**PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS  
NÚCLEO DE EDUCAÇÃO A DISTÂNCIA**

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

**Ângelo Rafael da Silva**

**Franclis Galdino da Silva**

**Paulo Henrique Fernandes Duarte**

**HomeCare**

Belo Horizonte

2022

**Ângelo Rafael da Silva**

**Franclis Galdino da Silva**

**Paulo Henrique Fernandes Duarte**

**HomeCare**

Projeto Integrado de Arquitetura de Software Distribuído como requisito parcial à obtenção do título de especialista.

Belo Horizonte

2022

**RESUMO**

Este trabalho de conclusão do curso de arquitetura de software distribuído, tem como foco o estudo e implementação de técnicas e tecnologias utilizadas para arquitetura e desenvolvimento de um aplicativo acessível para dispositivos móveis, que forneça aos usuários a possibilidade de localizar e contratar prestadores de serviço autônomos para serviços gerais sob demanda, sendo o aplicativo responsável por fornecer aos contratantes uma lista dos prestadores mais relevantes a sua necessidade e intermediar o processo de contratação e pagamento.

**Palavras-chave:** arquitetura de software, projeto de software, requisitos arquiteturais, casos de uso, design thinking, pwa, frontend, backend, event driven design, microserviços, arquitetura baseada em eventos

**SUMÁRIO**

[1. Objetivos do trabalho](#_heading=h.gjdgxs) 5

[**2. Descrição geral da solução**](#_heading=h.txgdhx38xsrf) **5**

[2.1. Apresentação do problema](#_heading=h.803zn7t1v35d) 5

[2.2. Descrição geral do software (Escopo)](#_heading=h.4da4hpc9fnq1) 6

[**3. Definição conceitual da solução através de Requisitos funcionais**](#_heading=h.jlo44dlfsiqf) **7**

[Requisitos Não Funcionais](#_heading=h.nvoxtd77d9dh) 8

[Regras de Negócio](#_heading=h.c4vdabengjm5) 9

[Casos de Uso](#_heading=h.nuoejhx38ya8) 10

[Sequência caso de uso UC03 - Pagar Serviço](#_heading=h.1qb4rcoq3l6l) 11

[**Arquitetura**](#_heading=h.7g5jhitb0bmq) **12**

[Desenho da Arquitetura do Software](#_heading=h.v2qg36b644ei) 12

[**4. Modelagem e projeto arquitetural**](#_heading=h.q6c2g1hp8jrs) **13**

[**5. Prova de Conceito (POC) / protótipo arquitetural**](#_heading=h.ucx2vnm17blj) **14**

[5.1. Implementação e Implantação](#_heading=h.40dpxu6bqof) 14

[6. Architecture Decision Records (ADRs)](#_heading=h.2hpih2bfpybz) 17

[6.1. Resultado](#_heading=h.2ze84co8z06e) 24

[7. Conclusão](#_heading=h.z1w499h7jbz7) 24

[**REFERÊNCIAS**](#_heading=h.8ycweiq7x1ot) **25**

[**APÊNDICES**](#_heading=h.tpc5i8axqs15) **25**

## **1. Objetivos do trabalho**

O objetivo deste trabalho é desenvolver um aplicativo PWA para gerenciamento do processo de contratação e confirmação de prestação de serviços, utilizando arquitetura em microserviços e orientada a eventos. Para encontrar o contexto utiliza-se técnicas como design thinking com o objetivo de orientar e estimular a definição do contexto através de um workshop em formato de debate.

Os objetivos específicos do trabalho são:

* Identificar os componentes necessários para desenvolver uma arquitetura em microserviços;
* Elaborar uma estratégia para garantir resiliência do processo na integração com serviços externos.
* Elaborar uma estratégia para tornar a experiência do usuário, simples, fluida e intuitiva.
* Realizar integrações do aplicativo com sistemas de terceiros.

# **2. Descrição geral da solução**

A solução tem como foco o projeto e avaliação de uma arquitetura baseada em microserviços, que tem tido cada vez mais adoção no mercado por oferecer maior flexibilidade e segurança para apliacaoes corporativas principalmente B2B.

Essa abordgagem é relevante por permitir que nossa aplicação seja escalável e com maior controle sobre os custos de operação e manutenção, uma vez que podemos estabelecer planos de escala da aplicação de forma sazonal ou sob demanda em momentos de pico, enquanto em momentos de menos acesso podemos reduzir a operação dos serviços economizando em processamento em nuvem. Também teremos maior controle no custo de manutenção por temos os serviços de forma isolada, sendo mais fácil testar cada serviço e evitar que defeitos se propaguem facilmente por toda aplicação, e em caso de algum defeito em produção, o rollback de um serviço é suficiente para restaurar a responsividade do aplicativo.

## **2.1. Apresentação do problema**

Seguindo a metodologia do design thinking:

Sobre Empatia, elaboramos um cenário que serviu como referência e que foi enriquecido com as experiências reais dos integrantes do workshop e conseguimos identificar alguns problemas que o usuário teria ao tentar procurar e contratar um serviço.

Decidir: Conseguimos separar problemas e identificar qual seria o foco principal que traz maior risco, ou que causaria mais problemas.

Idear : Definimos uma proposta de negócio simplificada, focada em viabilizar a prestação de serviço e gerenciamento do pagamento com garantia de resiliência.

Prototipar : Como protótipo iremos ter um aplicativo web PWA contendo 2 telas principais, sendo 1 para o cliente e outra para o prestador e 1 para login batendo num backend que possuirá as regras de negócio e lógicas de acesso a dados e integração com parceiros.

Testar : Cliente entra com seus dados no aplicativo e é redirecionado para a respectiva página de acordo com o tipo de seu acesso. Sendo um cliente, ele poderá buscar por um serviço específico, incluindo sua localização e a abrangência da busca medida pelo raio de sua localização, e o critério de priorização dos resultados, sendo eles, por preço ou por qualidade. O aplicativo irá exibir todos prestadores disponíveis e ele poderá contratar um clique no botão "Contratar" onde terá acesso a outra tela onde ele fornecerá os dados de pagamento. Após o pagamento ser aprovado ele terá acesso aos dados do prestador de serviço. O Prestador de serviço receberá um e-mail contendo o link do aplicativo onde ele poderá informar a finalização para que o dinheiro seja liberado.

## **2.2. Descrição geral do software (Escopo)**

O objetivo da aplicação é permitir que o cliente encontre prestadores de serviço conforme sua necessidade, pagar o serviço e receber os dados de contato do fornecedor para poder combinar o serviço.

História:

* Cliente busca por prestadores disponíveis na região (num raio de X km).
* Ele visualiza uma lista de prestadores ordenados pelo preço ou avaliação.
* Escolhe um prestador e paga pelo serviço.
* Após o pagamento é liberado os dados de contato e o prestador é notificado.
* O cliente e o prestador negociam o serviço externamente.
* O prestador de serviço informa a conclusão do serviço.
* O cliente confirma a conclusão, avalia a qualidade do serviço prestado e libera o pagamento para o pprestador.

