Arquitetura de Microsserviços: Conceituação, Projeto e Implementação

Alexandre Peluchi de Oliveira Junior

**1. Apresentação**

Boa tarde!!! Inicio nesse momento a defesa do meu trabalho de diplomação. Apresento os meus sinceros cumprimentos aos avaliadores.

**2. Tema**

O meu nome é Alexandre Peluchi sou orientando do Professor Luiz Alberto Ferreira Gomes. E o tema do meu trabalho é “Arquitetura de Microsserviços: Conceituação Projeto e Implementação”.

**3. Introdução**

O foco do meu trabalho é na pesquisa e implementação da arquitetura de Microsserviços. Que é uma forma de desenvolver software onde cada pedaço do sistema, é pensado, desenvolvido e disponibilizado de forma independente, criando uma coleção de microsserviços desacoplados, cada serviço implementando uma única funcionalidade do sistema.

            Para aplicar os conhecimentos de Microsserviços na prática, uma pesquisa foi feita partindo do seguinte problema. Quais foram os motivos pelos quais, cerca de 21% das empresas quebram após o primeiro ano de atividade, segundo o IBGE. E após diversos motivos terem sido identificados, partiu-se da hipótese que conquistar clientes e a falta de capital, podem estar diretamente relacionados a má gestão de pedidos, caixa e preparo de produtos.

            Foi realizada então, uma pesquisa aplicada, descritiva com abordagem qualitativa e método hipotético-dedutivo, por meio de procedimento de pesquisa bibliográfica, documental e estudo de caso.

4. Problema

Apesar do grande crescimento do setor alimentício, para a saúde financeira de uma empresa é necessário melhorar o gerenciamento, mas é comum a empresa não utilizar nenhuma forma de gestão no seu negócio. Tornando difícil o atendimento e fechamento da conta por parte do garçom, e muitas vezes uma experiência desagradável para os clientes.

5. Objetivo

O objetivo deste trabalho é o desenvolvimento de um sistema utilizando microsserviços, para automatizar grande parte dos processos desses estabelecimentos, como o controle e gerenciamento de pedidos, o fechamento da conta dos clientes, cálculo de troco, fechamento de caixa, relatórios de gastos e de produtos mais vendidos.

6. Conceitos de Design

Para a implementação de sistemas que utilizam a arquitetura de microsserviços, alguns conceitos de *design* devem ser seguidos, tais como:

* **Alta coesão(Um sistema coeso é fortemente unido, conexo)** — dados e regras de negócio devem ser encapsulados juntos, garantindo facilidade na manutenção e implementação de novas funcionalidades.

Sem alta coesão, no fim a arquitetura será “monolitica distribuida”, será preciso a cada nova funcionalidade, fazer alterações e deploys em multiplos serviços com baixo acoplamento. (Porque )

* **Propósito único** — cada serviço deve possuir uma única responsabilidade.

Sem propósito único multiplos serviços monoliticos e não será possível ter todos os beneficios de microsserviços.

* **Baixo acoplamento** — quando um serviço possui pouca ou nenhuma dependência, ou relação com outros serviços.

Sem baixo acoplamento alterações em um único serviço vai impactar outros serviços, então não será possível fazer alterações e deploys mais rápidos com segurança.

Deixar de utilizar qualquer um destes conceitos, fará com que o padrão se torne um ou anti-padrão (*antipattern*), levando a resultados ruins, ao invés do esperado.

**7. API**

API (Application Programming Interface), basicamente uma API fornece uma interface que possui requisições bem estabelecidas, permitindo a utilização das suas funcionalidades por outros softwares ou aplicações.

As APIs permitem a **integração entre dois sistemas**, em que um deles fornece informações e serviços que podem ser utilizados pelo outro, sem a necessidade de o sistema que consome a API conhecer detalhes de implementação do software.

Um exemplo seria o **Google Maps**, existem páginas Web que consomem a API do **maps**, para exibir a localização de um estabelecimento ou a API de autenticação do Google ou do Facebook, alguns sites a utilizam para o usuário não ter que criar uma nova conta e simplesmente utilizar a sua conta do Google ou seu próprio Facebook.

Esse é um conceito muito importante, porque tem relação direta com **Microsserviço**, enquanto um é uma metodologia arquitetural para aplicações web, onde todas as funcionalidades são divididas entre pequenos serviços, APIs são as estruturas por meio das quais, os desenvolvedores podem interagir com a aplicação.

