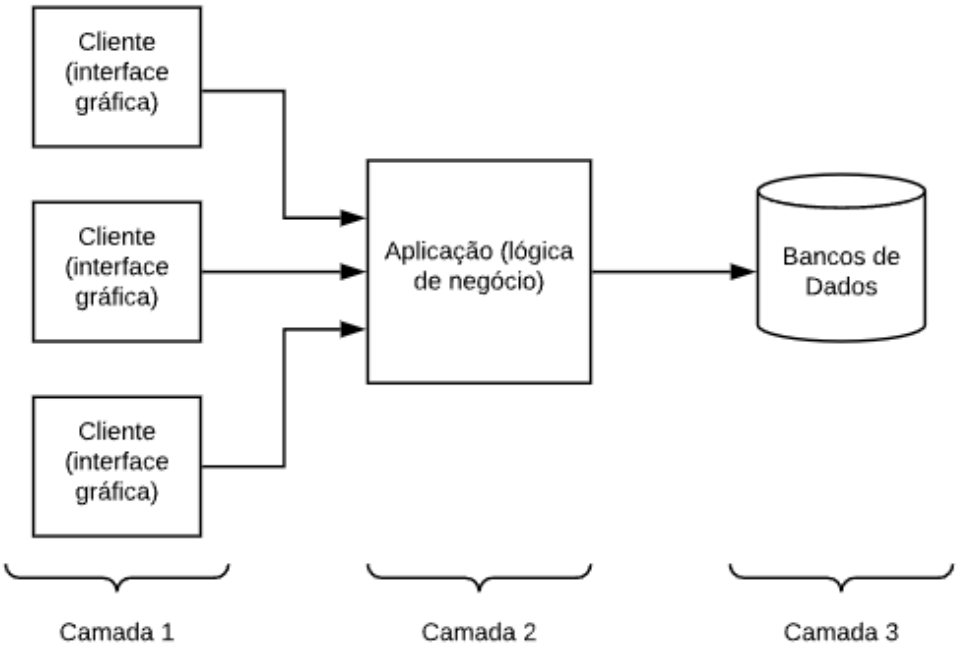
Arquitetura em camadas

As classes são organizadas em módulos de maior tamanho, chamado de camadas.As camadas são dispostas de forma hierárquica, assim uma camada só pode usar serviços da camada imediatamente inferior.

Arquitetura em três camadas

Interface com o Usuário
Lógica de negócio
Banco de dados

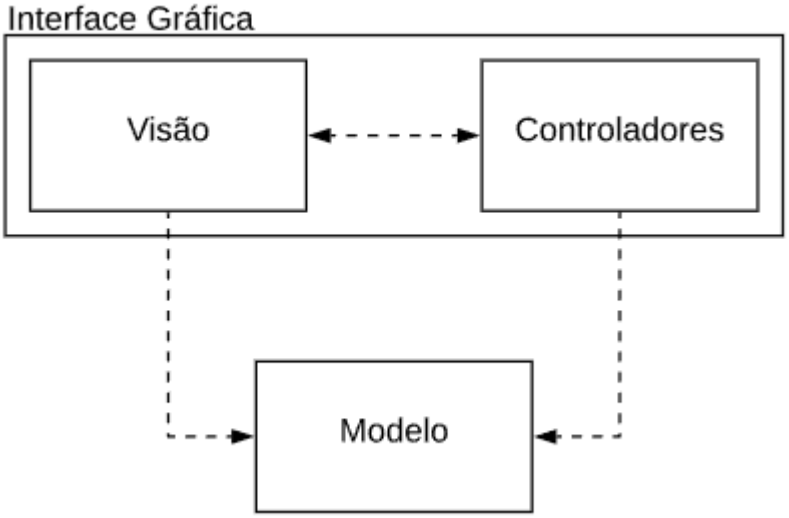


Pode ser ter a variante das duas camadas: a interface e aplicação rodam em uma camada só, no cliente. Desvantagem: todo processamento ocorre nos clientes, exigindo maior poder computacional.

Arquitetura MVC

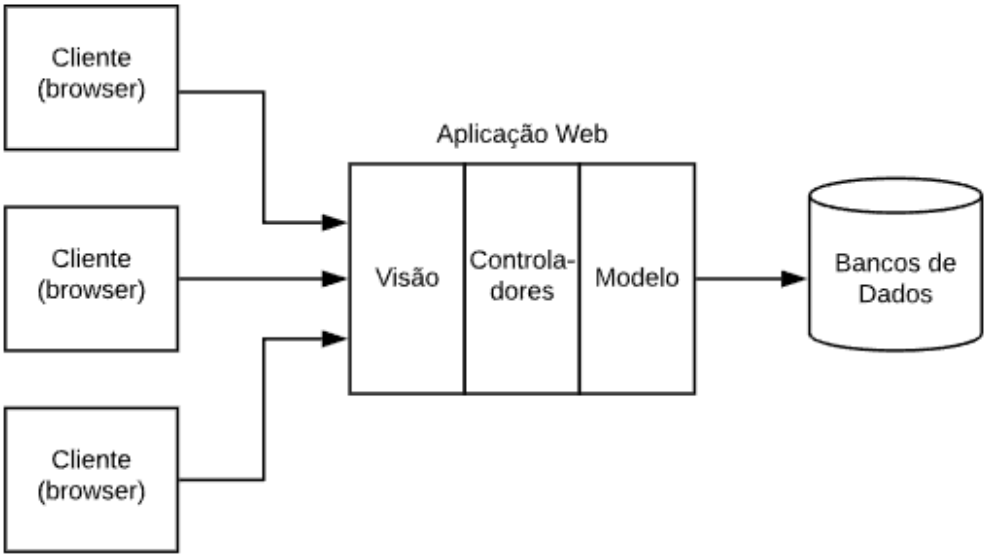
Visão = interface gráfica
Controladoras = eventos gerados por dispositivos de entrada como mouse e teclado
Modelo = armazenam os dados manipulados pela aplicação e métodos que alteram o estado dos objetos de domínio.

