Estratégia Concursos

Aula 00 - Prof. Diego Carvalho e Fernando Pedrosa

PB-Saúde (Analista de Rede Sistemas Software) Engenharia de software - 2024 (Pós-Edital)

Autor:

Diego Carvalho

09 de Novembro de 2024

Índice

APRESENTAÇÃO DO PROFESSOR

PROFESSOR DIEGO CARVALHO - WWW.INSTAGRAM.COM/PROFESSORDIEGOCARVALHO

Sobre o curso: galera, todos os tópicos da aula possuem Faixas de Incidência, que indicam se o assunto cai muito ou pouco em prova. Diego, se cai pouco para que colocar em aula? Cair pouco não significa que não cairá justamente na sua prova! A ideia aqui é: se você está com pouco tempo e precisa ver somente aquilo que cai mais, você pode filtrar pelas incidências média, alta e altíssima; se você tem tempo sobrando e quer ver tudo, vejam também as incidências baixas e baixíssimas. *Fechado?*

INCIDÊNCIA EM PROVA: BAIXÍSSIMA

INCIDÊNCIA EM PROVA: BAIXA

INCIDÊNCIA EM PROVA: MÉDIA

INCIDÊNCIA EM PROVA: ALTA

INCIDÊNCIA EM PROVA: ALTÍSSIMA

Além disso, essas faixas não são por banca – é baseado tanto na quantidade de vezes que caiu em prova independentemente da banca quanto nas minhas próprias avaliações sobre cada assunto.

#ATENÇÃO

Avisos Importantes

O curso abrange todos os níveis de conhecimento...

Esse curso foi desenvolvido para ser acessível a **alunos com diversos níveis de conhecimento diferentes.** Temos alunos mais avançados que têm conhecimento prévio ou têm facilidade com o assunto. Por outro lado, temos alunos iniciantes, que nunca tiveram contato com a matéria ou até mesmo que têm trauma dessa disciplina. A ideia aqui é tentar atingir ambos os públicos - iniciantes e avançados - da melhor maneira possível..

Por que estou enfatizando isso?

O **material completo** é composto de muitas histórias pessoais, exemplos, metáforas, piadas, memes, questões, desafios, esquemas, diagramas, imagens, entre outros. Já o **material simplificado** possui exatamente o mesmo núcleo do material completo, mas ele é menor e mais objetivo. *Professor, eu devo estudar por qual material?* Se você quiser se aprofundar nos assuntos ou tem dificuldade com a matéria, necessitando de um material mais passo-a-passo, utilize o material completo. Se você não quer se aprofundar nos assuntos ou tem facilidade com a matéria, necessitando de um material mais direto ao ponto, utilize o material simplificado.

Por fim...

O curso contém diversas questões espalhadas em meio à teoria. Essas questões possuem um comentário mais simplificado porque **têm o único objetivo de apresentar ao aluno como bancas de concurso cobram o assunto previamente administrado**. A imensa maioria das questões para que o aluno avalie seus conhecimentos sobre a matéria estão dispostas ao final da aula na lista de exercícios e **possuem comentários bem mais abrangentes**.

ESTRATÉGIA FLASHCARDS

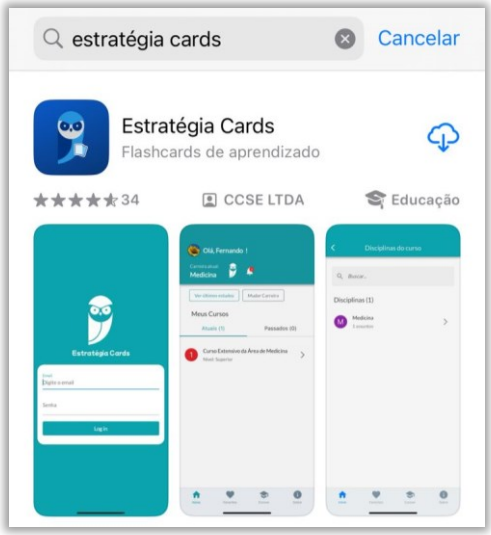
📚 Você tem dificuldade de estudar, memorizar e revisar os conteúdos que estuda em nossas aulas? Então nós temos a ferramenta perfeita para você!

Apresentamos o **Estratégia Cards**: app de flashcards que vai revolucionar sua forma de **estudar** e **revisar** conteúdos de provas de concurso público. Com nossa tecnologia inovadora e interface amigável, você dominará os tópicos mais complexos de maneira eficiente e divertida.

✨ Recursos do Estratégia Cards:

Curadoria de Flashcards	Flashcards criados e revisados por professores especializados em cada área, com qualidade e voltados para concursos públicos.
Flashcards Personalizados	Crie seus próprios flashcards, cobrindo os principais tópicos e matérias dos concursos públicos.
Repetição Espaçada	Técnica de aprendizagem que envolve revisar informações em intervalos crescentes para melhorar a retenção de longo prazo e combater o esquecimento.
Estatísticas Personalizadas	Visualize graficamente o percentual de acertos, erros ou dúvidas dos decks estudados.
Modo Offline	Estude em qualquer lugar, mesmo sem conexão à internet, fazendo o download dos decks.
Estudo por Áudio	*Está dirigindo ou fazendo esteira e quer continuar estudando?* Basta utilizar a opção "Escutar".
Decks Favoritos	Você pode escolher decks específicos como favoritos e visualizá-los em uma aba separada do app.
Opções de Estudo	Você poderá estudar todos os cards de um deck; ou apenas os que você errou; ou apenas os que você não estudou ainda; entre outras opções.

E como eu consigo baixar?



É muito fácil! Basta pesquisar por "Estratégia Cards" na loja oficial do seu smartphone.

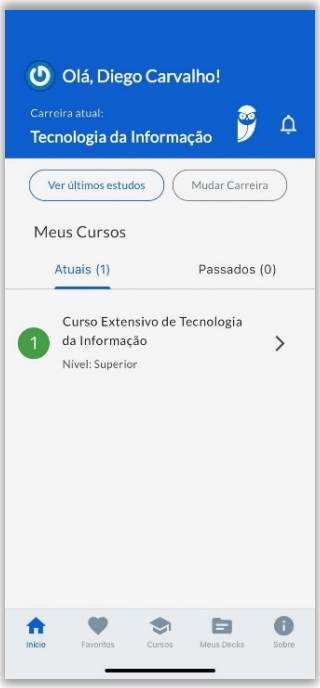
Se você tiver um Android, basta acessar a **Google Play**;

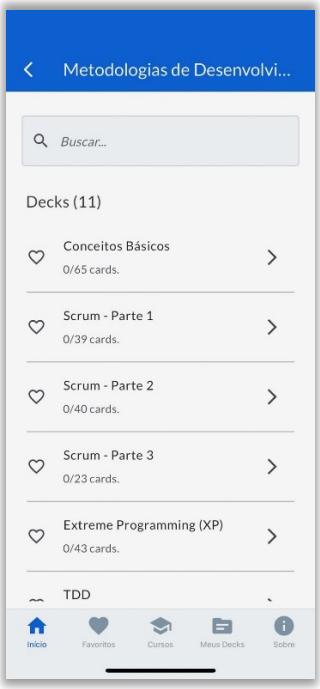
Se for tiver um iPhone, basta acessar a **App Store (iOS)**.

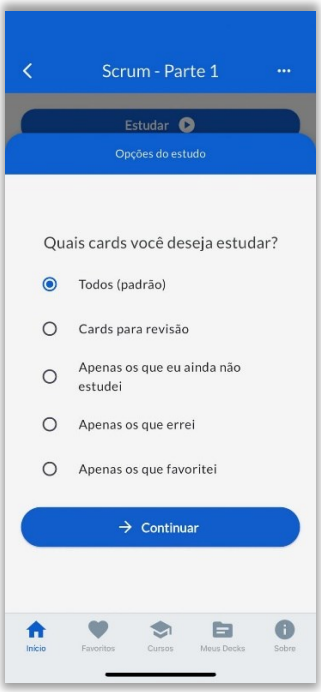
E para acessar?

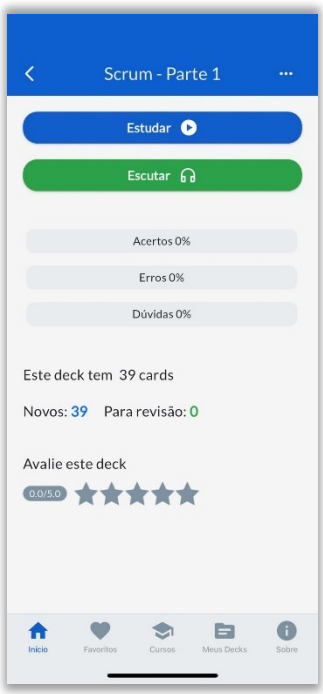
Para acessar, basta ter uma conta no Estratégia Concursos. Em seguida, utilize suas credenciais de login e senha para acessar o aplicativo. Por fim, acessa a carreira de Tecnologia da Informação.

Como utilizar o app:









ARQUITETURA DE SOFTWARE

Conceitos Básicos

INCIDÊNCIA EM PROVA: BAIXA

Ao se considerar a arquitetura de um edifício, vários atributos diferentes vêm à mente. No nível mais simplista, pensamos na forma geral da estrutura física. No entanto, na realidade, arquitetura é muito mais do que isso. Ela é a maneira pela qual os vários componentes do edifício são integrados para formar um todo coeso. É o modo pelo qual o edifício se ajusta em seu ambiente e integra com outros edifícios da vizinhança.

É o grau com que o edifício atende seu propósito expresso e satisfaz às necessidades de seu proprietário. É o sentido estético da estrutura – o impacto visual do edifício – e a maneira pela qual as texturas, cores e materiais são combinados para criar a fachada e o "ambiente de moradia". Ela engloba também os detalhes – o projeto dos dispositivos de iluminação, o tipo de piso, o posicionamento de painéis, enfim, a lista é interminável.

E, finalmente, ela é arte. Mas arquitetura também é algo mais. É constituída por "milhares de decisões, tanto as grandes como as pequenas". Algumas dessas decisões são tomadas logo no início do projeto e podem ter um impacto profundo sobre todas as ações subsequentes. Outras são postergadas ao máximo, eliminando, portanto, restrições demasiadas que levariam a uma implementação inadequada do estilo da arquitetura. *Mas o que dizer da arquitetura de software?*

Alguns autores definem esse termo difícil de descrever da seguinte maneira: "*A arquitetura de software de um programa ou sistema computacional é a estrutura ou estruturas do sistema, que abrange os componentes de software, as propriedades externamente visíveis desses componentes e as relações entre eles*". **De outra forma, a arquitetura de software é a organização ou a estrutura dos componentes significativos do sistema de software que interagem por meio de interfaces.**

A arquitetura não é o software operacional, mas – sim – uma representação que nos permite (1) analisar a efetividade do projeto no atendimento dos requisitos declarados, (2) considerar alternativas de arquitetura em um estágio quando realizar mudanças de projeto ainda é relativamente fácil e (3) reduzir os riscos associados à construção do software. Essa definição enfatiza o papel dos "componentes de software" em qualquer representação de arquitetura.

No contexto de projeto da arquitetura, um componente de software pode ser algo tão simples quanto um módulo de programa ou uma classe orientada a objetos, porém ele também pode ser estendido para abranger bancos de dados e "middleware" que possibilita a configuração de uma rede de clientes e servidores. As propriedades dos componentes são aquelas características necessárias para o entendimento de como eles interagem com outros componentes.

No nível da arquitetura, propriedades internas (por exemplo, detalhes de um algoritmo) não são especificadas. As relações entre componentes podem ser tão simples quanto a chamada procedural de um módulo a outro ou tão complexo quanto um protocolo de acesso a banco de dados. Uma arquitetura bem projetada deve ser capaz de atender aos requisitos funcionais e não-funcionais do sistema de software e ser suficientemente flexível para suportar requisitos voláteis.

A arquitetura é importante, na medida em que ela permite uma comunicação efetiva entre as partes interessadas, abrangendo a compreensão, negociação e consenso. **Além disso, permite decisões tempestivas, isto é, possibilita correção e validação do sistema antes da implementação.** Por fim, permite uma abstração reutilizável do sistema em situações diferentes com características similares.

Uma forma de organizar a arquitetura de um sistema complexo em partes menores é por meio da utilização de camadas de software. Seguindo essa abordagem, cada camada corresponderá a um conjunto de funcionalidades de um sistema de software – sendo que as funcionalidades de alto nível dependerão das funcionalidades de baixo nível. A separação em camadas fornece um nível de abstração através do agrupamento lógico de subsistemas relacionados entre si.

Parte-se do princípio de que as camadas de abstração mais altas devem depender das camadas de abstração mais baixas – isso permite que software seja mais portável/modificável. Eventuais mudanças em uma camada mais baixa, que não afetem a sua interface, não implicarão mudanças nas camadas mais superiores; e mudanças em uma camada mais alta, que não impliquem a criação de um novo serviço em uma camada mais baixa, não afetarão as camadas mais inferiores.

A arquitetura em camadas permite melhor separação de responsabilidades; decomposição de complexidade; encapsulamento de implementação; maior reúso e extensibilidade. No entanto, não são apenas benefícios! **Elas podem penalizar o desempenho do sistema e aumentar o esforço ou complexidade de desenvolvimento do software**. As camadas encapsulam bem algumas coisas, mas não todas. Isso, às vezes, resulta em alterações em cascata.

O exemplo clássico disto em uma aplicação corporativa em camadas é o acréscimo de um campo que precise ser mostrado na interface com o usuário e deva estar no banco de dados e assim deva também ser acrescentado a cada camada entre elas. **Camadas extras podem prejudicar o desempenho.** Em cada camada, os dados precisam, tipicamente, ser transformados de uma representação para outra.

O encapsulamento de uma função subjacente, no entanto, muitas vezes lhe dá ganhos de eficiência que mais do que compensam esse problema. Uma camada que controla transações pode ser otimizada e então tornará tudo mais rápido. Galera, camadas extras não necessariamente prejudicam o desempenho! Em geral, é isso que acontece! Porém, como foi dito, o encapsulamento de funções pode muitas vezes gerar ganhos de desempenho.

Esse assunto é polêmico e já caiu em prova, portanto vocês devem ter atenção! Bem, acredito que isso seja suficiente para posteriormente entender melhor cada arquitetura.

(MPOG/ATI – 2015) Embora normalmente os sistemas desenvolvidos se baseiem em padrões de arquitetura, cada um deles tem arquitetura totalmente específica, em consequência dos seus requisitos.

_____

Comentários: de fato, eu posso usar padrões de arquitetura, tais como uma arquitetura em camadas, uma arquitetura distribuída, uma arquitetura mainframe ou uma arquitetura orientada a serviços. Embora cada sistema tenha uma arquitetura baseada em seus requisitos, elas não são totalmente específicas (Errado).

Coesão e Acoplamento

INCIDÊNCIA EM PROVA: MÉDIA

Bem, falemos brevemente sobre esses dois importantes princípios de engenharia de software: coesão e acoplamento! Eles são fundamentais no conceito de arquitetura de software. Logo, preciso que vocês decorem a seguinte frase como se fosse um mantra: ***Uma boa arquitetura de software deve ter componentes de projeto com baixo acoplamento e alta coesão***. *Como é?* Acoplamento trata do nível de dependência entre módulos ou componentes de um software.

Já a Coesão trata do nível de responsabilidade de um módulo em relação a outros. *Professor, por que é bom ter baixo acoplamento?* **Porque se os módulos pouco dependem um do outro, modificações de um não afetam os outros, além de não prejudicar o reúso.** Se esse princípio não for observado durante a construção da arquitetura de um sistema de software, pode haver problemas sérios de manutenção futura!

Professor, por que é bom ter alta coesão? Porque se os módulos têm responsabilidades claramente definidas, eles serão altamente reusáveis, independentes e simples de entender.

A Coesão está ligada ao princípio da responsabilidade única, que diz que uma classe deve ter uma e apenas uma responsabilidade e realizá-la de maneira satisfatória, isto é, a força funcional relativa de um módulo ou componente de software. **O acoplamento está ligado ao grau de conexão entre módulos em uma estrutura de software.** Eu apresento na tabela abaixo os vários tipos de acoplamento:

TIPO DE ACOPLAMENTO	DESCRIÇÃO
ACOPLAMENTO POR CONTEÚDO	Ocorre quando um módulo faz uso de estruturas de dados ou de controle mantidas no escopo de outro módulo.
ACOPLAMENTO	Ocorre quando um conjunto de módulos acessa uma área global de dados.

COMUM	
ACOPLAMENTO POR CONTROLE	Ocorre quando módulos passam decisões de controle a outros módulos.
ACOPLAMENTO POR DADOS	Ocorre quando apenas uma lista de dados simples é passada como parâmetro de um módulo para o outro, com uma correspondência um-para-um de itens.
ACOPLAMENTO POR CHAMADAS DE ROTINAS	Ocorre quando uma operação chama outra. Nesse nível de acoplamento, é comum e, quase sempre, necessário. Entretanto, realmente aumenta a conectividade de um sistema.
ACOPLAMENTO POR USO DE TIPOS	Ocorre quando o Componente A usa um tipo de dados definido em um Componente B. Se a definição de tipo mudar, todo componente que usa a definição também terá de ser alterado.
ACOPLAMENTO POR INCLUSÃO OU IMPORTAÇÃO	Ocorre quando um Componente A importa ou inclui um pacote ou o conteúdo do Componente B.
ACOPLAMENTO EXTERNO	Ocorre quando um componente se comunica ou colabora com componentes de infraestrutura. Embora seja necessário, deve-se limitar a um pequeno número de componentes em um sistema.