**8. Diagrama de Microsserviço**

Gigantes da tecnologia como Netflix, Paypal, eBay, Twitter e Amazon são algumas empresas que utilizando está metodologia, nessa imagem podemos ver um exemplo de funcionamento do **Amazon S3 (Simple Storage Service)** que fornece armazenamento de objetos por meio de uma interface de serviço web.

O Amazon Elastic Container Service (Amazon ECS) é um serviço gerenciado de orquestração de contêineres. Clientes como Duolingo, Samsung usam o ECS para executar seus aplicativos mais confidenciais e essenciais à missão devido à segurança, confiabilidade e escalabilidade.

**Vamos supor que o Youtube utilize a arquitetura de microsserviços.**

**9. Tipos de Comunicação**

**Comunicação Síncrona**

Protocolo síncrono. HTTP é um protocolo síncrono. O cliente envia uma solicitação e espera uma resposta do serviço. Isso é independente da execução do código do cliente que pode ser síncrona (o thread é bloqueado).

**Comunicação Assíncrona**

Assíncrona (o thread não é bloqueado e a resposta acabará alcançando um retorno de chamada). O ponto importante aqui é que o protocolo (HTTP/HTTPS) é síncrono e o código do cliente somente poderá continuar sua tarefa quando receber a resposta do servidor HTTP.

**Publish/Subscribe**

Cada assinante precisa ser notificado de um evento específico uma vez, mas não deve ser notificado repetidamente do mesmo evento. O evento não pode ser considerado finalizado até que todos os assinantes tenham sido notificados. Uma vez que todos os assinantes tenham sido notificados, o evento pode ser considerado finalizado e deve desaparecer do canal de comunicação.

**10. Arquitetura Monolítica vs Microsserviços**

Arquitetura Monolítica é um sistema único, que roda em um único processo, uma aplicação de software em que diferentes componentes estão ligados a um único programa dentro de uma única plataforma.

Analogia sistema solar tudos planetas juntos é o Monolitico, sistema solar como conhecemos microsserviços.

**Vantagens da Arquitetura Monolítica**

* **Mais simples de desenvolver**: tudo em um único sistema.
* **Simples de testar**: é possível testar toda aplicação de um único lugar.
* **Simples de fazer o deploy para o servidor**: a alteração é simplesmente feita e pronto;
* **Simples de escalar**: como é só uma aplicação, se for preciso adicionar mais itens, é simplesmente ir adicionando o que for necessário.

**Desvantagens da Arquitetura Monolítica**

* **Manutenção**: a aplicação se torna cada vez maior de acordo com o seu tamanho, o código será cada vez mais difícil de entender e o desafio de fazer alterações rápidas e ter que subir para o servidor só cresce.
* **Alterações**: para cada alteração feita, é necessário realizar um novo deploy de toda a aplicação.
* **Linha de código**: uma linha de código que subiu errada pode quebrar todo o sistema e ele ficar totalmente inoperante.
* **Linguagens de programação**: não há flexibilidade em linguagens de programação. Aquela que for escolhida no início do projeto terá que ser seguida, sempre. Se o desenvolvimento de uma nova funcionalidade exigir outra linguagem de programação, existem duas possibilidades: ou todo o código é alterado ou a arquitetura do sistema precisará ser trocada.

**Vantagens dos Microsserviços**

* **Manutenção fácil e rápida**: tudo é feito de forma separada e mais rápida.
* **Flexibilidade**: é possível dividir em equipes para trabalhar de forma separada e totalmente independente em cada serviço.
* **Escalabilidade**: os deploys de cada microsserviço são totalmente independentes e mais rápidos;
* **Equipes autônomas:** Como os microsserviços são pequenos, pequenas equipes podem cuidar de cada um deles.
* **Serviços reutilizaveis:** Cada parte do sistema é desenvolvido como um quebra cabeça, portanto um serviço pode ser usado em diferentes aplicações.
* **Conteinerização**: Cada serviço configurado em seu próprio ambiente ou container, possibilitanto também criar cada microsserviço em uma linguagem de programação diferente.

**Desvantagens dos Microsserviços**

* **Arquitetura e planejamento cuidadoso:** é preciso um bom planejamento, para utilizar tudo que os microsserviços tem a oferentes, pois sem isso, o sistema pode não seguir está arquitetura.
* **Complexidade:** é um ponto a se tomar muito cuidado para que a organização sempre exista mesmo que novas features sejam implementadas no futuro.
* **Replicação de Código de resposta ou de infraestrutura:** o que existe de padrão em um serviço provavelmente existirá nos outros serviços também.
* **Latência:** Com muitos serviços comunicando alguns entre si e por fim com a API Gateway, pode ter uma demora na resposta de algumas requisições.
* **Complexidade na implementação de testes:**
* **Necessidade de um profissional DevOps:** DevOps está diretamente ligado a metodologias ágeis, que visa a entrega de softwares em larga escala.