MVC = (Visão + Controladores) + Modelo = Interface Gráfica + Modelo



MVC favorece a especialização do trabalho de desenvolvimento
MVC permite que classes de Modelo sejam usadas por diferentes Visões
MVC favorece testabilidade, pois é mais facil testar objetos não visuais, separando visual de modelo é mais facil testar

MVC PARA WEB



Single Page Applications

Modelo request-response

cliente requisita uma página do servidor(request)

servidor envia página e cliente a exibe (response)

Aplicação roda no browser, mas que é mais independente do servidor e “menos burra”(manipula sua própria interface e armazena os seus dados)

Microsserviços

A arquitetura monolítica, apesar de ser dividida em módulos, é executada em tempo de monólito, sendo lento. Em um monolito é necessário um processo rigoroso e burocrático para novos releases, para que os módulos não causem bugs nos outros módulos.

A ideia do micro serviço é que o sistema é decomposto em módulos em tempo de desenvolvimento e em execução.

Quando separa-se em módulos distintos não há mais possibilidade de que um módulo acesse um recurso interno de outro módulo.

Assim microsserviços são um instrumento para garantir que os times de desenvolvimento somente usem interfaces públicas de outros sistemas

Outra vantagem é a escalabilidade: é possível escalar apenas o serviço que necessita de mais recursos.

Como os microsserviços são autônomos e independentes eles podem ser implementados em tecnologias diferentes, incluindo linguagens de programação, frameworks e banco de dados.

Quando se usa um monolito, as falhas são fatais. Quando usa microsserviço as falhas são parciais, ou seja, uma funcionalidade pode parar de funcionar, mas as outras podem estar de pé.

Microsserviços são possíveis graças a computação em nuvem, onde pode ser alocar uma máquina em qualquer lugar para rodar um serviço

Os microsserviços são como empresas distribuindo várias equipes descentralizadas e autônomas, produzindo novas inovações.

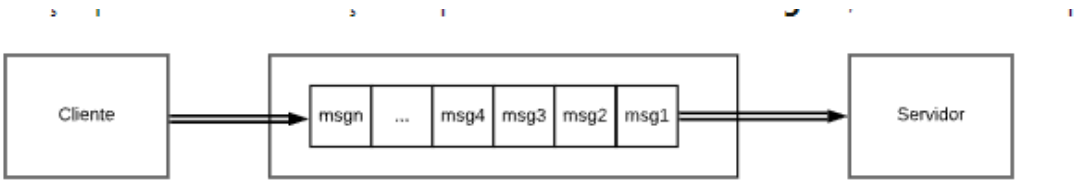
Os microsserviços devem ser autônomos também nos banco de dados, para cada serviço, um banco de dados.

Quando não usar microsserviços:

- Quando torna muito complexo o sistema
- Caso queira mais velocidade, o microsserviço é mais lento que o monolito
- Quando existem transações distribuídas.

Arquiteturas Orientadas a Mensagens

Nessa arquitetura a comunicação entre clientes e servidores é mediada por um terceiro serviço, que tem a única função de prover um fila de mensagens



Os clientes atuam como produtores

Os servidores atuam como consumidores

Uma mensagem é um registro(ou objeto) que entra na fila de mensagens do tipo FIFO. Tudo isso de maneira assíncrona, essas filas são chamadas de brokers

Além de permitir comunicação assíncrona essa arquitetura permite:

Desacoplamento no espaço: clientes não precisam conhecer os servidores e vice-versa.

