 | O sistema deve reclassificar automaticamente o risco de lavagem de dinheiro dos clientes | A | B |
| RF16 | O sistema deve calcular a data da renovação cadastral dos clientes | A | M |
| RF17 | O sistema deve obter e persistir informações de clientes falecidos | M | M |
| RF18 | O sistema deve obter e persistir a lista de clientes optantes pelo cadastro positivo | M | M |
| RF19 | O sistema deve atualizar automaticamente o porte dos clientes Pessoa Física de acordo com o valor do salário-mínimo vigente | M | B |
| RF20 | O cadastro de clientes originados no onboarding simplificado deve ser completado automaticamente através de consultas em bureaus | A | M |
| RF21 | O sistema deverá gerar o arquivo de dados cadastrais para envio ao E-Financeira | M | A |

\*B=Baixa, M=Média, A=Alta.

## Requisitos Não-funcionais

|  |  |  |
| --- | --- | --- |
| **ID** | **Descrição** | **Prioridade**  **B/M/A** |
| RNF01 | Deve ser registrado trilha de auditoria para todas as alterações que ocorrerem no cadastro de clientes | A |
| RNF02 | As API’s devem possuir um throughput de no mínimo 340 tps | A |
| RNF03 | O sistema deve possuir disponibilidade 24x7x365 | A |
| RNF04 | O front-end deve ser compatível com os navegadores: Chrome, Firefox e Edge | M |
| RNF05 | O front-end deve apresentar características de design que facilitem o uso por PCD’s | A |
| RNF06 | O deploy em produção das aplicações e a infraestrutura devem ser automatizadas usando pipelines CI/CD | M |
| RNF07 | O processamento de grandes quantidades de dados deve ser feito preferencialmente após as 20h | M |

## Mecanismos Arquiteturais

|  |  |  |
| --- | --- | --- |
| **Análise** | ***Design*** | **Implementação** |
| Persistência | ORM | Dapper |
| Persistência | Banco de dados relacional | Mysql |
| Front-end | Single Page Application | Angular |
| Back-end | Arquitetura em camadas | .Net Core |
| Gateway | API Gateway | AWS API Gateway |
| Log | Solução para gestão de logs | ELK |
| Teste de Software | Teste unitários | xUnit |
| Teste de carga | JMeter |
| Versionamento | Versionamento do código das aplicações | GIT |
| Distribuição | Integração e Entrega Continua (CI/CD) | AWS Code Pipelines |
| Barramento de mensagens | Integração entre sistemas através de mensageria | RabbitMQ |
| Documentação de API | Solução para documentação das APIs da solução | Swagger |

## Modelagem Arquitetural

Esta seção apresenta a modelagem arquitetural da solução proposta, de forma a permitir seu completo entendimento visando à implementação da prova de conceito do sistema My-Customers na seção 5.

Para esta modelagem arquitetural optou-se por utilizar o modelo C4 para documentação de arquitetura de software. Mais informações a respeito podem ser encontradas nos links: <https://c4model.com/> e <https://www.infoq.com/br/articles/C4-architecture-model/>. Dos quatro níveis que compõem o modelo C4 três serão apresentados aqui e somente o código será apresentado na próxima seção (5).

## 4.1 Diagrama de Contexto

**Figura 1 - Visão Geral da Solução**

Interface gráfica do usuário

Descrição gerada automaticamente

A Figura 1 mostra o contexto em que a aplicação My-Customers será inserida, demostrando seus principais módulos e integrações.

## 4.2 Diagrama de Container

**Figura 2 – Diagrama de container**

Interface gráfica do usuário

Descrição gerada automaticamente

A Figura 2 apresenta os containers da aplicação e a forma como estão organizados, a arquitetura visa manter somente uma fonte de verdade tratando-se dos dados de clientes, o database apresentado é a fonte de dados oficial do banco perante o Banco Central.

Os demais sistemas recebem novos clientes e atualizações através de mensageria, caso necessário eles armazenam informações pontuais em sua estrutura funcionado assim de forma desacoplada. Além disso, a implantação de novos sistemas fica mais simples, para receber informações de clientes basta que o novo sistema seja plugado na mensageria.