**11. Projeto do Sistema com Microsserviços**

Para iniciar o desenvolvimento do sistema, foi decidido que primeiro ele precisava ser todo planejado e documentado, ao invés de usar a abordagem de desenvolvimento *code-first*(criação das entidades de domínio antes de planejar a estrutura de banco de dados).

Primeiro foi feito um *brainstorm* e tudo foi anotado, as ideias foram apresentadas a uma empreendedora da cidade, proprietária de uma cafeteria, ficou confirmado alguns problemas, como:

* O fechamento do caixa ser feito manualmente sem saber tudo que foi vendido.
* A cozinha demora para receber o pedido feito pelo garçom, pois é utilizado comandas de papel e o mesmo deve levar o papel para os cozinheiros.
* Devido à anotação ilegível, algumas vezes o pedido é preparado errado.
* Inclusive nem tudo que o cliente consome é anotado quando há muito movimento, pela liberdade que o cliente tem de pegar produtos nos freezers.
* No fechamento da conta do cliente, o atendente tem que ficar buscando os preços no cardápio.

A partir disso, algumas funcionalidades do sistema foram decididas, como possíveis soluções para os problemas mencionados:

* Pegar pedidos via dispositivo móvel.
* Cálculo da conta do cliente em tempo real.
* Cardápio de fácil acesso.
* Cálculo de troco.
* Fechamento automático do caixa.

Modelagem da estrutura e fluxo básico da aplicação, seguindo os requisitos a partir da entrevista com a empresária.

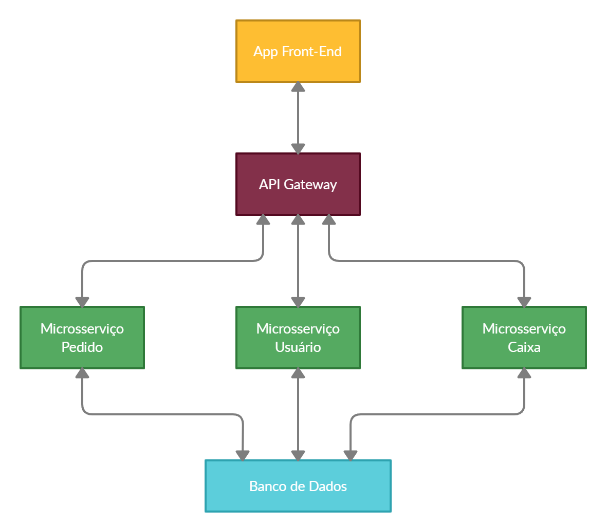
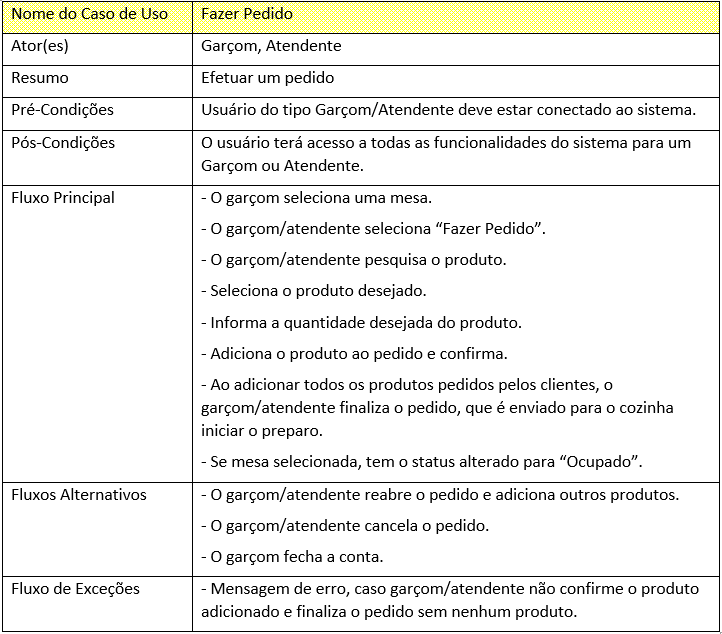


Diagrama de fluxo básico criado no Creately.

