UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE INFORMÁTICA

CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

ANAURY NORRAN PASSOS RITO

A ECONOMIA DAS APIS E MICROSSERVIÇOS

RECIFE

2017

UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE INFORMÁTICA

CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

ANAURY NORRAN PASSOS RITO

A ECONOMIA DAS APIS E MICROSSERVIÇOS

Monografia apresentada ao Centro de Informática (CIN) da Universidade Federal de Pernambuco (UFPE), como requisito parcial para conclusão do Curso de Ciência da Computação, orientada pelo professor José Carlos Cavalcanti.

RECIFE

2017

UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

ANAURY NORRAN PASSOS RITO

A ECONOMIA DAS APIS E MICROSSERVIÇOS

Monografia submetida ao corpo docente da Universidade Federal de Pernambuco, defendida e aprovada em ___ de _____de____.

Banca Examinadora:

_____ Orientador

Prof. Drº José Carlos Cavalcanti

_____ Examinador

Prof. Drª Carina Frota Alves

“A evolução do Homem passa, necessariamente, pela busca do conhecimento.”

Sun Tzu

Agradecimentos

Este trabalho é derivado de muito incentivo, apoio e dedicação pessoal. Contudo, não poderia deixar de agradecer a todas as pessoas que me auxiliaram em todo esse período de graduação que não foi fácil, porém muito gratificante, uma vez que a cada desafio solucionado nos tornamos pessoas mais fortes e autoconfiantes.

Desse modo, primeiramente eu gostaria de agradecer a Deus pela vida, pelo seu amor, por ter me proporcionado essa oportunidade em minha vida e também, por não ter me abandonado nesse processo tão longo e difícil, me abençoando e sempre estando ao meu lado.

Agradeço a meu pai Severino Ramos Rito, que me deu diversos incentivos e exemplos de que podemos superar as dificuldades da vida e conseguir crescer, a meu irmão Sânderson Passos Rito, que também estava comigo durante esse caminho me dando apoio e me ajudando em tudo que fosse de seu alcance, e principalmente a minha mãeJania Passos Rito, que me deu forças, nunca me abandonou, sempre acreditou no meu potencial, mesmo quando nem eu mesmo acreditava em mim, além do carinho, amor incondicional e incentivo.

Em segundo lugar, sou grato aos professores que contribuíram com a minha formação pessoal e acadêmica, e, especialmente, aos professores José Carlos Cavalcanti e Carina Frota Alves, o primeiro por ser meu orientador, por acreditar, me auxiliar e compreender durante o desenvolvimento desse trabalho, e o segundo por ter aceitado fazer parte da minha banca examinadora e por ter mencionado, mesmo que de forma rápida, sua história acadêmica, a qual eu me identifiquei e me deu inspiração.

Da mesma forma, agradeço aos meus familiares e amigos tanto de escola quanto que fiz durante esse tempo na universidade.

Agradeço, também, à Universidade Federal de Pernambuco, em especial ao Centro de Informática, pela minha formação e estrutura concedida.

Resumo

Na sociedade moderna, tornou-se crescente o uso de tecnologia para facilitar na solução de conflitos. Dessa forma, as empresas têm a necessidade de investir em informatização para se manterem no mercado e suprir a alta demanda de atividades. Diante disso, algumas tecnologias se destacam em relação a outras. Nesse caso, as APIs e os Microsserviços. Com o uso desses componentes, as aplicações conseguem executar funções de maneira mais fácil e eficiente. Além de tornar possível a geração de receita ao negócio. Para se ter sucesso economicamente, usando APIs, é necessário conhecer pontos fundamentais do negócio, gerenciar as APIs e traçar planos monetários para se obter renda. Com base nesse cenário, buscando entender o que as APIs e os Microsserviços podem proporcionar aos negócios os quais são adotados, este trabalho realizou uma pesquisa teórica a fim de abordar esses impactos, trazendo conceitos e comparativos com estruturas já consolidadas, de maneira a ilustrar esses efeitos com mais clareza. Como resultado dessa análise, foi possível observar que o uso de APIs e Microsserviços tem muito a contribuir com o crescimento econômico de um negócio. Todavia, da mesma forma que o uso contribui positivamente, ele trás consigo dificuldades que precisam ser avaliadas com cautela para se obter um resultado positivo.

Palavras-chave: API, Microsserviço, Negócio, Monetização, Gerenciamento.

Abstract

In modern society, the use of technology to facilitate conflict resolution has become increasingly popular. Thus, companies have the need to invest in technology to stay in the market and supply the high demand for activities. Therefore some technologies stand out in relation to others. In this case, the APIs and the Microservices. Using these components, applications can perform functions more easily and efficiently. In addition to make the possibility to generate revenue for the business. For the economic success, using APIs, it is necessary to know fundamental business points, manage the APIs and trace monetary plans to obtain income. Based on this scenario, seeking to understand what the APIs and Microservices can provide to the businesses that are adopted, this work carried out a theoretical research in order to approach these impacts, bringing concepts and comparisons with already consolidated structures, in order to illustrate these effects more clearly. As result of this analysis, it was possible to observe that the use of APIs and Microservices has much to contribute to the economic growth of a business. However, in the same way that use contributes positively, it brings with it difficulties that need to be carefully evaluated to obtain a positive result.

Keywords: API, Microservice, Business, Monetization, Management.

Sumário

Lista de Figuras

Lista de Tabelas

Tabela de Siglas

Sigla	Significado
API	Application Programming Interface
CEP	Código de Endereçamento Postal
CPF	Cadastro de Pessoas Físicas
HTTP	Hypertext Transfer Protocol
REST	Representational State Transfer
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
UX	User Experience

1 Introdução

Nesse capítulo será apresentada uma introdução a este Trabalho de Graduação, cuja estrutura é composta pela Seção 1.1, onde será contextualizado o uso das tecnologias, na Seção 1.2 haverá uma motivação para a defesa desse trabalho, na Seção 1.3 terá os objetivos, gerais e específicos, que foram pretendidos atingir, a Seção 1.4 conterá a metodologia em que o trabalho foi desenvolvido e, na 1.5, a estrutura que os capítulos foram abordados.

1.1 Contexto

A cada dia é mais comum termos aplicações que usem a internet como meio de comunicação, sendo muitas vezes usadas por diversos equipamentos tecnológicos, ou seja, independem de qual plataforma está sendo consumida. Com a crescente integração entre aplicações e sistemas é normal pensar como isso acontece, e é nessa situação onde as APIs se fazem presentes. A sigla API vem do termo em inglês *Application Programming Interface* ou, no português, Interface de Programação de Aplicações. Uma *API* disponibiliza dados e serviços para canais de desenvolvedores parceiros, colegas, ou terceiros para a criação de aplicações [1].

Historicamente, a API da Salesforce, criada em 7 de fevereiro de 2000, foi dada como o primeiro caso de sucesso. Mas o “boom” só veio mesmo em 2004, quando a API de fotografias do Flickr foi disponibilizada. Esse lançamento fez com que a empresa crescesse rapidamente e, em menos de um ano, fosse comprada pela Yahoo. Dois anos depois o Facebook lança a sua API e, depois de alguns meses, o Twitter age da mesma forma. Ainda em 2006, a Amazon apresenta sua própria API voltada para o armazenamento de dados e, seis meses após, outra API, dessa vez focada em infraestrutura [2]. Em meados de 2010, as aplicações móveis e os serviços em nuvem começavam a ganhar força. E esse novo padrão de aplicação permitiu que novos modelos de negócios fossem apresentados, dando origem ao termo Economia das APIs [3].

Paralelamente os microsserviços ganhavam força, porém o histórico é um pouco mais breve. Em 2005 o professor Peper Rogers apresentou o termo “*micro web services*”, ou, em português, microsserviços de rede. Mas a expressão

‘Microsserviço’ só ganhou força em 2011, quando foi apresentada uma arquitetura baseada em componentes distribuídos [4]. O conceito de microsserviços, segundo Martin Fowler, da empresa ThoughtWorks, pode ser entendido como pequenos sistemas autônomos que se comunicam entre si [5].

Atualmente é frequente encontrar aplicações que usem as duas tecnologias, e no caso das APIs, elas são usadas em redes sociais, como Facebook ou Twitter, em sites de comércio eletrônico, como Amazon e Ebay, em pagamentos eletrônicos, como PayPal, e em diversas outras aplicações [6]; já os microsserviços são usados, novamente, em empresas como Amazon e Ebay, além de Netflix, Uber, SoundCloud e tantas outras [7], pois ambas trazem benefícios significativos para o negócio onde são inseridas.

1.2 Motivação

Nos últimos anos, o tema APIs está ganhando um notório foco em discussões em todo o mundo. O aumento da conectividade e o fácil acesso à internet são grandes estímulos para que empresas olhem para as APIs como uma forma de revolucionar seus produtos e serviços, com intuito de atrair mais usuários/consumidores e parceiros, criando, com a ajuda desse último, os importantes ecossistemas digitais. Associados às APIs, os Microsserviços permitem que os desenvolvedores, tanto internos quanto parceiros, criem aplicações que possam agregar mais valor ao produto/serviço sem haver uma grande interdependência [1].

De acordo com dados do site ProgrammableWeb, no primeiro quadrimestre de 2017, o número de APIs públicas em seu repositório ultrapassou a marca de 17000, e desde 2014 até a data da análise, foram adicionadas mais de 5900 APIs [8]. Com diversas aplicações voltadas aos clientes, sistemas de comércio eletrônico, redes sociais, sistemas de *Internet Banking,* o crescimento da Internet das Coisas e outros, é que esse número continue a crescer.

1.3 Objetivos

O presente trabalho de graduação tem como propósito mais amplo fazer uma revisão bibliográfica para analisar os conceitos principais sobre APIs e Microsserviços, buscando entender a Economia das APIs.

De forma substantiva, objetivos gerais foram:

- Entender conceitos sobre APIs e Microsserviços;
- Estudar modelos de negócio das APIs;
- Entender a arquitetura de Microsserviços, diferenciando-a de outros modelos de arquiteturas.

E como objetivos específicos:

- Mostrar a relação entre os Microsserviços e as APIs, e apresentar A Economia das APIs, dando uma visão geral sobre o Gerenciamento de APIs e nos Modelos de Negócios.

1.4 Metodologia

A pesquisa realizada para o desenvolvimento desse trabalho foi iniciada a partir de um levantamento bibliográfico com o objetivo de coletar conhecimentos sobre a estrutura e a dinâmica das APIs e Microsserviços. As informações foram coletadas de livros, artigos, vídeos, blogs e sites especializados, e de *reports* de importantes empresas.

Para esse trabalho foi utilizada a abordagem qualitativa, para auxiliar na interpretação das informações levantadas acerca das duas tecnologias. Além disso, o estudo foi iniciado a partir de uma perspectiva exploratória, a fim de proporcionar maior domínio sobre o conteúdo.

1.5 Estrutura do Documento

Esse trabalho foi divido de uma forma que fosse possível apresentar conceitos gerais sobre APIs e Microsserviços, fazer uma associação entre os dois temas e, por fim, apresentar modelos relacionados à Economia das APIs.

O documento está dividido em 6 capítulos, cuja a estrutura está descrita a seguir:

- **Capítulo 1, Introdução**: Contém a contextualização da Economia das APIs, a motivação para o desenvolvimento do trabalho, os objetivos

que se pretende alcançar, a metodologia utilizada e a estrutura do documento;

- **Capítulo 2, APIs**: Essa seção contém os principais conceitos relacionados às APIs;
- **Capítulo 3, Microsserviços**: Esse capítulo contempla conceitos relacionados aos Microsserviços assim como a relação entre eles e as APIs;
- **Capítulo 4, Arquiteturas de Software**: Essa parte apresenta alguns modelos de arquitetura de software e um comparativo entre os padrões mostrados;
- **Capítulo 5, A Economia das APIs**: Essa seção contém conceitos sobre o tema, uma introdução ao Gerenciamento de APIs e os principais modelos de negócio usados;
- **Capítulo 6, Considerações Finais**: Conclusão do trabalho e possíveis trabalhos futuros.

2 APIs

2.1 Surgimento

Segundo Robert Broeckelmann, do site Medium [9] e principal consultor da Levvel (levvel.io), o termo API começou a ser usado para descrever qualquer interface para livrarias ou módulos entre grandes sistemas. Porém, nos últimos anos, a expressão passou a se referir, mais comumente, a um estilo arquitetural de cliente-servidor baseado no protocolo HTTP, sem estado e de uso mais simples.

Antes das APIs, era mais comum o uso de *Web Services* para fazer a comunicação entre as máquinas. Os modelos arquiteturais de *Web Services* mais usados em meados da década de 90 até a popularização das APIs foram o SOAP– *Simple Object Access Protocol,* e, posteriormente, o REST- *Representational State Transfer*. O padrão REST tornou-se uma popular alternativa aos *Web Services* SOAP, principalmente em aplicações móveis. Ele veio como opção para simplificar e melhorar o paradigma anterior (SOAP) e rapidamente ganhou notoriedade no mercado, devido a sua maior simplicidade e por não se basear em estados definidos, como no SOAP, e sim em requisições, assim como não se preocupar com detalhes de desenvolvimento [10].

Posteriormente, as APIs, evoluídas do REST, chegaram para formalizar o estilo. Porém, elas podem ser baseadas tanto em REST quanto em SOAP. Uma grande diferença entre o contexto onde eram usados os *Web Services*, antes do surgimento das APIs, e o contexto onde elas são usadas agora, é em relação à exposição. Os *Web Services* são mais expostos dentro da empresa, enquanto as APIs têm a opção de expor pra fora dela, a uma base maior de usuários, sujeito a usos inesperados e inovadores. Permitindo, de acordo com o público atingido, a criação de diferentes estratégias de monetização sobre as funções expostas. Dessa forma, as APIs também podem ser tratadas como produtos, além de apenas uma forma de estabelecer comunicação entre aplicações [11].