No contexto do projeto de componentes para sistemas orientados a objetos, coesão implica um componente ou classe encapsular apenas atributos e operações que estejam intimamente relacionados entre si e com a classe ou o componente em si. **A comunicação e a colaboração são elementos essenciais de qualquer sistema orientado a objetos.** Há, entretanto, o lado sinistro desse importante (e necessária) característica.

Como o volume de comunicação e colaboração aumenta (isto é, à medida que o grau de conexão entre as classes aumenta), a complexidade do sistema também cresce. **E à medida que a complexidade aumenta, a dificuldade de implementação, testes e manutenção do software também aumentam**. O acoplamento é uma medida qualitativa do grau com que as classes estão ligadas entre si.

Conforme as classes (e os componentes) se tornam mais interdependentes, o acoplamento aumenta – a ideia é tentar manter o acoplamento o mais baixo possível...

Arquitetura em Camadas

INCIDÊNCIA EM PROVA: ALTÍSSIMA

Inicialmente, os aplicativos eram combinados em uma única camada, incluindo Banco de Dados, Lógica do Aplicativo e Interface de Usuário. O aplicativo, em geral, era executado em um computador de grande porte, e os usuários o acessavam por meio de *terminais burros* que podiam apenas realizar a entrada e exibição de dados. Essa abordagem tem o benefício de ser mantida por um administrador central.

As arquiteturas de uma camada têm um grave empecilho: os usuários esperam interfaces gráficas que exigem muito mais poder computacional do que o dos simples terminais burros. A computação centralizada de tais interfaces exige muito mais poder computacional do que um único servidor tem disponível, **e assim as arquiteturas de uma camada não consegue suportar milhares de usuários**. Temos, então, a Arquitetura Cliente-Servidor de duas camadas.

Para explicá-la, precisamos passar alguns conceitos básicos. Bem, ela é organizada como um conjunto de serviços, além de servidores e clientes associados que os acessam e os usam. Compõem esse modelo: servidores, que oferecem serviços; clientes, que solicitam os serviços; e uma rede que permite aos clientes acessarem esses serviços.

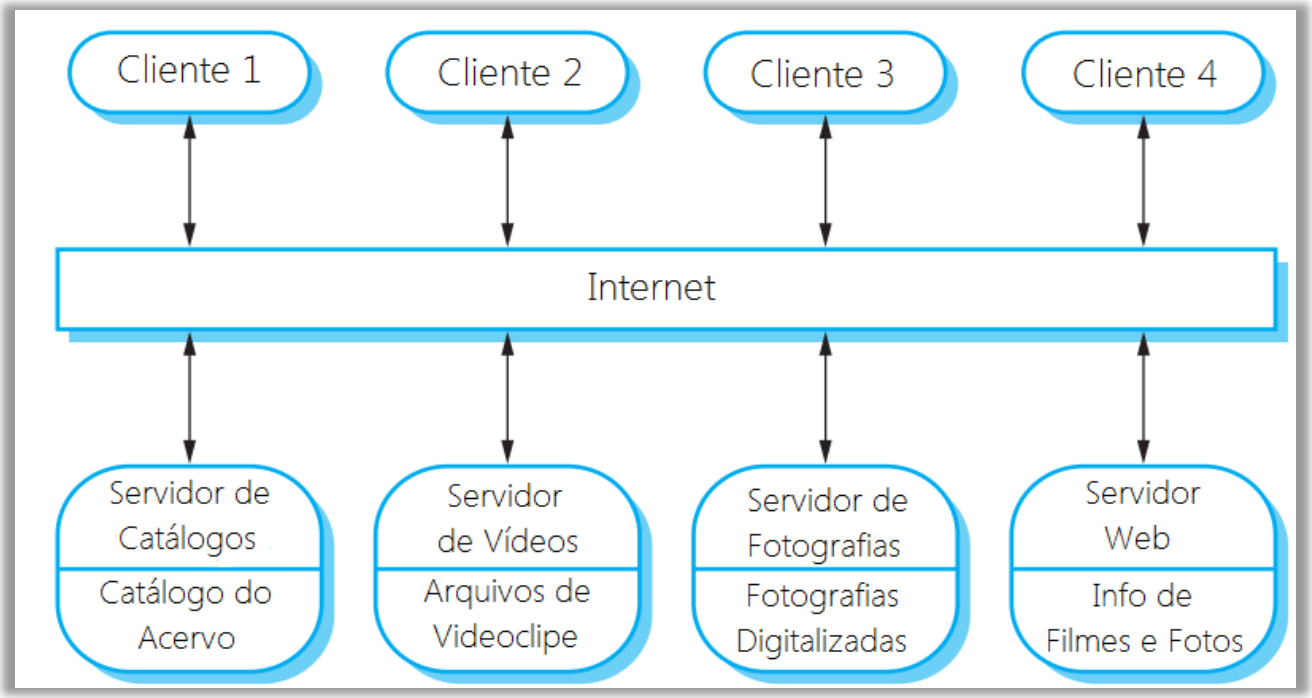
O processamento da informação é dividido em módulo ou processos distintos, sendo uma abordagem de computação que separa os processos em plataformas independentes que interagem, **permitindo que os recursos sejam compartilhados enquanto se obtém o máximo de benefício de cada dispositivo diferente**. Os principais componentes desse modelo são:

SERVIDORES	Oferecem serviços para outros subsistemas (Ex: Servidores de Impressão, Servidores de Arquivos, Servidor de Compilação, entre outros).
CLIENTES	Solicitam os serviços oferecidos pelos servidores. Geralmente são independentes, podendo ser executados simultaneamente.
REDE	Permite aos clientes acessarem esses serviços – quando clientes e servidores podem ser executados em uma única máquina, não são necessários.

Clientes iniciam/terminam a comunicação com servidores, solicitando/terminando serviços distribuídos. Eles não se comunicam com outros clientes diretamente, são responsáveis pela entrada e saída de dados e comunicação com o usuário e tornam a rede transparente ao usuário – em geral, são computadores pessoais conectados a uma rede. Já os servidores realizam uma execução contínua.

Eles recebem e respondem solicitações dos clientes, não se comunicam com outros servidores diretamente, prestam serviços distribuídos, atendem a diversos clientes simultaneamente – em geral, possuem um poder de processamento e armazenamento mais alto que de um cliente. Os clientes talvez precisem saber os nomes dos servidores e os serviços que eles fornecem, mas os servidores não precisam saber sobre os clientes.

Eles acessam os serviços pelo servidor por meio de chamadas remotas de procedimento usando, por exemplo, HTTP. **Um cliente faz um pedido a um servidor e espera até receber uma resposta**.

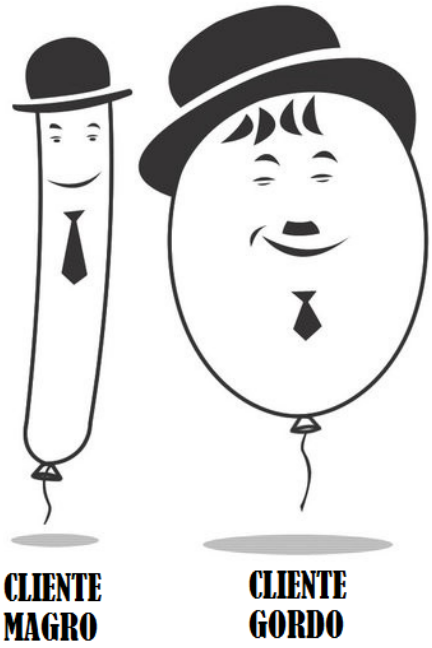


Essencialmente, um cliente faz um pedido a um servidor e espera até receber uma resposta. A imagem acima mostra um exemplo de sistema baseado no modelo cliente-servidor. Ele é multiusuário e baseado na Web, para fornecer um acervo de filmes e fotografias. Nesse sistema, vários servidores gerenciam e apresentam tipos diferentes de mídia. Os frames do vídeo precisam ser transmitidos rapidamente em sincronia, mas com uma resolução relativamente baixa.

Podem ser comprimidos em um repositório e, assim, o servidor pode cuidar da compressão/descompressão do vídeo. **Fotografias devem estar em alta resolução, portanto devem ser mantidas em um servidor dedicado**. O catálogo deve ser capaz de lidar com várias consultas e de fornecer links para Sistemas Web de informações com dados do filme e videoclipe, e um sistema de e-commerce que apoie a venda desses.

O programa cliente é simplesmente uma interface integrada com o usuário, construída com o uso de um navegador Web para esses serviços. A vantagem mais importante de um modelo cliente-servidor é que ele é uma arquitetura distribuída. O uso efetivo de sistemas em rede pode ser feito com muitos processadores distribuídos. É fácil adicionar um novo servidor e integrá-lo ao restante do sistema ou atualizar servidores de maneira transparente sem afetar outras partes do sistema.

As arquiteturas cliente-servidor de duas camadas podem ter duas formas: Cliente-Magro ou Cliente-Gordo. *Como é isso, professor?*



No Modelo Cliente-Magro, todo processamento da aplicação e o gerenciamento de dados é realizado no servidor. O cliente é responsável somente por executar o software de apresentação, portanto é magro! **No Modelo Cliente-Gordo**, o servidor somente é responsável pelo gerenciamento de dados e o software do cliente implementa a lógica da aplicação e as interações com os usuários, portanto é gordo!

VANTAGENS DE CLIENTES MAGROS	Baixo custo de administração; facilidade de proteção; baixo custo de hardware; menor custo para licenciamento de softwares; baixo consumo de energia; resistência a ambientes hosts; menor dissipação de calor para o ambiente; mais silencioso que um PC convencional; mais ágil para rodar planilhas complexas; entre outros.
DESVANTAGENS DE CLIENTES MAGROS	Se o servidor der problema e não houver redundância, todos os clientes-magros ficarão inoperantes; necessita de maior largura de banda na rede em que é empregado; em geral, possui um pior tempo de resposta, uma vez que usam o servidor para qualquer transação; apresenta um apoio transacional menos robusto; etc.

VANTAGENS DE CLIENTES GORDOS	Necessitam de requisitos mínimos para servidores; apresenta uma performance multimídia melhor; possui maior flexibilidade; algumas situações se beneficiam bastante, tais como jogos eletrônicos, em que se beneficiam por conta de sua alta capacidade de processamento e de hardware específico.
DESVANTAGENS DE CLIENTES GORDOS	Não há um local central para atualizar e manter a lógica do negócio, uma vez que o código do aplicativo é executado em vários locais de cliente; é exigida uma grande confiança entre o servidor e os clientes por conta do banco de dados; não suporta bem o crescimento do número de clientes.

Comparada à arquitetura de uma camada, as arquiteturas de duas camadas separam fisicamente a interface do usuário da camada de gerenciamento de dados. Para implementar arquiteturas de duas camadas, não se pode mais ter terminais burros no lado do cliente; precisa-se de computadores que executem código de apresentação sofisticado (e, possivelmente, a lógica do aplicativo).

Sistemas Cliente-Servidor em duas camadas foram dominantes durante aproximadamente toda a década de noventa e são utilizados até hoje. Todavia, para minimizar o impacto de mudanças nas aplicações, decidiu-se separar a camada de negócio da camada de interface gráfica, gerando três camadas[1]: Camada de Apresentação, Camada Lógica do Negócio e Camada de Acesso a Dados. Vejamos...

CAMADA DE APRESENTAÇÃO	Também chamada de Camada de Interface, possui classes que contêm funcionalidades para visualização dos dados pelos usuários. Ela tem o objetivo de exibir informações ao usuário e traduzir ações do usuário em requisições às demais partes dos sistemas. O amplo uso da internet tornou as interfaces com base na web crescentemente populares.
CAMADA DE NEGÓCIO	Também chamada Camada Lógica ou de Aplicação, possui classes que implementam as regras de negócio no qual o sistema será implantado. Ela realiza cálculos com base nos dados armazenados ou nos dados de entrada, decidindo que parte da camada de acesso de ser ativada com base em requisições provenientes da camada de apresentação.
CAMADA DE DADOS	Possui classes que se comunicam com outros sistemas para realizar tarefas ou adquirir informações para o sistema. Tipicamente, essa camada é implementada utilizando algum mecanismo de armazenamento persistente. Pode haver uma subcamada dentro desta camada chamada Camada de Persistência ou Camada de Acesso.

Nessa arquitetura, a apresentação, o processamento de aplicações e o gerenciamento de dados são processos logicamente separados executados por processadores diferentes. Seu uso permite a otimização da transferência de informações entre o servidor web e o de banco de dados. **As comunicações entre esses sistemas podem usar protocolos rápidos de comunicações de baixo nível**.

Um middleware eficiente que apoia consultas de banco de dados em SQL (Structured Query Language) é usado para cuidar da recuperação de informações do banco de dados. Em alguns casos, é adequado estender o modelo cliente-servidor de três camadas para uma variante com várias camadas, na qual servidores adicionais são incorporados ao sistema.

Esses sistemas podem ser usados quando as aplicações necessitam acessar e usar dados de bancos de dados diferentes. Nesse caso, um servidor de integração é colocado entre o servidor de aplicações e os servidores de banco de dados. O servidor de integração coleta os dados distribuídos e apresenta-os como se fossem provenientes de um único banco de dados. Um sistema de operações bancárias online é um exemplo de arquitetura cliente-servidor de três camadas.

O banco de dados de clientes fornece serviços de gerenciamento de dados; um servidor web fornece os serviços de aplicação, como recursos para transferir dinheiro, gerar extratos, pagar contas, etc; e uma interface gráfica amigável fornece a apresentação dos dados ao cliente. O

[1] Galera, diz-se Arquitetura em Três Camadas, 3-Layers Architecture ou 3-tiers Architecture. Apesar de muitas pessoas usarem os dois termos indiferentemente, eles não são iguais: Layers são camadas lógicas, isto é, pode haver três layers em uma única máquina e os Tiers são camadas físicas, isto é, pode haver apenas um tier por máquina. *Entenderam?* ;)

próprio computador com um browser é o cliente. **Esse sistema é escalonável, pois é relativamente fácil adicionar novos servidores web quando o número de clientes aumenta**.

O uso de uma arquitetura de três camadas, nesse caso, permite a otimização da transferência de informações entre o servidor web e o servidor de banco de dados. As comunicações entre esses sistemas podem usar protocolos de comunicações de baixo nível. Um middleware eficiente que apoia consultas de banco de dados é usado para cuidar da recuperação de informações do banco de dados.

A arquitetura de três camadas que distribuem o processamento de aplicações entre os clientes são mais escalonáveis. **O tráfego de rede é reduzido em comparação com arquiteturas cliente-magro de duas camadas**. O processamento de aplicações é mais volátil e pode ser facilmente atualizado, pois está no centro. O processamento pode ser distribuído entre os servidores de lógica de aplicações e de gerenciamento de dados, conseguindo respostas mais rápidas. *Vamos resumir?*

Bem, havia a arquitetura de uma camada (monolítica)! Nesse caso, um aplicativo era desenvolvido para ser usado em uma única máquina. **Esse aplicativo abarcava toda a funcionalidade em um único módulo**, isto é, a interface com usuário, lógica de aplicação e acesso a bancos de dados estavam presentes em um mesmo lugar. Logo, a necessidade de compartilhar a lógica de acesso a dados entre vários usuários fez surgir o modelo de arquitetura cliente-servidor em duas camadas.

Dessa forma, a base de dados foi colocada em uma máquina específica, separada das máquinas que executavam, de fato, as aplicações ou em uma mesma máquina, porém em processadores diferentes. Nessa arquitetura, os aplicativos eram instalados em estações clientes contendo toda a apresentação e lógica de aplicação. No entanto, quando havia alguma nova versão, tinha-se que reinstalar o software ou instalar a atualização em todas as (talvez milhares de) máquinas.