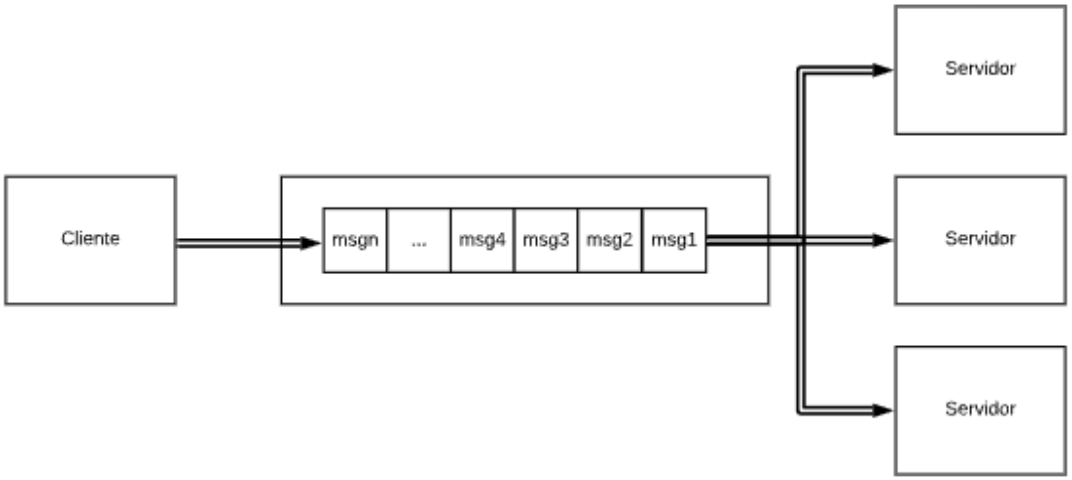
Desacoplamento no tempo: clientes e servidores não precisam estar simultaneamente disponíveis para se comunicarem.As mensagens continuaram na fila.

Com esse sistema de fila o time de cliente e o time de servidor pode trabalhar de forma autônoma, pois os atrasos de um time não interferem no outro.

O importante é que o broker de mensagens seja estável e capaz de armazenar uma grande quantidade de mensagens.

As filas de mensagens ainda permitem escalar mais facilmente o sistema distribuído.

próxima figura:

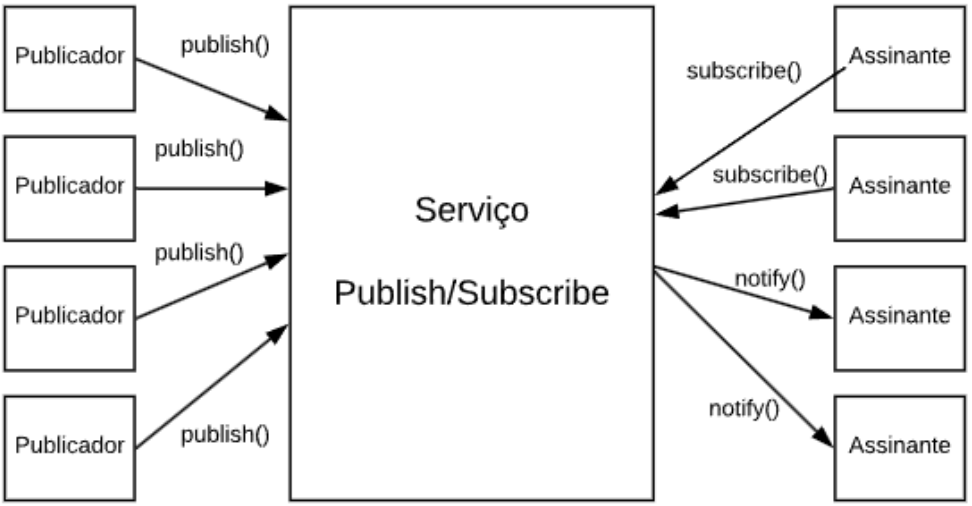


Arquiteturas Publish/Subscribe

Nessa arquitetura as mensagens são chamadas de eventos

Os publicadores produzem eventos e publicam no serviço de publish/subscribe

Os assinantes devem previamente assinar eventos de seu interesse, quando um evento é publicado, os seus assinantes são notificados.



Apesar de ser parecido com as filas de mensagens oferecendo também o desacoplamento no espaço e no tempo, existem algumas diferenças:

No publish/subscribe um evento gera notificações em todos seus assinantes. Já as filas de mensagens são sempre consumidas por um único servidor. Sendo assim o publish/subscribe se caracteriza por uma comunicação 1 para n. Enquanto as filas de mensagens se caracterizam por 1 pra 1.

No publish/subscribe os assinantes são avisados assincronamente. Na fila de mensagens, os servidores precisam puxar “pull” as mensagens da fila.

Essa arquitetura lembra o padrão de projeto Observador

Pipes e filtros

Orientada a dados nos quais os programas - chamados de filtros - tem como função processar os dados recebidos na entrada e gerar uma nova saída. Os filtros são conectados por meio de pipes, que agem como buffers, os pipes são usados para armazenar a saída de um filtro, enquanto ela não é lida pelo primeiro filtro da sequência.

Cliente/Servidor

Clientes e servidores são os dois únicos módulos dessa arquitetura, se comunicando por meio de uma rede. Os clientes solicitam serviços ao módulo servidor e aguardam o processamento.
Arquiteturas peer-to-peer são arquiteturas nas quais os módulos da aplicação podem desempenhar tanto o papel de cliente, como o papel de servidor, ou seja, são tanto provedores como consumidores do recurso.