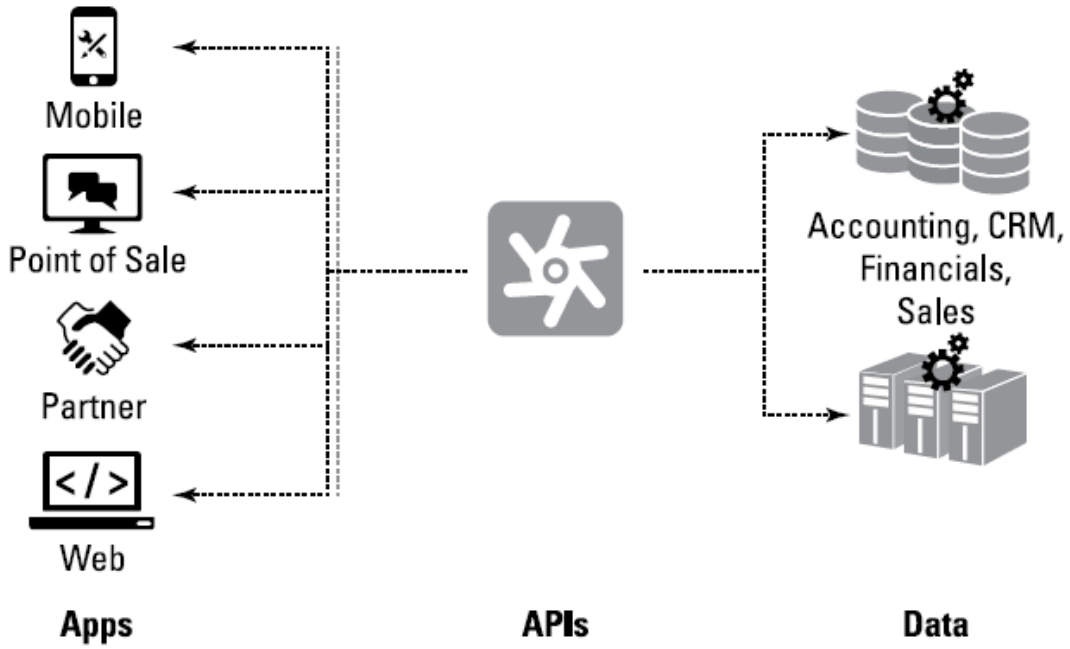
2.2 Definição

Assim como foi mencionado no Capítulo 1 desse trabalho, uma *Application Programming Interface*, no português, Interface de Programação de Aplicações ou simplesmente API é, segundo Daniel Jacobson, Greg Brail e Dan Woods, uma forma de dois programas se comunicarem, independente da plataforma usada, usando

uma linguagem comum [1]. Ou, de forma mais técnica, uma API é um conjunto de rotinas, ferramentas e protocolos para se criar uma aplicação [12].

Ela tem por objetivo conectar times internos, parceiros externos, processos e serviços para a criação de aplicações, como foi citado no Capítulo 1, ou seja, transmitir dados. Dessa forma, é possível imaginar uma API como uma “cola” digital (ver Figura 1) [13].

Figura 1: API como uma cola digital.



Fonte: API for Dunmies [13]

2.3 Entendendo a Cadeia de Valor de uma API

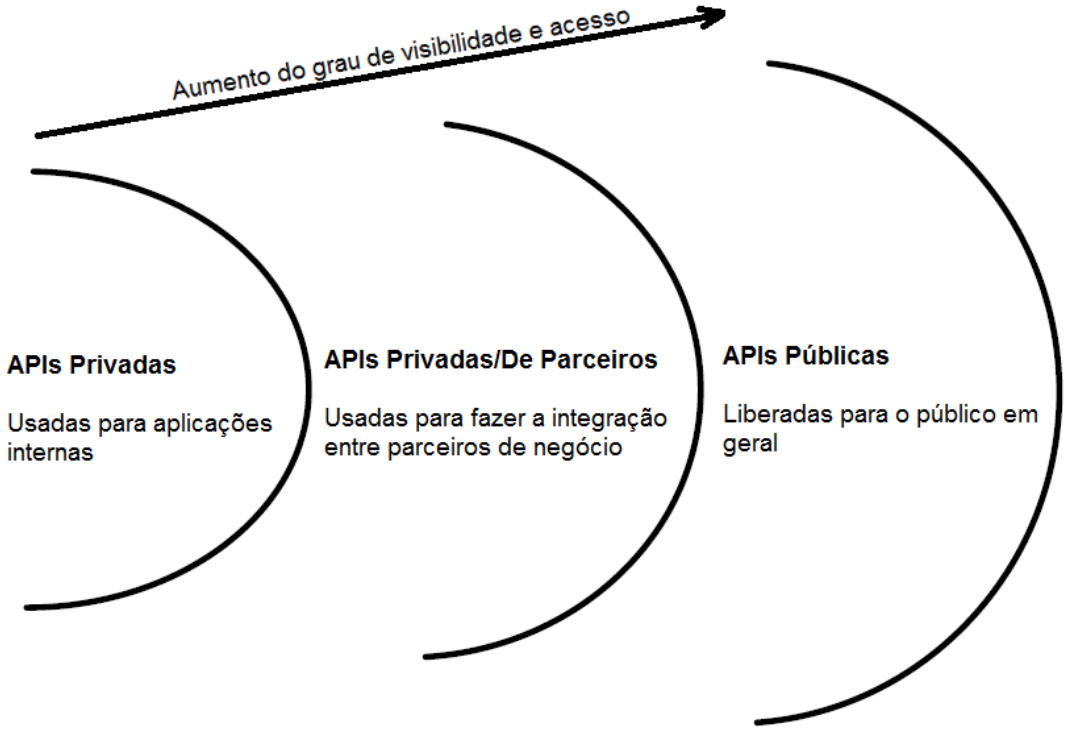
Para desenvolver uma API é necessário ter o conhecimento da sua cadeia de valor e os impactos que ela pode causar no negócio. Uma análise a ser feita previamente**,** busca-se entender os seguintes fatores [1]:

1. **Para quem os ativos comerciais que serão expostos?**
2. **Quem é o provedor da API?**
3. **Quem serão os desenvolvedores?**

4. **Quais aplicações farão uso dela?**

Analisando esses quatro pontos básicos, é possível obter uma noção de qual o segmento essa API vai se encontrar. Elas são comumente classificadas em três grupos, da forma: 1) Privadas, 2) Públicas ou 3) Privadas/De parceiros [14], onde cada uma delas tem públicos alvo, objetivos e cadeias de valor distintas.

Figura 2: Tipos de API



Fonte: Adaptada de API Management [14]

2.3.1 APIs Privadas

Como podem ser observadas na Figura 2, as APIs Privadas são as da camada mais interna, onde apenas os times internos têm acesso às funcionalidades. Nesse caso, essas APIs funcionam como uma interface entre a aplicação e o *Back-End* da empresa ou como meio de conectar diversos setores locais.

Esse segmento tem uma grande influência na Economia das APIs, além de ser, junto com as APIs de Parceiros (vistas na seção 2.3.2), a maioria em número de APIs, elas podem trazer, muitas vezes, mais benefícios ao negócio que uma API

Pública. Elas possibilitam transformar a estrutura negocial, reduzindo custos de operação, permitindo a colaboração entre times, simplificando a infraestrutura de TI, trazendo mais segurança aos dados compartilhados, e permitindo a modularização de componentes, entre outros benefícios [14] [15].

Respondendo os quatro pontos citados anteriormente, é possível ter uma ideia da cadeia de valor relacionada a uma API privada [1]:

1. **Para quem os ativos comerciais que serão expostos?**
   A empresa que usa uma API privada não tem interesse em expor seus ativos para fora dela. Logo, tais ativos só serão usados internamente;
2. **Quem é o provedor da API?**
   Por ser privada, geralmente o provedor da API é a própria empresa;
3. **Quem serão os desenvolvedores?**
   Seguindo o mesmo pensamento do item 2, os desenvolvedores são internos;
4. **Quais aplicações farão uso dela?**
   Geralmente aplicativos e sistemas criados para o uso interno, mas também podendo ser usadas para a criação de aplicações (de criação própria) para serem disponibilizadas ao público interno ou externo.

2.3.2 APIs Privadas/De Parceiros

Em seguida, existem as APIs de Parceiros (ou *Partner* APIs). Alguns autores, como por exemplo, Daniel Jacobson, Greg Brail e Dan Woods, autores do livro *APIs A Strategy Guide*, as encaixam como APIs Privadas, mesmo que também pareçam com as APIs Públicas, pelo fato de não serem usadas apenas internamente. Nesse caso, as funcionalidades e informações, além de serem acessadas pelas equipes internas, elas são disponibilizadas também para os parceiros de negócio [1].

Junto com as APIs Privadas, elas somam a maior fatia do mercado de APIs e têm como benefícios a criação de novos canais, a extensão de produtos e negócios, além de serem agregadores de novos valores ao produto/serviço. Fazendo a mesma associação que foi feita no item 2.3.1, as APIs de Parceiros também geram uma cadeia de valor:

1. **Para quem os ativos comerciais que serão expostos?**
   Uma empresa que possui uma API de Parceiros pretende expor ativos apenas para parceiros comerciais e internos;
2. **Quem é o provedor da API?**
   Nesse caso, o provedor é a própria empresa;
3. **Quem serão os desenvolvedores?**
   Nesse ponto, as APIs de Parceiros se parecem mais com as APIs Públicas, que serão vistas na seção seguinte, porém com uma pequena diferença: o público externo é restrito a afiliados;
4. **Quais aplicações farão uso dela?**
   As APIs de parceiros se comportam tanto como uma API Privada, quanto como uma API Pública, porém, nesse último caso, com um alcance menor. Portanto, elas podem ser usadas em aplicações internas e por aplicações externas, mas que tenham relação com o negócio do provedor.

2.3.3 APIs Públicas

Na outra extremidade estão as APIs Públicas, onde essas são desenvolvidas para expor funcionalidades e informações de um ou vários sistemas para pessoas fora do negócio, com o mínimo de arranjos contratuais.

Da mesma forma que o público externo pode acessar essa API, o interno tem a mesma possibilidade, podendo fazer o papel de uma API privada. Por terem um alvo maior, as APIs públicas têm grande potencial para agregar valor ao negócio sem investimento direto e reduzindo o custo no desenvolvimento de funcionalidades [16].

É dito que é nesse tipo onde existem as maiores chances de inovação [17]. Mas para tal objetivo ser alcançado, é necessário adotar medidas para chamar a atenção desses desenvolvedores e fornecer suporte e uma interface intuitiva para a execução das tarefas. Do outro lado, é indispensável que também sejam adotados métodos de segurança no acesso, mecanismos para resolver sobrecargas, além de tentar reduzir o impacto nos serviços quando houver alguma alteração na API [14].

Uma forma comum de se chegar a essa etapa, é seguir o caminho de uso de uma API Privada, compartilhar essa API com parceiros de negócios e, então, tornar essa API aberta.

Da mesma maneira feita nos dois outros casos de APIs, é possível analisar as mesmas questões e encontrar uma cadeia de valor para as APIs públicas [1]:

1. **Para quem os ativos comerciais que serão expostos?**
   Nesse caso os provedores da API desejam que elas alcancem uma maior escala de pessoas;
2. **Quem é o provedor da API?**
   Os usuários das APIs públicas podem usar as APIs disponibilizadas pela própria empresa ou usar APIs de terceiros;
3. **Quem serão os desenvolvedores?**
   Os desenvolvedores serão tanto do público interno, quanto do público externo;
4. **Quais aplicações farão uso dela?**
   Como, nessa ocasião, a API é pública, diversas aplicações podem ser criadas com vários objetivos distintos, internamente ou externamente.

2.4 Benefícios e desafios do uso

2.4.1 Benefícios

O desenvolvimento de uma API pode parecer uma tarefa complicada, porém o uso dessa tecnologia trará consigo diversos benefícios, tanto para os desenvolvedores, quanto para o empreendimento. Benefícios esses que variam de mercado para mercado, e de negócio para negócio. Todavia, é possível citar alguns que são comuns a qualquer negócio [18] [19]:

- **Eficiência:** Uma API permite que o conteúdo criado esteja automaticamente disponível para todos os canais;
- **Automatização:** As APIs permitem que as máquinas também lidem com as cargas de trabalho;

- **A criação de aplicativos:** Por ter acesso à informação e por ser uma interface, uma API possibilita que essas informações sejam usadas por aplicativos móveis;
- **Parceria:** Um caminho natural a se chegar a uma API Pública é sair de uma API Privada, serem feitas parcerias, passando pelo segundo tipo e, por fim, alcançar o título de Pública. Dessa forma, as APIs proporcionam que sejam feitas parcerias entre o provedor e os desenvolvedores;
- **Integração e experiência para o usuário:** As APIs permitem que seu conteúdo seja mais facilmente acessado por sites ou diferentes aplicações;
- **Personalização:** Usuários, funcionários, desenvolvedores, empresas, etc. têm a capacidade de personalizar as sessões com as informações e serviços que sejam mais interessantes para eles;
- **Transparência:** Para as APIs, todos os consumidores são mostrados da mesma forma, independente das especificações do dispositivo usado.

2.4.2 Desafios

Assim como qualquer outra tecnologia, o uso de APIs também traz desafios ao negócio ao qual elas são implantadas. Dessa forma, é viável citar [20] [21]:

- **Custo:** Desenvolver e fornecer recursos de uma API pode ser custoso em termos de desenvolvimento, tempo, manutenção e suporte;
- **Necessidade de conhecimento:** Para o desenvolvimento de uma API é necessário ter um bom conhecimento de programação e a curva de aprendizado pode ser maior, quando comparada com outras tecnologias;
- **Segurança:** Por uma API ser uma interface, ela está mais suscetível à conexão de aplicações e módulos os quais podem trazer potenciais ataques;
- **Lidar com diferentes públicos:** O nível de acesso, funções, documentação, etc., variam de acordo com o tipo de API escolhido;
- **Integração com sistemas legado:** Em empresas menores ou *start-ups*, onde há uma grande dependência de funções externas, o uso de APIs é um caminho natural. Porém isso muda quando uma empresa é dependente de

um sistema legado, é necessário mais esforço, tanto para integrar uma API ao sistema, quanto criar uma.

3 Microsserviços

3.1 Surgimento

A criação de aplicações monolíticas sempre teve suas dificuldades, principalmente com o seu crescimento. Com a evolução da tecnologia, o modelo de arquitetura monolítica tradicional começou a apresentar problemas. Devido a sua grande complexidade interna, tornou-se difícil definir a granularidade de um projeto. O fato de todas as funcionalidades serem agregadas ao sistema principal, faz com que o sistema cresça, aumente de complexidade e aumente a dependência entre as funções.