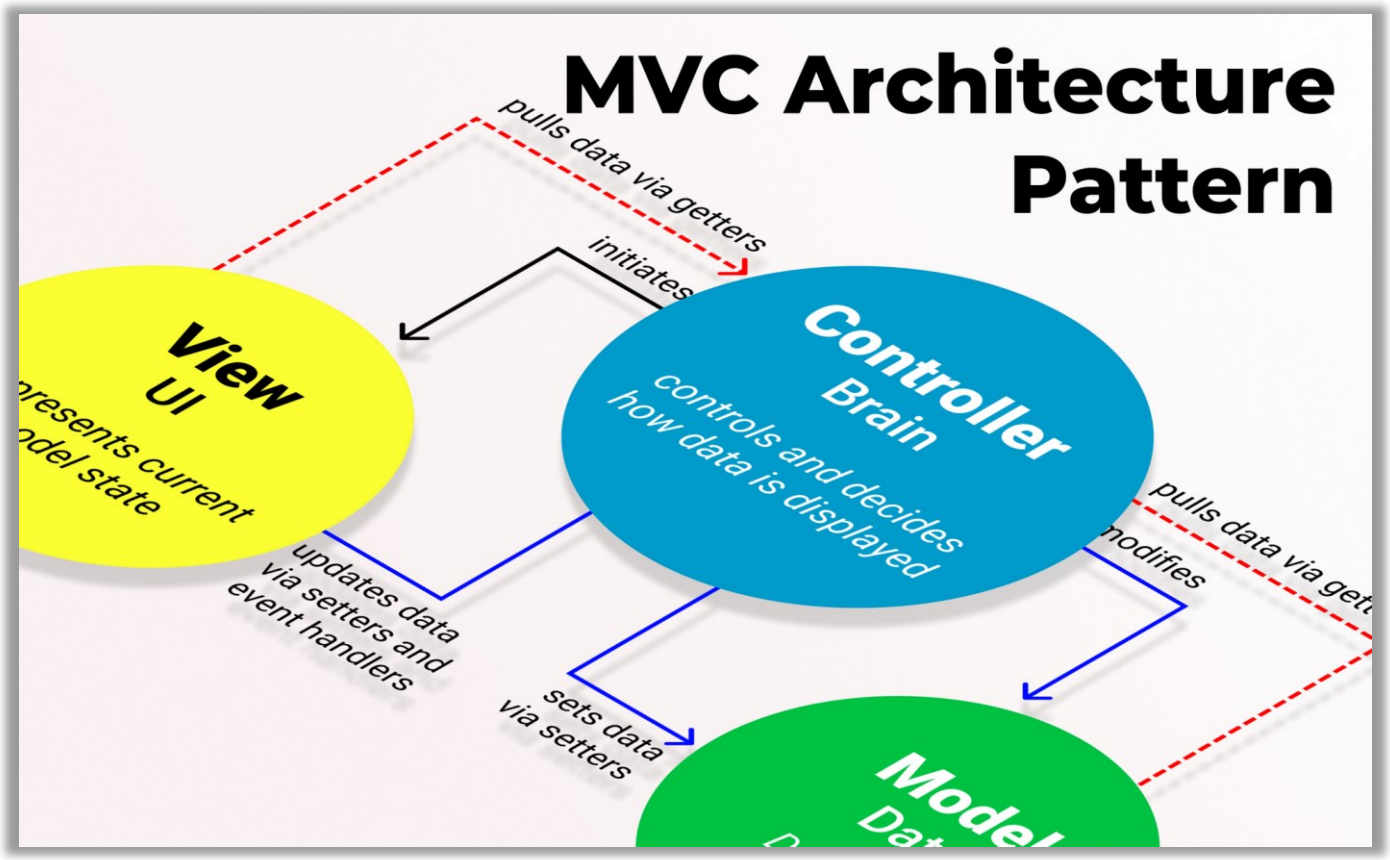
Esse problema fez surgir a Arquitetura Cliente-Servidor em Três Camadas. **Essa abordagem retirava a lógica de negócio da máquina do cliente e a centralizava em um servidor chamado Servidor de Aplicação**. Assim, o acesso ao Banco de Dados era feito seguindo regras de negócio contidas no Servidor de Aplicação, facilitando a atualização dos aplicativos. Contudo, atualizações na Camada de Apresentação precisavam ser distribuídas em todas as máquinas da rede.

Logo depois, surgiu uma nova abordagem: Arquitetura Cliente-Servidor em Quatro Camadas! **Ela surgiu com a ideia de retirar a Apresentação do cliente e centralizá-la em um Servidor Web**. Assim, **não havia necessidade de instalar o aplicativo na máquina do cliente**, o acesso era feito por meio de um navegador. As camadas eram: Dados, Aplicação, Apresentação e Cliente (Navegador Web).

Grosso modo: um usuário faz uma requisição por meio de um Navegador (Camada do Cliente), essa requisição é passada para um Servidor Web (Camada de Apresentação), que a processa e procura a regra de negócio correspondente no Servidor de Aplicação (Camada de Aplicação), que procura os dados no banco de dados (Camada de Dados).

Arquitetura MVC

INCIDÊNCIA EM PROVA: ALTÍSSIMA



O Model-View-Controller (MVC) é um padrão arquitetural de software para implementar interfaces de usuário. **Ele divide uma aplicação de software em três partes interconectadas, de modo a separar representações internas de informação das formas em que a informação é apresentada para o usuário.** Galera, esse é um assunto que pode ser bastante aprofundado, vou tentar simplificá-lo nessa aula.

Em uma linguagem bem simples e direta, ele é um padrão arquitetural de software que separa uma aplicação em três camadas. Você pode entendê-lo como uma forma de organizar o código de uma aplicação de forma que sua manutenção fique mais fácil. Trata-se da separação de responsabilidades, sendo uma maneira de quebrar uma aplicação (ou parte dela) em camadas: Modelo, Visão e Controle.

O MVC promove a estrita separação de responsabilidade entre componentes de uma interface gráfica onde temos componentes responsáveis pela manutenção do estado da aplicação, denominado de Modelo, pela exibição de parte deste modelo para o usuário, ao que chamamos de Visão e pela coordenação entre atualizações no modelo e interações com o usuário, feita através do Controlador.

Durante a década de setenta, surgiu a necessidade de criação de uma arquitetura para ser utilizada em projetos de interface visual na linguagem de programação Smalltalk**. A ideia original era organizar o código, separar responsabilidades, aumentar a manutenibilidade, promover um baixo acoplamento e uma alta coesão, fomentar a reusabilidade do código e tornar o sistema escalável.**

Passou um bocadinho de tempo e, com o surgimento da WWW, algumas pessoas pensaram em adaptar esse padrão arquitetural para o mundo web. **Muitos frameworks de aplicação comerciais e não comerciais foram criados tendo como base esse modelo.** Estes frameworks variam em suas interpretações, principalmente no modo que as responsabilidades MVC são divididas entre o cliente e servidor.

Camada Model

Essa é a camada responsável pela representação dos dados, provendo meios de acesso (leitura/escrita). **Cara, sempre que você pensar em manipulação de dados, (leitura, escrita ou validação de dados[1]), pense na Camada de Modelo!** Ela gerencia não só os dados, mas também os comportamentos fundamentais da aplicação – representados por regras de negócio (Sim, elas ficam na Camada de Modelo!).

A Camada de Modelo encapsula as principais funcionalidades e dados do sistema. Ela notifica suas visões e respectivos controladores quando surge alguma mudança em seu estado, isto é, ela é responsável pela manutenção do estado da aplicação. Estas notificações permitem que as visões produzam saídas atualizadas e que os controladores alterem o conjunto de comandos disponíveis.

Camada Controller

Essa é a camada responsável por receber todas as requisições do usuário. Seus métodos – chamados *actions* – são responsáveis por uma página, controlando qual modelo usar e qual visão será mostrada ao usuário. **Ele é capaz de enviar comandos para o modelo atualizar o seu estado.** Ele também pode enviar comandos para a respectiva visão para alterar a apresentação da visão do modelo.

A Camada de Controle atende às requisições do usuário e seleciona o modelo e a visão que o usuário usará para interagir com o modelo. O usuário interage com controladores – por meio de visões – que interpretam eventos e entradas enviadas (*Input*), mapeando ações do usuário em comandos que são enviados para o modelo e/ou para a visão para efetuar as alterações apropriadas (*Output*).

[1] A validação ocorre na Camada de Modelo. *Pode ocorrer na Camada de Visão?* Sim, eu posso utilizar um JavaScript para fazer algumas validações de dados, mas isso é inseguro e se trata de uma violação do modelo. Logo, aceitem que validações ocorrem na camada de modelo.

Um controlador define o comportamento da aplicação, interpretando as ações do usuário e mapeando-as em chamadas do modelo. **Em um cliente de aplicações web, essas ações do usuário poderiam ser cliques em botões ou seleções de menus.** As ações realizadas pelo modelo poderiam ser ativar processos de negócio ou alterar o estado do modelo.

Vocês já pensaram no porquê de essa camada ter esse nome? Porque ela controla o fluxo da aplicação, interpretando os dados de entrada e coordenando/orquestrando as manipulações do modelo e as interações com o usuário. Trata-se de uma camada intermediária entre a Visão e o Modelo. **Em geral, há um controlador para cada visão, apesar de poder existir várias controladoras para uma mesma visão.**

Camada View

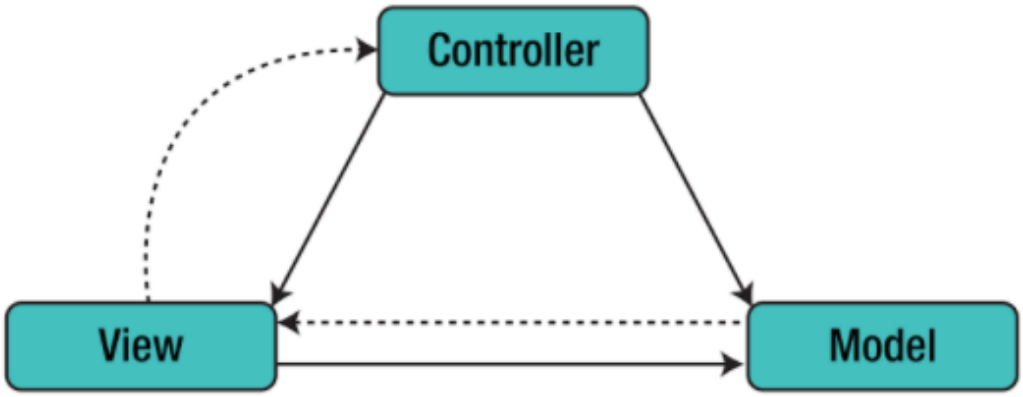
Essa é a camada responsável pela interação com o usuário, sendo responsável apenas pela exibição de dados. Trata-se de uma representação visual do modelo. Ela permite apresentar, de diversas formas diferentes, os dados para o usuário. A visão não sabe nada sobre o que a aplicação está fazendo atualmente, ela recebe instruções do controle, notifica o controle e recebe informações do modelo.

Prosseguindo com nossa linha de pensamento: geralmente, a visão contém formulários, tabelas, menus e botões para entrada e saída de dados. **Além disso, existem diversas visões para cada modelo.** Galera, agora um ponto importante que de vez em quando cai em prova e é importante saber bem para não confundir e acabar errando a questão por bobeira.

À primeira vista, a Arquitetura MVC parece não ter diferença alguma em relação à Arquitetura em Três-Camadas, com o Modelo substituindo a Camada de Dados, a Visão substituindo a Camada de Apresentação e o Controlador substituindo a Camada de Lógica de Negócio. **No entanto, essas duas arquiteturas são diferentes em relação a interação entre suas camadas.**

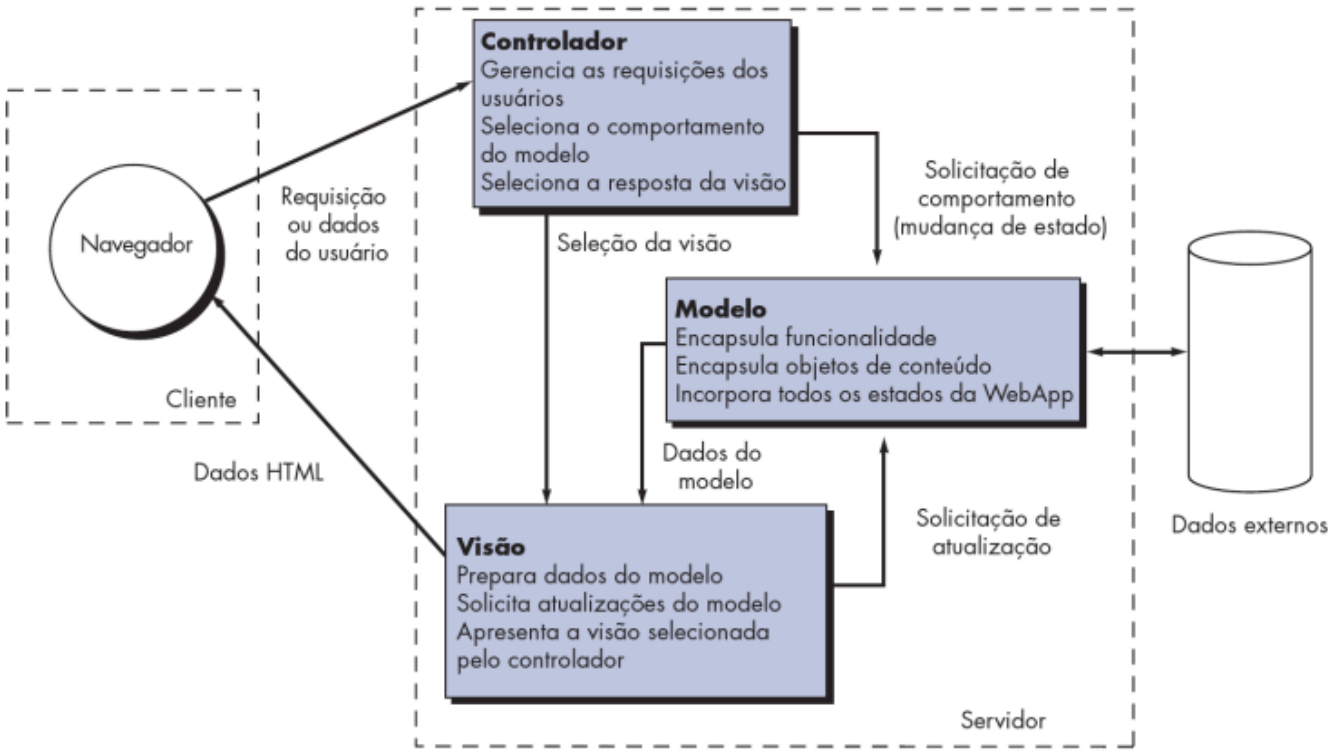
Na Arquitetura em Três-Camadas, a comunicação entre camadas é rigidamente linear, isto é, a Camada de Apresentação e a Camada de Dados só se conversam bidirecionalmente com a Camada de Lógica, mas nunca entre si. Já no MVC, a comunicação é triangular – **existem diversas implementações diferentes dessa arquitetura, uma comunicação típica é apresentada na imagem a seguir.**

Observem que a Visão pode tanto gerar eventos a serem tratados pelo Controle quanto obter os dados a serem exibidos diretamente do modelo. O Controle trata os eventos da Visão, mas também pode manipular diretamente o Modelo. Finalmente, o Modelo pode reagir diretamente tanto à Visão quanto ao Controle, mas também pode gerar eventos a serem tratados pela visão.



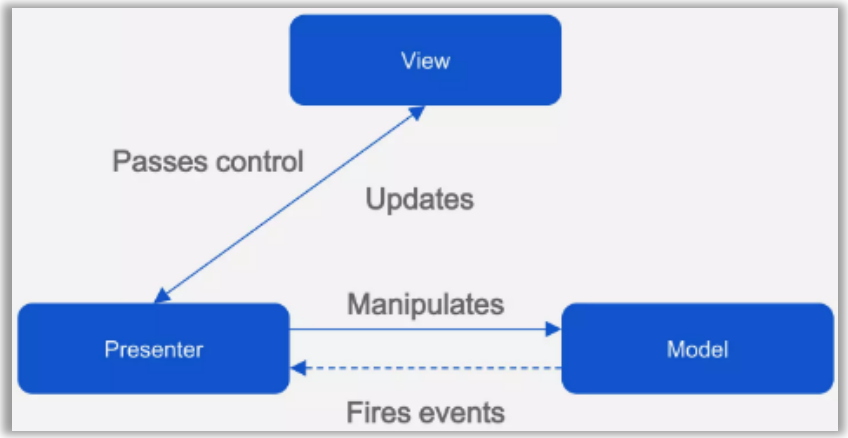
Essa última sentença é extremamente polêmica – vocês encontrarão muitos lugares dizendo que não é possível que a visão solicite diretamente o estado do modelo, mas é possível, sim! **Vamos supor que você esteja comprando cursos no site do Estratégia!** Colocou um no carrinho, colocou dois e colocou o terceiro. Você, então, clica no botão de listar os itens que estão no carrinho.

Nesse momento, a visão não precisa necessariamente fazer uma requisição para o controle, para que o controle peça a lista de itens para o modelo. Ora, ela pode pedir diretamente para o modelo! **Fiquem ligados também que existem diversas visões para cada modelo.** Na imagem abaixo, podemos ver as possíveis interações na Arquitetura MVC!



Arquitetura MVP

INCIDÊNCIA EM PROVA: BAIXA



A Arquitetura MVP (Model-View-Presenter) é um padrão de design amplamente utilizado no desenvolvimento de aplicações de software, especialmente em interfaces gráficas (UI), com o objetivo de separar as responsabilidades e melhorar a manutenibilidade e a escalabilidade da aplicação. O MVP é uma evolução do padrão MVC e é utilizado para desacoplar a interface do usuário da lógica de negócios, garantindo uma maior flexibilidade no desenvolvimento e na testabilidade do código. Vejamos detalhes sobre as suas camadas:

Camada Model

O Modelo é responsável por gerenciar a lógica de negócios e os dados da aplicação. Ele encapsula o estado da aplicação e as regras de negócios, podendo representar uma base de dados, APIs, ou qualquer outro tipo de fonte de dados. O Modelo é independente da interface do usuário e não tem conhecimento direto sobre a View ou o Presenter.