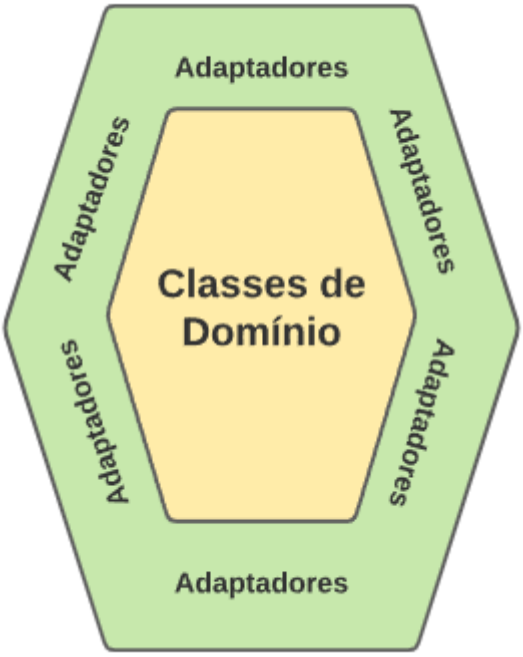
Anti-padrões Arquiteturais

Big ball of mud:

Nesse padrão qualquer módulo se comunica com praticamente qualquer outro. Não possuindo uma arquitetura definida, existindo uma explosão no número de dependências, que dá origem a um espaguete de código. Consequentemente, a manutenção do sistema torna-se muito difícil e arriscada.

Arquitetura Hexagonal

Divide-se em classes de domínio e de infraestrutura/tecnologias.
As classes de domínio devem ser independentes das relacionadas com infraestrutura.
Dessa forma mudando a tecnologia não se impacta as classes de domínio, e as classes de domínio podem ser compartilhadas por mais de uma tecnologia.
Nessa arquitetura a comunicação entre essas duas classes é feita a partir de adaptadores.



As portas, as interfaces usadas para comunicação entre interface e infraestrutura.

Portas de entrada são fora para dentro, quando uma classe externa precisa de um método de um classe de domínio.

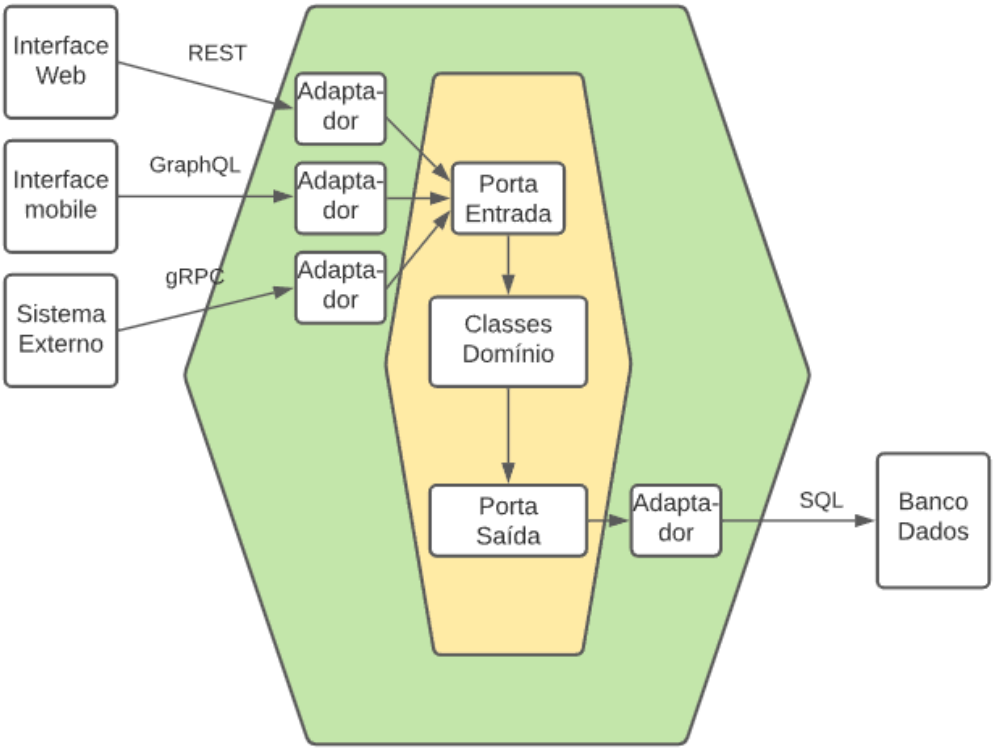
Portas de saída são de dentro para fora, quando uma classe de domínio precisa de um método de uma classe externa.

As portas são independentes de tecnologia, estando no hexágono interior, já os sistemas externos usam alguma tecnologia.