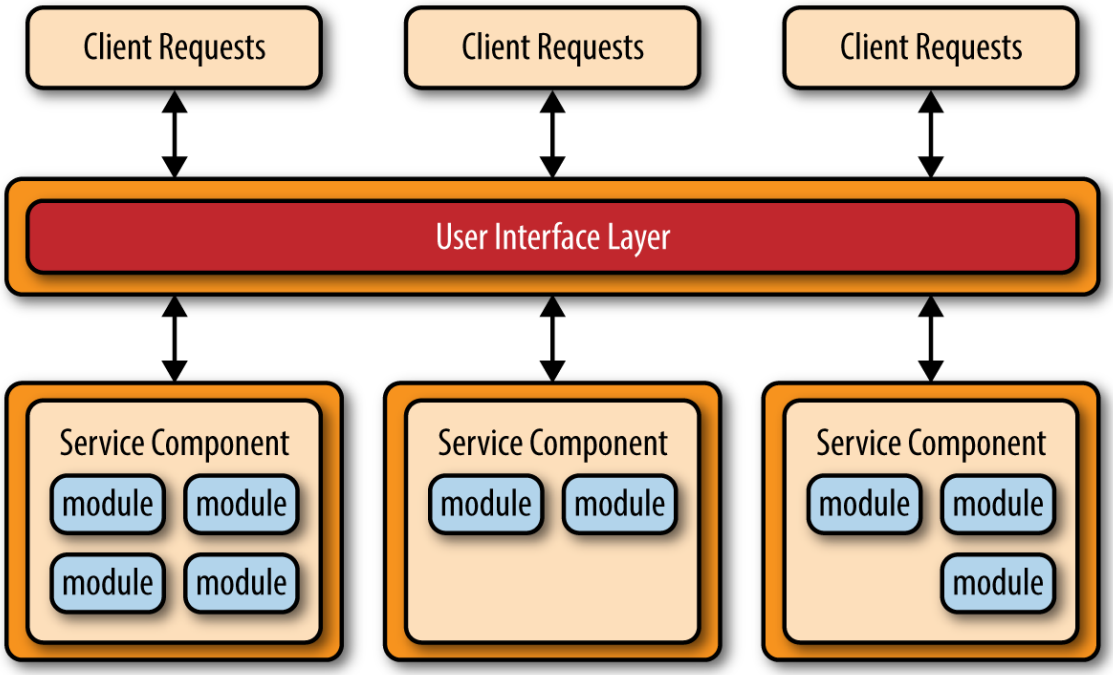
Como consequências de uma forte interdependência entre funções, é comum de, o sistema como um todo, apresentar dificuldades na mudança de tecnologias, no desenvolvimento de funções, na escalabilidade do sistema, na instalação e na configuração do sistema, etc. Além do que foi citado, há também: problemas de impacto das mudanças feitas, adversidade nas entregas continuas e problemas em módulos isolados que afetariam a aplicação como um todo. Porém, um grande ponto positivo dessa arquitetura seria a arquitetura externa simplificada [22].

A arquitetura de microsserviços veio para facilitar na criação e desenvolvimento de aplicações em torno do negócio, decompondo o sistema maior em partes menores que se comunicam entre si. Nesse caso, há uma troca válida entre simplicidade na arquitetura interna do componente, com uma arquitetura externa mais complexa. A simplicidade interna proporciona diversos benefícios que possibilita a diminuição do tempo de desenvolvimento e, consequentemente, na oportunidade de entrega de novas funcionalidades em menor espaço de tempo [23].

3.2 Definição de Microsserviço

Segundo Eberhard Wolff, autor do livro *Microservices: Flexible Software Architecture* [24], Microsserviço é um conceito de modularização, com propósito de dividir um sistema maior em módulos menores. Nesse caso, esses componentes podem ser desenvolvidos separadamente, em diferentes tecnologias, com seus próprios recursos e suporte, trazendo maior independência entre os módulos. Em outras palavras, microsserviços são pequenos sistemas autônomos que podem compor um maior e estabelecer comunicação entre si.

Figura 3: Modelo básico da Arquitetura de Microsserviços



Fonte: Software Architecture Patterns [25]

3.3 Características das arquiteturas baseadas em Microsserviços

A Figura 3 é uma representação básica de uma arquitetura baseada em microsserviços (o foco do Capítulo 4). Assim como qualquer outro tipo de arquitetura, ela tem suas próprias características, as quais proporcionam vantagens e desvantagens ao negócio (vistas na seção 3.5). Podem ser vistos como traços principais [26]:

1. **Componentização:** Como característica mais aparente, é observado que, diferente de outras formas arquiteturais, a arquitetura baseada em Microsserviços propõe que as funcionalidades de uma aplicação sejam divididas em serviços menores, contanto que haja comunicação entre eles, e que cada um possa ser executado independente de outro;
2. **Contexto limitado:** Do inglês *Bounded Context*, esse princípio é um dos mais importantes nesse tipo de arquitetura. É fundamental que cada componente execute apenas uma função, de forma satisfatória, do sistema maior;

3. **Desenhado visando falhas:** Como consequência da componentização, a estrutura dos Microsserviços tem que ser confiável e tolerante a falhas. Caso ocorra alguma falha em algum componente, não haverá impactos em outros componentes, e ocorrendo, será de mínima importância;
4. **Tratados como produtos:** Também em decorrência da componentização, cada componente é visto, na maioria das vezes, como um produto único, que ao fim do desenvolvimento, quando o item é entregue, tem-se o objetivo como concluído e grupo de desenvolvimento é desfeito;
5. **Gerenciamento de dados descentralizados:** Diferente da arquitetura monolítica, os Microsserviços têm características de sistemas distribuídos. Faz-se necessário o gerenciamento de transações entre eles, e o acompanhamento de diferentes bancos de dados. Caso seja usado apenas um banco de dados compartilhado, isso violaria a independência entre os componentes;
6. **Comunicação entre componentes:** Contrapondo-se à arquitetura monolítica, onde a comunicação entre as “partes” do sistema é feita de forma mais simples, na arquitetura baseada em Microsserviços é importante que seja estabelecido um meio de comunicação entre os componentes, assegurando que não haja dependência entre eles;
7. **Complexidade:** Por ser um sistema distribuído, e devido à divisão em componentes, é natural que esse tipo de arquitetura necessite de mais recursos para realizar diversas atividades. Por ter, muitas vezes, diversos processos sendo executados, é fundamental desenvolver mecanismos de execução de testes, monitoramento, balanceamento, comunicação e determinar uma infraestrutura para que esses mecanismos sejam executados;
8. **Design evolutivo:** Pelo fato dos elementos serem componentizados e possuírem independência entre si, essa arquitetura permite que a aplicação se comporte de forma evolutiva, onde novas funcionalidades podem ser adicionadas e funções obsoletas podem ser removidas. Dessa forma, é possível realizar mudanças de forma que haja mínimo impacto entre os elementos.

3.4 Benefícios e Desafios da abordagem de Microsserviços

Diante das características apresentadas na seção anterior, diversos impactos são trazidos por essa estrutura ao negócio, e cada um traz consigo benefícios e desafios que precisam ser ponderados no momento em que é pensado sobre a adesão da estrutura arquitetural. Dessa forma, serão apresentadas, nos itens a seguir, essas consequências.

3.4.1 Benefícios

Para se adotar um modelo de arquitetura é intuitivo que ela tenha que trazer benefícios ao sistema como um todo. E com essa estrutura de arquitetura não é diferente. Esse formato é capaz de trazer vantagens, tanto técnicas, quanto negociais, agregando valor ao resultado final.

Entre os benefícios, é possível citar [23] [26] [27]:

- **Forte modularização:** Como consequência da característica de componentização, os módulos dos microsserviços desempenham suas funções havendo comunicação com outros componentes apenas por meio de interfaces, diminuindo a dependência entre eles;
- **Facilita a substituição das partes e o desenvolvimento em diferentes tecnologias:** Por serem serviços distintos, com mínima interação com outros componentes e por serem acoplados a uma interface, essa arquitetura facilita a troca de módulos, caso haja mudanças, ou, seja necessária haver a troca, além de permitir que cada serviço seja desenvolvido em diferentes tecnologias;
- **Códigos menos complexos:** Diferindo dos sistemas monolíticos, onde o sistema executa todas as funções e pode haver milhões de linhas de código para que isso seja realizado, na arquitetura de microsserviços os módulos executam apenas a função a eles designada (característica de Contexto Limitado, vista na seção 3.3) e, consequentemente, diminuindo o tamanho e a complexidade dos códigos;
- **Falhas isoladas:** Como cada componente tem dependência mínima de outro, caso haja algum problema, esse acontecimento vai ter maior impacto somente no módulo em que aconteceu;

- **Desenvolvimento contínuo:** O desenvolvimento de partes especializadas em realizar apenas uma função facilita que o sistema esteja em constante crescimento;
- **Facilidade de escalar os serviços:** Os microsserviços são oferecidos através de interfaces ligadas a uma rede, a qual pode ser acessada de diversos locais e dispositivos.

3.4.2 Desafios

Por outro lado, se os provedores do negócio decidirem adotar essa estratégia de arquitetura, é necessário que eles tenham em mente que além dos benefícios que a empresa pode ter, serão trazidos, também, alguns desafios que eles devem ficar em alerta para que a estrutura não se torne um grande problema. Como aspectos a serem observados, são destacados os seguintes pontos [23] [26] [27]:

- **Complexidade:** A partir do momento em que a decisão de usar uma arquitetura distribuída é feita, o aumento na complexidade é inevitável. E com a abordagem de microsserviços não é diferente. Mesmo que a estrutura dos componentes seja, de certa forma, mais simples que nos sistemas monolíticos, ao agregar muitas dessas estruturas, a complexidade se torna maior;
- **Teste:** Se há, de um lado, nos testes unitários, menor complexidade, do outro lado, nos testes de integração, a dificuldade aumenta. Pois, diferentemente dos sistemas monolíticos, onde a integração é feita em poucas aplicações, no modelo de microsserviços, a integração é feita em diversos componentes;
- **Consistência de dados:** Como cada componente é autônomo e responsável por seu próprio banco de dados, a consistência dos dados pode se tornar um problema;
- **Monitoramento:** Da mesma forma que nos testes, os microsserviços trazem benefícios, eles também trazem questões sobre monitoramento a serem avaliadas. Verificar a saúde de um microsserviço isolado é, relativamente, fácil. A grande preocupação é avaliar a saúde do sistema como um todo, além de monitorar falhas e seus possíveis impactos;

- **Latência e congestionamento na rede:** Isso se torna possível devido ao uso de grandes números de componentes, onde esses se comunicam através de requisições apenas por meio da rede;
- **Versionamento e atualização:** Por estarem em partes separadas, cada componente pode sofre atualizações a qualquer momento, possibilitando a incompatibilidade entre versões, além de, por exemplo, bibliotecas usadas por diversos serviços serem atualizadas e se tornarem incompatíveis com eles;
- **Tamanho do componente:** É necessário ter noção da funcionalidade do módulo, para, caso seja preciso, dividir o componente em partes menores, com objetivo de que cada parte execute uma função diferente;
- **Manter a coordenação entre os módulos:** Como cada parte possui independência, é preciso que seja coordenada a execução dos componentes, visando cumprir o objetivo final de forma satisfatória;
- **Custo:** Por se tratar de uma arquitetura distribuída, onde cada componente tem sua própria infraestrutura, a manutenção desses serviços pode se tornar mais custosa.

3.5 A Relação entre Microsserviços e APIs

No Capítulo 4 são apresentadas três topologias distintas da arquitetura de microsserviços e uma delas é o modelo onde é usada uma API para fazer a integração entre os componentes e o *Front-End* da aplicação.

É válido deixar claro que a comunicação entre os componentes é feita, na maioria das vezes, através de uma API REST, usando o padrão de API *Gateway* (melhor apresentado na seção 4.4 e 5.3.3), do inglês API *Gateway Pattern*, como forma de arquitetura entre eles [28], a qual está contida no modelo apresentado na seção 4.4.1.1.

É importante entender que um API Gateway de microsserviços é um mecanismo que faz a conexão entre o *Front-End* da aplicação e os módulos. Porém eleé mais que apenas um ponto de entrada. O mecanismo é responsável por orquestrar as requisições entre os componentes junto à aplicação, além de abstrair a complexidade dos componentes, criar pontos de acesso aos módulos, prover segurança e fazer a comunicação entre as partes, usando uma linguagem comum [29].

4 Arquiteturas de Software

Existem diversos padrões de arquitetura de software no mercado atualmente, todos com suas vantagens, desvantagens e situações as quais cada tipo se aplica melhor. Nesse capítulo, serão apresentados quatro modelos de arquitetura: em camadas, baseada em eventos, baseada em *microkernel* e a baseada em microsserviços, seguindo o livro *Software Architecture Patterns*, de Mark Richards [25], para então, ser mostrado um quadro comparativo entre eles, assim como situações onde cada um pode ser encaixado.

4.1 Arquitetura em Camadas

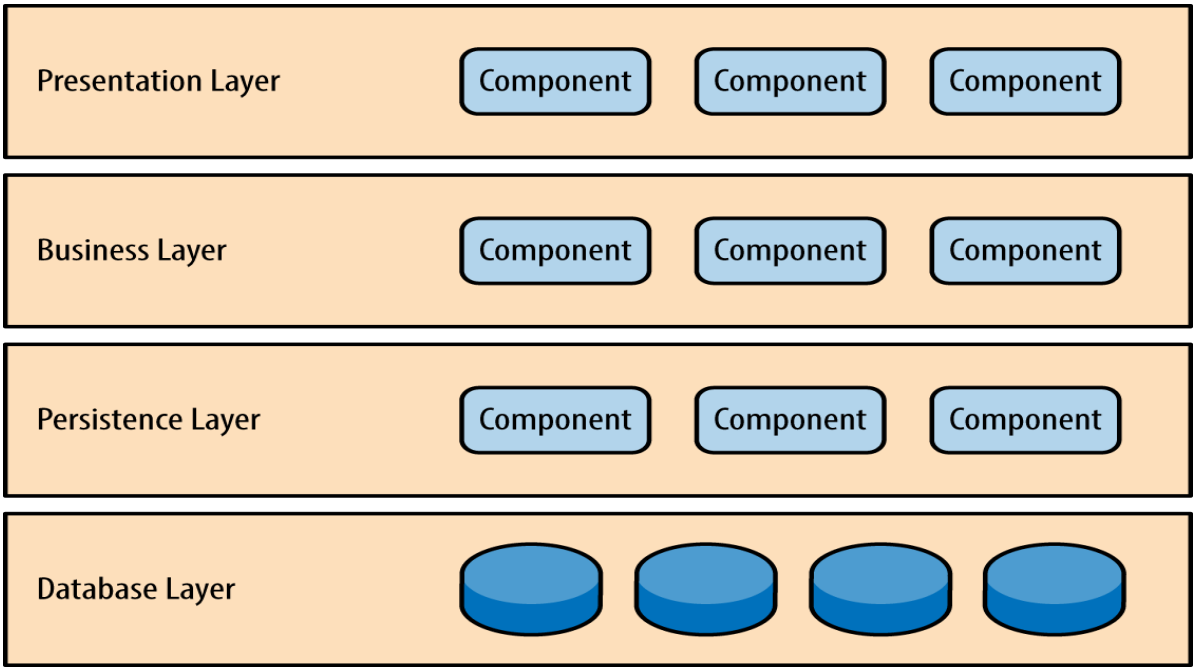
4.1.1 Visão Geral

O modelo em camadas [25] é o tipo mais comum de arquitetura. Esse padrão é usado em diversas aplicações e amplamente conhecido pelos desenvolvedores, designers e arquitetos de software. Portanto, ela se torna uma escolha natural para muitas outras aplicações.