Para garantir a segurança da aplicação, o cenário descrito acima é protegido por um conjunto de regras que evitam prejuízos para os usuários e plataforma. Abaixo às principais regras identificadas durante a fase de design thinking:

* Após a notificação de conclusão do serviço, caso o dinheiro não seja liberado pelo cliente em até 15 dias o aplicativo irá liberar automaticamente o dinheiro para o prestador.
* Caso o serviço não seja informado como concluído pelo prestador até 60 dias, o aplicativo irá liberar automaticamente o dinheiro para o cliente.
* O cliente pode abrir uma disputa para reclamar sobre a não prestação do serviço a fim de evitar que o dinheiro seja liberado indevidamente.

# **3. Definição conceitual da solução através de Requisitos funcionais**

* **Requisitos Funcionais**

| **Cód.** | **Nome** | **Descrição** |
| --- | --- | --- |
| RF-001 | Pesquisar prestadores | Cliente busca por prestadores disponíveis na região (num raio de X km) |
| RF-002 | Ordenar pesquisa por preço | Cliente visualiza uma lista de prestadores ordenados pela oferta mais barata. |
| RF-003 | Ordenar pesquisa por avaliação | Cliente visualiza uma lista de prestadores ordenados pela avaliação de qualidade do prestador. |
| RF-004 | Visualizar prestador | Cliente escolhe um prestador e paga pelo serviço. |
| RF-005 | Selecionar prestador | Cliente escolhe um prestador e paga pelo serviço. |
| RF-006 | Pagar serviço | Cliente efetua o pagamento de um serviço para um prestador. |
| RF-007 | Confirmar pagamento | Plataforma confirma o recebimento do pagamento. |
| RF-008 | Liberar dados de contato | Plataforma libera os dados de contato do prestador para o cliente. |
| RF-009 | Notificar prestador | Plataforma notifica o prestador sobre a contratação informando os dados do cliente. |
| RF-010 | Informar conclusão do serviço | Prestador informa para a plataforma que o serviço está concluído. |
| RF-011 | Notificar cliente | Plataforma notifica o cliente pedindo confirmação de conclusão. |
| RF-012 | Confirmar a conclusão | Cliente confirma a conclusão. |
| RF-013 | Liberar o dinheiro manualmente | Plataforma libera o dinheiro para o prestador após confirmação do cliente. |
| RF-014 | Liberar dinheiro automaticamente | Caso o dinheiro não seja liberado pelo cliente em até 15 dias o aplicativo irá liberar automaticamente o dinheiro para o prestador. |
| RF-015 | Devolver dinheiro automaticamente | Caso o serviço não seja informado como concluído pelo prestador até 60 dias, o aplicativo irá liberar automaticamente o dinheiro para o cliente. |
| RF-016 | Abrir disputa | O cliente pode abrir uma disputa para reclamar sobre a não prestação do serviço a fim de evitar que o dinheiro seja liberado indevidamente. |

## Requisitos Não Funcionais

| **Cód.** | **Nome** | **Descrição** |
| --- | --- | --- |
| RNF-002 | Assíncrono | Evita que o cliente um pagamento falhe caso o cliente feche o aplicativo ou que precise esperar por um longo tempo até que o pagamento seja confirmado. |
| RNF-003 | Resiliente | Garante que estágios importantes do sistema não sejam perdido em caso de falha, por exemplo registros no banco de dados ou notificações para o cliente. |

## Regras de Negócio

| **Cód.** | **Nome** | **Descrição** |
| --- | --- | --- |
| RN-001 | Deve realizar negociações fora da plataforma | O cliente e o prestador negociam o serviço externamente |
| RN-002 | Deve liberar dinheiro para o prestador automaticamente após 15 dias | Caso o serviço não seja confirmado pelo cliente em até 15 dias o aplicativo irá liberar automaticamente o dinheiro para o prestador. |
| RN-003 | Deve devolver o dinheiro para o cliente após 60 dias | Caso o serviço não seja informado como concluído pelo prestador até 60 dias, o aplicativo irá liberar automaticamente o dinheiro para o cliente. |
| RN-004 | Deve bloquear liberação do dinheiro para o prestador caso o pagamento esteja em disputa | O cliente pode abrir uma disputa para reclamar sobre a não prestação do serviço a fim de evitar que o dinheiro seja liberado indevidamente. |
| RN-005 | Deve coletar taxa pelo pagamento do serviço | As operações de pagamento devem descontar um percentual como taxa do serviço pelo aplicativo, já incluso no valor informado ao cliente |
| RN-006 | Deve liberar o dinheiro somente após confirmação de conclusão |  |
| RN-007 | Deve liberar o contato somente após confirmação de pagamento |  |
| RN-008 | Deve exibir dados de contato apenas se o pagamento estiver confirmado |  |

## Casos de Uso

Esses são todos os casos de uso identificados através da análise dos requisitos.

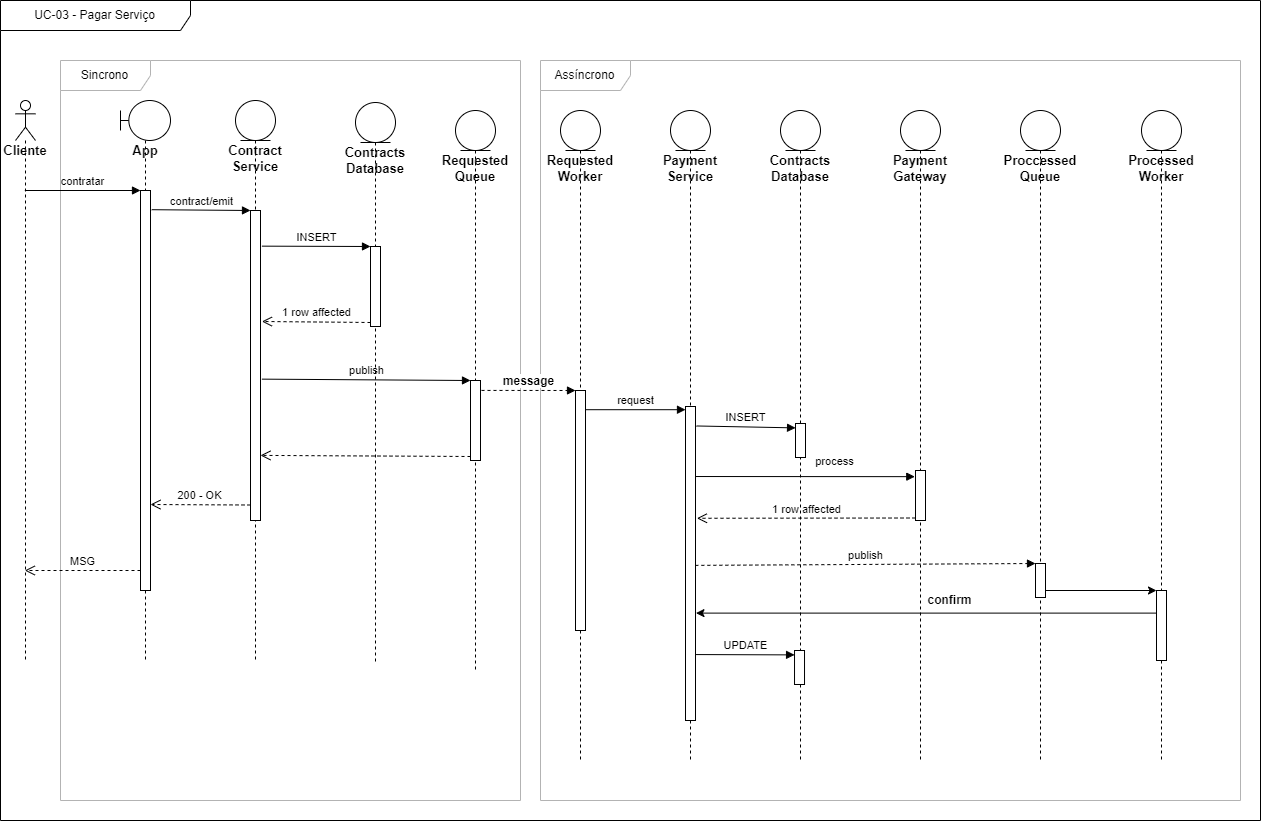
Em destaque estão os casos de uso mais críticos, principalmente o **UC-03 Pagar Serviço**, que precisa ter duas capacidades, ser resiliente e assíncrono, porque não queremos que o cliente fique muito tempo esperando e caso ocorra algum erro por falha no sistema, não queremos que o cliente precise refazer seu pagamento.