Toda manipulação de dados cadastrais é feita pela Web-Api do My-Customers ou pelos serviços de atualização, isso garante a consistência de dados já que a regra de negócio sempre será respeitada.

A rastreabilidade de alterações cadastrais é garantida pelo LOG, a solução será implementada em uma plataforma permite a pesquisa em segundos da trilha de auditoria, possibilitando rápida resposta em caso de questionamento de órgãos reguladores ou até mesmo em identificação de fraudes.

## 4.3 Diagrama de Componentes

**Figura 3 – Diagrama de componentes**

Diagrama, Desenho técnico, Esquemático

Descrição gerada automaticamente

A Figura 3 apresenta os componentes da aplicação, estes componentes podem ser detalhados como:

* Angular – Plataforma/Framework de desenvolvimento de sistemas WEB.
* MyCustomers.Web – SPA que disponibiliza UI para utilização de usuários internos do banco.
* Fornecedores Externos – Empresas parceiras que disponibilizam informações a serem utilizados pelo banco, mas que não são de seu domínio.
* API Gateway – Ferramenta responsável por orquestrar chamadas de API’s.
* .NET Core – Framework utilizado no desenvolvimento do back-end da solução My-Customers e do robô My-Robot.
* MyCustomers.Api – Projeto responsável pela organização e exposição das rotas da API.
* MyCustomers.Core – Projeto que contém toda a regra de negócio da API.
* MuCustomers.InfraStructure – Projeto responsável por centralizar a comunicação com o banco de dados e a mensageria.
* MyCustomers.Tests – Projeto com os testes automatizados da API.
* MyGenerics.Logging – Pacote responsável pela comunicação com o logstash.
* MyCustomers.Messaging – Projeto responsável por orquestrar a comunicação com o serviço de mensageria.
* Dapper – ORM responsável pela comunicação com o banco de dados.
* MyRobot.Schedule – Projeto responsável por organizar todos os serviços que vão executar no MyRobot.
* MyRobot.Services – Projeto com a regra de negócio dos serviços que serão executados no MyRobot.
* MuRobot.InfraStructure – Projeto responsável por centralizar a comunicação com o banco de dados e a mensageria.
* MyRobot.Tests – Projeto com os testes automatizados do robô.
* MyRobot.Messaging – Projeto responsável por orquestrar a comunicação com o serviço de mensageria.
* MySql – Banco de dados relacional que vai persistir os dados de clientes.
* RabbitMQ – Ferramenta responsável pela mensageria.
* ELK – Junção das ferramentas ElasticSearch, Logstash e Kibana responsáveis pelo armazenamento e consulta de logs.
* Sistemas Internos – Demais sistemas que compõem o ambiente interno do banco.

## Prova de Conceito (PoC)

Para validar algumas implementações propostas para a aplicação foram utilizadas provas de conceitos (POC), três aspectos de alta importância para a arquitetura foram contemplados:

1. **Persistência de dados:** O objetivo dessa POC foi avaliar o micro ORM Dapper a fim de entender sua implementação e funcionamento, buscando comprovar sua *performance* que é considerada uma vantagem em relação a outras bibliotecas semelhantes. Espera-se que com a utilização da biblioteca seja possível obter maior controle na performance de querys;
2. **Design pattern para API:** O objetivo dessa poc foi comprovar que a arquitetura em camadas alinhada a conceitos de DDD (Domain-Driven Design) pode organizar bem o código diante da grande quantidade de regras de negócio que compõem a aplicação. Espera-se que seja possível incluir, retirar ou modificar regras de negócio de forma simples e rápida;
3. **Angular:** O objetivo dessa POC foi estudar a utilização do framework Angular para desenvolvimento do front-end que será utilizado pelo *backoffice* da área de cadastro. Espera-se que com a utilização desse framework seja possível desenvolver um sistema responsivo, performático e que não dependa de muitos componentes de terceiros para seu funcionamento.