O padrão em camadas é dividido em níveis horizontais, onde cada nível é especializado em um papel da aplicação. Essas camadas são empilhadas verticalmente, uma sobre a outra e a comunicação é explicita e fracamente acoplada [30]. Não há uma quantidade pré-determinada de camadas. Porém é comum que haja quatro: apresentação, negócio, persistência e banco de dados, assim como é apresentado na Figura 4. Mas esse número varia de acordo com a complexidade do negócio e como ele foi estruturado, podendo ser maior ou menor.

Cada nível possui uma responsabilidade específica no trabalho, e cada um necessita satisfazer as requisições do negócio. De forma mais simples, por exemplo, a camada de negócio não tem a necessidade de saber como é feita a persistência dos dados, ela só está responsável por pegar os dados do nível de persistência, aplicar as lógicas de negócio e repassá-los para a camada acima.

Figura 4: Modelo em Camadas



Fonte: Software Architecture Patterns [25]

Um dos pontos mais fortes dessa arquitetura é separar os componentes por interesses, fazendo assim que cada componente, de uma camada específica, se preocupe, apenas, com a lógica pertencente àquele nível. Devido a essa limitação de escopo, esse tipo de classificação facilita na criação de modelos de funções para cada camada, assim como torna mais fácil o desenvolvimento, a governança, a execução de testes e a manutenção das aplicações.

Para criar um modelo em camadas, é importante conhecer o conceito de “abertura” e “fechamento” da camada, pois a aplicação desse conceito é que vai definir como será o relacionamento e o fluxo das requisições entre elas. No padrão de camada fechado, as requisições deverão, obrigatoriamente, passar de camada em camada, impossibilitando o acesso direto entre camadas com níveis intermediários. Por exemplo: a camada de apresentação não pode fazer uma requisição diretamente à camada de persistência, sem passar pela de negócio. O acesso imediato traria mais velocidade na transação, porém iria de encontro ao princípio de isolamento da camada. O qual significa que cada nível é independente, com conhecimento mínimo ou nenhum de outra camada, e, geralmente, mudanças feitas em uma camada só haverá impacto nela, ou em uma associada.

Dessa forma, seguindo o raciocínio do exemplo, se houver uma mudança feita na camada de persistência, impactará, além da própria camada e da de negócios, também terá efeito sobre o nível de apresentação, criando mais dependência entre elas. A “abertura” da camada faz sentido quando o numero de níveis é maior e existem camadas de serviços compartilhados, podendo, dessa forma, uma categoria fazer acesso direto à outra, passando através da camada de serviços compartilhados.

4.1.2 Considerações e Análise

A arquitetura baseada em camadas é um padrão já consolidado e um bom modelo de arquitetura inicial para aplicações. Porém, esse modelo tende a se tornar em uma aplicação monolítica, mesmo que os módulos de negócio e apresentação sejam divididos em unidades menores.

De um lado, por se tratar de uma arquitetura relativamente simples de se desenvolver, e pelo critério de divisão das camadas serem por especialidades, a execução dos testes, o desenvolvimento da aplicação e a manutenção da camada, se tornam tarefas mais simples.

Por outro lado, esse tipo de arquitetura possui dificuldades a responder de forma rápida às mudanças no ecossistema, pela falta de flexibilidade nas camadas. Além de que, para que mudanças sejam feitas, é necessário rearranjar o sistema inteiro. Há problemas, também, tanto no desempenho (nesse quesito há ineficiência por ser necessário que as requisições passem por todas as camadas) quanto na escalabilidade, pelo sistema possuir deficiência na granularidade dos módulos.

4.2 Arquitetura Baseada em Eventos

4.2.1 Visão Geral

A arquitetura de software baseada em eventos [25] é um modelo distribuído e assíncrono usado principalmente em grandes aplicações. Esse padrão é composto por elementos independentes que processam eventos assincronamente.

A arquitetura consiste de duas topologias principais, a mediadora (*Mediator Topology*) e a corretora (*Broker Topology*), ambas com características e objetivos distintos.

4.2.1.1 Topologia Mediadora

A topologia mediadora é útil quando o evento tem mais de uma etapa e quando é necessário haver orquestração, para determinar a ordem dessas etapas, no processo do evento. Os elementos mais comuns nesse tipo de arquitetura são: a fila de eventos, o mediador, os canais e os processadores, cada um com uma função específica. Porém, para entender essas funções e como é executado o processo, é necessário distinguir os dois tipos de evento:

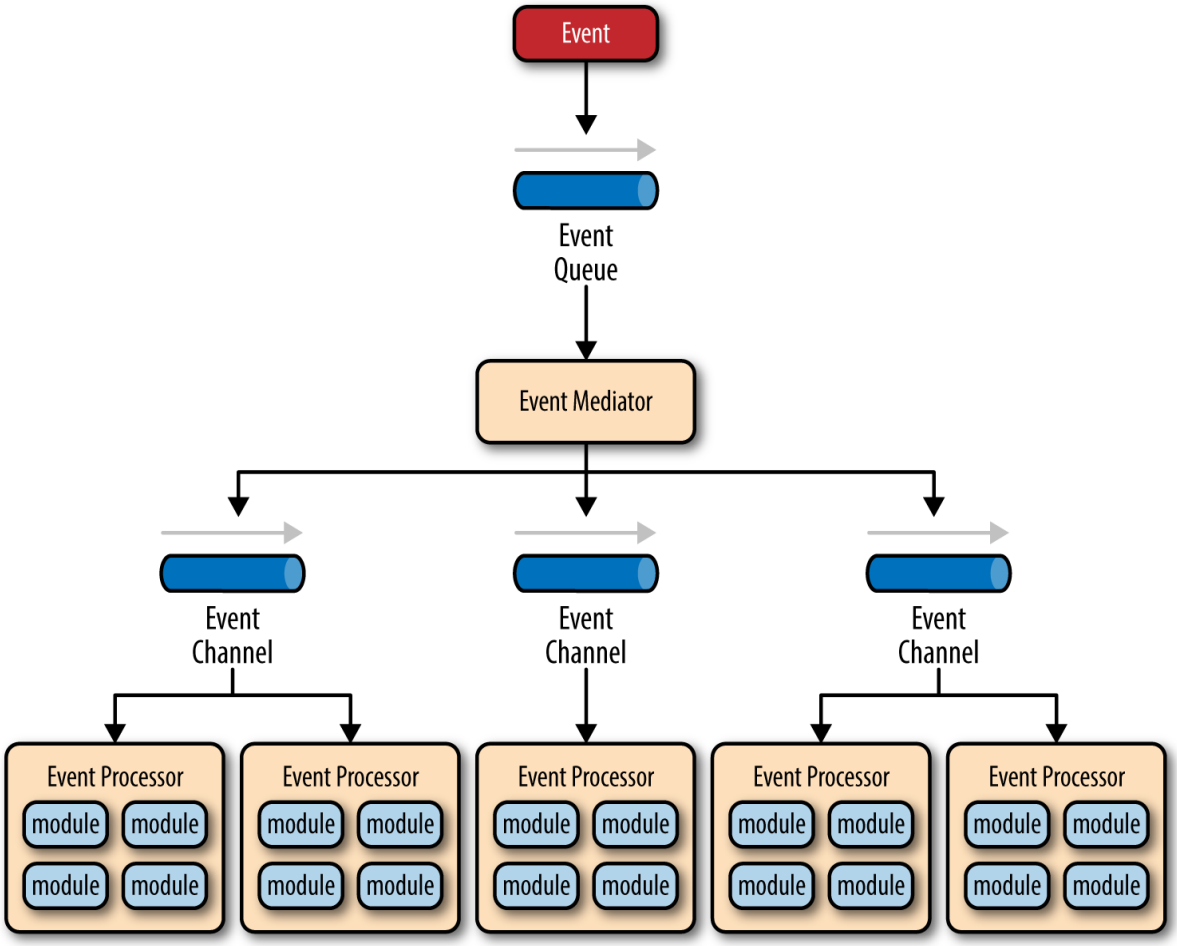
1. **Evento inicial:** é o evento original, recebido pela fila;
2. **Evento de processamento:** é o evento gerado pelo mediador e recebido pelos outros componentes.

Havendo essa diferenciação entre os eventos, é possível explanar a função de cada componente:

- **A fila (*event queue*)**: transporta o evento ao mediador;
- **O mediador (*event mediator*)**: é o componente responsável por fazer a orquestração dos passos contidos no evento inicial. Para cada passo é gerado um evento de processamento e, esse, é repassado para um canal;
- **O canal (*event channel*)**: ele tem como função repassar os eventos de processamento para o processador de forma assíncrona;
- **O processador (*event processor*)**: esses contêm os módulos e a lógica da aplicação para poder executar os eventos recebidos.

Dessa forma, o fluxo e o modelo básico dessa topologia podem ser ilustrados na Figura 5.

Figura 5: Topologia Mediadora



Fonte: Software Architecture Patterns [25]

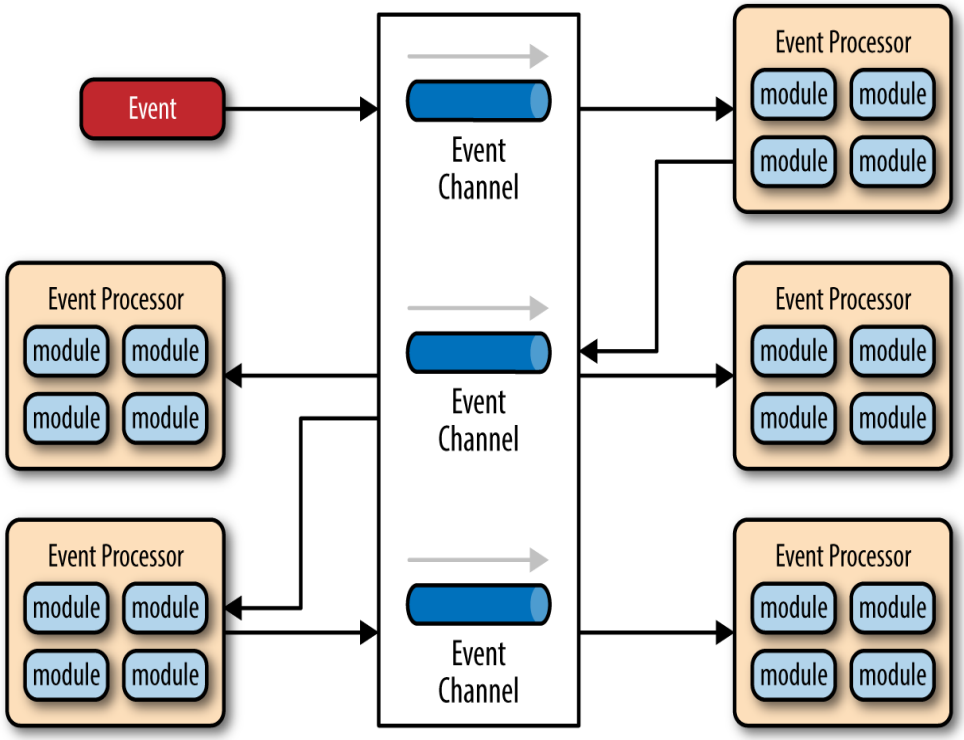
4.2.1.2 Topologia Corretora

A Topologia Corretora é o segundo modelo de arquitetura baseada em eventos. Nesse formato, a principal diferença entre ela e a Topologia Mediadora se dá pelo fato de, na Corretora, não haver um mediador central para orquestrar o evento inicial. Esse padrão é útil quando o fluxo de eventos é mais simples, e quando não é necessária tanta orquestração entre os processadores.

Nessa arquitetura, o fluxo de mensagens é distribuído em cadeia para os processadores através de um corretor. De forma mais simples, um evento chega ao corretor, onde esse distribui para os processadores que tem interesse nesse evento. Depois de manipular o evento, o processador libera o resultado trabalhado de volta no corretor, o qual repassará, novamente, aos processadores que vêem importância

no evento processado, e o fluxo continua até não haver mais interessados, como pode ser visto na Figura 6.

Figura 6: Topologia Corretora



Fonte: Software Architecture Patterns [25]

Como componentes-chave da topologia, é válido dar atenção ao:

- **O corretor (*broker*):** esse componente contém todos os canais que são usados nos fluxos dos eventos;
- **Os canais (*event channels*):** semelhantes aos da Topologia Mediadora, porém podem ser baseados em filas de mensagens, tópicos (onde as mensagens são separadas por assunto), ou uma combinação entre as duas;
- **Os processadores (*event processor*):** equivalentes aos da topologia anterior.

4.2.2 Considerações e Análise

Esse modelo de arquitetura é relativamente difícil de desenvolver, principalmente pela sua natureza distribuída e pelo seu comportamento assíncrono. Como consequência do forte desacoplamento entre os componentes, é difícil manter uma transação entre eles, por isso, para ser implantada, é necessário verificar quais processadores são independentes e quais devem possuir um meio de comunicação entre eles. Ainda pelo fato do alto desacoplamento e pela falta de sincronia entre os elementos, o desenvolvimento, os testes, a manutenção e a governança dos componentes, são prejudicados.