## Sequência caso de uso UC03 - Pagar Serviço

Este foi o caso de uso mais crítico, por incluir validações de regras de negócio, integração com parceiros, persistência de dados e possuir um processo que começa síncrono e depois se torna assíncrono.

Essas complexidades tornam a visão de diagrama de sequência útil para a analisarmos os componentes necessários e como ocorre a troca de informações entre eles.



# Arquitetura

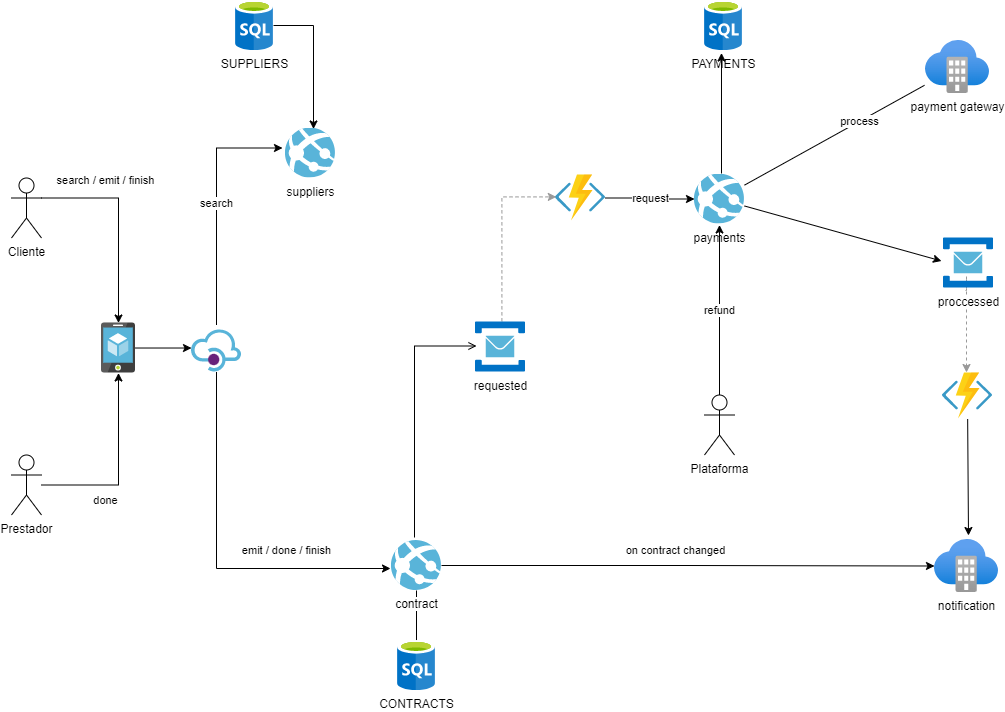
## Desenho da Arquitetura do Software

O uso de uma arquitetura baseada em eventos é importante para garantir a consistência das informações. Lidando com informações de transações bancárias realizadas por um parceiro, isso garante que eventuais problemas de rede não afetem a confiabilidade do aplicativo, tão pouco, gere risco financeiro.

Nesse projeto identificamos a necessidade de utilizarmos 3 microserviços, cada um deles com um escopo específico e com necessidades diferentes de escala. Por exemplo, o microserviço de prestadores é provavelmente o que será mais acessado pelos usuários, porém com propósito apenas de leitura, o que pode ser otimizado com uso de cache e hospedados em um servidor localizado próximo dos usuários para diminuir a latência, enquanto, os microserviços de contratação precisam ser rápidos no processamento.

Para garantir a resiliência, as operações que dependem de um processamento externo são protegidas por filas do RabbitMq, dessa forma é naturalmente realizado uma retentativa quando houver uma falha.

As filas são consumidas por workers que fazem o callback para o microserviço de pagamento, informando o quando for o momento de tentar o envio para o Gateway de pagamentos ou para a plataforma de notificações.



# **4. Modelagem e projeto arquitetural**

Nesta seção é apresentado um diagrama que permite entender a arquitetura da aplicação, detalhando-a suficientemente para viabilizar sua implementação. Iremos utilizar um diagrama macro de implantação em nuvem para descrever os principais componentes e suas ligações através de alguma ferramenta de diagrama. Exemplos:

* + **AWS**: <https://www.cloudcraft.co/>;
  + **Azure**: <https://online.visual-paradigm.com/pt/diagrams/features/azure-architecture-diagram-tool/>;
  + **Outras nuvens**: <http://draw.io>.

# **5. Prova de Conceito (POC) / protótipo arquitetural**

Para nossa Prova de Conceito, criaremos uma interface gráfica integrada com nossa aplicação webApi para fornecer ao usuário a possibilidade de buscar prestadores, selecionar um prestador e contrata-lo informando os dados de pagamento.

No Backend, a aplicação irá utilizar banco de dados em memória para fornecer a lista de usuários e prestadores, além de processar os contratos e integrar com um gateway de pagamentos fictício, salvar os pagamentos em filas do RabbitMq e também enviar notificações para os e-mails dos clientes e prestadores de acordo com alterações na situação atual do contrato.

A evolução da construção do protótipo será registrada no servidor github página <https://github.com/phduarte/HomeCare>.