## Integrações entre Componentes

Para apresentação da POC foi desenvolvido um protótipo navegável utilizando a ferramenta Figma, o wireframe e o protótipo estão disponíveis através dos links:

Wireframe: <https://www.figma.com/file/61sc72Z37exQtdckIfP4qK/MyCustomers?node-id=0%3A1>

Protótipo:

<https://www.figma.com/proto/61sc72Z37exQtdckIfP4qK/MyCustomers?node-id=4%3A25&scaling=min-zoom&page-id=0%3A1&starting-point-node-id=2%3A2>

## Código da Aplicação

**Figura 4 – Código da aplicação**

Diagrama

Descrição gerada automaticamente

A Figura 4 demonstra como a estrutura da aplicação foi organizada e como os componentes implementados se relacionam.

Os componentes implementados são divididos em:

- Controllers: Responsáveis por fazerem o roteamento das requisições recebidas pela API e implementar as definições que permitem a geração dinâmica da documentação da API.

- Services: Responsáveis por comportar toda a regra de negócio da API, trata-se do core da API onde a maior parte do trabalho é executado.

- Repositorys: Responsáveis por fazer a persistência de dados.

- AppException: Middleware implementado para controlar as exceções lançadas pela API e adequar o retorno utilizando http status.

Link API: <http://mycustomersapi.us-east-1.elasticbeanstalk.com/swagger/index.html>

Link Front: <http://mycustomersfront.s3-website-us-east-1.amazonaws.com/>

Repositório: <https://github.com/cleidson-jesus/pucminas>

Usuário: usuario

Senha: senh@123

Existem clientes incluídos no banco de dados com o nome “Cleidson”, após o login pesquisar por esse nome utilizando o input “Procure por um usuário”, a pesquisa é ativada quando a tecla enter é pressionada.

Após esse procedimento será exibida uma lista de clientes que possuem o nome pesquisado, clique em cima de algum dos clientes exibidos para abrir o cadastro, os campos exibidos podem ser editados e persistidos no banco de dados através do botão salvar.

## Avaliação da Arquitetura (ATAM)

A avaliação da arquitetura desenvolvida neste trabalho é abordada nesta seção visando avaliar se ela atende ao que foi solicitado pelo cliente, segundo o método ATAM.

## 6.1. Análise das abordagens arquiteturais

|  |  |  |  |
| --- | --- | --- | --- |
| **Atributos de Qualidade** | **Cenários** | **Importância** | **Complexidade** |
| Rastreabilidade | Cenário 1: O sistema deve manter um registro de log para todas as alterações que ocorrerem no cadastro dos clientes. | A | M |
| Performance | Cenário 2: O sistema deve atender a no mínimo 340 tps. | A | A |
| Disponibilidade | Cenário 3: O sistema deve estar disponível 24x7x365. | A | M |
| Compatibilidade | Cenário 4: O front-end deve ser compatível com os navegadores: Crhome, Firefox e Edge | B | M |
| Usabilidade | Cenário 5: O front-end deve apresentar características que facilitem a utilização por PCD’s. | M | B |
| Manutenibilidade | Cenário 6: O deploy em produção das aplicações e a infraestrutura devem ser automatizadas usando pipelines CI/CD | M | M |
| Eficiência | Cenário 7: O processamento de grandes quantidades de informação deve ser executado preferencialmente após 20h. | A | B |

## 6.2. Cenários

Cenário 1 – Rastreabilidade: Ao efetuar operações de inclusão, alteração ou exclusão de dados o sistema deve registrar um log onde devem constar quais dados foram alterados, data da alteração, sistema e usuário responsáveis pela alteração.

Cenário 2 – Performance: Ao ser requisitada por sistemas externos a API deve apresentar boa performance e ser capaz de responder a no mínimo 340 transações por segundo.

Cenário 3 – Disponibilidade: As funcionalidades do sistema devem manter o máximo de disponibilidade possível visto que se trata de um negócio *b2c* onde milhões de clientes vão acessar o aplicativo 24 horas por dia, 7 dias por semana.

Cenário 4 – Compatibilidade: O front-end deve se manter funcional e preservar o design sendo executado nos navegadores Chrome, Firefox e Edge.

Cenário 5 – Usabilidade: O front-end deve oferecer fácil navegação, com fontes e elementos de design que favoreçam a leitura, o layout deve facilitar a localização e funções e permitir que elas sejam executadas de forma rápida.