Do outro lado, por serem componentes separados, definidos para a execução de um único propósito e por serem assíncronos, há maior agilidade na execução de operações, assim como são favorecidos o desempenho e a escalabilidade [31].

4.3 Arquitetura Baseada em *Microkernel*

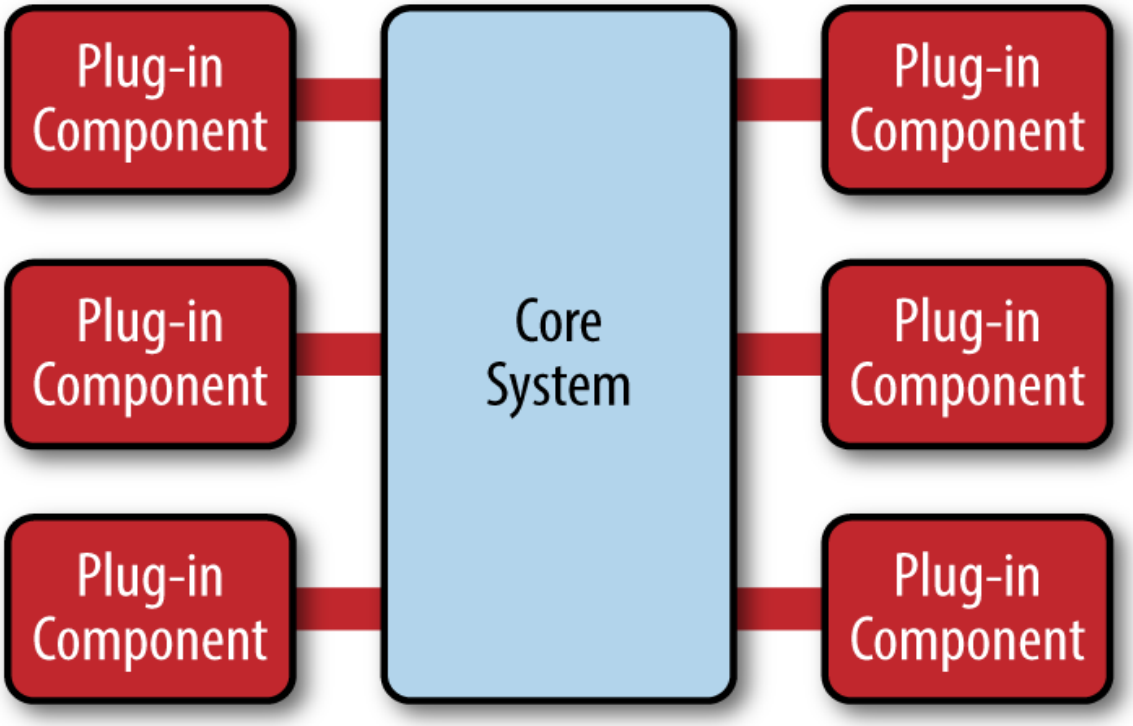
4.3.1 Visão Geral

A arquitetura baseada em *Microkernel* [25] tem uma ideia de um modelo *plug-in,* ou seja, é um padrão natural para aplicações baseadas em produtos, onde há versões delas disponíveis para *download* e funcionalidades que podem ser adicionadas ao negócio principal.

Esse padrão, ilustrado na Figura 7, é composto de dois tipos de componentes, o sistema principal (*core system*) e os módulos adicionais (*plug-in components*), e a lógica da aplicação é dividida entre eles, provendo extensibilidade, flexibilidade e isolamento das funcionalidades.

Nesse modelo, o módulo principal possui as funcionalidades mínimas para que o sistema opere. Do ponto de vista do negócio, o módulo principal muitas vezes só contém os códigos gerais, sem tratamento ou regras de ocasiões especiais. Já os módulos adicionais são geralmente independentes entre si e podem conter códigos para situações ocasionais ou novas funcionalidades; dessa forma, isso contribui para que o sistema, como um todo, fique mais complexo. O padrão não tem interesse em saber como o componente principal e os adicionais serão desenvolvidos, o ponto principal é que os adicionais sejam independentes entre si.

Figura 7: Modelo da Arquitetura *Microkernel*



Fonte: Software Architecture Patterns [25]

4.3.2 Considerações e Análise

Um grande ponto positivo dessa arquitetura é o fato dela poder ser combinada com outras arquiteturas, como por exemplo, a arquitetura de microsserviços. Dessa forma, não é necessário que toda a aplicação seja desenvolvida seguindo esse padrão. Isso traz mais flexibilidade ao negócio.

Porém, para aplicações que são tratadas como produtos, esse modelo é o mais indicado para se ter como padrão inicial, principalmente para aplicações, que têm em seu negócio a perspectiva de adição de funcionalidades ao longo do tempo.

Caso a arquitetura não esteja mais satisfazendo os requisitos do negócio, há a possibilidade de refatoração ou mudança para outro modelo, devido ao fato da aplicação já ser, naturalmente, dividida em partes.

Por sua natureza evolutiva em funcionalidades e pelo seu comportamento “plugável”, esse modelo consegue responder de forma satisfatória a mudanças no ambiente e nos módulos. Pelo fato dos componentes serem isolados e

personalizáveis, os testes podem ser executados de formas distintas, e o desempenho atinge níveis adequados ao negócio. Porém, pelos componentes serem muito dependentes do sistema principal, a escalabilidade e a facilidade de desenvolvimento são prejudicadas.

4.4 Arquitetura Baseada em Microsserviços

4.4.1 Visão Geral

O modelo de arquitetura, que é apresentado nessa seção, vem ganhando espaço nas empresas por ser uma alternativa viável às aplicações monolíticas.

A evolução dessa arquitetura se deu para resolver problemas associados a outros modelos. Pelo lado da estrutura monolítica, evoluiu para corrigir o problema na adição de funcionalidades, para buscarem seguir o conceito das entregas de funções de forma contínua, e para facilitarem o desenvolvimento e os testes; e, do lado da arquitetura SOA, mesmo tendo seus benefícios ao negócio, o padrão de microsserviços busca melhorar a complexidade interna, facilitando o desenvolvimento, reduzindo o custo (tanto de desenvolvimento, quanto de manutenção e testes) e a facilitando no entendimento da estrutura [25].

Independente da topologia escolhida é necessário ter um entendimento sobre os principais conceitos associados ao modelo geral. Esses conceitos já foram apresentados no Capítulo 3 – Microsserviços, porém serão retomados, de forma mais simplificada, para seguir com o raciocínio. Dessa forma, é importante ter conhecimento de que cada unidade tem uma função distinta, e que são entregues como produtos únicos. Outro ponto a levar em consideração, é o fato da sua natureza distribuída, ondecada componente é independente dos outros.

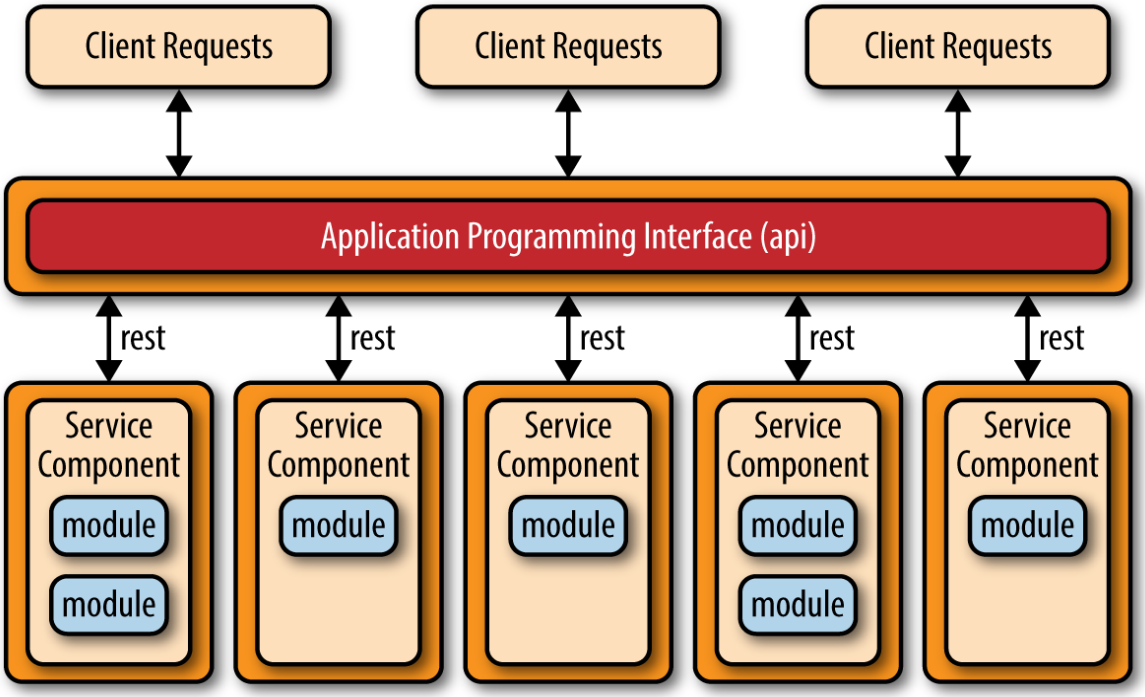
Existem diversas formas de uma arquitetura ser implementada; porém, três modelos têm maior relevância entre todos os outros [25]: I) os baseados em aplicações REST, II) os baseados em mensagens centralizadas e III) os baseados em API REST.

4.4.1.1 Topologia baseada em REST API

Essa topologia tem maior utilidade em aplicações web que disponibilizam suas funções através de uma interface de programação de aplicações (API). Esse modelo (mostrado na Figura 8) é constituído de diversos componentes

independentes, com funções distintas, os quais são acessados por uma interface REST através de uma API.

Figura 8: Topologia baseada em API REST



Fonte: Software Architecture Patterns [25]

Todos os componentes devem se comunicar com os outros, assim como com as aplicações e seu próprio banco de dados, em tempo real. Assim sendo, todos eles devem possuir uma interface, e é nesse ponto onde se dá a importância de uma API. Uma API REST proporciona um modelo lógico e simplificado para construir interfaces entre os microsserviços. Por esse motivo, as APIs devem ser baseadas em mensagens, para não afetar a interoperabilidade dos serviços quando ocorrer mudanças ou atualizações, e dirigidos à hipermídia, significando que também podem ser enviadas descrições de possíveis ações, além de, apenas, dados, dessa forma maximizando o baixo acoplamento [32] [33].

4.4.1.1.1 API Gateway

Para essa topologia e para a próxima apresentada (seção 4.4.2) é interessante apresentar pontos importantes sobre um API *Gateway* e como ele influencia nesse modelo.

Um API *Gateway,* mencionado na seção 3.5, é um dispositivo de rede que atua de forma que as APIs não tenham que interagir diretamente com os clientes. Ele é ponto central onde estão todas as abstrações das funcionalidades e as administra através de políticas [34]. Dessa forma, todas as requisições dos clientes terão que passar, obrigatoriamente, por eles.

Por estarem conectados a todas as interfaces (também chamadas de APIs) dos microsserviços, esses *Gateways* são usados comumente para fornecer três funções essenciais à arquitetura [35]:

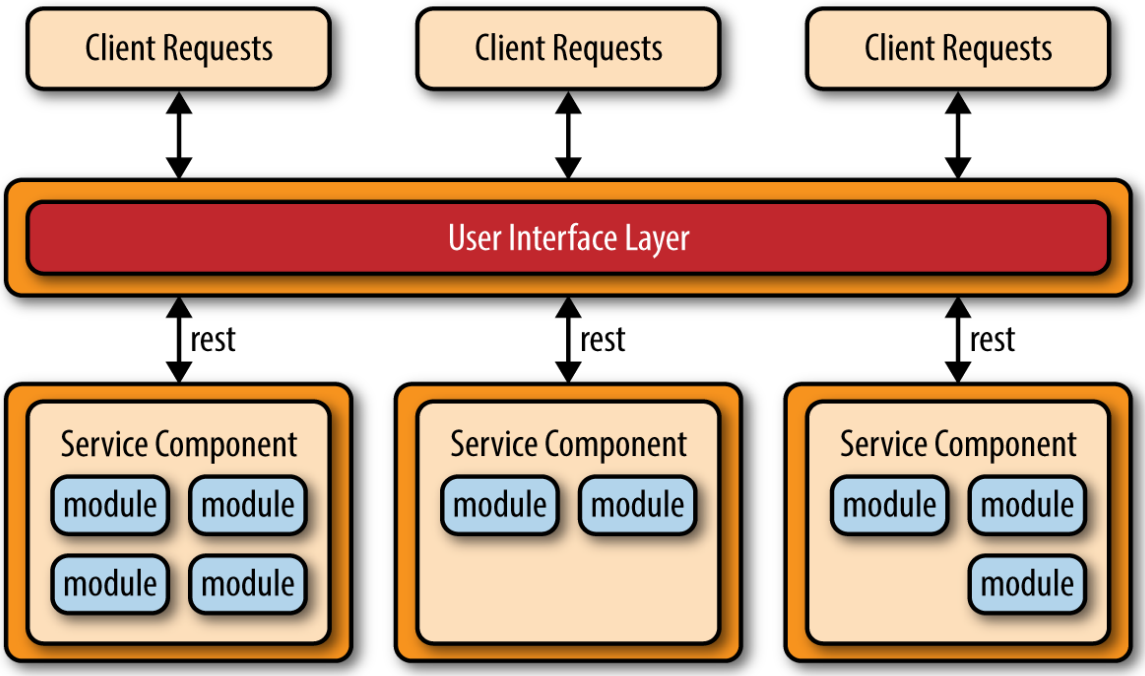
1. **Segurança:** Pelo alto desacoplamento, tendo como consequência uma grande liberdade entre os microsserviços, é importante que grandes aplicações, com diversos módulos, todas as interações com clientes externos sejam feitas através de APIs, assim como na forma de comunicação entre os componentes, pois, para cada elemento, outro componente também é um “cliente externo”. E um *gateway* é usado para proteger os pontos de conexão dessas APIs;
2. **Orquestração:** Como já foi referido no Capitulo 3 sobre os microsserviços, eles são desenvolvidos para exercer uma função única, de maneira satisfatória. Portanto, é imprescindível que haja um mecanismo de organização nas requisições às interfaces dos componentes, para evitar o problema de múltiplas chamadas;
3. **Roteamento:** Nesse ponto, o API *Gateway* tem a função de esconder a complexidade de roteamento entre a aplicação-cliente e os microsserviços.