## **5.1. Implementação e Implantação**

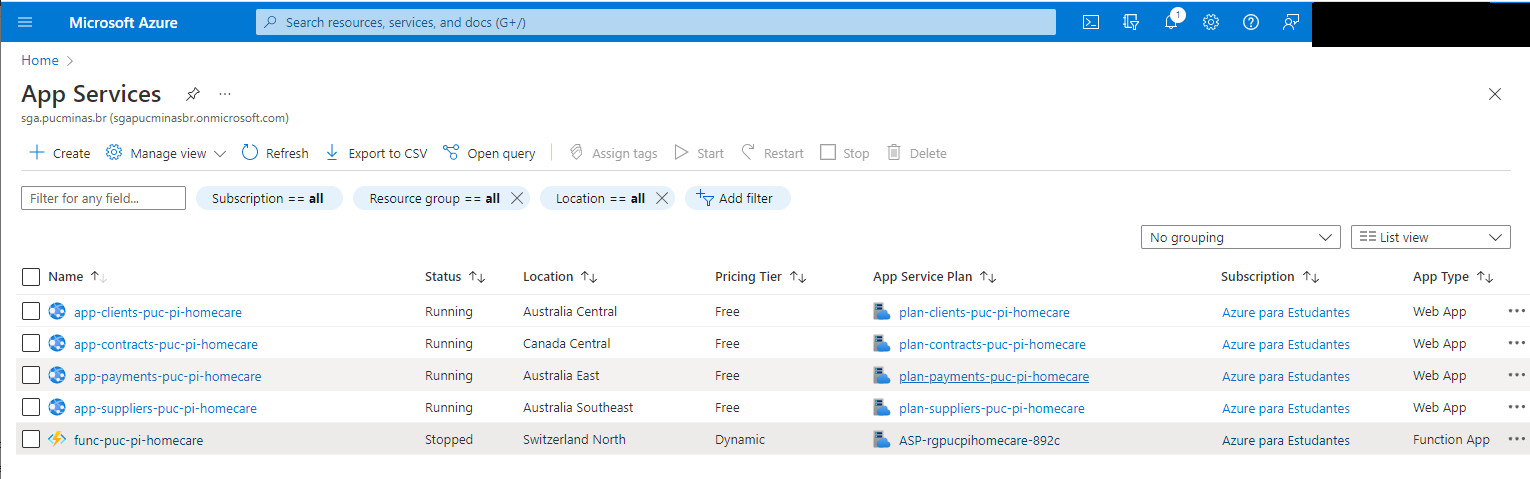
**Back-End**

Adotamos o padrão de webAPI conhecida como Minimal APIs, disponibilizada no .NET 6 e que pareceu bem aderente ao conceito de microserviços, de fato, simplificou a estrutura dos projetos de microserviços.

Também utilizamos padrões de abordagens como DDD e Clean Architecture, o que permitiu gerenciar bem as regras de negócio, ter uma boa separação de escopos o que resultou em consistência de cada um dos componentes, ou seja, conseguimos localizar rapidamente componentes que devem ser alterados ou substituídos.

**Microserviços**

Através da conta de estudante disponibilizada pela PUC criamos vários serviços de aplicativo, cada um em uma localização diferente. Apesar de não ter sido identificado como requisito não funcional para o sistema, a possibilidade de utilizar diferentes localização pode ser aproveitada como uma estratégia de otimização de custo, em situações onde um microserviço possui menor criticidade e a baixa latência não é um risco para a aplicação, pode se adotar datacenters conforme o menor custo.

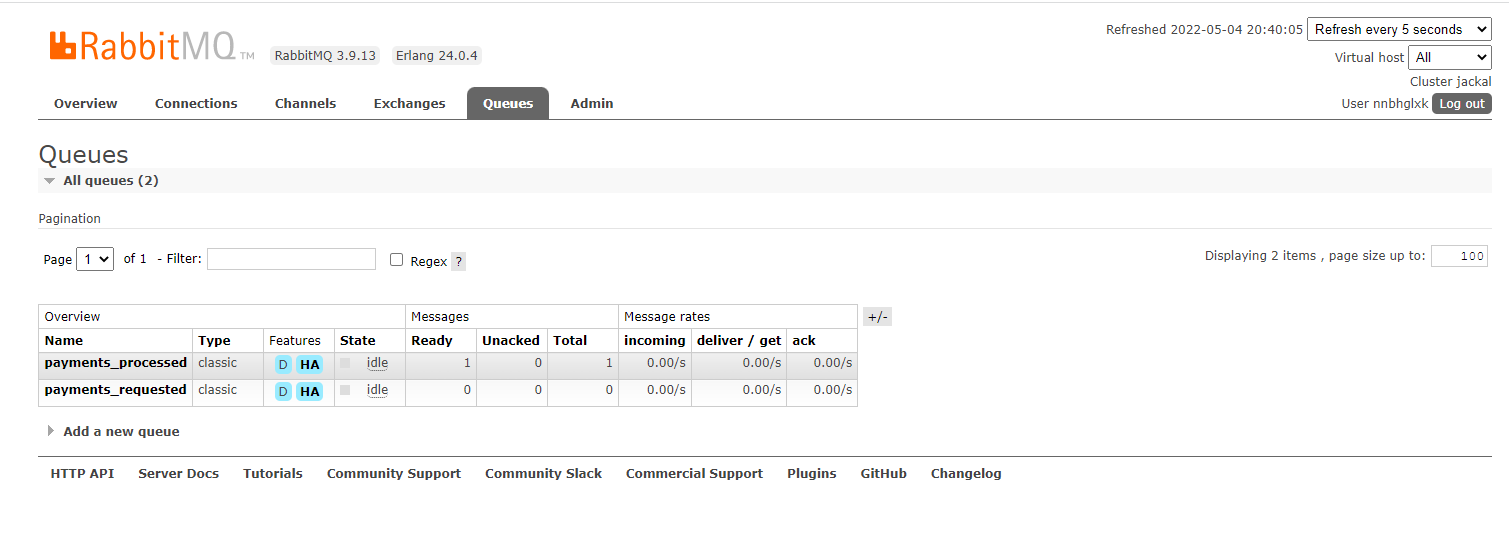


Abaixo deixamos disponível o swagger de cada um dos serviços:

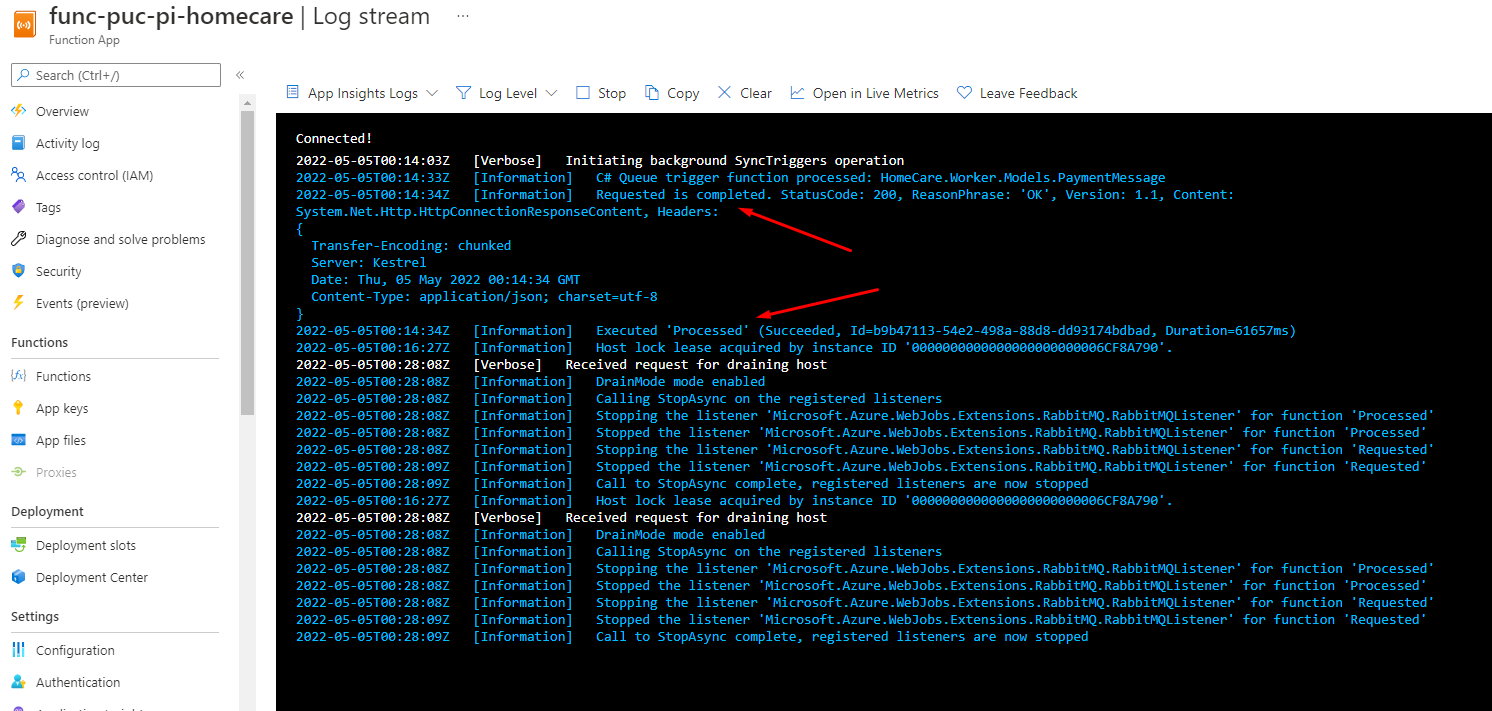
* Clientes: <https://app-clients-puc-pi-homecare.azurewebsites.net/swagger>
* Prestadores: <https://app-suppliers-puc-pi-homecare.azurewebsites.net/swagger>
* Contratos: <https://app-contracts-puc-pi-homecare.azurewebsites.net/swagger>
* Pagamentos: <https://app-payments-puc-pi-homecare.azurewebsites.net/swagger>

**Eventos**

Utilizamos o provedor cloudamqp que oferece o serviço de RaaS (RabbitMq as a Service).

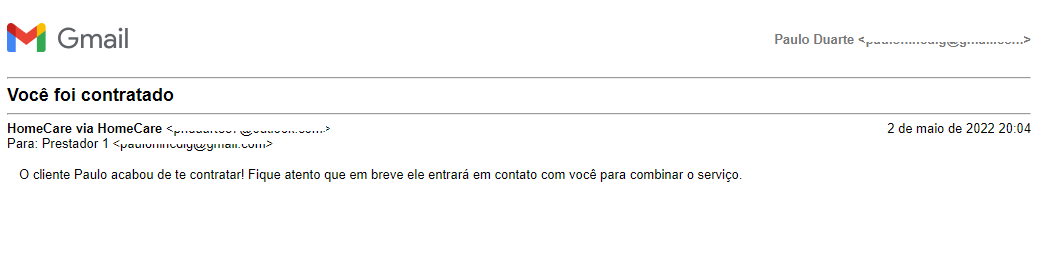


Quando uma mensagem uma Azure Functions realiza o tratamento da mensagem enviando para o microserviço responsável pela execução da regra.