Camada View

A View é responsável por apresentar a interface do usuário. No MVP, a View é passiva, ou seja, ela não tem lógica de negócios e não interage diretamente com o Modelo. A View exibe as informações fornecidas pelo Presenter e envia as interações do usuário (como cliques, toques, etc.) de volta para o Presenter. A View depende completamente do Presenter para realizar qualquer ação, incluindo a recuperação de dados e a atualização da interface.

Camada Presenter

O Presenter atua como intermediário entre o Modelo e a View. Ele contém toda a lógica da interface e a lógica de interação com o Modelo. Ele manipula a comunicação entre a View e o Modelo, obtendo dados do Modelo e formatando-os para a exibição na View. O Presenter também processa os eventos da interface do usuário (UI), como cliques e entradas do usuário, que são enviados pela View. Ao contrário do Controller no MVC, o Presenter sabe sobre a View e mantém uma referência a ela, mas a View não conhece o Presenter diretamente.

No MVP, o usuário interage com a View, acionando um evento (como clicar em um botão). A View informa o Presenter sobre o evento. O Presenter processa o evento, realizando alguma lógica, como buscar dados do Modelo. O Modelo retorna os dados processados ao Presenter. O Presenter manipula os dados e diz à View como exibi-los (atualizando a interface com os dados).

PRINCIPAIS CARACTERÍSTICAS
A principal vantagem do MVP é que ele desacopla a lógica de exibição da lógica de negócios, o que facilita a manutenção do código e a reutilização de componentes.
A View e o Modelo não se comunicam diretamente, todo o fluxo é mediado pelo Presenter.
O Presenter, sendo independente da interface gráfica, é fácil de testar em comparação com a View. A lógica de negócios contida no Presenter pode ser testada em isolamento, sem a necessidade de renderizar a interface do usuário.
O fato de o MVP dividir responsabilidades torna a manutenção mais simples, já que mudanças em uma parte da aplicação (como o layout da View) não afetam a lógica da aplicação (Presenter e Model).
A separação facilita a escalabilidade da aplicação, permitindo que diferentes partes sejam trabalhadas de forma independente.
O MVP proporciona maior flexibilidade para ajustar a interface do usuário sem tocar na lógica do aplicativo, o que é útil em situações em que diferentes versões da interface são necessárias (por exemplo, em diferentes plataformas como desktop, web ou dispositivos móveis).

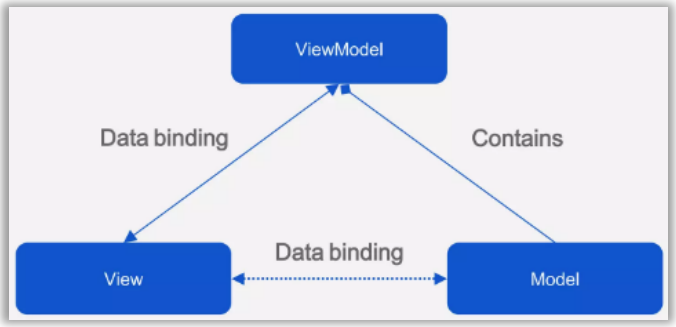
O padrão MVP é amplamente utilizado em aplicações que demandam uma clara separação entre lógica de apresentação e interface gráfica, principalmente em ambientes como aplicações desktop (Windows Forms), dispositivos móveis (Android), e em frameworks como GWT (Google Web Toolkit). Ele é ideal para interfaces mais complexas, onde a lógica de interface precisa ser testada de forma isolada, promovendo maior manutenibilidade e escalabilidade ao código.

MVC x MVP

Por fim, não podemos confundir MVP com MVC. No MVC, a View pode se comunicar diretamente com o Modelo, enquanto no MVP toda a interação entre View e Modelo é mediada pelo Presenter. No MVC, o Controller é responsável por responder a eventos de entrada e pode delegar tarefas ao Modelo e à View. No MVP, o Presenter tem controle total sobre a lógica de apresentação e a comunicação entre Modelo e View, sendo uma parte mais centralizada da arquitetura.

Arquitetura MVVW

INCIDÊNCIA EM PROVA: BAIXA



A Arquitetura MVVM (Model-View-ViewModel) é um padrão de design usado principalmente no desenvolvimento de aplicações que envolvem interfaces gráficas, como aplicativos de desktop e mobile, sendo bastante popular em frameworks como WPF (Windows Presentation Foundation), Xamarin e Angular. O MVVM tem como principal objetivo separar a lógica de negócios, a lógica de apresentação e a interface do usuário, permitindo maior flexibilidade, organização e facilidade de manutenção do código. Vejamos detalhes de suas camadas:

Camada Model

O Modelo é responsável por representar os dados e a lógica de negócios da aplicação. Ele lida com o gerenciamento de dados (como bancos de dados ou serviços de API) e contém as regras de negócios associadas. O Modelo não tem conhecimento direto sobre a ViewModel ou a View, sendo completamente independente da interface do usuário.

Camada View

A View é responsável por exibir a interface do usuário. Ela contém a camada de apresentação e é composta por controles de interface gráfica, como botões, listas e caixas de texto. A View deve ser o mais "passiva" possível, ou seja, sem conter lógica de negócios. A lógica é transferida para a ViewModel e a View apenas "observa" mudanças nos dados. A comunicação entre a View e a ViewModel é feita através de mecanismos de *data binding*.

Camada ViewModel

A ViewModel é a camada intermediária entre a View e o Modelo. Ela é responsável por expor os dados e comandos que a View utiliza para interagir com o Modelo. Ela processa as entradas da View, transforma os dados do Modelo em algo que a View pode apresentar, e também mantém o estado da interface do usuário. O data binding bidirecional permite que as mudanças feitas na ViewModel se reflitam automaticamente na View, e vice-versa.

No MVVW, o usuário interage com a View, por exemplo, clicando em um botão ou digitando em uma caixa de texto. A ViewModel processa essas interações e, se necessário, se comunica com o Modelo para realizar a lógica de negócios ou manipular dados. As mudanças no Modelo são refletidas na ViewModel, que por sua vez, atualiza automaticamente a View por meio do data

binding. O data binding bidirecional permite que a ViewModel e a View mantenham os dados sincronizados sem a necessidade de código adicional. Vamos detalhar o *Data Binding*...

O data binding é um dos aspectos mais poderosos da arquitetura MVVM. Ele permite a sincronização automática entre a View e a ViewModel sem a necessidade de codificação manual. Isso significa que qualquer alteração feita nos dados da ViewModel reflete automaticamente na View, e alterações na interface (como entradas do usuário) são automaticamente transmitidas para a ViewModel.

PRINCIPAIS CARACTERÍSTICAS
A arquitetura MVVM oferece uma separação clara entre a lógica de negócios (Modelo), a lógica de apresentação (ViewModel) e a interface do usuário (View). Isso facilita a manutenibilidade, já que mudanças em uma camada não afetam diretamente as outras.
A ViewModel é facilmente testável, pois ela não depende diretamente da interface gráfica. Isso permite realizar testes unitários na lógica de apresentação sem a necessidade de interagir com a View.
A ViewModel pode ser reutilizada em diferentes Views. Isso é útil em cenários onde você deseja exibir os mesmos dados de maneiras diferentes em diferentes partes da aplicação.
O MVVM organiza a aplicação de uma forma que facilita a manutenção e o crescimento do projeto. A separação de responsabilidades torna o código mais modular e escalável.
A capacidade de sincronização automática entre a View e a ViewModel reduz a quantidade de código boilerplate, tornando a implementação mais limpa e eficiente.

O padrão MVVM é amplamente utilizado em aplicações que envolvem interfaces ricas e interativas, como aplicações desktop com WPF (Windows Presentation Foundation), aplicativos móveis com Xamarin, e aplicações web com frameworks como Angular. Ele é ideal em cenários que demandam data binding bidirecional, permitindo sincronização automática entre a interface do usuário e a lógica de apresentação, o que facilita a manutenção, testabilidade e escalabilidade do código.

MVC x MVVW

No MVC, o Controller é responsável por mediar as interações entre a View e o Modelo. No MVVM, a ViewModel serve como intermediária, mas utiliza o data binding para facilitar a comunicação automática entre a View e a ViewModel. No MVC, a View pode ter uma comunicação mais direta com o Modelo e o Controller. Já no MVVM, a View não deve interagir diretamente com o Modelo, mas sim com a ViewModel.

O MVVM é amplamente baseado em data binding bidirecional, enquanto o MVC geralmente requer a atualização manual de dados entre a View e o Modelo.

Arquitetura Distribuída

INCIDÊNCIA EM PROVA: BAIXA

Os sistemas de computação estão passando por uma evolução. Desde 1945, quando começou a era moderna dos computadores, até aproximadamente 1985, os computadores eram grandes e caros. **Contudo, mais ou menos a partir de meados da década de 80, dois avanços tecnológicos começaram a mudar essa situação**. O primeiro foi o desenvolvimento de microprocessadores de grande capacidade.

O segundo desenvolvimento foi a invenção de redes de computadores de alta velocidade. Nesse cenário, surgem os sistemas distribuídos, os quais são plataformas formadas por um conjunto de computadores independentes que se apresenta a seus usuários como um sistema único e coerente. Uma breve introdução histórica e agora podemos partir para o assunto: a arquitetura mainframe (Grande Porte) é conhecida por ser altamente centralizada.

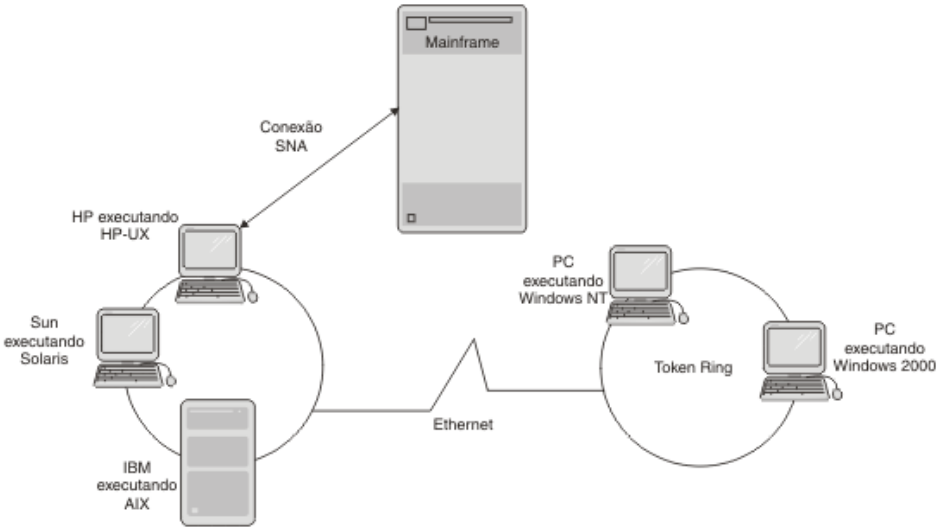
Todo processamento ocorre no mainframe e há um bocado de terminal burro que o acessa. **Já a Arquitetura Distribuída inverte essa concepção, na medida em que o processamento é dispersado através da organização e seus desktops e servidores**. *Professor, quais são as vantagens dessa abordagem?* Cara, há ganhos de responsividade, escalabilidade, redundância, disponibilidade, compartilhamento de recursos, controle e envolvimento do usuário, etc.

Componentes podem ser hospedados em diferentes plataformas e tecnologias, além de se comunicarem por meio de uma rede – sendo que cada nó tem sua responsabilidade específica no processamento de uma tarefa comum. *Galera, o usuário nota que o processamento é distribuído?* Não, é tudo transparente! **Para ele, é como se tudo estivesse sendo executado como um único sistema**.

Os computadores em um sistema distribuído podem estar fisicamente próximos e conectados por uma rede local ou podem estar geograficamente distantes e conectados por uma rede remota. Uma arquitetura distribuída pode consistir em uma série de configurações possíveis, como mainframes, computadores pessoais, estações de trabalho, minicomputadores e assim por diante. **A meta da computação distribuída é fazer um trabalho de rede como um computador único.**

Os sistemas de computação distribuída podem ser executados no hardware fornecido por muitos fornecedores e podem utilizar diversos componentes de software baseados em padrões. **Esses sistemas são independentes do software subjacente**. Eles podem ser executados em vários sistemas operacionais e podem utilizar vários protocolos de comunicação. Algum hardware pode utilizar Unix como o sistema operacional, enquanto outro hardware pode utilizar Windows.

Para comunicação entre máquinas, esse hardware pode utilizar SNA ou TCP/IP em Ethernet ou Token Ring. A imagem seguinte apresenta uma possível arquitetura distribuída descentralizada:



Esse sistema contém duas LANs conectadas entre si. Uma consiste em estações de trabalho UNIX (HP-UX, Solaris, AIX) de vários fabricantes (HP, Sun, IBM); já a outra consiste em PCs executando vários sistemas operacionais (NT e 2000) em Token Ring. Há uma LAN conectada a um mainframe por meio de uma conexão SNA. **Cliente/Servidor e P2P são exemplos famosos de arquitetura distribuída**.

Galera, não confundam arquitetura distribuída com arquitetura paralela. No primeiro caso, processos são executados concorrentemente em máquinas diferentes dentro de uma rede; é uma arquitetura fracamente acoplada; é mais imprevisível devido ao uso de redes de computadores e suas falhas; apresenta controle e acesso descentralizado dos recursos. No segundo caso, processos são executados concorrentemente em uma mesma máquina;

Trata-se de uma arquitetura fortemente acoplada, que pode compartilhar hardware ou se comunicar através de um barramento de alta velocidade; é mais previsível, porque falhas são menos comuns em barramentos de alta velocidade; apresenta controle e acesso centralizado dos recursos; utiliza melhor o poder de processamento; apresenta melhor desempenho e confiabilidade; permite compartilhar dados e recursos; reutiliza serviços já disponíveis; atende mais usuários; etc.

Claro que nem tudo são flores – há também desvantagens: desenvolver, gerenciar e manter sistemas distribuídos; controlar o acesso concorrente a dados e a recursos compartilhados; evitar que falhas de máquinas ou da rede comprometam o funcionamento do sistema; garantir a segurança do sistema e o sigilo dos dados trocados entre máquinas; lidar com a heterogeneidade do ambiente; entre outros.

Um último conceito sobre esse assunto: Middleware. **Um Middleware é uma camada de software que permite que elementos de aplicações interoperem através de redes de computadores** – imaginem um software que padroniza interfaces de programação para que você não se preocupe se existem protocolos, arquiteturas, sistemas operacionais ou bancos de dados diferentes. Ele provê um modo para obter dados de um lugar para outro.

Além disso, é capaz de mascarar diferenças existentes entre sistemas operacionais, plataformas de hardware e protocolos de rede. O Middleware oculta do desenvolvedor da aplicação a complexidade do processo de transporte da rede. Os exemplos mais comuns são: RPC, RMI, CORBA, DCOM, etc. Quando utilizamos linguagens orientadas a objetos em arquiteturas distribuídas, chegamos ao conceito de objetos distribuídos e invocação remota.

Como o nome bem diz, objetos distribuídos são instâncias que podem ser acessadas remotamente e trazem para as aplicações distribuídas todas as vantagens da orientação a objetos: maior reusabilidade, segurança, padronização. Nas linguagens estruturadas, as funções remotas são utilizadas a partir de chamadas remotas, as famosas Remote Procedure Call (RPC). Os objetos não possuem funções, mas sim métodos, e não são "chamados", mas invocados.

Alguém consegue descobrir o nome similar ao RPC para objetos distribuídos? **Trata-se do Remote Method Invocation (RMI). O RMI permite que um objeto executado em uma Java Virtual Machine (JVM) possa invocar métodos de outro objeto remoto, ou seja, executado em outra JVM.** RMI permite comunicação remota entre programas escritos na linguagem Java. *Professor, quer dizer que, de uma máquina eu posso invocar um método de um objeto de outra máquina?*

E se eu não tiver acesso ao código remoto? Para isso, utilizamos um recurso muito importante da orientação a objetos: interfaces. **Desse modo, basta que o objeto que invoca o objeto distribuído tenha a interface com as assinaturas dos métodos remotos.** Além da interface, existe um objeto local que implementa a interface remota, mas na verdade não faz nada além de repassar tudo ao objeto remoto (que também implementa a interface, obviamente).

O nome desse objeto local que implementa a interface remota é o proxy *(qualquer semelhança com o padrão de projeto não é mera coincidência). Ah, professor, o senhor QUASE me enganou! Como pode um objeto num local saber onde está o objeto remoto, e como saber qual das classes que implementam a interface remota estão disponíveis para uso?* Esse aluno vai passar! Meu querido, você tem toda razão...

Para saber qual objeto implementa o serviço da interface remota a ser invocado e onde ele se encontra, existe um serviço chamado Binder. **Ele é responsável por transformar o nome de um serviço remoto que o objeto local quer invocar numa referência real de um objeto distribuído.** Os servidores com os objetos distribuídos devem registrar os serviços no Binder para que os clientes possam acessá-los.