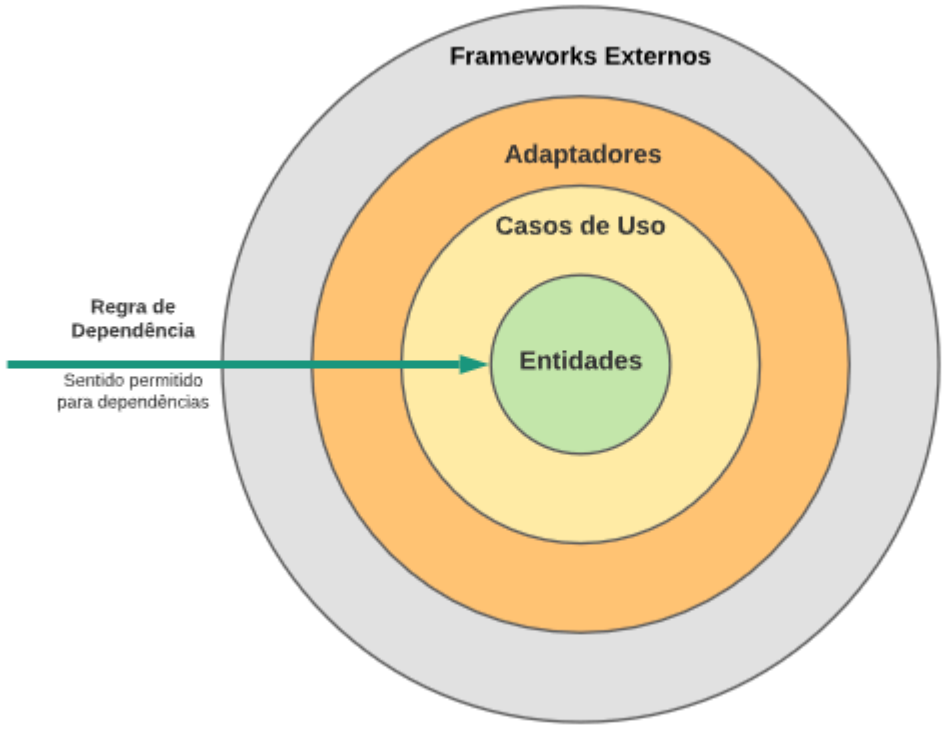
Os adaptadores atuam de dois modos:

Recebem chamadas de métodos vinda de fora do sistema e encaminham para as portas de entrada

Recebem chamadas vindas de dentro do sistema e as direcionam para o sistema externo.



Arquitetura limpa



Arquitetura Limpa

No centro da arquitetura temos as entidades e os casos de uso

Entidades: são classes comuns a vários sistemas da empresas e regras de negócio genérica

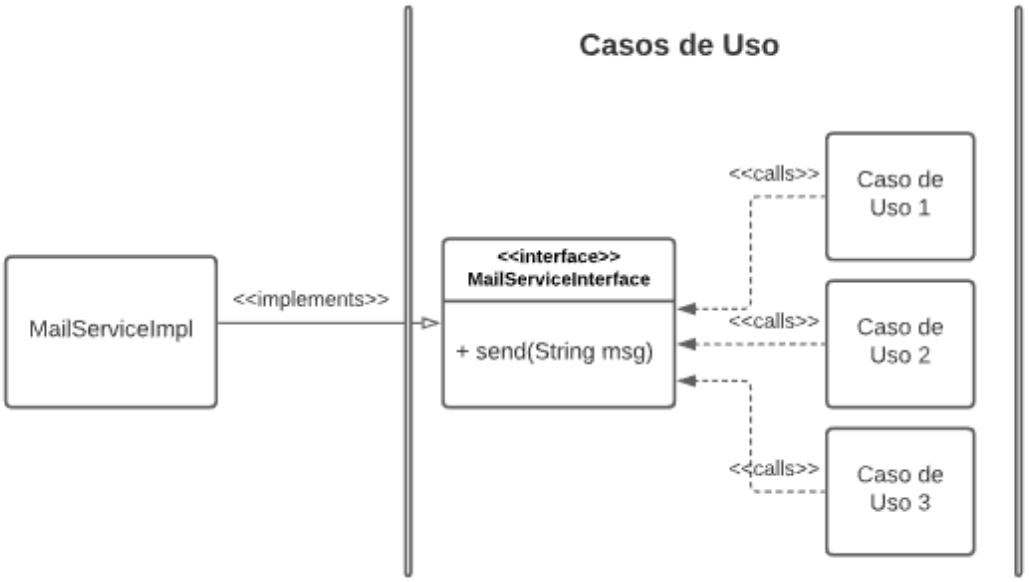
Casos de uso: regras de negócio específicas de um sistema

Já externamente temos os adaptadores.

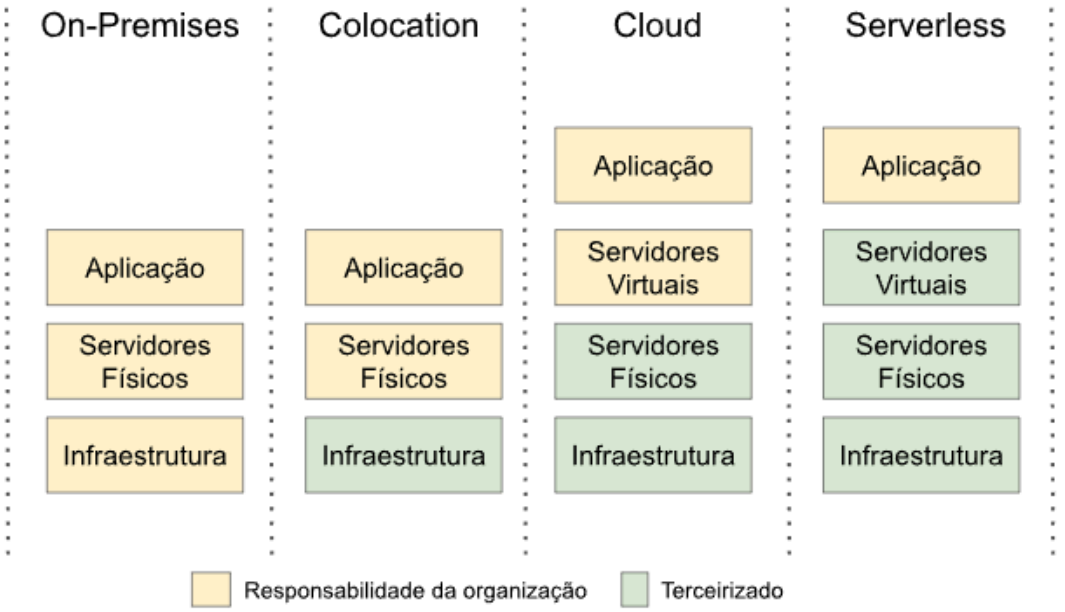
Adaptadores: Mediar a interação mais externa (sistemas externos) e camadas centrais(Casos de uso e entidades). Por exemplo, os adaptadores podem ser responsáveis por implementar uma API REST.