Cenário 6 – Manutenibilidade: Os sistemas envolvidos na solução devem possuir esteiras de deploy automático, sendo assim, havendo a necessidade de publicação de versões em produção somente será necessário gerar um novo commit no branch determinado que a versão será atualizada em produção.

Cenário 7 – Eficiência: Os serviços do banco têm a maior parte de sua utilização em horário diurno, para que não ocorra impacto na experiência dos usuários nesse horário o processamento de grandes quantidades de informação deverá ser feito a partir de 20h, que é quando as agências já estão fechadas e a utilização do app começa a diminuir.

## 6.3. Evidências da Avaliação

|  |  |
| --- | --- |
| Atributo de Qualidade: | Rastreabilidade |
| Requisito de Qualidade: | O sistema deve manter um registro de log para todas as alterações que ocorrerem no cadastro dos clientes. |
| Preocupação: | |
| O sistema deve permitir a identificação do responsável pelas alterações no cadastro de clientes, além disso deve registrar quais informações foram alteradas. | |
| Cenário(s): | |
| Cenário 1 | |
| Ambiente: | |
| Sistema em operação normal | |
| Estímulo: | |
| Sistemas enviam requisições para a API rest do sistema My-Customers | |
| Mecanismo: | |
| Elaborar inteligência que receba as requisições que mudem o estado do cadastro do cliente e envie para o sistema responsável por persistir os logs. | |
| Medida de resposta: | |
| Persistir os logs enviados e possibilitar consulta através de interface para usuário. | |
| Considerações sobre a arquitetura: | |
| Riscos: | Instabilidade no servidor responsável por armazenamento dos logs causando perda de pacotes. |
| Pontos de Sensibilidade: | Não há |
| Tradeoff: | Não há |

|  |  |
| --- | --- |
| Atributo de Qualidade: | Performance |
| Requisito de Qualidade: | O sistema deve atender a no mínimo 340 tps. |
| Preocupação: | |
| O sistema deve atender a um número mínimo de requisições por segundo pois terá de atender a um cenário onde milhares de usuários farão sua utilização simultaneamente. | |
| Cenário(s): | |
| Cenário 2 | |
| Ambiente: | |
| Sistema em operação extrema | |
| Estímulo: | |
| Teste de carga no sistema. Não é possível reproduzir pois o hardware disponível para testes não consegue atingir a métrica definida. | |
| Mecanismo: | |
| A infraestrutura que suporta o sistema deve ser escalável a fim de atender a métrica definida caso necessário. | |
| Medida de resposta: | |
| Escalar a infraestrutura de forma que o tempo de resposta não aumente diante ao aumento no número de requisições. | |
| Considerações sobre a arquitetura: | |
| Riscos: | Infraestrutura não suportar o aumento no número de requisições e aumentar o tempo de resposta para os usuários. |
| Pontos de Sensibilidade: | Não há |
| Tradeoff: | Não há |

|  |  |
| --- | --- |
| Atributo de Qualidade: | Disponibilidade |
| Requisito de Qualidade: | O sistema deve estar disponível 24x7x365. |
| Preocupação: | |
| O sistema deve apresentar a maior disponibilidade possível já que atende a um modelo de negócio *b2c*. | |
| Cenário(s): | |
| Cenário 3 | |
| Ambiente: | |
| Sistema em operação normal | |
| Estímulo: | |
| Usuários utilizando o sistema | |
| Mecanismo: | |
| Acessar sistema em diferentes horários e dias. | |
| Medida de resposta: | |
| Implantar sistema em infraestrutura resiliente a falhas. Para implementação das provas de conceito foi utilizado a nuvem da AWS, em sua versão paga ela pode garantir até 99.9% de disponibilidade. | |
| Considerações sobre a arquitetura: | |
| Riscos: | Sistema ficar indisponível, impossibilitando clientes de acessarem seus recursos financeiros. |
| Pontos de Sensibilidade: | Não há |
| Tradeoff: | Não há |