4.4.1.2 Topologia baseada em aplicações REST

Esse modelo difere do anterior no quesito de que, contrário ao uso de uma API REST, as requisições dos clientes são recebidas por aplicações *web*. Como é

ilustrada na Figura 9, a interface entre os clientes e os serviços é feita por aplicações *web* que, remotamente, acessam os serviços através de uma interface REST. Interfaces REST são interfaces que se comportam de acordo com o modelo, de forma assíncrona, sem estados definidos e sem se preocupar com detalhes de desenvolvimento dos componentes [10]. Outra diferença desse padrão para o modelo da seção 4.4.1 é no tamanho dos componentes, nessa topologia, os componentes tendem a serem maiores, pois podem apresentar mais funções do negócio. Esse padrão é mais comum em aplicações de pequeno e médio porte, com menos complexidade.

Figura 9: Topologia baseada em Aplicações REST



Fonte: Software Architecture Patterns [25]

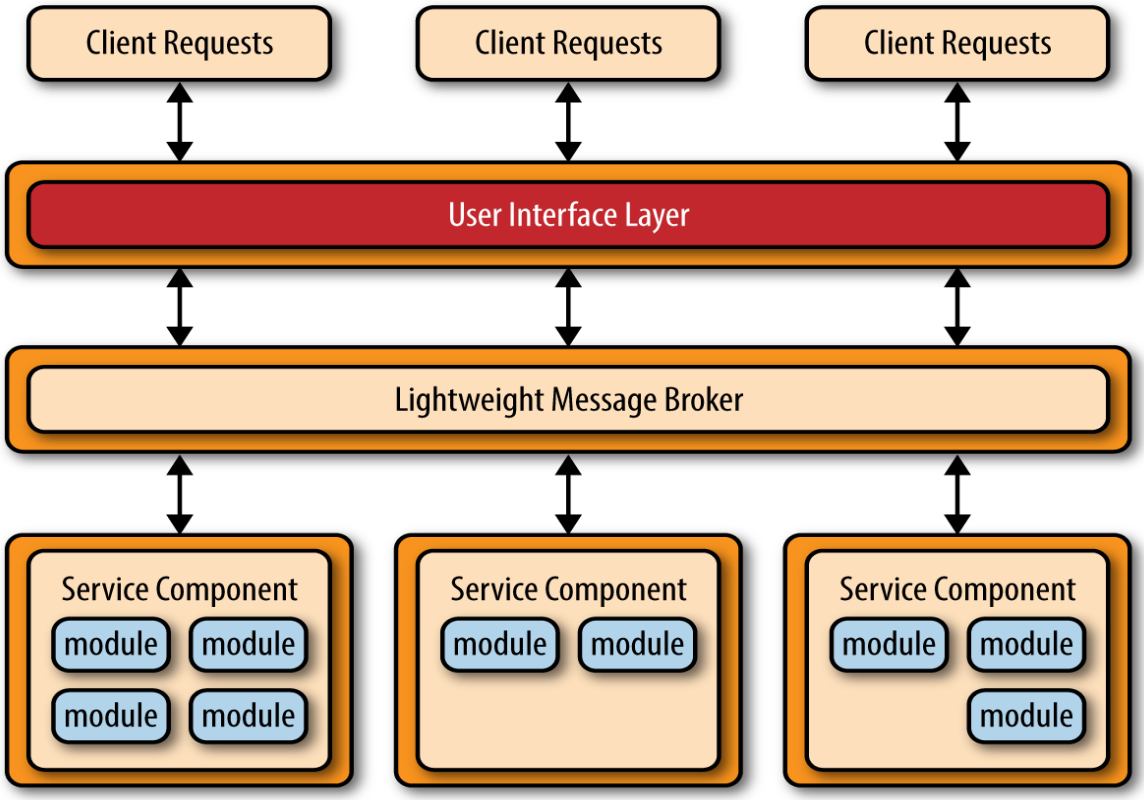
4.4.1.3 Topologia baseada em Mensagens Centralizadas

Outra topografia comum de microsserviços é o modelo baseado em mensagens centralizadas (Figura 10). Esse formato é semelhante ao apresentado anteriormente, porém há uma troca no acesso à camada de interface. Nesse padrão, o acesso é feito através de um corretor de mensagens centralizado, no lugar de uma

aplicação *web*. O corretor não executa nenhuma função de orquestração ou roteamento, ele é só um meio de acesso remoto aos componentes. Esse modelo pode ser usado em casos onde seja necessário um controle mais sofisticado na camada de transporte entre a interface e os componentes.

Como benefícios sobre a topologia baseada em aplicações REST, é dada a importância aos mecanismos de fila de mensagens, assim como na assincronia entre elas, no monitoramento, no tratamento de erros e na escalabilidade.

Figura 10: Topologia baseada em Mensagens Centralizadas



Fonte: Software Architecture Patterns [25]

4.4.2 Considerações e Análise

Devido à maioria das funções de um sistema baseado na arquitetura de Microsserviços estar contida em módulos menores, esse modelo pode resolver diversos benefícios de facilidade de desenvolvimento, escalabilidade, robustez e suporte, encontrados em aplicações monolíticas ou em aplicações baseadas em SOA. Outros benefícios, assim como já foi citado no Capítulo 3, são a possibilidade

entrega de funções de forma contínua e a possibilidade de executar testes, fazer e reagir a mudanças em componentes isolados, sem afetar o sistema todo.

Todavia, por ter a mesma natureza distribuída da Arquitetura baseada em Eventos (seção 4.2), ela compartilha dos mesmos problemas de manutenção, disponibilidade, autenticação, criação de contratos e governança dos componentes.

4.5 Quadro comparativo dos modelos

Para facilitar a visualização das características comuns entre as arquiteturas, sem levar em consideração as aplicações onde cada tipo se enquadra melhor, a seguir é mostrada na Tabela 1, um quadro ‘característica/modelo’ avaliando oito pontos usando os seguintes critérios:

1. **Agilidade**: Capacidade de reação a mudanças no ecossistema;
2. **Testes na unidade**: Facilidade de execução de testes nos componentes;
3. **Testes no sistema**: Facilidade de execução de testes no sistema como um todo;
4. **Desempenho**: Capacidade de execução das funções de forma satisfatória;
5. **Escalabilidade**: Facilidade de escalar o sistema;
6. **Desenvolvimento**: Facilidade de desenvolver o sistema;
7. **Mudança**: Possibilidade de fazer mudanças sem afetar o sistema todo ou outros componentes;
8. **Entrega**: Possibilidade de entregas de funções de forma contínua.

Usando a seguinte legenda:

- **+**: representando como favorável;
- **-**: representando como desfavorável.

Tabela 1: Quadro comparativo das Arquiteturas

	Camadas	Eventos	*Microkernel*	Microsserviços
Agilidade	-	+	+	+
Testes na unidade	+	+	+	+
Testes no sistema	+	-	-	-
Desempenho	-	+	+	-
Escalabilidade	-	+	-	+
Desenvolvimento	+	-	-	+
Mudança	-	+	+	+
Entrega	-	+	+	+

Fonte: Adaptada de Software Architecture Patterns [25]

Assim como seus pontos positivos e negativos, cada modelo tem melhor aplicação em diferentes situações [36]:

- **Camadas**
  - Aplicações novas e que devem ser construídas rapidamente;
  - Negócios onde há necessidade de réplica do departamento de TI;
  - Times com pouca experiência.
- **Baseada em Eventos**
  - Sistemas assíncronos;
  - Aplicações onde existe pouca interação entre os módulos;
  - Interfaces de usuários.
- ***Microkernel***
  - Ferramentas usadas por um grande público;
  - Aplicações *Plug-In*
  - Aplicações que recebem atualizações frequentes e com funções principais fixas.
- **Microsserviços**
  - Aplicações com pequenos componentes;
  - Negócios com rápido desenvolvimento de aplicações;
  - Times espalhados.

Analisando a Tabela 1, e sem levar em consideração as situações onde cada modelo tem melhor aplicação, é possível visualizar que o modelo de Microsserviços, junto com o Baseado em Eventos, são os modelos que trazem mais fatores positivos ao sistema como um todo. Levando em consideração a arquitetura de Microsserviços, a qual é dada maior ênfase nesse trabalho, comparada ao modelo em Camadas, comumente encontrado em aplicações nos dias de hoje, além do primeiro modelo ser um modelo viável financeiramente quando comparado ao segundo, ele trás diversos benefícios sobre o outro e só peca em um ponto, quando comparados: na execução de testes no sistema por completo.

Dessa forma, é válido renunciar um pouco na simplificação da complexidade externa, por uma maior simplicidade na complexidade interna, levando em consideração as vantagens e desvantagens que esses modelos podem trazer.

5 A Economia das APIs

5.1 API *Disruption*

O conceito de “disrupção” foi trazido ao contexto da inovação pelo professor de Harvard, Clayton Chistensen [37]. Ele descreve inovações que criam um novo mercado, mais acessível ao consumidor, e desestabilizam as empresas líderes do ramo [38].

As APIs têm grande capacidade de promover essa disrupção nos negócios, e estão por trás da maioria das disrupções digitais. Elas estão aumentando a conectividade e provendo negócios que eram, antes, inesperados, mudando a forma de interação com o mundo. Ainda, permitem cortar custos de desenvolvimento em diversas plataformas, por não ser mais necessário desenvolver o sistema inteiro.

Todavia, a maior forma de disrupção que as APIs vêm trazendo está na forma em que as pessoas refletem sobre seus atos. Pensamentos como “*por que não existe um aplicativo para fazer isso?*”, ou “*por que eu não posso fazer isso do meu telefone?*”, mostram que as pessoas estão, cada vez mais, dependentes da tecnologia para resolver as necessidades do dia-a-dia.

Existem diversas razões para as APIs proporcionarem tantas mudanças aos negócios, e para ilustrar esses motivos, são aqui apresentados três deles [39]:

1. **Voltada ao mercado:** As APIs provêm velocidade no desenvolvimento de aplicações e no consumo dessas pelos usuários;
2. **Alavancagem:** Com APIs, empresas menores, ou *startups,* conseguem produzir tanto quanto empresas maiores;
3. **Produção e lucro sobre o sistema principal:** Os negócios podem entregar suas funções ou ativos centrais mais longe.

As APIs estão em todo o lugar, mesmo que as pessoas não esperem por isso. Contanto que os consumidores comuns tenham suas necessidades supridas, eles não se preocupam com a tecnologia que está sendo usada por trás, e, em muitos casos, são APIs. Como já foi mencionado no Capitulo 1 desse trabalho, as APIs são comumente intituladas como as colas digitais.

5.2 Definição

A 3scale [40] define a Economia das APIs como uma consequência dos efeitos econômicos do uso das APIs para prover acesso direto aos sistemas e processos dos provedores dessa API. Este processo se soma à capacidade de inovar e fornecer benefícios a desenvolvedores internos e externos, parceiros e clientes. Assim, a economia resultante habilita a criação de diversas aplicações com potencial para mudar o negócio.

Existem três papeis fundamentais para o desenvolvimento da economia, são eles [41]:

1. **Os fornecedores da API**: Os fornecedores, ou provedores, são os que querem expor as informações. Eles têm tanto a função de detectar as informações importantes, quanto como criar, disponibilizar e gerenciar a API. Além de decidir o público que terá acesso a elas;
2. **Os consumidores da API:** Essas são as pessoas que vão usar as informações das APIs. Elas têm a função de criar novos negócios apoiados nas APIs;
3. **Os consumidores finais:** Esse papel não cria nada de novo ao consumir as informações das APIs, porém é a parte que impulsiona a economia, ou seja**,** são os clientes, os usuários finais.

5.3 O Núcleo do Gerenciamento de APIs

A partir do momento em que uma API é criada, é imprescindível o uso de uma ferramenta de gerenciamento de APIs para fazer a sua administração. Uma ferramenta desse tipo auxilia na disponibilização da API ao público em que ela deseja atingir, provendo ativos únicos. A ferramenta fornece recursos para manter os desenvolvedores engajados, informações comerciais, *analytics*, segurança e proteção. Além disso, uma plataforma de gerenciamento de APIs auxilia na divulgação do negócio através de canais digitais, na adoção por parceiros, na monetização dos ativos e na otimização de investimentos [14].

5.3.1 Definição

É possível entender o Gerenciamento de APIs como uma prática adotada por uma empresa para fazer o gerenciamento das suas APIs. Dentre os recursos que fazem parte deste gerenciamento estão inclusos [39]:

- Fornecer meios para catalogar as APIs, com dados como assunto, descrição, versão, recursos, etc.;
- Prover formas de atuação no catálogo, expondo as APIs ao público, com segurança, direitos de consumo e diferentes versões;
- O gerenciamento de APIs deve fornecer um sistema de registro para a utilização da API, contendo as características dessa API, incluindo métricas e indicadores de desempenho. Esses dados podem ser usados para a criação de melhorias, no monitoramento e na monetização das APIs expostas.

Uma plataforma de gerenciamento de APIs é uma forma de executar a prática de gestão de forma mais unificada. Para prover as funcionalidades, citadas nos itens anteriores, essa ferramenta possui componentes básicos com funções específicas, os quais serão apresentados nos próximos itens.

5.3.2 Registro de API

Como foi mencionada anteriormente, a prática de gerenciamento de APIs deve ter meios de fazer um inventário das APIs, de forma que os consumidores possam verificar suas características e especificações, como funcionalidades, descrição, capacidades e exposição de dados.

O registro de API tem por função auxiliar na organização do ciclo de vida da API, controlando versões, melhorias ou inatividade, também mantendo os dados a controle do *Gateway* (seção 5.4.3), e possibilitando e fornecendo informações sobre a API, o Portal do Desenvolvedor (seção 5.4.4) prover o catálogo aos consumidores. Em outras palavras, esse componente é desenvolvido para receber as informações sobre as APIs [39].

5.3.3 API *Gateway*

Assim como já foi citado no capítulo anterior, um API *Gateway* é um mecanismo que controla as requisições e orquestra as funcionalidades. Seu papel é expor as APIs da organização aos seus consumidores provendo controle de requisições, segurança, gerenciando o tráfego, e apresentando um conjunto uniforme de APIs e capturando informações úteis às finalidades do provedor.