Devido ao grande escopo do projeto, pois serão desenvolvidos a aplicação de *back-end*, *front-end* e aplicativo *mobile*, o escopo inicial para está primeira parte do trabalho, será a criação do *back-end* da aplicação.

12. Casos de Uso

Baseado nas funcionalidades básicas do sistema, foram criados Casos de Uso, para se ter uma idéia melhor de como seria o fluxo de execução e uso do sistema.



Caso de uso “Fazer Pedido”

13. Diagrama de Entidade-Relacionamento

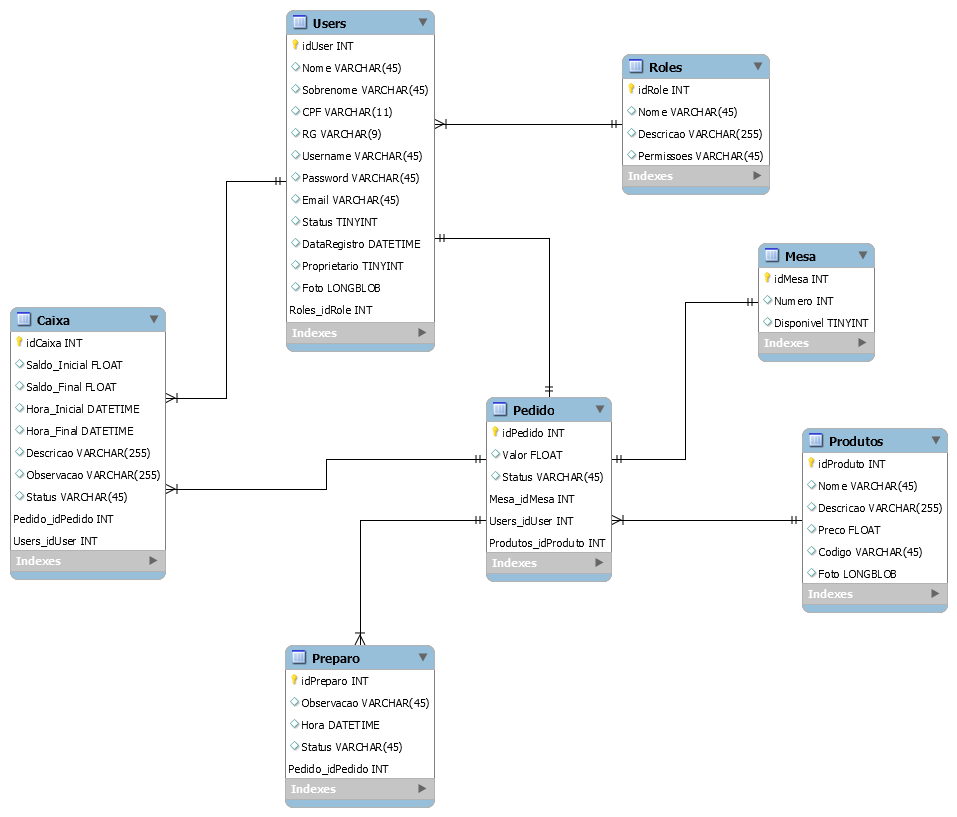


Diagrama de entidade relacional desenvolvido na ferramenta MySQL Workbench.

**14. Tecnologias e Ferramentas**

A IDE para o desenvolvimento utilizada foi o Visual Studio Community e para banco de dados o SQL Server Express 2019.

A linguagem de programação utilizada foiC#, para o desenvolvimento das APIs (microsserviços) o *framework* ASP.NET Core e para manipulação de dados o *Entity Framework.* Para o teste dos serviços foi utilizado a ferramenta Postman, que oferece um ambiente para execução e requisições de testes em APIs.

Com a escolha do *framework* e devido a grande popularidade por ser um repositório *open-source* no GitHub, exclusivo para .NET, o Ocelot foi utilizado como API Gateway para os microsserviços.

Basicamente Ocelot é uma coleção de *middlewares* que “consomem” os endpoints das APIs. A utilização de um API Gateway traz algumas vantagens como, roteamento de tráfego, *endpoints* unificados, autenticação, autorização, *load balance*, *caching*, *logging* e a composição robusta da API.

15. Implementação do Sistema com Microsserviços

Inicialmente foi criado uma “Blank Solution”, e a partir da mesma foram sendo adicionados os microsserviços. O primeiro a ser desenvolvido foi o serviço Usuário, foi criado um modelo do tipo ASP.NET Core Web Application com o nome de UsuarioService, a entidade (representa os dados da aplicação) Usuario foi definida e o contexto do banco de dados estruturado conforme Figura 3.