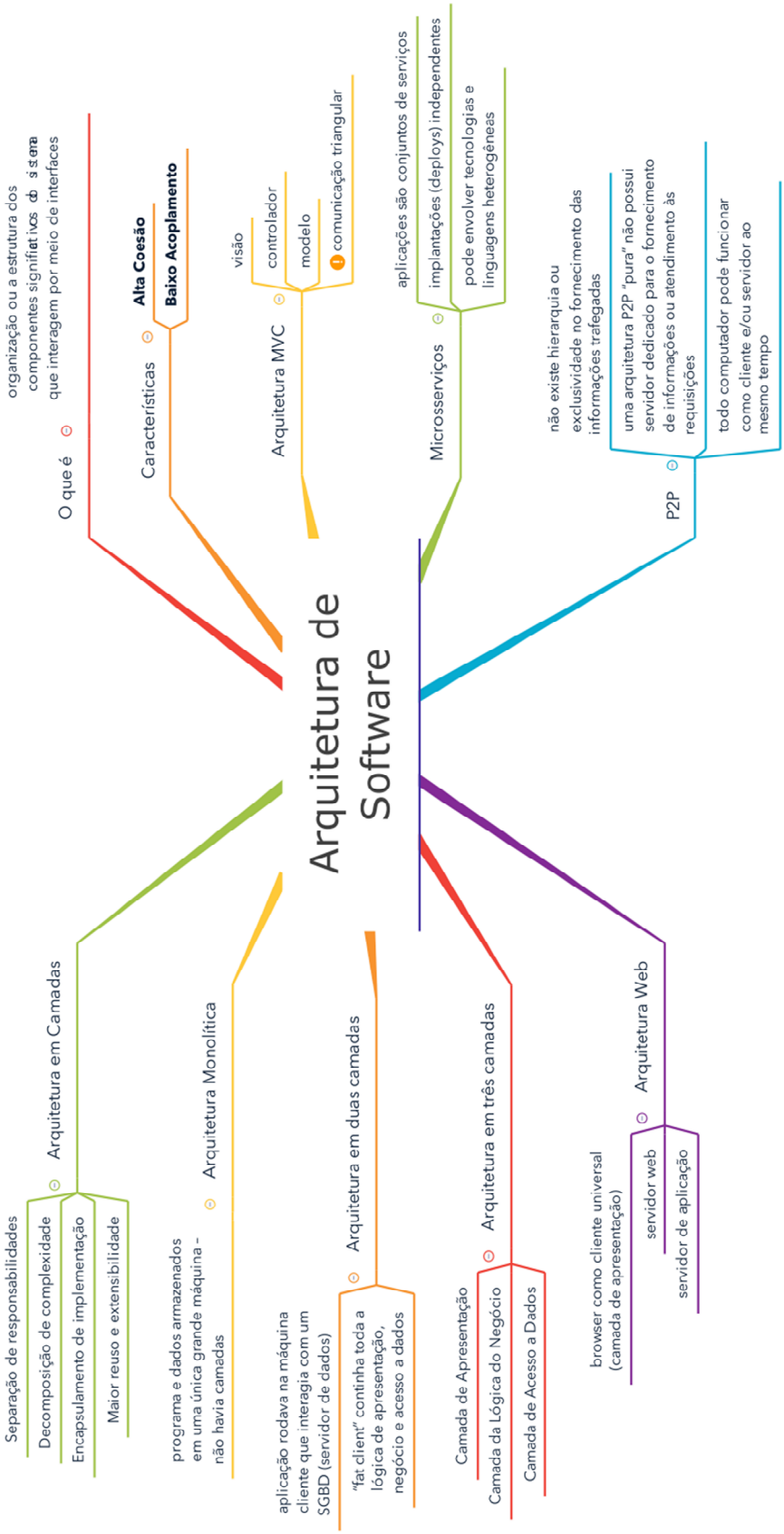
Basta que, antes, haja um contrato para que os nomes dos serviços sejam bem conhecidos por todos: clientes, servidores com objetos remotos e Binder. **Os objetos remotos do lado do servidor são dois para cada objeto distribuído: skeleton e dispatcher.** O skeleton implementa a interface remota, mas ainda não é quem faz o trabalho braçal (que pode ser uma classe "normal" que implementa a interface remota). Já o dispatcher faz parte do arcabouço para receber as invocações remotas e chamar o skeleton certo.

Proxy (lado cliente), skeleton (lado servidor) e dispatcher (lado servidor) compõe o middleware de um sistema de objetos distribuídos. Em algumas implementações, o skeleton e dispatcher são unidos numa classe. Finalmente, o objeto distribuído, que vai realizar cálculos, acessar bancos de dados e tudo o que o cliente precisar, é acessado pelo Skeleton, que recebe os resultados e faz todo o caminho de volta.

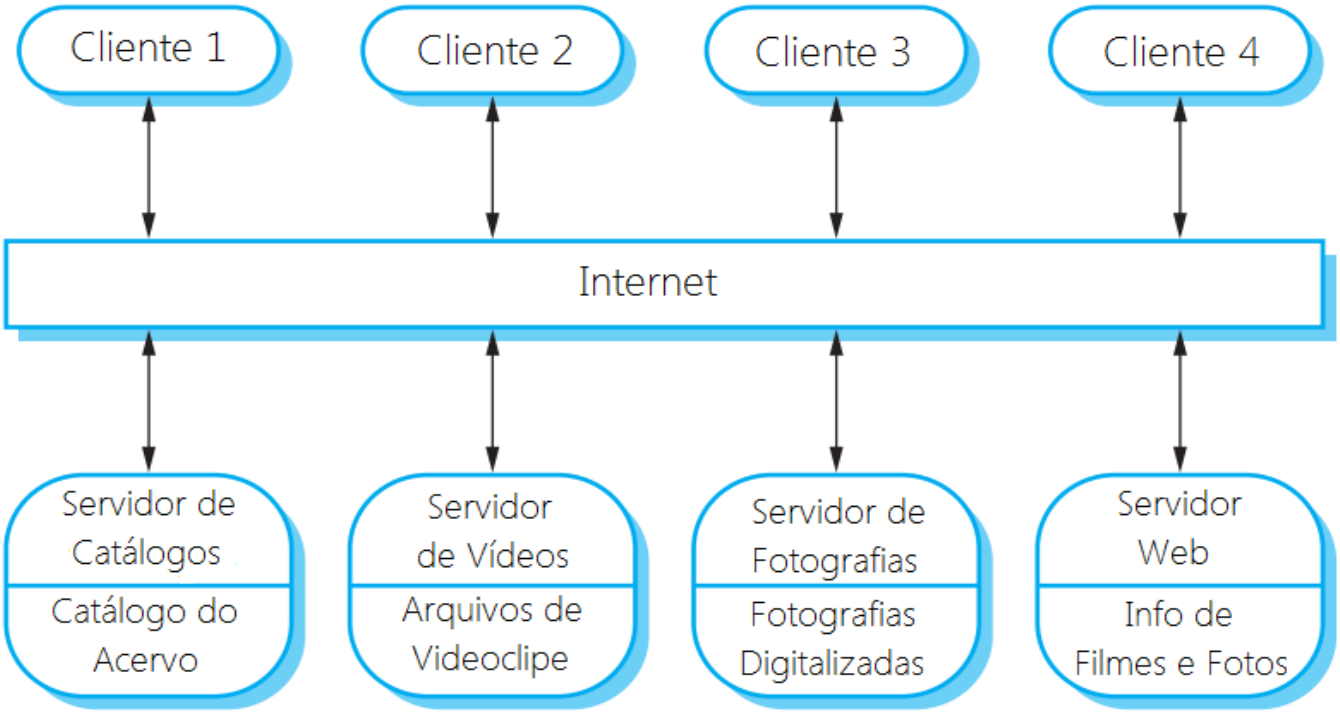
RESUMO

ARQUITETURA DE SOFTWARE	A arquitetura de software de um programa ou sistema computacional é a estrutura ou estruturas do sistema, que abrange os componentes de software, as propriedades externamente visíveis desses componentes e as relações entre eles

TIPO DE ACOPLAMENTO	DESCRIÇÃO
ACOPLAMENTO POR CONTEÚDO	Ocorre quando um módulo faz uso de estruturas de dados ou de controle mantidas no escopo de outro módulo.
ACOPLAMENTO COMUM	Ocorre quando um conjunto de módulos acessa uma área global de dados.
ACOPLAMENTO POR CONTROLE	Ocorre quando módulos passam decisões de controle a outros módulos.
ACOPLAMENTO POR DADOS	Ocorre quando apenas uma lista de dados simples é passada como parâmetro de um módulo para o outro, com uma correspondência um-para-um de itens.
ACOPLAMENTO POR CHAMADAS DE ROTINAS	Ocorre quando uma operação chama outra. Nesse nível de acoplamento, é comum e, quase sempre, necessário. Entretanto, realmente aumenta a conectividade de um sistema.
ACOPLAMENTO POR USO DE TIPOS	Ocorre quando o Componente A usa um tipo de dados definido em um Componente B. Se a definição de tipo mudar, todo componente que usa a definição também terá de ser alterado.
ACOPLAMENTO POR INCLUSÃO OU IMPORTAÇÃO	Ocorre quando um Componente A importa ou inclui um pacote ou o conteúdo do Componente B.
ACOPLAMENTO EXTERNO	Ocorre quando um componente se comunica ou colabora com componentes de infraestrutura. Embora seja necessário, deve-se limitar a um pequeno número de componentes em um sistema.


Arquitetura de Software
O que é
organização ou a estrutura dos componentes significativos do sistema que interagem por meio de interfaces
Características
Alta Coesão
Baixo Acoplamento
Arquitetura MVC
visão
controlador
modelo
comunicação triangular
Microsserviços
aplicações são conjuntos de serviços
implantações (deploys) independentes
pode envolver tecnologias e linguagens heterogêneas
P2P
não existe hierarquia ou exclusividade no fornecimento das informações trafegadas
uma arquitetura P2P "pura" não possui servidor dedicado para o fornecimento de informações ou atendimento às requisições
todo computador pode funcionar como cliente e/ou servidor ao mesmo tempo
Arquitetura Web
browser como cliente universal (camada de apresentação)
servidor web
servidor de aplicação
Arquitetura em três camadas
Camada de Apresentação
Camada da Lógica do Negócio
Camada de Acesso a Dados
Arquitetura em duas camadas
"fat client" continha toda a lógica de apresentação, negócio e acesso a dados
aplicação rodava na máquina cliente que interagia com um SGBD (servidor de dados)
Arquitetura Monolítica
programa e dados armazenados em uma única grande máquina - não havia camadas
Arquitetura em Camadas
Separação de responsabilidades
Decomposição de complexidade
Encapsulamento de implementação
Maior reuso e extensibilidade

SERVIDORES	Oferecem serviços para outros subsistemas (Ex: Servidores de Impressão, Servidores de Arquivos, Servidor de Compilação, entre outros).
CLIENTES	Solicitam os serviços oferecidos pelos servidores. Geralmente são independentes, podendo ser executados simultaneamente.
REDE	Permite aos clientes acessarem esses serviços – quando clientes e servidores podem ser executados em uma única máquina, não são necessários.



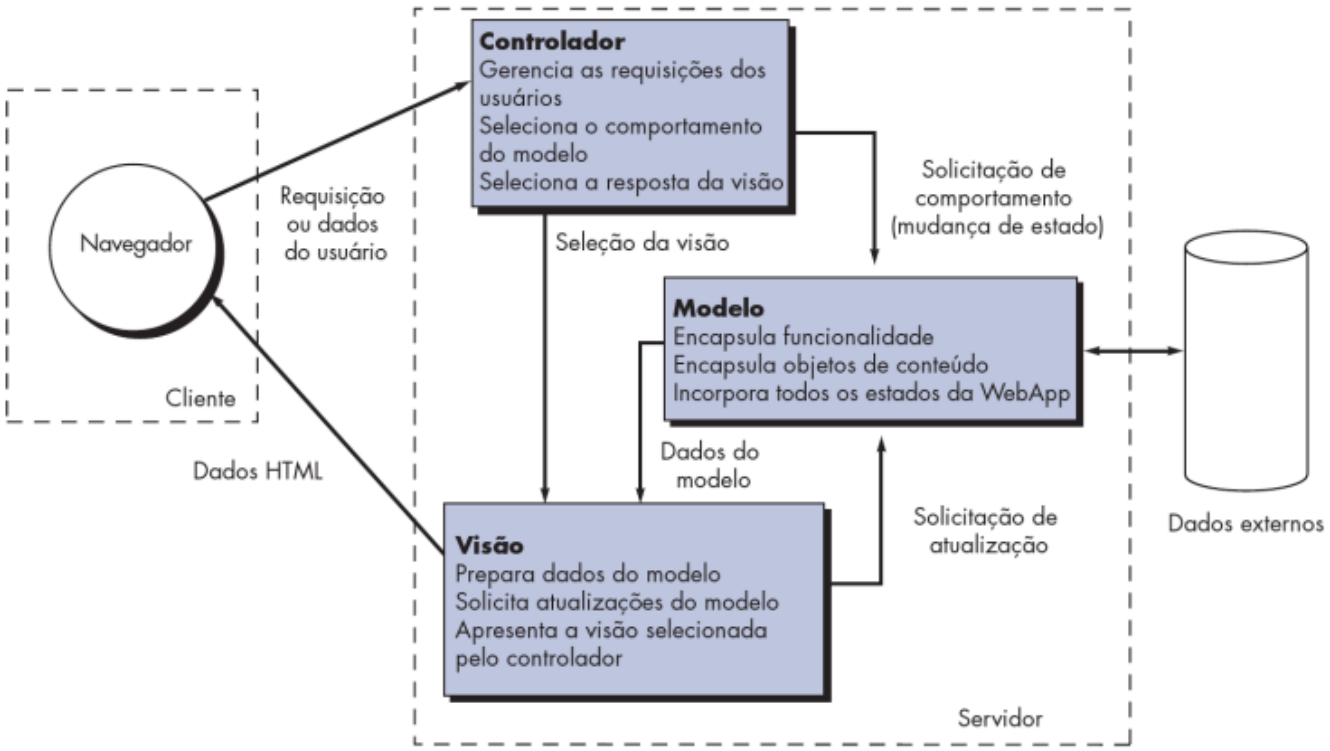
VANTAGENS DE CLIENTES MAGROS	Baixo custo de administração; facilidade de proteção; baixo custo de hardware; menor custo para licenciamento de softwares; baixo consumo de energia; resistência a ambientes hosts; menor dissipação de calor para o ambiente; mais silencioso que um PC convencional; mais ágil para rodar planilhas complexas; entre outros.
DESVANTAGENS DE CLIENTES MAGROS	Se o servidor der problema e não houver redundância, todos os clientes-magros ficarão inoperantes; necessita de maior largura de banda na rede em que é empregado; em geral, possui um pior tempo de resposta, uma vez que usam o servidor para qualquer transação; apresenta um apoio transacional menos robusto; etc.

VANTAGENS DE CLIENTES GORDOS	Necessitam de requisitos mínimos para servidores; apresenta uma performance multimídia melhor; possui maior flexibilidade; algumas situações se beneficiam bastante, tais como jogos eletrônicos, em que se beneficiam por conta de sua alta capacidade de processamento e de hardware específico.
DESVANTAGENS DE CLIENTES GORDOS	Não há um local central para atualizar e manter a lógica do negócio, uma vez que o código do aplicativo é executado em vários locais de cliente; é exigida uma grande confiança entre o servidor e os clientes por conta do banco de dados; não suporta bem o crescimento do número de clientes.

CAMADA DE APRESENTAÇÃO	Também chamada de Camada de Interface, possui classes que contêm funcionalidades para visualização dos dados pelos usuários. Ela tem o objetivo de exibir informações ao usuário e

	traduzir ações do usuário em requisições às demais partes dos sistemas. O amplo uso da internet tornou as interfaces com base na web crescentemente populares.
CAMADA DE NEGÓCIO	Também chamada Camada Lógica ou de Aplicação, possui classes que implementam as regras de negócio no qual o sistema será implantado. Ela realiza cálculos com base nos dados armazenados ou nos dados de entrada, decidindo que parte da camada de acesso de ser ativada com base em requisições provenientes da camada de apresentação.
CAMADA DE DADOS	Possui classes que se comunicam com outros sistemas para realizar tarefas ou adquirir informações para o sistema. Tipicamente, essa camada é implementada utilizando algum mecanismo de armazenamento persistente. Pode haver uma subcamada dentro desta camada chamada Camada de Persistência ou Camada de Acesso.

MODEL	Essa classe também é responsável por gerenciar e controlar a forma como os dados se comportam por meio das funções, lógica e regras de negócios estabelecidas.
VIEW	Essa camada é responsável por apresentar as informações de forma visual ao usuário. Em seu desenvolvimento devem ser aplicados apenas recursos ligados a aparência como mensagens, botões ou telas.
CONTROLLER	Essa camada é responsável por intermediar as requisições enviadas pelo View com as respostas fornecidas pelo Model, processando os dados que o usuário informou e repassando para outras camadas.