Os API *Gateways* funcionam colaborativamente com os Registros, consumindo informações sobre as APIs e como eles devem as expor. Com as informações capturadas, eles repõem os Registros com novas informações das APIs a fim de manter a imagem de como elas estão sendo usadas para auxiliar nas decisões gerenciais [39].

5.3.4 Portal dos Desenvolvedores

O componente final da arquitetura básica de gerenciamento de API é o portal do desenvolvedor. É o meio de comunicação entre o provedor e os desenvolvedores. O portal serve de interface entre as pessoas e as APIs, proporcionando experiência de qualidade, ferramentas e recursos para o desenvolvimento de aplicativos e facilidades de envolvimento com a organização.

Para melhorar a experiência do usuário, o portal deve ter funções para facilitar o entendimento das APIs, como por exemplo, documentação para o consumidor conhecer a API antes de começar o desenvolvimento real, para auxiliar no gerenciamento das aplicações que interagem com a API, e mecanismos que promovam *feedback* das métricas capturadas pelo *Gateway*, para uso em monitoramento ou monetização.

Dessa forma, o Portal do Desenvolvedor fornece as informações ao desenvolvedor para facilitar o entendimento das APIs, assim como suporte para entender como as aplicações estão associadas às APIs [39].

5.4 Monetização de APIs

Com o uso de APIs, fica mais fácil compartilhar dados e serviços com diversas aplicações. Porém, há ativos digitais que podem render receita ao negócio. Como exemplo é possível citar [14]:

- Ativos os quais existem à disposição do pagamento pelos desenvolvedores, como mapas, dados analíticos, etc.;
- Serviços digitais, como funções de verificação em geral (por exemplo, CPF, CEP...), de mensagens...;
- Mecanismos de pagamento onde há uma taxa para cada transação;
- Inclusão de produtos em *marketplaces*.

5.5.1 Conceitos importantes

Para monetizar uma API é importante ter conhecimento sobre alguns conceitos relacionados ao tema [14]:

1. **API como produto (API *product*):** As APIs podem ser vendidas como produtos, produtos esses compostos por uma coleção de recursos relacionados à API, e agregados em um novo produto e disponibilizados à comunidade. O provedor pode criar produtos diferentes combinando APIs para usos distintos, contrariamente a ideia de criar uma lista de recursos relacionados à mesma API [42] [43];
2. **Pacote de APIs (API *package*):** Um pacote de APIs é um conjunto de API *products* que o provedor quer monetizar;
3. **Plano de tarifa (*rate plan*):** Esse plano especifica o modelo de cobrança feito sobre o uso da API, ou o compartilhamento de receitas. Ele pode ser pré-pago ou pós-pago, com formas de cobrança com tarifas fixas, variáveis, *freemium* (onde existem serviços, na maioria das vezes o núcleo principal, sem tarifas e outros que são tarifados), combinados com o desenvolvedor, ou, até mesmo, sem tarifas relacionadas. E os planos dependem do modelo usado para monetizar a API [44];
4. **Documentos de cobrança (*billing documents*):** Uma vez uma API ter sido monetizada, é necessário gerar registros de cobrança aos consumidores de forma regular, para tornar possível que o provedor da API, após estudos, faça ajustes nos documentos para aumentar ou diminuir as taxas e/ou o compartilhamento de receitas;

5. **Relatórios de Monetização (*monetization reports*):** Para auxiliar na apresentação e conciliação dos dados e informações, é importante a criação de relatórios. Como por exemplo:
    - **Relatório de cobranças:** Esse documento deve apresentar os detalhes relacionados às cobranças aos desenvolvedores;
    - **Relatório de saldo pré-pago:** Esse relatório deve mostrar o saldo de reabastecimento do desenvolvedor que pagou previamente pelo serviço, para poder conciliar os pagamentos e os serviços disponibilizados;
    - **Relatório de receitas:** Esse parecer deve fornecer uma visão das receitas geradas através do uso da API, e ele ajuda na avaliação do desempenho e da popularidade da API entre os desenvolvedores;
    - **Relatório de variância:** Esse documento fornece informações úteis para auxiliar na comparação de atividades ou receitas geradas em um intervalo de tempo.

5.5.2 Como aumentar o lucro usando APIs

Além de saber quais ativos digitais podem ser vendidos, é interessante ter a informação de como as APIs podem ser usadas para aumentar a receita do negócio [14].

1. **Aumentar os canais de clientes:** Uma função das APIs é expor os dados e serviços da empresa para parceiros, onde esses desenvolvem aplicações que consumem esses serviços por meio das APIs. Dessa forma, o tráfego através das APIs aumenta, à medida que o número de aplicações e o consumo aumentam. APIs também são usadas para fazer a integração entre serviços, gerando tráfego adicional através dela. O objetivo do provedor da API é criar um ecossistema de desenvolvedores parceiros, que criarão aplicações, que, dessa forma, proporcionarão a chegada de novos clientes e, consequentemente, terá impacto na receita;
2. **Aumentar a retenção de clientes:** Depois de certo momento é mais provável que os clientes se repitam, do que conseguir novos. Dessa forma, reter

clientes é um aspecto chave para manter o sucesso de uma empresa. Após atingir um número alto de clientes, fica mais difícil dos desenvolvedores migrarem para outras APIs e, a partir do momento em que os clientes se acostumam com uma API, dificulta a conquista deles pelos competidores. Como provedor, é importante desenvolver plataformas que aumente a base e retenha os clientes. É necessário desenvolver APIs de uso fácil e bem documentadas, para encorajar os desenvolvedores a usá-las, dessa maneira, facilitando a criação de boas aplicações;

3. **Incentivar a compra de serviços Premium ou de maior valor agregado:** Funções mais interessantes ou com maior valor agregado podem ser disponibilizadas para quem pagar pelo acesso. Esse modelo é mais sugerido para empresas que já possuem o negócio consolidado e com uma clientela fiel;
4. **Aumentar os canais de afiliados:** Para fornecedor de APIs, é interessante disponibilizar sua API a terceiros, os quais desenvolverão aplicações ligadas à API como afiliados. Portanto, com mais aplicações desenvolvidas, mais tráfego de informações, e, consequentemente, mais receita;
5. **Aumentar os canais de distribuição:** Em muitos casos, o negócio da empresa depende da quantidade de acesso a seus conteúdos e serviços. O crescimento da receita é diretamente proporcional ao crescimento no número de acessos. Nessa situação é vantajoso o incremento no número de canais de distribuição para levar seus conteúdos a mais clientes.

5.5.3 Modelos de Negócio

Atualmente existem diversos modelos de negócios que podem ser associados às APIs, cada um com características próprias e formas de gerar receitas distintas. Porém todos eles possuem três objetivos em comum [45]:

1. Estender o alcance atingido pelas APIs;
2. Expor a marca a um público maior;
3. Gerar oportunidades de receita.

Para o modelo ser bem sucedido, os papéis do provedor, dos consumidores das APIs e dos consumidores finais, apresentados na seção 5.2, devem se beneficiar do uso da API. Esses modelos podem ser simples ou complexos dependendo do uso dos ativos e do valor entregue. Porém, na maioria dos casos eles se encaixam em uma das quatro categorias e que serão vistos nas seções seguintes [1] [14] [46] [47] [48]:

- Gratuito (*Free*);
- O desenvolvedor paga (*Developer Pays*);
- O desenvolvedor recebe (*Developer Gets Paid*);
- Indireto (*Indirect*).

5.5.3.1 O Modelo Gratuito (*Free*)

Assim como é sugerido no título do modelo, as APIs gratuitas estão disponíveis para o consumo sem que seja necessário o pagamento de taxas do desenvolvedor ou do consumidor final. Esse formato auxilia no aumento da adoção e da popularidade. Com isso, o valor da marca da empresa cresce e torna-se possível o investimento em outros canais, a fim de aumentar o alcance de consumidores.

- **O totalmente gratuito:** Onde não há cobrança alguma sobre a API;
- **O baseado na duração (*Duration-based free model*):** Nesse formato, o consumidor possuirá um tempo limitado para aproveitar os recursos de forma gratuita e, após o fim do prazo, serão cobradas taxas para continuar o uso;
- **O baseado na quantidade (*Quantity-based free model*):** Nessa configuração, o consumidor terá um número limite de acessos e, se ele desejar continuar o uso, deverá pagar por isso;
- **Hibrido (*Hybrid free model*):** Esse modelo é uma combinação dos dois anteriormente citados e a cobrança é feita caso o limite de tempo ou de uso sejam atingidos.
- **O *Freemium*:** Alguns autores categorizam o modelo *Freemium* como um modelo gratuito, como Brajesh De, autor do livro “*API Management”*, assim como é encontrado numa postagem da APIgee. Porém existem pessoas que

discordem e o categorizem como um modelo onde o desenvolvedor paga, como apontado em um *White Paper* da IBM [48]. Para fins de apresentação, o modelo em questão aqui será incluso na categoria de modelos gratuitos. O modelo *Freemium* tem como característica promover o uso do núcleo das funções da API de forma gratuita e cobrar por funções adicionais.

Esses modelos podem ser encontrados em APIs como: Facebook e Google Maps.

5.5.3.2 O Modelo onde o Desenvolvedor Paga (*Developer Pays*)

O também conhecido como Modelo baseado em Tarifas (*Fee-Based Model*), é usado quando uma organização expõe ativos com maior valor aos consumidores e, com isso, aplicam tarifas sobre os recursos os quais os desenvolvedores estão dispostos a pagar. O modelo baseado em tarifas também possui variações, como:

- **Apenas um pagamento (*One-time fee*):** Nesse modelo o provedor cobra uma tarifa de registro e concede ao consumidor acesso ilimitado à API;
- **O modelo de tarifa de inscrição (*Subscription fee*):** Nesse formato é cobrada uma tarifa em intervalos regulares pelo uso da API;
- **Pago por transação (*Pay-per-API transaction*):** Nessa configuração não há uma tarifa mínima. O consumidor deverá pagar pelo número de transações feitas;
- **Pago pelo volume das transações (*Pay by transaction volume*):** Esse modelo de monetização é baseado no volume de requisições das APIs ou no volume de dados que passam por elas. Quando o limite é atingido, é cobrado pelo uso adicional;
- **Modelo de níveis de preços (*Tiered pricing model*):** Nessa forma não há uma tarifa comum. O consumidor pagará taxas baseadas em pacotes de requisições das APIs. Por exemplo, um número X de requisições custa um valor por requisição, enquanto uma quantidade Y custa outro valor. Geralmente uma banda, ou pacote, maior, diminui o preço de uma requisição individual.

Esses modelos podem ser encontrados em APIs como: Amazon, PayPal e Google AdWords.

5.5.3.3 O Modelo onde o Desenvolvedor é Pago (*Developer Gets Paid*)

Nos formatos onde os desenvolvedores são pagos, conhecido também pelo nome de “Modelo de Compartilhamento de Receita” (*Revenue-Sharing Model*), os provedores expõem os ativos digitais aos desenvolvedores, onde esses fazem o trabalho de vendê-los através de aplicações. Dessa forma, os provedores compartilham um percentual das receitas adquiridas com os desenvolvedores. Esse modelo tanto ajuda os consumidores, com o recebimento monetário, quanto os provedores, que expandem o negócio e aumentam as vendas através dos afiliados.

Da mesma forma que as duas categorias anteriores, os gratuitos e os baseados em tarifas, os Modelos de Compartilhamento de Receita também podem ser divididos em tipos:

- **Custo por ação (*Cost per Action*):**O provedor paga apenas quando uma ação especifica acontece;
- **Compartilhamento de receitas (*Revenue Sharing*)**: Nessa configuração, o provedor compartilha uma parte das receitas ganhas através do trafego em aplicações dos desenvolvedores. O compartilhamento pode ser fixo, onde apenas um percentual estipulado é distribuído, pode ser variável, o qual o percentual vai depender do volume de vendas em um determinado período, e de receita única, onde o afiliado receberá para cada cliente encaminhado através das suas aplicações;
- **Receita recorrente (*Recurring revenue*):** Nesse modelo, o afiliado receberá para cada cliente encaminhado para a API através das aplicações enquanto o inscrito continuar sendo cliente da API do provedor.

Esses modelos podem ser encontrados em APIs como: Google AdSense e amazon.com.

5.5.3.4 O Modelo Indireto (*Indirect Model*)

No modelo indireto [1] [48], as receitas são geradas quando um objetivo que impulsiona o modelo de negócio principal é atingido, dessa forma, varia de negócio para negócio. Esses objetivos podem ser estender a marca, levar a API a mais consumidores, aumentar a produtividade interna, fazer integração entre sistemas, etc. Esse modelo pode ser combinado com outros modelos que criem receitas diretas, pois, na maioria das vezes, eles só as geram de maneira indireta.

Esses modelos podem ser encontrados em APIs como: Netflix, eBay e Twitter.

5.5.4 O Modelo mais usado

Como já foi mostrado na seção anterior, existem vários modelos de negócio que uma empresa pode adotar, quando se trata a questão das APIs. Porém, no mercado atual, de acordo com uma análise da NordicAPIs [49] sobre as mais de 1800 APIs Públicas contidas no Site RapidAPI, o modelo onde o desenvolvedor é cobrado diretamente pelas chamadas e requisições tem maior notoriedade (*Developers Pays*), tornando-o o de maior adoção. Adotado o modelo, existem três variações as quais tem maior destaque:

1. **O *pay as you go*,** onde o desenvolvedor paga pelo que usa;
2. **O de cotas fixas (*fixed quotes*)**, o qual cabe ao desenvolvedor pagar por um plano com um número fixo de requisições por mês;
3. **O de cotas flexíveis (*flexible quotes*)**, onde esse é composto por uma cota fixa mensal podendo ser acrescido de tarifas por requisições extras.