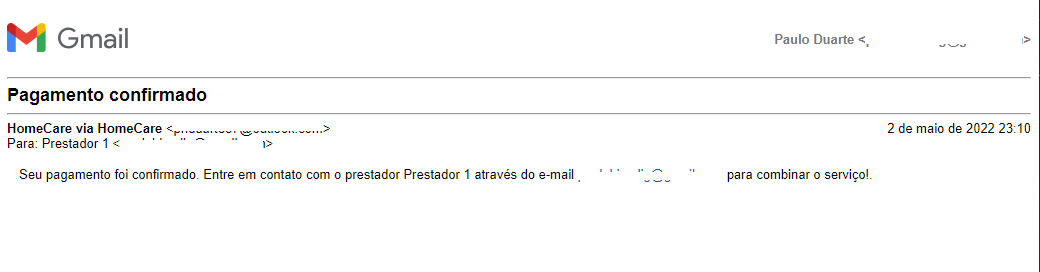


**Integrações**

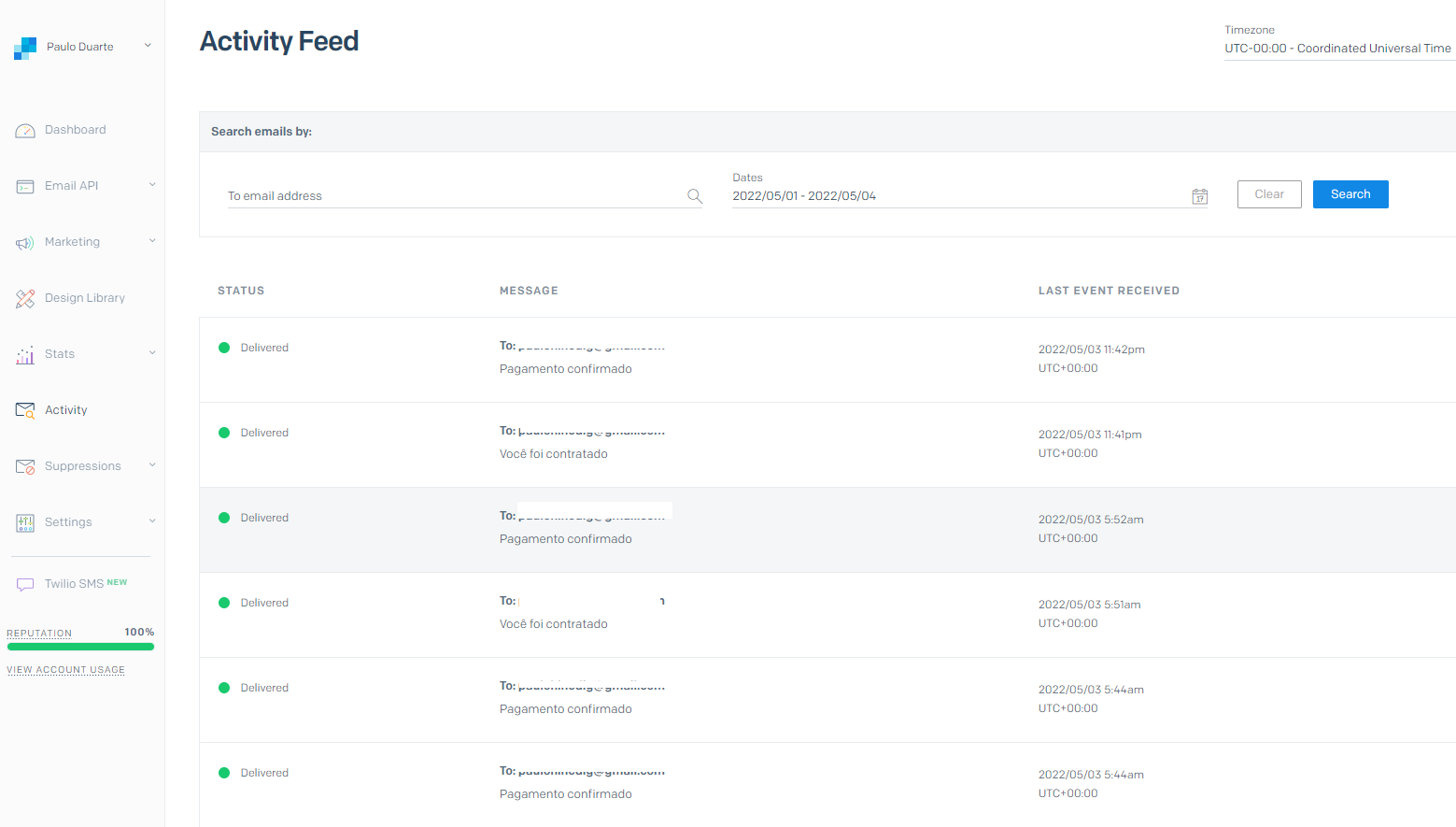
No nosso fluxo, sempre que um prestador é contratado é enviado um e-mail para lhe informar que o cliente entrará em contato e também um e-mail para o cliente informando os dados de contato do prestador.



Também é enviado um e-mail para o cliente quando o prestador informa que o serviço já foi feito e para o prestador quando o cliente finaliza o contrato.



Criamos uma conta na plataforma SendGrid da Twillo para utilizamos sua api de envio de e-mails.



## **6. Architecture Decision Records (ADRs)**

**Objetivo**

Detalhamento do motivo de escolha de tecnologia, provedor ou ferramenta mais importante que foram escolhidas de acordo com o contexto do momento.

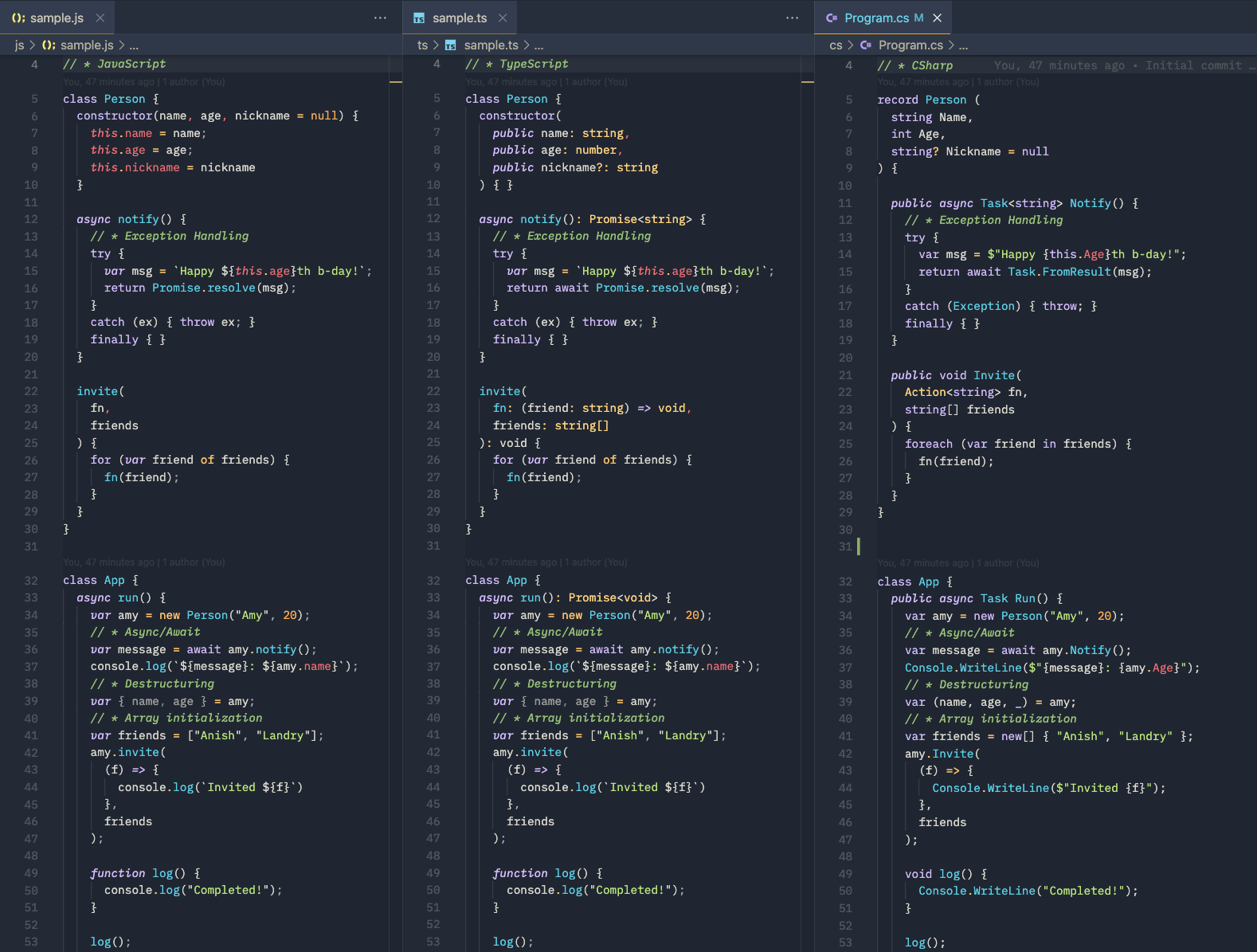
**Roteiro**

* Listar atributos de qualidade necessários
* Atributos mais críticos devem ser priorizados (possuem um peso maior, estão ligadas ao negócio, mercado, riscos e desafios de sucesso do sistema)
* Diferenciar decisões de design e implementação
* Analisar Trade-offs (ponderação)