QUESTÕES COMENTADAS – CESPE

1. **(CESPE / CAU-BR – 2024)** Ao se migrar para uma arquitetura cliente/servidor multinível, a mesma aplicação pode assumir simultaneamente as funções de cliente e de servidor.

Comentários:

Em uma arquitetura cliente/servidor multinível (ou multicamada), uma aplicação pode assumir simultaneamente as funções de cliente e de servidor. Por exemplo, em uma arquitetura de três camadas, uma aplicação intermediária (middleware) pode atuar como cliente ao solicitar dados de um banco de dados (servidor de dados) e como servidor ao fornecer esses dados para uma aplicação cliente (interface de usuário). Isso permite maior modularidade e escalabilidade no design do sistema.

Gabarito: Correto

---

2. **(CESPE / FINEP – 2024)** O principal objetivo de se reduzir o acoplamento entre módulos em um sistema de software é:

a) minimizar a necessidade de documentação.
b) melhorar a eficiência de comunicação entre módulos.
c) diminuir o tamanho dos módulos.
d) aumentar a coesão entre módulos.
e) facilitar a integração de novos módulos.

Comentários:

(a) Errado. Minimizar a necessidade de documentação não é o principal objetivo da redução do acoplamento entre módulos. Embora um sistema menos acoplado possa ser mais intuitivo e, em certos aspectos, exigir menos documentação explicativa, essa não é a razão principal para sua implementação;

(b) Errado. Melhorar a eficiência de comunicação entre módulos não é o principal objetivo da redução do acoplamento. Na verdade, a redução do acoplamento geralmente significa reduzir a quantidade ou complexidade das interações entre módulos, o que pode resultar em uma comunicação menos direta mas mais modular e gerenciável;

(c) Errado. Diminuir o tamanho dos módulos não é diretamente relacionado com a redução do acoplamento. O acoplamento refere-se mais à dependência funcional entre módulos do que ao seu tamanho;

(d) Errado. Aumentar a coesão entre módulos é um objetivo desejável em design de software, mas não é o principal objetivo da redução do acoplamento. Coesão refere-se à extensão em que as responsabilidades dentro de um único módulo estão relacionadas e focadas;

(e) Correto. Facilitar a integração de novos módulos é um dos principais objetivos de reduzir o acoplamento entre módulos em um sistema de software. Quando módulos são pouco acoplados, eles dependem menos uns dos outros, o que torna mais fácil modificar, substituir ou adicionar novos módulos sem afetar significativamente o restante do sistema.

Gabarito: Letra E

---

3. **(CESPE / BNB – 2022)** No padrão MVC, o componente de modelo gerencia as requisições dos usuários.

Comentários:

O Modelo é responsável por fornecer os dados para a view e para o controlador. Ele não gerencia as requisições dos usuários. O controlador é responsável por gerenciar as requisições dos usuários e, com base nas informações do modelo, ele processa as requisições e decide a ação a ser tomada.

Gabarito: Errado

---

4. **(CESPE / BNB – 2022)** Na arquitetura em camadas, os componentes da camada mais interna opera o sistema operacional, ao passo que os da camada mais externa interagem com o usuário.

Comentários:

A arquitetura em camadas é uma abordagem de projeto de sistemas que divide o sistema em diferentes camadas, cada uma responsável por uma função específica. A camada mais interna opera diretamente o hardware e gerencia os recursos de sistema, como processamento, memória, armazenamento e rede. Esta camada trabalha diretamente com o sistema operacional, fornecendo o suporte necessário para o funcionamento do sistema. A camada mais externa contém os componentes responsáveis por interagir com o usuário e é responsável por criar a interface de usuário e executar a lógica de aplicativos e processos.

Gabarito: Correto

---

5. **(CESPE / Petrobrás - 2022)** Enquanto a arquitetura é responsável pela infraestrutura de alto nível do software, o design é responsável pelo software a nível de código, como, por exemplo, o que cada módulo está fazendo, o escopo das classes e os objetivos das funções.

Comentários:

A arquitetura de um software descreve a estrutura de nível mais alto de uma aplicação e identifica seus principais componentes. No projeto detalhado (design) detalha-se o projeto do software para cada componente identificado na arquitetura.

Gabarito: Correto

---

6. **(CESPE / TJ-RJ - 2021)** Na arquitetura MVC (Model-View-Controller), as funcionalidades de cada segmento são mais bem descritas como:

a) o modelo encapsula as funcionalidades; o view gerencia as requisições dos usuários; o controlador prepara dados do modelo.

b) o modelo encapsula objetos de conteúdo; a visão solicita atualizações do modelo; o controlador seleciona o comportamento do modelo.

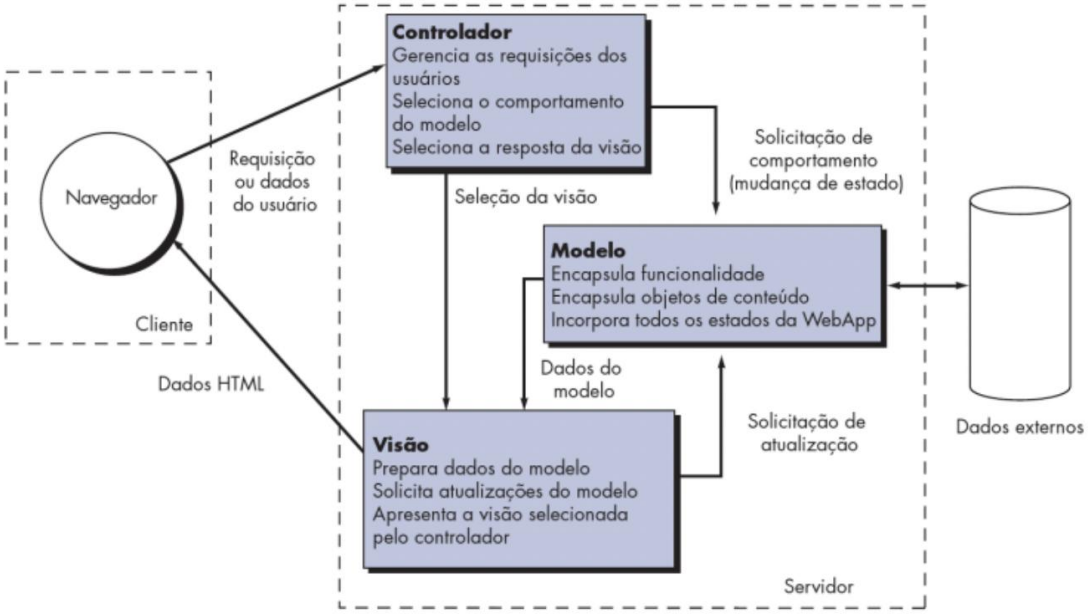
c) o modelo incorpora todos os estados; o view gerencia as requisições dos usuários; o controlador encapsula objetos de conteúdo.

d) o modelo encapsula objetos de conteúdo; o view seleciona o comportamento do modelo; o controlador solicita atualizações do modelo.

e) o modelo seleciona a resposta da visão; a visão apresenta a visão selecionada pelo controlador; o controlador encapsula objetos de conteúdo.

Comentários:

O MVC promove a estrita separação de responsabilidades entre componentes de uma interface gráfica. O Modelo é responsável pela representação dos dados, provendo meios de acesso (escrita/leitura), além disso ela encapsula as principais funcionalidades de dados do sistema. A camada de Visão é responsável pela interação com o usuário, além disso, é possível que a camada de Visão solicite diretamente o estado (atualizações) do Modelo. Por fim, a camada de Controle é responsável por receber todas as requisições do usuário, além disso, um controlador define o comportamento da aplicação e interpreta as ações do usuário.



Gabarito: Letra B

---

7. **(CESPE / TJ-RJ - 2021)** Em um ambiente cliente/servidor, a arquitetura que permite a mesma aplicação assumir tanto o papel de cliente quanto o de servidor é conhecida como arquitetura C/S:

a) simples.
b) de dois níveis.
c) multinível.
d) de três camadas.
e) par-par.

Comentários:

A arquitetura par-a-par (ou peer-to-peer) trata-se do modelo não hierárquico de rede mais simples em que máquinas se comunicam diretamente, sem passar por nenhum servidor dedicado, podendo compartilhar dados e recursos umas com as outras. Nesse tipo de rede, todas as máquinas oferecem e consomem recursos umas das outras, atuando ora como clientes, ora como servidoras.

O gabarito definitivo foi arquitetura multinível – eu não consigo concordar com esse gabarito. Caso alguém tenha uma opinião divergente, manda no fórum :)

Gabarito: Letra C

---

8. **(CESPE / TELEBRÁS - 2021)** Na arquitetura de software, a arquitetura cliente/servidor tem como vantagem uma maior facilidade de manutenção e segurança dos dados, e como desvantagens possíveis bloqueios no tráfego da rede, além de problemas de atualização da interface de aplicação.

Comentários:

Uma arquitetura cliente/servidor é formada por servidores, que oferecem serviços para outros subsistemas; pelos clientes, que solicitam os serviços oferecidos pelos servidores. Uma das vantagens desse modelo é a manutenção e a segurança dos dados, visto que os servidores podem ter um melhor controle sobre quem acessam os recursos. Já uma das desvantagens desse modelo é que os servidores podem ficar sobrecarregados com o excesso de solicitações pelos clientes, ocasionando bloqueios no tráfego da rede. Outro ponto é sobre a atualização da interface de aplicação, isso pode também paralisar a rede.

Gabarito: Correto

---

9. **(CESPE / TELEBRÁS - 2021)** Por se tratar de uma arquitetura distribuída, o modelo cliente-servidor pressupõe facilidades para atualizar os servidores de forma transparente, sem que isso afete outras partes do sistema.

Comentários:

Perfeito! Como os servidores são centralizados, e em quantidade menor que os clientes, há mais facilidade em se administrar as atualizações deles. Em suma, em um modelo cliente-servidor, uma de suas vantagens é a facilidade de manutenção dos servidores.

Gabarito: Correto

---

10. **(CESPE / TRE-BA – 2017)** Com referência às arquiteturas multicamadas de aplicações para o ambiente web, assinale a opção correta.

a) Se, na camada de dados, for realizada uma alteração no banco de dados, o restante das camadas também será afetado.

b) O modelo de três camadas recebe essa denominação caso um sistema cliente-servidor seja desenvolvido mantendo-se a camada de negócio do lado do cliente e as camadas de apresentação e dados no lado do servidor.

c) Cada camada é normalmente mantida em um servidor específico para tornar-se o mais escalonável e independente possível em relação a outras camadas, estando entre as suas principais características o eficiente armazenamento e a reutilização de recursos.

d) O objetivo das arquiteturas multicamadas consiste na junção de responsabilidades entre os componentes das aplicações web, de modo a atender aos requisitos funcionais e não funcionais esperados pela aplicação.

e) Na arquitetura de duas camadas — apresentação e armazenamento —, o computador que contiver a base de dados terá de ficar junto com os computadores que executarem as aplicações.

Comentários:

(a) Errado, uma alteração no banco de dados alteraria apenas as classes da camada de dados, mas o restante da arquitetura não seria afetado por essa alteração; (b) Errado, ela recebe essa denominação quando um sistema cliente-servidor é desenvolvido retirando-se a camada de negócio do lado do cliente; (c) Correto. Cada camada desta arquitetura é normalmente mantida em um servidor específico para tornar-se mais escalonável e independente das demais. Cada camada é auto-contida o suficiente de forma que a aplicação pode ser dividida em vários computadores em uma rede distribuída; (d) Errado, o objetivo consiste na separação de responsabilidades entre os componentes das aplicações web, de modo que tenham alta coesão; (e) Errado. Nesta estrutura, a base de dados é colocada em uma máquina específica, separada das máquinas que executavam as aplicações.

Gabarito: Letra C

---

11. (CESPE / STJ – 2015) Na arquitetura em camadas MVC (modelo-visão-controlador), o modelo encapsula o estado de aplicação, a visão solicita atualização do modelo e o controlador gerencia a lógica de negócios.

Comentários:

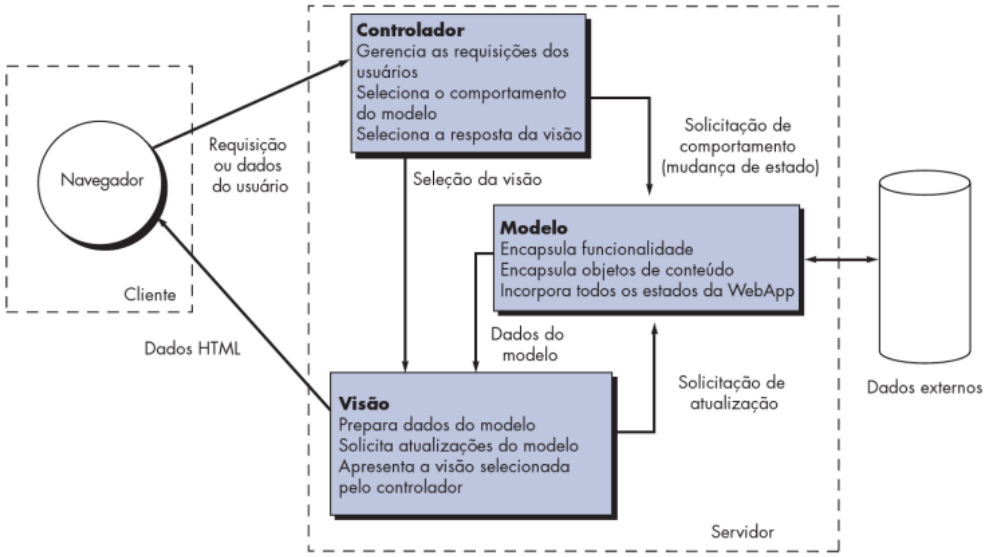
A questão está errada! *O modelo encapsula o estado da aplicação?* Sim, isso é verdade. *A visão solicita atualização do modelo?* Sim, vimos que ela faz isso por meio de eventos que notificam o controlador. *O Controlador gerencia a lógica de negócios?* Não, quem gerencia a lógica de negócios é o modelo.

Gabarito: Errado

---

12. (CESPE / MEC – 2015) O controlador gerencia as requisições dos usuários encapsulando as funcionalidades e prepara dados do modelo.

Comentários:



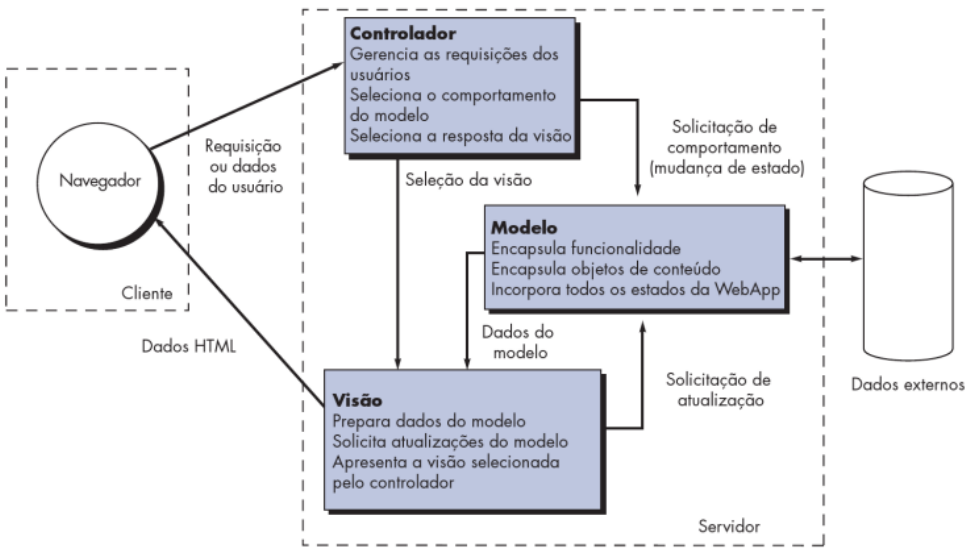
O controlador gerencia as requisições dos usuários, o modelo encapsula funcionalidades e a visão prepara dados do modelo.

Gabarito: Errado

---

13. (CESPE / MEC – 2015) A visão encapsula objetos de conteúdo, solicita atualizações do modelo e seleciona o comportamento do modelo.

Comentários:



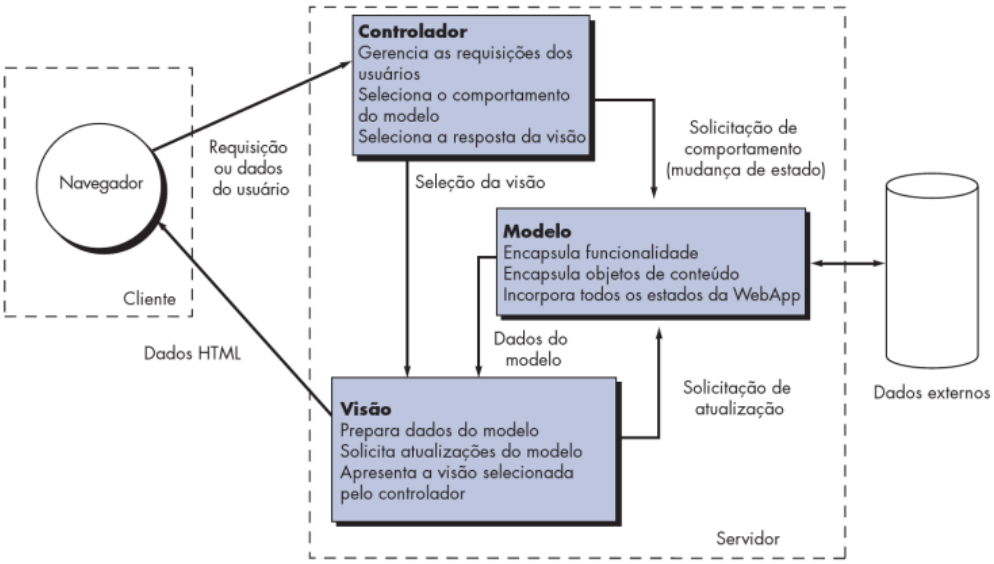
O modelo encapsula objetos de conteúdo, a visão solicita atualizações do modelo e o controle seleciona o comportamento do modelo.