Como pode ser visto na Tabela 2, cada modelo tem suas vantagens e desvantagens. Porém pode ser dada maior visibilidade, para o provedor, ao modelo de Cotas Flexíveis, pois, avaliando conforme a sua descrição, ele tem as vantagens do modelo de Cotas Fixas, como a previsibilidade na receita, e, ao atingir o limite de cotas, a receita pode continuar crescendo, assim como é no modelo *Pay as You Go.*

Tabela 2: Modelos mais Usados

Pay as YouGo	Cotas Fixas	Cotas Flexíveis
O preço aumenta com o uso	Os preços são previsíveis	Os preços são previsíveis
O preço é maior em aplicativos com maior público	A receita é previsível	A receita é previsível
A receita é imprevisível	Maior receita para desenvolvedores com menor consumo	Aplicação sempre no ar
Preços variáveis por desenvolvedor	Ao fim da cota, a aplicação é comprometida	Difícil comunicação em torno de excessos

Fonte: Adaptada de NordicAPIs [49]

5.5.5 Determinando o modelo adequado ao negócio

Tendo em vista todos esses modelos apresentados na seção 5.5.3, é difícil os empresários definirem qual modelo implantar às APIs. É de grande valia entender que cada modelo raiz (não as folhas), possui uma motivação para ser escolhido. Como justificativa breve para cada modelo é possível mostrar [1]:

- **Gratuito:** O uso de modelos gratuitos, e suas variações, são indicados para quando o negócio deseja estender o alcance da empresa, adquirindo maior adoção e popularidade;
- **O desenvolvedor paga:** Esse modelo é adequando para quando a API tem funções de grande valor agregado, as quais os desenvolvedores necessitam e se dispõem a pagar para usar;
- **O desenvolvedor é pago:** Nesse cenário, as empresas deixam o trabalho de venda dos ativos digitais nas mãos dos desenvolvedores.
- **Indireto:** O modelo vai variar de acordo com objetivos específicos da empresa.

Porém, além de entender, de forma simplificada, os usos de cada modelo existem alguns pontos que, quando analisados, auxiliam na decisão de qual modelo seguir [50].

- **Verificar qual é o propósito principal da API:** Dessa forma será possível definir se a API será usada como produto (seção 5.5.1) para gerar receita; se a API vai melhorar um produto já existente; ou se a API vai promover um produto;
- **Verificar os ativos digitais disponibilizados:** É necessário ter em mente quais os ativos que podem gerar receita através de APIs;
- **Verificar os consumidores da API:** Podendo ser internos, parceiros ou um maior público externo.

Após entender o objetivo principal da API e ter feito uma relação entre as informações do negócio e os três pontos, anteriormente citados, é possível visualizar qual o modelo se encaixa melhor no contexto. Tornando viável a possibilidade de melhores decisões para monetizar a API e, assim, criar receita.

6 Conclusão

O investimento em informatização do negócio por parte das empresas tem avançado cada vez mais em decorrência da crescente dependência de tecnologia para solucionar conflitos e pelo fácil acesso à internet. O uso de novas tecnologias tem contribuído para que as aplicações possam executar funções, de maneira a resolver problemas, de forma mais fácil e eficiente, e, dessa forma, direta ou indiretamente, gerar receita no negócio.

As APIs surgem com o objetivo de integrar diversas plataformas e equipamentos ao sistema responsável por solver as questões, e fazer com que desenvolvedores terceiros criem produtos associados ao seu serviço. Através do uso das APIs, a comunicação é feita sem que haja intervenção dos usuários, onde esses podem ser pessoas ou outros sistemas ou serviços.

Os microsserviços, por outro lado, aparecem como alternativa para um modelo de arquitetura muito comum atualmente, a arquitetura em camadas. Por ter natureza distribuída, esse modelo propõe a quebra da estrutura monolítica em partes independentes, provendo maior desacoplamento entre os módulos, os quais passam a ser responsáveis por, na maioria das vezes, apenas uma função, porém sem perder a comunicação entre eles.

A presente pesquisa buscou fazer uma análise relacionada às APIs e aos Microsserviços, assim como apresentar seus impactos nos negócios. Foi possível avaliar que na economia moderna, o uso dessas tecnologias se tornou um ponto forte e diferencial para obter sucesso em um negócio. Associando os pontos fortes das duas tecnologias, as aplicações se tornam mais valiosas, tanto economicamente, para os provedores e afiliados, gerando receita e benefícios ao sistema, quanto na eficiência na resolução de problemas, nesse ponto, favorável para os três papeis integrantes da economia, o fornecedor, o parceiro e usuário final. Pelo baixo acoplamento e pela grande facilidade de integração, a combinação dessas duas tecnologias possibilita entregas contínuasde funcionalidades com maior aptidão. A qual viabiliza o constante crescimento do sistema, a partir dessas entregas, e traz valor a todas as partes interessadas no negócio.

6.1 Trabalhos Futuros

O uso de APIs e Microsserviços tende a continuar crescendo nos próximos anos e, com isso, essas tecnologias se farão cada vez mais presentes na vida das pessoas. Entretanto, para haver esse progresso é necessário que sejam estabelecidos contratos onde os provedores e os afiliados das APIs sejam mutuamente beneficiados.

Recentemente, no ano de 2016, Oliver Hart e Bengt Holmström venceram o Prêmio Nobel da Economia devido a suas grandes contribuições à Teoria dos Contratos. É possível expressar que, de certo modo, os acordos entre os fornecedores e os parceiros também são impactados com as descobertas. Portanto, é interessante que seja feita uma relação entre esse estudo e as APIs de modo que se torne mais aparente de como esses contratos, anteriormente citados, são feitos e beneficiam ambas as partes.

7 Referências

[1] JACOBSON, Daniel; BRAIL, Greg; WOODS, Dan. *APIs: A Strategy Guide*. 1. ed. Estados Unidos da América: O'Reilly Media, 2012. 136 p.

[2] API *Evangelist History of APIs*. Disponível em: <https://history.apievangelist.com>. Acesso em: 07 ago. 2017

[3] UMA BREVE HISTÓRIA DAS APIS COM A PLUGA. Disponível em: <https://mundoapi.com.br/materias/uma-breve-historia-das-apis-com-a-pluga/>. Acesso em: 07 ago. 2017

[4] *A brief history of microservices*. Disponível em: <http://blog.leanix.net/en/a-brief-history-of-microservices>. Acesso em: 08 ago. 2017

[5] *What are Microservices*?. Disponível em: <https://martinfowler.com/microservices/#what>. Acesso em: 16 dez. 2017

[6] Exemplos de APIs que você usa todo dia e não sabe. Disponível em: <https://sensedia.com/blog/apis/exemplos-de-apis/>. Acesso em: 16 dez. 2017

[7] *Microservice Architecture*. Disponível em: <http://microservices.io/articles/whoisusingmicroservices.html>. Acesso em: 16 dez. 2017

[8] *API Directory Eclipses 17,000 as API Economy Continues Surge*. Disponível em: <https://www.programmableweb.com/news/programmableweb-api-directory-eclipses-17000-api-economy-continues-surge/research/2017/03/13>. Acesso em: 08 ago. 2017

[9] *What are APIs? (The Technology Perspective)*. Disponível em: <https://medium.com/@robert.broeckelmann/what-are-apis-the-technology-perspective-ca7e33d383c1>. Acesso em: 11 ago. 2017

[10] FERREIRA, Cleber de F.; MOTA, Roberto Dias. COMPARANDO APLICAÇÃO WEB SERVICE REST E SOAP. S.l. Disponível em: <http://web.unipar.br/~seinpar/2014/artigos/pos/Cleber_de_F_Ferreira_Roberto_Dias_Mota%20(1).pdf>. Acesso em: 12 ago. 2017

[11] *Integration architecture: Comparing web APIs with service-oriented architecture and enterprise application integration*. Disponível em: <https://www.ibm.com/developerworks/websphere/library/techarticles/1503_clark/1305_clark.html>. Acesso em: 12 ago. 2017

[12] O que é API?. Disponível em: <https://www.tecmundo.com.br/programacao/1807-o-que-e-api-.htm>. Acesso em: 12 ago. 2017

[13] NIJIM, Sharif; PAGANO, Brian. *APIs For Dummies: Apigee Special Edition*. Estados Unidos da América: John Wiley & Sons, 2014. 44 p.

[14] DE, Brajesh. API *Management: An Architect's Guide to Developing and Managing APIs for Your Organization*. 1. ed. Índia: Apress, 2017. 195 p.

[15] *The Why and How of APIs: The Internal API Model*. Disponível em: <https://apigee.com/about/blog/digital-business/why-and-how-apis-internal-api-model>. Acesso em: 13ago. 2017

[16] *Private, Partner or Public: Which API Strategy Is Best For Business*?. Disponível em: <https://www.programmableweb.com/news/private-partner-or-public-which-api-strategy-best-business/2014/02/21>. Acesso em: 13 ago. 2017

[17] *The Why and How of APIs: The Open API Model*. Disponível em: <https://apigee.com/about/blog/technology/why-and-how-apis-open-api-model>. Acesso em: 19 ago. 2017

[18] *Benefits of APIs*. Disponível em: <https://api-all-the-x.18f.gov/pages/benefits_of_apis/>. Acesso em: 26 ago. 2017

[19] *What are the Benefits of APIs*?. Disponível em: <https://www.programmableweb.com/news/what-are-benefits-apis/analysis/2015/12/03>. Acesso em: 26 ago. 2017

[20] *What are some challenges faced when building APIs to expose and integrate systems of large enterprises*?. Disponível em: <https://www.quora.com/What-are-some-challenges-faced-when-building-APIs-to-expose-and-integrate-systems-of-large-enterprises>. Acesso em: 16 dez. 2017

[21] *Application Programming Interface (API): What it is and How it is Utilized in the Transportation Industry*. Disponível em: <http://ltxsolutions.com/application-programming-interface-api-transportation/>. Acesso em: 16 dez. 2017

[22] *Microservices Architectures*. Disponível em: <https://www.happiestminds.com/Insights/microservices/>. Acesso em: 02 set. 2017

[23] GOETSCH, Kelly. *Microservices for Modern Commerce*. 1. ed. Estados Unidos da América: O'Reilly Media, 2016. 68 p.

[24] WOLFF, Eberhard. Microservices: *Flexible Software Architecture*. 1. ed. Estados Unidos da América: Pearson Education, 2017. 395 p.

[25] RICHARDS, Mark. *Software Architecture Patterns*. 3. ed. Estados Unidos da América: O'Reilly Media, 2016. 47 p.

[26] *Microservices*. Disponível em: <https://martinfowler.com/articles/microservices.html>. Acesso em: 10 set. 2017

[27] *Microservices architecture style*. Disponível em: <https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices>. Acesso em: 19 set. 2017

[28] *Microservices & API Gateways, Part 1: Why an API Gateway*?. Disponível em: <https://www.nginx.com/blog/microservices-api-gateways-part-1-why-an-api-gateway/#WhyanAPIGateway>. Acesso em: 16 dez. 2017

[29] *Using an API Gateway in Your Microservices Architecture*. Disponível em: <https://smartbear.com/learn/api-design/api-gateways-in-microservices/>. Acesso em: 16 dez. 2017

[30] *Microsoft Application Architecture Guide*, 2nd Edition. Disponível em: <https://msdn.microsoft.com/en-us/library/dd673617.aspx>. Acesso em: 04 out. 2017

[31] *Understand Event-Driven Software Architecture*. Disponível em: <https://msdn.microsoft.com/en-us/library/dd673617.aspx> Acesso em: 04 out. 2017

[32] *API DESIGN 304: API DESIGN FOR MICROSERVICES*. Disponível em: <http://www.apiacademy.co/resources/api-design-304-api-design-for-microservices/>. Acesso em: 10 out. 2017

[33] *API MANAGEMENT 101: API MANAGEMENT BASICS*. Disponível em: <http://www.apiacademy.co/resources/api-management-101-api-management-basics/>. Acesso em: 10 out. 2017

[34] *API MANAGEMENT 302: USING AN API GATEWAY IN MICROSERVICE ARCHITECTURE*. Disponível em: <http://www.apiacademy.co/resources/api-management-302-using-an-api-gateway-in-microservice-architecture/>. Acesso em: 10 out. 2017

[35] NADAREISHVILI, Irakli. et. al. *Microservice Architecture*. 1. ed. Estados Unidos da América: O'Reilly Media, 2016. 128 p.

[36] *The top 5 software architecture patterns: How to make the right choice*. Disponível em: <https://techbeacon.com/top-5-software-architecture-patterns-how-make-right-choice>. Acesso em: 21 out. 2017

[37] *Disruptive Innovation*. Disponível em: <http://www.claytonchristensen.com/key-concepts/>. Acesso em: 04 nov. 2017

[38] Entenda o que é 'disrupção' e saiba como ele ameaça empresas. Disponível em: <http://odia.ig.com.br/noticia/economia/2015-06-28/entenda-o-que-e-disrupcao-e-saiba-como-ele-ameaca-empresas.html>. Acesso em: 04 nov. 2017

[39] DOERRFELD, Bill. et. al. *The API Economy: Disruption and the Business of APIs*. S.l: s.n., 2016. 107 p.

[40] WILLMOTT, Steven; BALAS, Guillaume; 3cale*. Winning in the API Economy*. S.l: s.n., [201?]. 69 p.

[41] O que são APIs – Parte 3: A Economia de APIs. Disponível em: <https://sensedia.com/blog/negocios-digitais/o-que-sao-apis-parte-3-a-economia-de-apis/>. Acesso em: 07 nov. 2017

[42] CA. *API Strategy and Architecture: A Coordinated Approach*. S.l: s.n., [201?].

[43] *API STRATEGY 301: API-AS-A-PRODUCT*. Disponível em: <http://www.apiacademy.co/resources/api-strategy-lesson-301-api-as-a-product/>. Acesso em: 07 nov. 2017

[44] APIgee. *The Definitive Guide to API Management*. S.l: s.n., [201?].

[45] CA. *API Monetization: Unlocking the Value of Your Data*. S.l: s.n., [201?].

[46] *Business Models for APIs*. Disponível em: <https://developer.ibm.com/apiconnect/documentation/api-101/business-models-apis/>. Acesso em: 18 nov. 2017

[47] *How To Pick the Best Business Models for Your APIs*. Disponível em: <https://www.programmableweb.com/news/how-to-pick-best-business-models-your-apis/analysis/2017/09/27>. Acesso em: 18 nov. 2017

[48] GLICKENHOUSE, Alan; ENGLAND, Larry. *API Monetization – Understanding your Business Model Options*. Disponível em: <https://www-01.ibm.com/common/ssi/cgi-bin/ssialias?htmlfid=KUW12387USEN>. Acesso em: 20 nov. 2017

[49] *The Ultimate Guide to Pricing Your API*. Disponível em: <https://nordicapis.com/the-ultimate-guide-to-pricing-your-api>. Acesso em: 20 nov. 2017

[50] *A Guide to Picking the Right Business Model for Your API Strategy*. Disponível em: <https://www.epam.com/ideas/blog/a-guide-to-picking-the-right-business-model-for-your-api-strategy>. Acesso em: 16 dez. 2017