|  |  |
| --- | --- |
| Atributo de Qualidade: | Compatibilidade |
| Requisito de Qualidade: | O front-end deve ser compatível com os navegadores: Chrome, Firefox e Edge. |
| Preocupação: | |
| O front-end deve se manter funcional e preservar o design sendo executado nos navegadores Chrome, Firefox e Edge. | |
| Cenário(s): | |
| Cenário 4 | |
| Ambiente: | |
| Sistema em operação normal. | |
| Estímulo: | |
| Usuário acessa o front-end em diferentes navegadores. | |
| Mecanismo: | |
| Adequar código do front-end para diferentes navegadores caso necessário. | |
| Medida de resposta: | |
| Design é apresentado de forma idêntica e sistema permanece funcional em diferentes navegadores.  Edge:  Chrome:  Firefox: | |
| Considerações sobre a arquitetura: | |
| Riscos: | O código do front-end pode ser interpretado de maneiras diferentes entre os navegadores, causando diferença entre o layout e até impedir que funcionalidades sejam executadas corretamente. |
| Pontos de Sensibilidade: | Não há |
| Tradeoff: | Não há |

|  |  |
| --- | --- |
| Atributo de Qualidade: | Usabilidade |
| Requisito de Qualidade: | O front-end deve apresentar características que facilitem a utilização por PCD’s. |
| Preocupação: | |
| O front-end deve fornecer boa usabilidade. | |
| Cenário(s): | |
| Cenário 5 | |
| Ambiente: | |
| Sistema em operação normal. | |
| Estímulo: | |
| Usuários utilizando o sistema. | |
| Mecanismo: | |
| Acompanhar utilização do sistema verificando tempo que tarefas levam para serem completadas. | |
| Medida de resposta: | |
| Sistema com design intuitivo e de fácil visualização.  Na prova de conceito implementada é possível efetuar a atualização de um cadastro com a execução de apenas dois comandos.  Foram implementados campos grandes que facilitam a visualização, o botão de ação foi destacado com uma cor forte que torna fácil sua identificação. | |
| Considerações sobre a arquitetura: | |
| Riscos: | Um sistema difícil de utilizar causa perda de produtividade e pode até mesmo impossibilitar a utilização de alguns usuários. |
| Pontos de Sensibilidade: | Não há |
| Tradeoff: | Não há |

|  |  |
| --- | --- |
| Atributo de Qualidade: | Manutenibilidade |
| Requisito de Qualidade: | Pipelines de deploy automático para todos os sistemas envolvidos na solução. |
| Preocupação: | |
| Mitigar erros com publicações manuais. | |
| Cenário(s): | |
| Cenário 6 | |
| Ambiente: | |
| Sistema em operação normal. | |
| Estímulo: | |
| Desenvolvedor faz publicação de versão em produção. | |
| Mecanismo: | |
| Pipeline faz todos os processos necessários para publicação e publica a versão em produção.  Para o desenvolvimento da POC não foram implementados pipelines para deploy automático, porém, como foi utilizado o Elastic Beanstalk é possível fazer o deploy automático utilizando o próprio Visual Studio. | |
| Medida de resposta: | |
| Versão é publicada automaticamente em produção. | |
| Considerações sobre a arquitetura: | |
| Riscos: | Algum agente de publicação falhar impossibilitando o deploy automático. |
| Pontos de Sensibilidade: | Não há |
| Tradeoff: | Não há |

|  |  |
| --- | --- |
| Atributo de Qualidade: | Eficiência |
| Requisito de Qualidade: | O processamento de grandes quantidades de informação não deve afetar a operação do sistema. |
| Preocupação: | |
| Sistema tornar-se lento devido a interferência de processamentos de grandes quantidades de informação. | |
| Cenário(s): | |
| Cenário 7 | |
| Ambiente: | |
| Sistema com grande carga de processamento. | |
| Estímulo: | |
| Processamentos que demandam grande quantidade de recursos sendo executados. | |
| Mecanismo: | |
| Executar processamento de dados que necessitam de maior quantidade de recursos fora do horário crítico de utilização do sistema. | |
| Medida de resposta: | |
| Em um horário com menor utilização do sistema a tendência é que ocorra pouca ou nenhuma interferência para os usuários. | |
| Considerações sobre a arquitetura: | |
| Riscos: | Algum processamento não terminar de executar antes do início do próximo dia. |
| Pontos de Sensibilidade: | Não há |
| Tradeoff: | Não há |