Por último temos o framework externo: bibliotecas, frameworks e quaisquer sistemas externos.

Nessa arquitetura temos a Regra de Dependência: as classes da camada X não devem conhecer nenhuma classe da camada Y mais externa. Além disso, o fluxo de controle deve ser de fora para dentro.



Arquitetura Serverless



O nome serverless explica-se pelo fato que os desenvolvedores não precisam se preocupar com instalação, configuração e escalabilidade de servidores. Com o serverless, paga-se pelo tempo de execução das funções serverless. Essa solução nem sempre é a mais barata.

As funções serverless tem as seguintes características:

Stateless, não guardam qualquer estado entre uma execução e outra.

Elas executam por um intervalo de tempo máximo, normalmente da ordem de alguns minutos.

Elas podem ser implementadas em uma variedade de linguagens de programação.

Desvantagens:

Complexidade em gerenciar uma arquitetura constituída por um grande número de pequenas funções

Maior latência

Riscos de alto acoplamento com a plataforma de cloud, tornando mais difícil uma mudança para outra plataforma(dependência de fornecedores).

É possível ter uma arquitetura híbrida, onde apenas alguns exercícios são por meio de funções serverless.

Consistências de dados em Microsserviços usando-se Sagas

Em bancos de dados centralizados a atomicidade é garantida a partir do commit e do rollback

Já em bancos de dados distribuídos uma possível solução é o Two Phase Commit (2PC), porém tem custo e latência altos. Para solucionar esse problema temos a Sagas.
Sagas são definidas por dois conjuntos:

Um conjunto de transações t1,t2,...

Um conjunto de compensações para cada transação. Toda transação tem uma segunda transação que reverte.Ou seja, se der ruim em uma transação, reverte as que deram certo.

Respondendo as questões do livro

Cap 7

1. Dada a sua complexidade, sistemas de bancos de dados são componentes relevantes na arquitetura de qualquer tipo de sistema. Verdadeiro ou falso? Justifique a sua resposta.

R: Verdadeiro, é importante entender onde e como o banco de dados vai se encaixar na arquitetura.

2. Descreva três vantagens de arquiteturas MVC.

R: Favorece a especialização do trabalho
Favorece a testabilidade
Permite que várias classes do modelo sejam usadas por diferentes classes de visão.

3. Qual a diferença entre classes Controladoras em uma Arquitetura MVC tradicional e classes Controladoras de um sistema Web implementado usando um framework MVC como Ruby on Rails?

R: As classes controladoras tradicionais controlam os eventos gerados pelos dispositivos de entrada. Já no sistema web essa classe é responsável por processar uma solicitação e gerar uma nova solicitação para a classe de visão.

4. Descreva resumidamente quatro vantagens de microsserviços.

R: escalabilidade, implementação individual da equipe, uso de várias tecnologias, isolamento de falhas.

5. Por que microsserviços não são uma bala de prata? Isto é, descreva pelo menos três desvantagens do uso de microsserviços.

R: Latência alta, não é bom para transações distribuídas e não é recomendado para sistemas complexos.

6. Explique a relação entre a Lei de Conway e os microsserviços.

R: A lei de Conway mostra que as empresas tendem a adotar arquiteturas de software que são cópias de suas estruturas organizacionais. Com as empresas se “modularizando”, tendo times independentes e com responsabilidades e ideias próprias, traz à tona as arquiteturas de microsserviços, onde cada serviço é trabalhado por uma equipe, com suas próprias tecnologias e ideias, sendo independente das outras.

7. Explique o que significa desacoplamento no espaço e desacoplamento no tempo. Por que arquiteturas baseadas em filas de mensagens e arquiteturas Publish/Subscribe oferecem essas formas de desacoplamento?

R: desacoplamento no espaço: clientes não precisam conhecer os servidores e vice-versa. desacoplamento no tempo: clientes e servidores não precisam estar simultaneamente disponíveis para se comunicarem.As mensagens continuaram na fila.