Gabarito: Errado

---

14. (CESPE / STJ – 2015) No padrão em camadas modelo-visão-controle (MVC), o controle é responsável por mudanças de estado da visão.

Comentários:



O controle é responsável por solicitar mudanças de estado, mas quem realiza a mudança é o modelo.

Gabarito: Errado

---

15. (CESPE / ANTAQ – 2014) O modelo MVC é um padrão de arquitetura que consiste na definição de camadas para a construção de softwares.

Comentários:

A questão foi extremamente rigorosa e considerou que o Modelo MVC não possui camadas. Sendo bastante rigoroso, em uma definição acadêmica, ele trata de separação de responsabilidades e, não, de camadas exatamente. Essa é a única questão que eu já vi cobrando esse nível de detalhamento.

Gabarito: Errada

---

16. (CESPE / ANTAQ – 2014) O controller tem a responsabilidade de armazenar e buscar os dados que deverão ser exibidos pelo view.

Comentários:

Na verdade, essa é uma responsabilidade do modelo.

Gabarito: Errado

---

17. (CESPE / INPI – 2013) De acordo com os princípios da engenharia de software relacionados à independência funcional, os algoritmos devem ser construídos por módulos visando unicamente ao alto acoplamento e à baixa coesão, caso a interface entre os módulos dê-se pela passagem de dados.

Comentários:

Não, está invertido! Visa-se ao baixo acoplamento e à alta coesão!

Gabarito: Errado

---

18. (CESPE / STF – 2013) Quanto maior for o número de camadas, menor será o desempenho do software como um todo.

Comentários:

Esse é um assunto polêmico! Em geral, é isso que acontece, isto é, quanto mais camadas, mais overhead. No entanto, há casos em que isso não é válido! Portanto, essa questão caberia recurso.

Gabarito: Correto

---

19. (CESPE / STF – 2013) Cada camada tem comunicação (interface) com todas as demais camadas, tanto inferiores quanto superiores.

Comentários:

Na verdade, as camadas só possuem comunicação/interface com suas camadas adjacentes.

Gabarito: Errado

---

20. (CESPE / STF – 2013) Em uma arquitetura em camadas, a camada de persistência é responsável por armazenar dados gerados pelas camadas superiores e pode utilizar um sistema gerenciador de banco de dados para evitar, entre outros aspectos, anomalias de acesso concorrente dos dados e problemas de integridade de dados.

Comentários:

Questão ótima! Persistência guarda os dados e, sim, pode utilizar um SGBD. *Por que?* Porque ele é capaz de tratar concorrência de dados e problemas de integridade! Tudo certinho...

Gabarito: Correto

---

21. (CESPE / FUB – 2013) Aplicações cliente-servidor multicamadas são usualmente organizadas em três camadas principais: apresentação, lógica e periférico.

Comentários:

Na verdade, é apresentação, lógica e dados.

Gabarito: Errado

---

22. (CESPE / FUB – 2013) Entre as desvantagens de se executar todas as camadas de uma aplicação cliente-servidor no lado do servidor se destaca a dificuldade de atualização e correção da aplicação.

Comentários:

Primeiro, se todas as camadas de uma aplicação cliente-servidor são executados no servidor, então não é uma arquitetura cliente-servidor. Executar a maioria das camadas no lado do cliente é uma dificuldade de atualização e correção de aplicação, na medida em que é necessária fazer essa manutenção em todas as máquinas clientes.

Gabarito: Errado

---

23. (CESPE / BACEN – 2013) MVC (Model-View-Controller) é um modelo de arquitetura de software que separa, de um lado, a representação da informação e, de outro, a interação do usuário com a informação.

Comentários:

A camada de visão trata da representação da informação e a camada de controle trata da interação do usuário com a informação (modelo).

Gabarito: Correto

---

24. (CESPE / TCE-ES – 2013) No Padrão MVC, as regras do negócio que definem a forma de acesso e modificação dos dados são geridas pelo controlador.

Comentários:

Na verdade, essa é uma função do modelo.

Gabarito: Errado

---

25. **(CESPE / Banco da Amazônia – 2012)** De acordo com o princípio da coesão de classes, cada classe deve representar uma única entidade bem definida no domínio do problema. O grau de coesão diminui com o aumento contínuo de código de manutenção nas classes.

Comentários:

Uma classe deve ter uma única responsabilidade e executá-la de maneira satisfatória. Além disso, o grau de coesão tende a diminuir com o aumento contínuo de código de manutenção nas classes, mas isso não é regra.

Gabarito: Correto

---

26. **(CESPE / Banco da Amazônia – 2012)** O acoplamento de métodos expressa o fato de que qualquer método deve ser responsável somente por uma tarefa bem definida.

Comentários:

Acoplamento? Não! Na verdade, é a coesão!

Gabarito: Errado

---

27. **(CESPE / CET – 2011)** No padrão de desenvolvimento modelo-visualização-controlador (MVC), o controlador é o elemento responsável pela interpretação dos dados de entrada e pela manipulação do modelo, de acordo com esses dados.

Comentários:

Ele é responsável pelo fluxo da aplicação e, por isso, é o responsável pela interpretação dos dados de entrada e pela manipulação do modelo.

Gabarito: Correto

---

28. **(CESPE / MEC – 2011)** O modelo MVC pode ser usado para construir a arquitetura do software a partir de três elementos: modelo, visão e controle, sendo definidas no controle as regras de negócio que controlam o comportamento do software a partir de restrições do mundo real.

Comentários:

Regras de negócio ficam no modelo; o controlador só orquestra as solicitações vindas da visão para o modelo.

Gabarito: Errado

---

29. **(CESPE / MEC – 2011)** O controlador, no modelo MVC, realiza a comunicação entre as camadas de visão e modelo.

Comentários:

Perfeito! O controlador realmente realiza a comunicação entre as camadas de visão e modelo.

Gabarito: Correto

---

30. **(CESPE / MEC – 2011)** No MVC, o modelo representa o estado geral do sistema.

Comentários:

É realmente no modelo que se encontram os dados (estado) do sistema.

Gabarito: Correto

---

31. **(CESPE / MEC – 2011)** Apesar do seu amplo emprego em aplicações web, o MVC deve ser utilizado apenas em interfaces gráficas em função de sua arquitetura de componentes.

Comentários:

Quando ela começou nem existiam aplicações web. Ele é usado primariamente em interfaces gráficas, mas não se limita a isso!

Gabarito: Errado

---

32. **(CESPE / MEC – 2011)** No MVC, é o modelo que permite apresentar, de diversas formas diferentes, os dados para o usuário.

Comentários:

Opa... essa é uma função da visão!

Gabarito: Errado

---

33. **(CESPE / MEC – 2011)** O controlador é o responsável pelas regras de negócio e pelos dados de uma aplicação no MVC.

Comentários:

Na verdade, essa é uma função do Modelo.

Gabarito: Errado

---

34. (CESPE / MEC – 2011) A independência dos componentes é um dos atributos que reflete a qualidade do projeto. O grau de independência pode ser medido a partir dos conceitos de acoplamento e coesão, os quais, idealmente, devem ser alto e baixo, respectivamente.

Comentários:

Na verdade, é o contrário! Devem ser baixo e alto, respectivamente.

Gabarito: Errado

---

35. (CESPE / MEC – 2011) O termo cliente é usado para designar uma parte distinta de um sistema de computador que gerencia um conjunto de recursos relacionados e apresenta sua funcionalidade para usuários e aplicativos.

Comentários:

Na verdade, a questão trata de um serviço!

Gabarito: Errado

---

36. (CESPE / MEC – 2011) A arquitetura cliente/servidor proporciona a sincronização entre duas aplicações: uma aplicação permanece em estado de espera até que outra aplicação efetue uma solicitação de serviço.

Comentários:

Exato! A redação é meio complicada, mas o que essa questão quis dizer é: uma aplicação (servidor) fica em estado de espera aguardando solicitações e outra aplicação (cliente) efetua a solicitação de serviços.

Gabarito: Correto

---

37. (CESPE / MEC – 2011) A arquitetura cliente/servidor enseja o desenvolvimento de um sistema com, no máximo, duas camadas, quais sejam, cliente e servidor.

Comentários:

Na verdade, pode ter quantas camadas forem necessárias.

Gabarito: Errado

---

38. **(CESPE / ABIN – 2010)** Em sistemas de grande porte, um único requisito pode ser implementado por diversos componentes; cada componente, por sua vez, pode incluir elementos de vários requisitos, o que facilita o seu reúso, pois os componentes implementam, normalmente, uma única abstração do sistema.

Comentários:

Vocês se lembram da coesão e acoplamento? Um componente que inclui elementos de vários requisitos tem baixa coesão, reduzindo sua reusabilidade.

Gabarito: Errado

---

39. **(CESPE / INMETRO – 2010)** A coesão e o acoplamento são formas de se avaliar se a segmentação de um sistema em módulos ou em componentes foi eficiente. Acerca da aplicação desses princípios, assinale a opção correta.

a) O baixo acoplamento pode melhorar a manutebilidade dos sistemas, pois ele está associado à criação de módulos como se fossem caixas-pretas.

b) Os componentes ou os módulos devem apresentar baixa coesão e um alto grau de acoplamento.

c) Os componentes ou os módulos devem ser fortemente coesos e fracamente acoplados.

d) Um benefício da alta coesão é permitir realizar a manutenção em um módulo sem se preocupar com os detalhes internos dos demais módulos.

e) A modularização do programa em partes especializadas pode aumentar a qualidade desses componentes, mas pode prejudicar o seu reaproveitamento em outros programas.

Comentários:

(a) Errado. Na verdade, essa é uma função da coesão e, não, do acoplamento; (b) Errado. Os componentes ou os módulos devem apresentar alta coesão e baixo grau de acoplamento; (c) Correto. Essa é a regra geral: alta coesão e baixo acoplamento; (d) Errado. Esse é um benefício do baixo acoplamento; (e) Errado. Pelo contrário, quanto mais especializado, maior a coesão e maior a probabilidade de reaproveitamento em outros programas.

Gabarito: Correto

---

40. **(CESPE / BASA – 2010)** Nessa arquitetura (arquitetura multicamadas), quando são consideradas três camadas, a primeira camada deve ser implementada por meio do servidor de aplicação.

Comentários:

Por lógica, a primeira camada seria a Camada de Usuário ou a Camada de Dados. A Camada de Aplicação jamais seria a primeira camada.

Gabarito: Errado

---

41. **(CESPE / BASA – 2010)** Em arquitetura multicamadas, o servidor de aplicação nada mais é do que um programa que fica entre o aplicativo cliente e o sistema de gerenciamento de banco de dados.

Comentários:

É exatamente isso! Ele está entre a interface e os dados.

Gabarito: Correto

---

42. **(CESPE / BASA – 2010)** Uma desvantagem dessa arquitetura (arquitetura multicamadas) é o aumento na manutenção da aplicação, pois alterações na camada de dados, por exemplo, acarretam mudanças em todas as demais camadas.

Comentários:

Não, a divisão em camadas reduz a manutenção da aplicação.

Gabarito: Errado

---

43. **(CESPE / BASA – 2010)** Em uma arquitetura cliente-servidor, os clientes compartilham dos recursos gerenciados pelos servidores, os quais também podem, por sua vez, ser clientes de outros servidores.

Comentários:

Exato! Os servidores gerenciam seus recursos que são compartilhados pelas dezenas, milhares, milhões de clientes. Além disso, um servidor pode acabar sendo um cliente de outro servidor.

Gabarito: Correto

---

44. **(CESPE / BASA – 2010)** A Internet baseia-se na arquitetura cliente-servidor, na qual a parte cliente, executada no host local, solicita serviços de um programa aplicativo denominado servidor, que é executado em um host remoto.

Comentários:

Exato! A internet é isso: uma arquitetura cliente-servidor distribuída em que a parte cliente é executada no host local e solicita serviços de diversos servidores, que são executados em um host remoto.

Gabarito: Correto

---

45. (CESPE / BASA – 2010) A arquitetura cliente-servidor viabiliza o uso simultâneo de diferentes dispositivos computacionais, do seguinte modo: cada um deles realiza a tarefa para a qual é mais capacitado, havendo a possibilidade de uma máquina ser cliente em uma tarefa e servidor em outra.

Comentários:

Apesar de a redação da questão estar confusa *("tarefa para a qual é mais capacitado"*), a questão está certa em afirmar que uma máquina pode ser cliente em uma aplicação e servidora em outra.

Gabarito: Correto

---

46. (CESPE / EMBASA – 2010) O MVC promove a estrita separação de responsabilidades entre os componentes de uma interface.

Comentários:

Essa questão foi anulada, porque ela não especificou qual tipo de interface. Considerando tratar-se de interface gráfica, a questão seria correta.

Gabarito: Anulada

---

47. (CESPE / EMBASA – 2010) No MVC, a visão é responsável pela manutenção do estado da aplicação.

Comentários:

Na verdade, a questão trata da Camada de Modelo.

Gabarito: Errado

---

48. (CESPE / EMBASA – 2010) O modelo no MVC tem como atribuição exibir a parte que é responsável pela manutenção da aplicação para o usuário.

Comentários:

Na verdade, essa atribuição é da camada de visão.

Gabarito: Errado

---

49. (CESPE / EMBASA – 2010) O controlador é responsável pela coordenação entre atualizações no modelo e interações com o usuário.

Comentários:

Essa é realmente uma das responsabilidades do controlador.

Gabarito: Correto

---

50. (CESPE / EMBASA – 2010) Por meio do MVC, é possível o desenvolvimento de aplicações em 3 camadas para a Web.

Comentários:

Perfeito! Ela realmente permite o desenvolvimento de aplicações em três camadas para a web.

Gabarito: Correto

---

51. (CESPE / UNIPAMPA – 2009) Normalmente, a arquitetura em três camadas conta com as camadas de apresentação, de aplicação e de dados.

Comentários:

Exato! Arquitetura em Três Camadas: Apresentação, Aplicação e Dados.

Gabarito: Correto

---

52. (CESPE / UNIPAMPA – 2009) Em uma arquitetura em três camadas, na camada de aplicação, usualmente está um servidor de banco de dados que gerencia o conjunto de requisições.

Comentários:

Não, o Servidor de Banco de Dados usualmente se encontra na Camada de Dados.

Gabarito: Errado

---

53. (CESPE / UNIPAMPA – 2009) O uso de middlewares é comum em aplicações de n camadas.

Comentários:

Exato! Utilizam-se middlewares para realizar uma comunicação mais eficiente.

Gabarito: Correto

---

54. **(CESPE / UNIPAMPA – 2009)** Na camada de persistência dos dados em aplicações n camadas, podem ser utilizados o banco de dados orientado a objetos e o banco de dados relacionais.

Comentários:

É isso aí! É independente desse tipo de tecnologia.

Gabarito: Correto

---

55. **(CESPE / UNIPAMPA – 2009)** Nas aplicações cliente-servidor, em duas camadas, é simples acessar fontes de dados heterogêneas porque o legado de base de dados não precisa de drivers de conexões diferentes.

Comentários:

Na verdade, é necessário diversos drivers de conexões diferentes para acessar às bases de dados de fontes heterogêneas e não é simples! Em uma arquitetura cliente-servidor de três camadas, continua sendo necessário os drivers de conexões diferentes, porém é mais simples por haver uma camada responsável por gerenciar essas conexões.

Gabarito: Errado

---

56. **(CESPE / ANTAQ – 2009)** Os principais componentes da arquitetura cliente-servidor, que é um modelo de arquitetura para sistemas distribuídos, são o conjunto de servidores que oferecem serviços para outros subsistemas, como servidores de impressão e servidores de arquivos, o conjunto de clientes que solicitam os serviços oferecidos por servidores, e a rede que permite aos clientes acessarem esses serviços.

Comentários:

Perfeito, são clientes, servidores e uma rede que permite a comunicação.

Gabarito: Correto

---

57. **(CESPE / BASA – 2009)** Em arquiteturas cliente-servidor multicamadas, na maior parte das aplicações, o browser é adotado como cliente universal.

Comentários:

Sim, pessoal! Nessa arquitetura, o browser (navegador) geralmente é o cliente.

Gabarito: Correto

---

58. **(CESPE / ANAC – 2009)** O framework modelo visão controlador (MVC – model view controller) é muito utilizado para projeto da GUI (graphical user interface) de programas orientados a objetos.