## 6.4. Resultados Obtidos

|  |  |  |
| --- | --- | --- |
| **Requisitos Não Funcionais** | **Teste** | **Homologação** |
| RNF01: Deve ser registrado trilha de auditoria para todas as alterações que ocorrerem no cadastro de clientes | N.A | N.A |
| RNF02: As API’s devem possuir um *throughput* de no mínimo 340 tps | N.A | N.A. |
| RNF03: O sistema deve possuir disponibilidade 24x7x365 | OK | OK |
| RNF04: O front-end deve ser compatível com os navegadores: Chrome, Firefox e Edge | OK | OK |
| RNF05: O front-end deve apresentar características de design que facilitem o uso por PCD’s | OK | OK |
| RNF06: O deploy em produção das aplicações e a infraestrutura devem ser automatizadas usando pipelines CI/CD | OK | OK |
| RNF07: O processamento de grandes quantidades de dados deve ser feito preferencialmente após as 20h | N.A | N.A |

## Avaliação Crítica dos Resultados

A arquitetura proposta mostrou-se eficaz em resolver os principais problemas encontrados em sistemas de gestão de clientes em bancos, a utilização da arquitetura orientada a eventos permitiu centralizar as informações de clientes, o que facilita o mapeamento de informações, além disso, com a utilização da *stack* ELK a rastreabilidade de alterações nessas informações é feita de forma eficaz e rápida.

Através da implementação das POCS foi possível validar que as decisões em relação a tecnologias foram assertivas, desde a utilização de Angular como linguagem para o front end, passando pela utilização do Dapper como ORM até chegar no design pattern da API, para o último ponto foi observado que apesar de facilitar a implementação de regras de negócio pela equipe técnica não é possível que a área usuária altere as regras dinamicamente.

Com a utilização de um API Gateway toda a estrutura de integrações com o My-Customers é mapeada, essa solução também impede que novas integrações sejam feitas de forma desconhecida, assim obtemos um ambiente conhecidos por todos o que facilita a manutenibilidade e identificação de problemas.

Diante da implementação do Open Banking no Brasil torna-se essencial pensar em soluções que atendam a essa demanda, não foram especificadas soluções de segurança robustas que permitam tranquilidade em expor as informações da instituição para as demais instituições participantes.

## Conclusão

Esse trabalho teve como objetivo apresentar a aplicação da arquitetura de software através da implementação de um sistema fictício para controle de clientes em bancos, o que se mostrou totalmente vantajoso.

O desenvolvimento do relatório técnico deixou claro a importância de projetar software antes de desenvolver, os tópicos abordados trouxeram à tona problemas e permitiram a discussão para solução antes que esses ocorressem, sem a análise arquitetural esses problemas possivelmente seriam identificados já com o sistema em produção.

Os diagramas construídos possibilitaram visualizar todo o ecossistema da solução, incluindo integrações com os demais sistemas internos e externos, esse ponto de vista facilitou a tomada decisões sobre onde investir em segurança, performance e usabilidade.

A materialização de soluções através das POC’s foi crucial para identificar a aderências das tecnologias escolhidas, ampliar o conhecimento técnico e obter visibilidade para evoluções no projeto.

A partir dessas percepções pretende-se desenvolver melhorias no projeto, como por exemplo a implementação de *feature* *toggles* para controle dinâmico de regras de negócio pela área usuária, além disso, é necessário implementar soluções mais robustas de segurança que permitam atender a demanda do Open Banking com tranquilidade.

## Referências

## SOUZA, Ludmila. Pandemia mudou a relação dos brasileiros com tecnologias bancárias. São Paulo: Agência Brasil, 2021. Disponível em: <https://agenciabrasil.ebc.com.br/economia/noticia/2021-06/pandemia-mudou-relacao-dos-brasileiros-com-tecnologias-bancarias> . Acesso em: 03 de março de 2022.