8. Quando uma empresa deve considerar o uso de uma arquitetura baseada em filas de mensagens ou uma arquitetura publish/subscribe?

R: Quando quer trabalhar de modo individual entre cliente e servidor, além de poder trabalhar de forma assíncrona cliente e servidor. Ainda é possível escalonar bem.

9. Explique o objetivo do conceito de tópicos em uma arquitetura publish/subscribe.

R: São categorias de eventos, quando um cliente produz informa um tópico. Assim os assinantes não precisam assinar todos os eventos, apenas os eventos de um determinado tópico.

10. (POSCOMP, 2019, adaptado) Marque V ou F.

(F) O padrão MVC é uma adaptação do padrão arquitetural Camadas. A Camada Visão lida com a apresentação e a manipulação da interface, a Camada Modelo organiza os objetos específicos da aplicação, e a Camada Controle posiciona-se entre estas duas com as regras do negócio.

(V) O padrão Broker é voltado a problemas de ambientes distribuídos. Sugere uma arquitetura na qual um componente (broker) estabelece uma mediação que permite um desacoplamento entre clientes e servidores.

(V) Mesmo que um dado padrão arquitetural ofereça uma solução para o problema sendo resolvido, nem sempre ele é adequado. Fatores como contexto e o sistema de forças que afeta a solução fazem também parte do processo de avaliação e da escolha de padrões adequados.

O que é uma Arquitetura Hexagonal?

1. Em uma Arquitetura Hexagonal, um adaptador é uma implementação do padrão de projeto de mesmo nome. E as portas? Elas podem ser vistas como sendo uma implementação – pelo menos aproximada – de qual padrão de projeto? Se necessário, consulte o Capítulo 6 para responder.

R: Como o padrão bridge, interligando duas grandes classes sendo a classe de domínio e as externas.

2. Na figura que mostra a arquitetura hexagonal do sistema de bibliotecas, por que os adaptadores de interface externa (HTTP, GraphQL e REST) e o adaptador de persistência (SQL) estão em faces distintas do hexágono? Eles poderiam ser desenhados na mesma face?

R: Estão em faces distintas, pois são coisas diferentes, na face de sql se trata sobre banco de dados, já na outra se trata de interfaces, cada face tem sua funcionalidade.

3. A definição do termo hexagonal é arbitrária, pois, dependendo da aplicação, ela poderia ser chamada de quadrangular, pentagonal, heptagonal, octogonal, etc. Justifique essa afirmação.

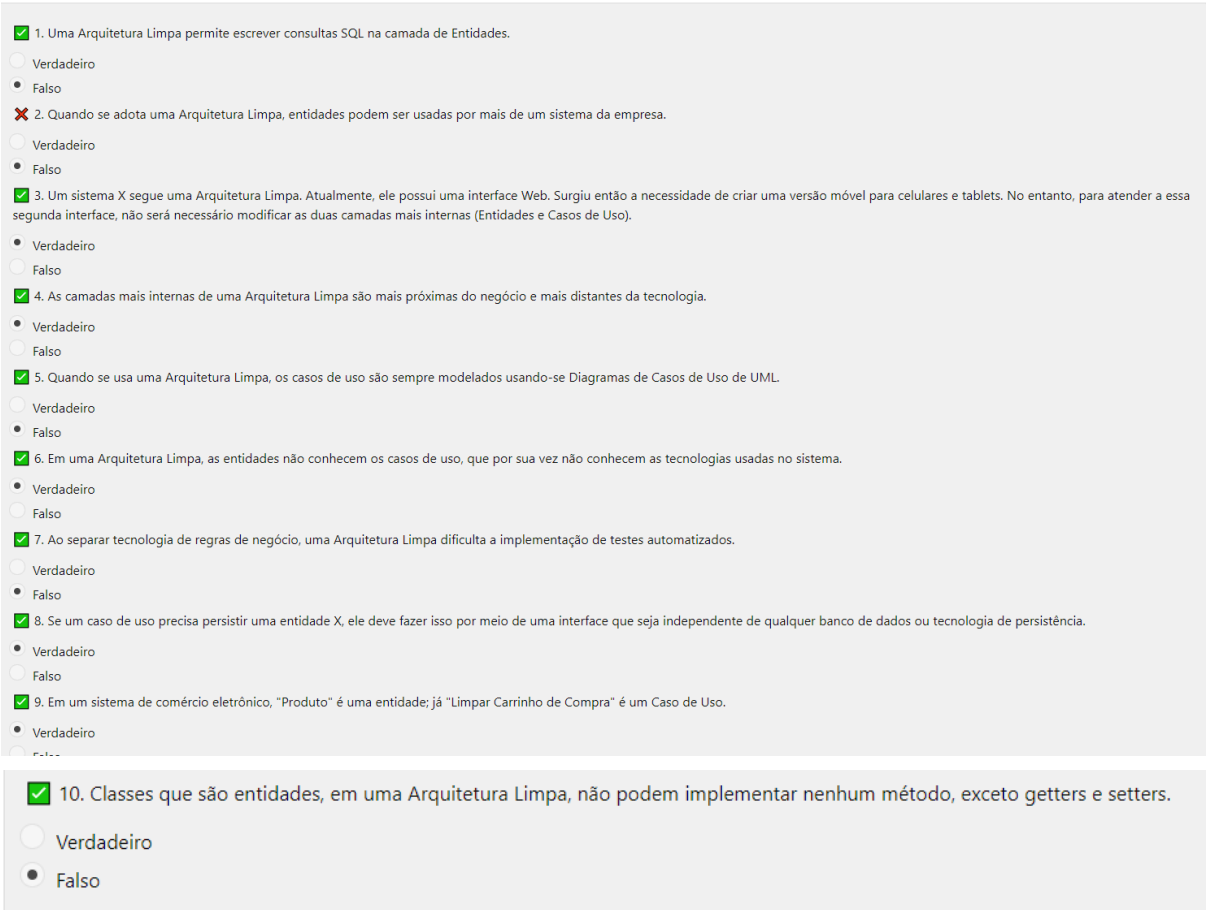
R: Sim poderia ser qualquer desses nomes, pois está atrelado a número de faces do polígono, esse número de faces varia dependendo do que o sistema precisa.