**Backend**

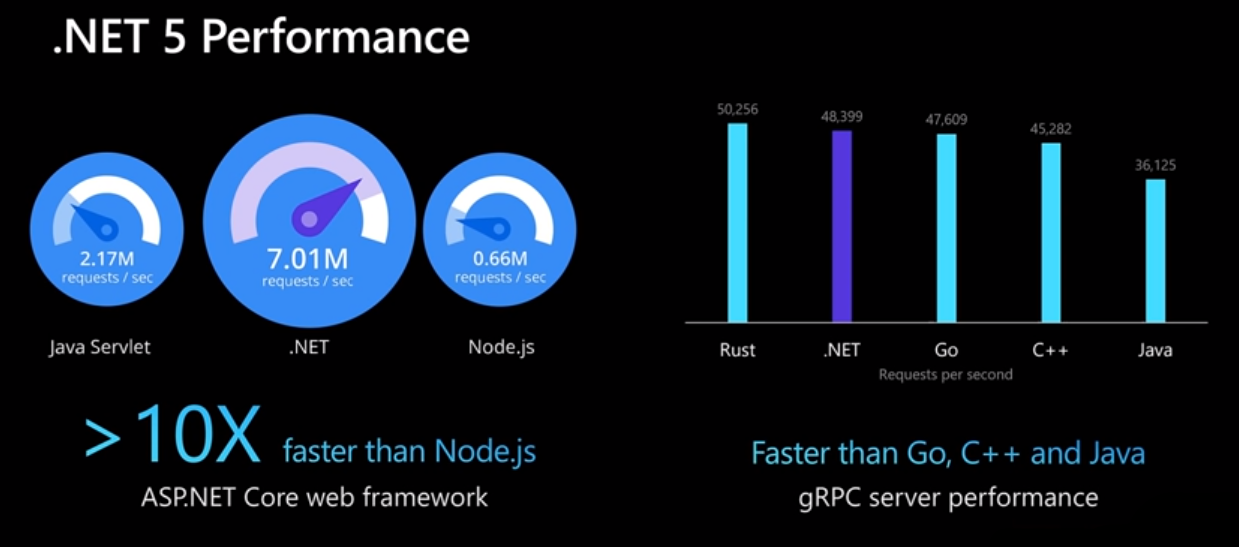
* Curva de Aprendizado

Durante o curso tivemos acesso à 3 tecnologias para desenvolvimento backend, sendo, DotNet, Java e NodeJs. Sintaticamente falando, as 3 são bastante semelhantes, principalmente DotNet e NodeJs.



* Benchmark Performance

O redesenho do framework .NET que resultou no .NET Core o tornou uma ótima alternativa para execução de código em servidores cloud devido sua velocidade de processamento e baixa carga, otimizando assim as requisições http e o warmup.



* Experiência da Equipe

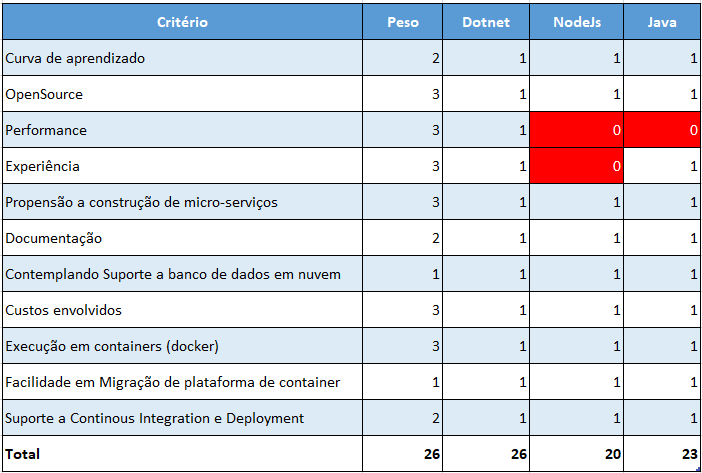
A curva de aprendizado acabou sendo injusta com o NodeJs, pois os integrantes da equipe não possuem proficiência em NodeJs e a disciplina começou ser ministrada após o Projeto Integrado já estar em andamento. Como o objetivo da PoC não é comparar a adoção de uma plataforma, optamos em utilizar .NET Core dessa forma podemos focar os estudos no modelo de desenvolvimento orientado a eventos que é um objetivo da PoC.

* Conclusão

Fizemos uma avaliação no formato de matriz com critérios e ponderações baseadas no grau de importância para o projeto no momento da análise.

**Peso dos critérios:**

* 1 - Útil, mas não é obrigatório.
* 2 - Importante, se tiver será um diferencial.
* 3 - Crítico, é obrigatório ter.



**Mensageria**

RabbitMQ é um servidor de mensageria Open Source desenvolvido em Erlang, implementado para suportar mensagens em um protocolo denominado Advanced Message Queuing Protocol (AMQP). Ele possibilita lidar com o tráfego de mensagens de forma rápida e confiável, além de ser compatível com diversas linguagens de programação, possuir interface de administração nativa e ser multiplataforma.

Por ser uma tecnologia de código aberto, madura e bem apoiada pela comunidade, o RabbitMQ se torna uma unanimidade até o momento dessa análise.

Nessa PoC, optamos por utilizar o RabbitMQ as a Service, o que traz a facilidade de manutenção da infraestrutura. O provedor Cloud AMQP possui um plano de onboarding que oferece o ambiente gratuito até completar 1 milhão de mensagens, o que parece ser suficiente para esse projeto até o momento da apresentação.

**Front End**

Para definição da tecnologia utilizada no front end, seguimos os criterios, orientações e discussões aprendidos nas aulas de Arquitetura de Front Ends. O framework ecolhido foi o React, seguindo os critérios abaixo:

* Facilidade de desenvolvimento:

Possui ótima documentação e é um dos frameworks mais utilizados para desenvolvimento front-end, com 186mil estrelas no github, e tem uma comunidade bastante ativa e solicita em fóruns. A configuração inicial é mais simples e rápida, podendo ir adicionando ferramentas JavaScript conforme a aplicação cresce, sendo recomendada inúmeras ferramentas para aplicações grandes, sendo que elas exigem poucas ou nenhuma configuração. Como toda tecnologia possui uma curva de aprendizado, sendo necessário estudo e prática, sendo um framework desenvolvido com a linguagem JavaScript é necessário conhecimento prévio.