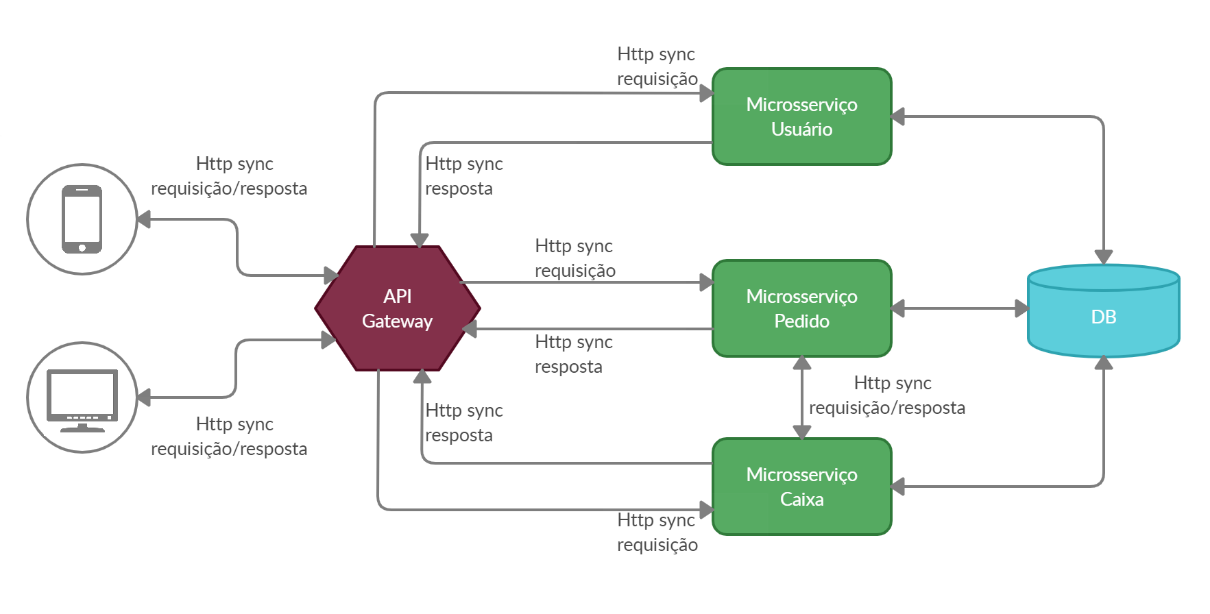


Figura 4. Diagrama de Comunicação entre os microsserviços.

Com as entidades criadas, foi instalado os pacotes do EF Core (ORM), utilizando o gerenciado de pacotes do NuGet e a *migration* inicial foi criada, gerando o banco de dados e as tabelas “usuarios” e “roles”, com um usuário administrador e as roles (do tipo administrador, gerente, atendente e garçom) pré-cadastradas, configurações definidas no arquivo de contexto do banco de dados da aplicação.

Com a estrutura do banco de dados pronta, foi simplesmente necessário criar o *controller* (define as rotas da *web* API) do Usuário, e já foi possível testar o CRUD do mesmo, para o teste foi utilizado a ferramenta Postman, que oferece um ambiente para execução e documentação de testes em APIs e suas requisições.

# 21. Conclusão

Foi importante a aprendizagem da arquitetura de microsserviços, conceitos de entidade e domínio, UML, API, autenticação, comunicação entre serviços utilizando um API Gateway e por fim, um aprofundamento em ORM (Mapeamento Objeto-Relacional) , causou grande dificuldade na implementação do *back-end* do sistema, assim como o aprendizado de como é feita a comunicação entre serviços.

Foram observados resultados muito positivos em relação à conversa com a empreendedora, sobre as questões relacionados ao seu empreendimento, esclarecendo diversas dúvidas durante o processo de projeção do sistema.

Por se tratar de um repositório relativamente novo, o Ocelot soluciona com facilidade e simplicidade no que condiz a sua proposta, que é unificar os *endpoints* de todos os serviços de uma aplicação em um só lugar, tornando-se cada vez mais popular.

Com a finalização do *back-end* do sistema, pôde-se concluir que o objetivo traçado para esta primeira etapa foi alcançado, pois o sistema disponibiliza todo o necessário conforme o planejamento inicial, o que implicará no bom desenvolvimento do *front-end* e aplicativo para dispositivos *mobiles*.