5. Descreva, resumidamente, as diferenças entre a Arquitetura Hexagonal e a Arquitetura Limpa (que estudamos em um outro artigo didático).

R: Na arquitetura hexagonal existe a classe de domínio, na limpa é separada por entidades e casos de uso. Para ligar classes na hexagonal se usa portas e adaptadores, na limpa se usa interfaces próprias de cada classe.

1 Construindo Sistemas com uma Arquitetura Limpa

1. Para fixar os principais conceitos de uma Arquitetura Limpa, responda ao seguinte exercício de V ou F, com correção online.



2. Em uma arquitetura limpa o nome de um elemento declarado em uma camada externa não deve ser mencionado pelo código de uma camada interna? Qual a principal vantagem ou benefício dessa regra?

R: Com isso as camadas internas ficam mais estáveis, dessa forma é mais fácil de mudar as tecnologias e deixa as regras de negócio sem modificações.

5. Suponha que um sistema use tecnologias X, Y e Z. E suponha que temos certeza de que elas nunca vão mudar no futuro. Ou seja, não existe chance de amanhã o sistema ter que usar uma tecnologia X', Y' ou Z'. Nesse cenário, você acha que ainda pode ser útil a adoção de uma Arquitetura Limpa? Justifique.

R: Seria útil pela organização que essa arquitetura propõe, sua testabilidade e sua escalabilidade, mas não traz o benefício de mudar tecnologias, que é um dos pontos interessantes da arquitetura.

6. Quando não vale a pena usar uma Arquitetura Limpa?

R: Projetos pequenos, que não vão ter mudanças.

1 O que é uma Arquitetura Serverless?

1. Quando usamos serverless não precisamos nos preocupar com as questões abaixo, EXCETO:

1. Planejamento de capacidade
2. Balanceamento de carga
3. Escalabilidade
4. Tolerância a falhas
5. Persistência de dados

Persistência de dados

2. Por que o termo serverless não deve ser interpretado de forma literal, isto é, como sendo sinônimo de computação sem servidores?

R: Pois ainda existem servidores, o termo serverless é pelo fato que a empresa não precisa ter um hardware de servidores e computadores, eles contratam alguma empresa que tenha esse recurso, usando nuvem por exemplo.

3. Suponha uma agenda de compromissos construída usando-se uma arquitetura baseada em funções serverless. Mostra-se abaixo uma das funções dessa aplicação, a qual retorna todos os compromissos inseridos na agenda (esse código foi copiado do seguinte repositório). Qual a desvantagem de serverless, conforme discutido na seção final do artigo, fica mais clara ao analisarmos o código dessa função? Justifique brevemente sua resposta.

R: Grande dependência com o provedor do recurso serverless e maior latência.

1 Consistência de Dados em Microsserviços usando-se Sagas

1. Por que microsserviços não devem compartilhar um único banco de dados? Para responder, você pode consultar a Seção 7.4.1 do Capítulo 7 e também o início da Seção 7.4.

R: Pois com um só banco de dados para todos os microsserviços, pode criar um gargalo na evolução do sistema.

2. Qual a diferença entre uma transação distribuída e uma saga? Mais especificamente:

1. Quando consideradas individualmente, as transações de uma saga são atômicas?

   Sim

2. Se não houvesse compensações, as transações de uma saga, quando consideradas em conjunto, seriam sempre atômicas?

   Sim

3. Suponha uma transação Ti de uma saga. Uma transação T’ que não faz parte da saga pode observar os resultados de Ti antes da execução completa da saga?

   não

4. Suponha uma transação distribuída T. Uma segunda transação T’ pode observar os resultados ainda intermediários de T?

   não

5. Com sagas, temos que escrever a lógica de rollback, isto é, o código das compensações. O mesmo acontece com transações distribuídas? Sim ou não? Justifique.

   não

3. Como um desenvolvedor deve proceder quando uma compensação Ci falhar (isto é, não puder ser executada com sucesso)?

R: Executa de novo.

4. Qual problema de transações de longa duração é resolvido por meio de sagas?

R: Que caso falhe alguma transação, é possível reverter todo o processo que ocorreu.