* Arquitetura:

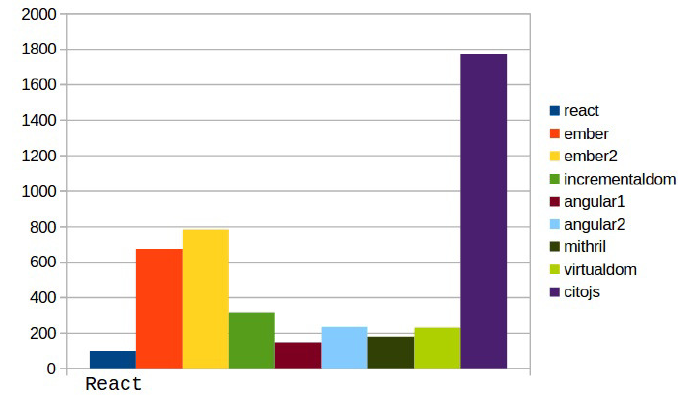
O React é fundamentado em componentes, que são objetos JavaScript e possuem propriedades, estado e um ciclo de vida gerenciado pelo framework, com o objetivo de atualizar o DOM de forma organizada e fácil. Utiliza a sintaxe JSX para organizar a renderização dos componentes. JSX foi criado pela equipe de desenvolvimento do React, para criação de componentes usados como template, misturando código JavaScript com código HTML. O JSX não é interpretado pelos navegadores, sendo necessário utilizar um transpilador para fazer a conversão, que são utilizados para facilitar o desenvolvimento dos componentes, porém não é obrigatório o uso, o React oferece APIs nativas do ECMAScript. O ciclo de vida de um componente é dividido em montagem, atualização, desmontagem e tratamento de erros, cada parte do ciclo de vida possui métodos que podem ser sobrescritos conforme necessidade. Sendo o único de uso obrigatório o método render, que quando executado atualiza o DOM, ocorrendo na alteração do estado ou criação do componente. Para realizar a leitura dos dados de um componente é utilizado o método this.state e para atualizar é utilizado o método setState, ou por meio de Hooks que são uma nova adição ao React 16.8, e permitem que você use o state e outros recursos do React sem escrever uma classe.

* Componentização:

Em React os componentes são definidos como classes ou funções, quando definidos como classes eles possuem mais funcionalidades, porém os Hooks introduzidos a partir do React 16.8 vieram para resolver esse problema. Para definir um componente como classe é preciso estender React.Component, sendo preciso definir o método render na subclasse. A renderização dos componentes em React são feitas por meio do método render, dentro do próprio componente.

* Performance:

A usabilidade e a performance esperadas em um PWA praticamente exigem ser uma single page application (SPA). O React utiliza o virtual DOM para renderizar os componentes, a manipulação do virtual DOM é mais performática do que a manipulação o DOM, quando é feito a alteração do componente, o React cria uma nova árvores com os componentes no virtual DOM, calcula a diferença entre a árvore antiga e nova e atualiza o DOM com apenas as diferenças. Por ser uma PWA, o JavaScript será baixado e ficará no Application Cache do navegador. A página precisa estar disponível em poucos segundos, e isso inclui o primeiro acesso. Outro ponto importante é a utilização de componentes conforme necessidade do projeto. O react é one-way binding, a transição dos dados entre o DOM e os componentes são somente em uma direção, o estado e as propriedades do componente são copiados para o DOM somente por meio do método render, e os dados do DOM são enviados por meio de eventos. Fazendo com que o desenvolvimento seja mais simples e baseado em funções e facilita o entendimento do código. Do ponto de vista de performance, o melhor é atualizar quando for necessário atualizar, como clicando em no botão para submeter o formulário, ao invés de atualizar os atributos no JavaScript todas vez que as tags são alteradas. O React ganha sentido porque ele enfatiza fortemente a atualização do estado dos objetos via eventos. Ele condena o acesso aos componentes explicitamente, e isso torna-o simples, rápido e de fácil entendimento. O livro Progressive Web APPs trás a comparação de renderização no navegador entre os principais frameworks front de mercado. O gráfico representa em milissegundo o tempo necessário para renderização no navegador redesenhar a página e o React se destaca nitidamente.



*Pontes, Guilherme. Progressive Web Apps . Casa do Código. Pesqueisa realizada pelo site auth0.com (PEYROTT, 2016)*

* Documentação:

A documentação assume o conhecimento prévio sobre programação com a linguagem JavaScript, recomendando uma revisão sobre ela. Ela considera os diferentes conhecimentos e métodos de aprendizado, possuindo seções específicas de tutorial prático ou guia detalhado. A documentação sempre reflete a última versão estável do framework. Para iniciantes que tiverem dificuldades com a documentação, possui um link para um tutorial com os principais conceitos do React. Possui referência para determinadas API específica do React.Possui um glossário com os principais termos da documentação. Possui uma FAQ para perguntas e respostas sobre assuntos comuns. Faz recomendações de cursos e livros.

* Suporte da comunidade (dúvida em fóruns, stackoverflowe etc):

Possui o fórum do React que está no GitHub da Meta (facebook) ou o fórum do steackoverflow, onde conseguimos identificar a participação ativa e interação entre as pessoas respondendo as perguntas e compartilhando conhecimentos. Analisando o gráfico de download da npm trends, podemos perceber o número gigantesco de pessoas interessadas no React no último ano.

## **6.1. Resultado**

Através da conta de estudante disponibilizada pela PUC criamos vários serviços de aplicativo, cada um em uma localização diferente. Apesar de não ter sido identificado como requisito não funcional para o sistema, a possibilidade de utilizar diferentes localização pode ser aproveitada como uma estratégia de otimização de custo, em situações onde um microserviço possui menor criticidade e a baixa latência não é um risco para a aplicação, pode se adotar datacenters conforme o menor custo.

## **7. Conclusão**

A utilização de uma arquitetura em microserviços e orientada a eventos para o contexto estudado demonstrou ser eficiente ao fornecer a capacidade de escalarmos os caminhos mais críticos da nossa aplicação para aumentar seu desempenho e também protegê-los com a estratégia de utilização de filas.

Toda a infraestrutura do sistema foi construída em nuvem e diversos recursos da aplicação puderam ser terceirizados, o que resultou em economia de tempo e dinheiro, agilizou a construção da POC e viabilizou a hospedagem sem precisar ter o custo de propriedade dos servidores.

# **REFERÊNCIAS**

Aula 2 - Análise, Projeto e Avaliação de Arquitetura de Software - Eduardo.

Trabalho Final - Arquitetura Java - Fabio Leandro Rodrigues Cordeiro.

RabbitMQ o que é e como utilizar. cedrotech, 2018. Disponível em: https://blog.cedrotech.com/rabbitmq-o-que-e-e-como-utilizar#:~:text=Dentre%20as%20aplicabilidades%20do%20RabbitMQ,fila%20de%20trabalhos%20em%20background. Acesso em: 01 de maio de 2022.

Digital, Charlie. Added JS and TS versions of battleships. github, 2022. Disponível em: https://github.com/CharlieDigital/js-ts-csharp. Acesso em: 01 de maio de 2022.

React. Reactjs, 2022. Documentação. Disponível em: https://pt-br.reactjs.org/docs/getting-started.html. Acesso em: 01 de maio de 2022.

Pontes, Guilherme. Progressive Web Apps. Casa do Código, 2018.

# **APÊNDICES**

<https://github.com/phduarte/HomeCare>