Comentários:

Perfeito! Foi nesse contexto que ele foi criado e com esse objetivo.

Gabarito: Correto

---

59. **(CESPE / TCU – 2009)** No MVC (model-view-controller), um padrão recomendado para aplicações interativas, uma aplicação é organizada em três módulos separados. Um para o modelo de aplicação com a representação de dados e lógica do negócio, o segundo com visões que fornecem apresentação dos dados e input do usuário e o terceiro para um controlador que despacha pedidos e controle de fluxo.

Comentários:

Perfeito! O MVC promove a estrita separação de responsabilidade entre componentes de uma interface gráfica onde temos componentes responsáveis pela manutenção do estado da aplicação, denominado de Modelo, pela exibição de parte deste modelo para o usuário, ao que chamamos de Visão e pela coordenação entre atualizações no modelo e interações com o usuário, feita através do Controlador.

Gabarito: Correto

---

60. **(CESPE / ANATEL – 2009)** Uma das vantagens da arquitetura distribuída é o compartilhamento de recursos, que permite que sistemas, aplicativos e dispositivos periféricos, como discos, impressoras, arquivos, estejam associados a diferentes computadores em uma rede. Uma segunda vantagem é a concorrência, uma vez que vários processos podem operar ao mesmo tempo em diferentes computadores na rede. E, por fim, uma terceira vantagem é a proteção, pois o acesso é feito de forma centralizada.

Comentários:

Opa... o acesso é feito de forma descentralizada.

Gabarito: Errado

---

61. (CESPE / ANTAQ – 2009) Uma das desvantagens da arquitetura distribuída é sua complexidade, uma vez que é mais difícil compreender as propriedades emergentes dos sistemas que as dos sistemas centralizados.

Comentários:

É realmente complexo compreender e controlar as propriedades de sistemas distribuídos. *Já imaginaram controlar as propriedades de concorrência e transação de um sistema que está distribuído?* Pois é!

Gabarito: Correto

---

62. (CESPE / INMETRO – 2009) Em uma arquitetura distribuída, middleware é definido como uma camada de software cujo objetivo é mascarar a heterogeneidade e fornecer um modelo de programação conveniente para os programadores de aplicativos. Como exemplos de middlewares é correto citar: Sun RPC, CORBA, RMI Java e DCOM da Microsoft.

Comentários:

Perfeito! Middleware é uma camada de software que permite que elementos de aplicações interoperem através de redes de computadores. Ele oculta do desenvolvedor da aplicação a complexidade do processo de transporte da rede. Os exemplos mais comuns são: RPC, RMI, CORBA, DCOM, etc.

Gabarito: Correto

---

63. (CESPE / IPEA – 2008) Na arquitetura cliente-servidor com três camadas (three tier), a camada de apresentação, a camada de aplicação e o gerenciamento de dados ocorrem em diferentes máquinas. A camada de apresentação provê a interface do usuário e interage com o usuário, sendo máquinas clientes responsáveis pela sua execução. A camada de aplicação é responsável pela lógica da aplicação, sendo executada em servidores de aplicação. Essa camada pode interagir com um ou mais bancos de dados ou fontes de dados. Finalmente, o gerenciamento de dados ocorre em servidores de banco de dados, que processam as consultas da camada de aplicação e enviam os resultados.

Comentários:

Questão perfeita! Refere-se a Camadas Físicas (Tiers), portando cada camada está localizada em uma máquina.

Gabarito: Correto

---

64. (CESPE / STJ – 2008) A arquitetura de um sistema de software pode se basear em determinado estilo de arquitetura. Um estilo de arquitetura é um padrão de organização. No estilo cliente-servidor, o sistema é organizado como um conjunto de serviços, servidores e clientes associados que acessam e usam os serviços. Os principais componentes desse estilo são servidores que oferecem serviços e clientes que solicitam os serviços.

Comentários:

Exato! Trata-se de um conjunto de clientes e servidores que acessam e fornecem diversos serviços.

Gabarito: Correto

---

65. (CESPE / TJ-CE – 2008) A arquitetura MVC fornece uma maneira de dividir a funcionalidade envolvida na manutenção e apresentação dos dados de uma aplicação web.

Comentários:

O MVC promove a estrita separação de responsabilidade entre componentes de uma interface gráfica onde temos componentes responsáveis pela manutenção do estado da aplicação, denominado de Modelo, pela exibição de parte deste modelo para o usuário, ao que chamamos de Visão e pela coordenação entre atualizações no modelo e interações com o usuário, feita através do Controlador

Gabarito: Correto

---

66. (CESPE / TJ-CE – 2008) A arquitetura MVC foi desenvolvida recentemente para mapear as tarefas complexas de saída do sistema do usuário.

Comentários:

Durante a década de setenta, surgiu a necessidade de criação de uma arquitetura para ser utilizada em projetos de interface visual na linguagem de programação Smalltalk. A ideia original era organizar o código, separar responsabilidades, aumentar a manutenibilidade, promover um baixo acoplamento e uma alta coesão, fomentar a reusabilidade do código e tornar o sistema escalável

Gabarito: Errado

---

67. (CESPE / TJ-CE – 2008) Na arquitetura MVC, um controlador define o comportamento da aplicação, já que este é o responsável por interpretar as ações do usuário e as relaciona com as chamadas do modelo.

Comentários:

Ele realmente interpreta ações do usuário e direciona para as chamadas do modelo.

Gabarito: Correto

---

68. (CESPE / TJ-CE – 2008) A arquitetura MVC não separa a informação de sua apresentação, porque, em sistemas web, informação e apresentação estão na mesma camada.

Comentários:

Ela separa a informação (Modelo) da apresentação (Visão).

Gabarito: Errado

---

69. (CESPE / TJ-CE – 2008) O desenvolvimento de sistemas web ocorre tipicamente em três camadas. A arquitetura MVC aumenta o escopo do desenvolvimento para, no máximo, quatro camadas, sendo a quarta camada o processamento dos dados do usuário.

Comentários:

Não, tipicamente temos três camadas. No entanto, realmente não há restrição de escopo para quatro camadas. Pode-se acrescentar outras camadas principalmente nas camadas de controle e visão.

Gabarito: Errado

---

70. (CESPE / IPEA – 2008) A arquitetura distribuída é caracterizada pelo compartilhamento de recursos computacionais e serviços por meio da comunicação direta e descentralizada entre os sistemas envolvidos e inclui, entre outras coisas, a troca de informações, ciclos de processamento e espaço de armazenamento em disco.

Comentários:

Ela se caracteriza por seu compartilhamento de recursos e serviços – além da comunicação direta e descentralizada entre os sistemas. Enfim, a questão está perfeita!

Gabarito: Correto

---

71. (CESPE / SERPRO – 2008) Uma arquitetura distribuída permite a divisão de uma mesma tarefa em diferentes processadores em uma mesma CPU. Essa característica aumenta a velocidade de processamento de uma informação.

Comentários:

Não se deve confundir Arquitetura Distribuída com Arquitetura Paralela. Observem que, no caso apresentado, não há uma rede de computadores – tudo ocorre em um uma mesma CPU! Logo, a questão trata de Arquitetura Paralela.

Gabarito: Errado

---

72. (CESPE / TCU – 2007) A arquitetura cliente-servidor tem por motivação sincronizar a execução de dois processos que devem cooperar um com outro. Assim, dadas duas entidades que queiram comunicar-se, uma deve iniciar a comunicação enquanto a outra aguarda pela requisição da entidade que inicia a comunicação.

Comentários:

Parece complicado, mas é simples! Uma entidade começa a comunicação e a outra aguarda a requisição da entidade iniciadora.

Gabarito: Correto

---

73. (CESPE / DATAPREV – 2006) Uma arquitetura cliente/servidor caracteriza-se pela separação do cliente, o usuário que acessa ou demanda informações, do servidor. Um exemplo típico é um navegador que acessa páginas na Internet. É uma arquitetura que permite o acesso a serviços remotos através de rede de computadores, e que tem como principal deficiência a falta de escalabilidade.

Comentários:

Na verdade, um dos benefícios da arquitetura cliente/servidor é a escalabilidade.

Gabarito: Errado

---

74. (CESPE / DATAPREV – 2006) Arquiteturas cliente/servidor podem ser decompostas em mais de duas camadas. Uma configuração muito utilizada é aquela em que os clientes acessam informações por meio de servidores de aplicação, que por sua vez acessam servidores de banco de dados. Este tipo de arquitetura é conhecida como arquitetura em 3 camadas, ou three-tier.

Comentários:

Exato! O usuário utiliza um servidor de aplicação que acessa um servidor de banco de dados.

Gabarito: Correto

---

75. **(CESPE / CENSIPAM – 2006)** O padrão MVC organiza um software em modelo, visão e controle. O modelo encapsula as principais funcionalidades e dados. As visões apresentam os dados aos usuários. Uma visão obtém os dados do modelo via funções disponibilizadas pelo modelo; só há uma visão para um modelo. Usuários interagem via controladoras que traduzem os eventos em solicitações ao modelo ou à visão; podem existir várias controladoras associadas a uma mesma visão.

Comentários:

Opa... pode haver várias visões para um modelo.

Gabarito: Errado

---

76. **(CESPE / CENSIPAM – 2006)** No padrão MVC, se um usuário modifica o modelo, as visões que dependem desse modelo refletem essas modificações, pois o modelo notifica as visões quando ocorre uma modificação nos seus dados. Portanto, é usado um mecanismo para propagação de modificações que mantém um registro dos componentes que dependem do modelo.

Comentários:

Mudanças em um modelo podem modificar as visões que dependem desse modelo, gerando uma rastreabilidade.

Gabarito: Correto

---

77. **(CESPE / SEAD-PA – 2004)** Em um modelo cliente-servidor em que o processamento é concentrado nos clientes e o armazenamento concentrado no servidor, observa-se uma baixa carga de tráfego na rede.

Comentários:

É um caso de Cliente-Gordo, em que o tráfego na rede é menor. No entanto nada garante que o tráfego na rede será baixo, ele pode ser altíssimo, porém é menor comparado ao Cliente-Magro.

Gabarito: Correto

---

78. **(CESPE / SERPRO – 2004)** Uma das vantagens da arquitetura cliente-servidor é que parte da carga de processamento é retirada do servidor e colocada nos vários clientes.

Comentários:

Essa é uma das grandes vantagens da arquitetura cliente-servidor. Ela retira uma parte do processamento dos serviços (desonerando o servidor) e transfere para os clientes.

Gabarito: Correto

79.(CESPE / STJ – 2004) As camadas da arquitetura cliente-servidor de três camadas são: camada de interface de usuário, camada de regras de negócio e camada de acesso ao banco de dados.

Comentários:

Esses nomes mudam bastante, mas a questão está certa!

Gabarito: Correto

80.(CESPE / STJ – 2004) Na arquitetura cliente-servidor multicamadas, uma alteração na camada de acesso aos dados não afeta a camada de interface de usuário, desde que essas camadas estejam na mesma máquina.

Comentários:

Na verdade, mesmo que essas camadas estejam na mesma máquina, uma alteração na camada de acesso aos dados não afeta a camada de interface de usuário.

Gabarito: Errado

81.(CESPE / STJ – 2004) A arquitetura cliente-servidor multicamadas possui a vantagem de que a camada de interface de usuário pode se comunicar diretamente com qualquer outra camada, ou seja, não existe hierarquia entre camadas.

Comentários:

Não, a camada de interface se comunica diretamente apenas com a camada de negócio.

Gabarito: Errado

82.(CESPE / TRE-RS – 2003) Aplicações com arquitetura cliente-servidor são assimétricas, no sentido de que o cliente e o servidor possuem papéis diferentes na arquitetura de comunicações.

Comentários:

Perfeito, eles possuem papéis distintos.

Gabarito: Correto

83.(CESPE / TRE-RS – 2003) O servidor, por possuir normalmente um hardware mais robusto, sempre deve executar a parte mais pesada do processamento.

Comentários:

Não! Por exemplo, quando se utiliza um cliente-gordo, a maior parte do processamento ocorre no cliente e, não, no servidor.

Gabarito: Errado

84.(CESPE / TRE-RS – 2003) Do ponto de vista das funcionalidades de usuários, o servidor não precisa necessariamente de uma interface de usuário.

Comentários:

Perfeito, percebam que foi dito *"do ponto de vista das funcionalidades de usuários"*. Uma vez que sejam executadas as funcionalidades esperadas, não há necessidade de uma interface de usuário.

Gabarito: Correto

QUESTÕES COMENTADAS – FCC

1. **(FCC / TRF3 – 2019)** Os conceitos alta coesão e baixo acoplamento, utilizados no processo de desenvolvimento de software, são princípios essenciais de:

a) Abstração.
b) Modularidade.
c) Incrementação.
d) Separação de Interesses.
e) Generalidade.

Comentários:

De acordo com Pressman: "*Na modularidade o software é dividido em componentes separadamente especificados e localizáveis, algumas vezes denominados módulos, que são integrados para satisfazer os requisitos de um problema*". Essa divisão em módulos facilita a compreensão e, consequentemente, reduz o custo necessário para se construir um software.

Como a coesão trata da divisão de responsabilidades e o acoplamento do nível de dependência entre os módulos, desejamos uma alta coesão e um baixo acoplamento em processo de modularização.

Gabarito: Letra B

---

2. **(FCC / DPE-AM – 2018)**

Trecho 1:
```
public int pensaoAlimenticia(){
  return Util.getFuncoes.getFuncoesData.calculaPensao(processo);
}
```

Trecho 2:
```
public int pensaoAlimenticia(){
  return Util.calculaPensao(processo);
}
```

Em um sistema Orientado a Objetos bem desenvolvido, os princípios relativos a acoplamento e coesão devem ser respeitados. O código Java apresentando no trecho 1 mostra um exemplo de

a) baixo acoplamento e o trecho 2 o corrige para alto acoplamento.
b) alta coesão e o trecho 2 o corrige para baixa coesão.

c) alto acoplamento e o trecho 2 o corrige para baixo acoplamento.
d) baixo acoplamento e o trecho 2 mostra um exemplo de baixa coesão.
e) baixa coesão e o trecho 2 mostra um exemplo de alto acoplamento.

Comentários:

O acoplamento trata do nível de dependência entre módulos ou componentes de um software. Dito isso, podemos analisar os trechos de código:

O trecho 1 trata da função pensãoAlimenticia(), em que há grande dependência de outros métodos no retorno; já no trecho 2, essa mesma função é construída, mas com menos dependência em relação ao trecho 1. Logo, há alto acoplamento no trecho 1 e baixo acoplamento no trecho 2.

Gabarito: Letra C

---

3. **(FCC / TCM-GO – 2015 – Adaptada)** Quanto à Arquitetura em 3 Camadas, é necessário um arranjo que possibilite a reutilização do código e facilite sua manutenção e seu aperfeiçoamento. Deve- se separar Apresentação, Regra de Negócio e Acesso a Dados. Busca-se a decomposição de funcionalidades de forma a permitir aos desenvolvedores concentrarem-se em diferentes partes da aplicação durante a implementação.

Comentários:

Perfeito! Essa é uma situação bastante comum em uma arquitetura em três camadas.

Gabarito: Letra C

---

4. **(FCC / CNMP – 2015)** Há algumas variantes possíveis de arquitetura a serem utilizadas em um sistema de bancos de dados. Sobre essas variantes, é correto afirmar que:

a) na arquitetura de 3 camadas, não há uma camada específica para a aplicação.

b) a camada de apresentação da arquitetura de 2 camadas situa-se, usualmente, no servidor de banco de dados.

c) na arquitetura de 3 camadas, a camada de servidor de banco de dados é denominada cliente.

d) a arquitetura de 3 camadas é composta pelas camadas cliente, aplicação e servidor de banco de dados.

e) na arquitetura de 2 camadas não há necessidade de uso de um sistema gerenciador de bancos de dados.

Comentários:

(a) Errado, é a camada intermediária; (b) Errado, fica na camada de apresentação; (c) Errado, é chamada camada de dados; (d) Correto; (e) Errado, é claro que há necessidade.

Gabarito: Letra D

---

5. **(FCC / TJ-AP – 2014)** Uma arquitetura muito comum em aplicações web é o Modelo Arquitetural 3 Camadas:

I. Camada de Persistência.
II. Camada de Lógica de Negócio.
III. Camada de Apresentação.

Neste modelo, a correta associação dos componentes com as camadas é:

a) I-Servidor de Banco de Dados - II-Servidor de Aplicação - III-Máquina Cliente.
b) I-Servidor Web - II-Servidor Cliente - III-Servidor de Aplicação.
c) I-Servidor Web - II-Servidor de Banco de Dados - III-Máquina Cliente.
d) I-Servidor de Banco de Dados - II-Máquina Cliente - III-Servidor de Aplicação.
e) I-Máquina Cliente - II-Servidor de Banco de Dados - III-Servidor Web.

Comentários:

Servidor de Banco de Dados se associa com... Camada de Persistência;
Servidor de Aplicação se associa com... Camada de Lógica de Negócio;
Máquina Cliente se associa com... Camada de Apresentação.

Gabarito: Letra A

---

6. **(FCC / TST – 2012)** Uma arquitetura em camadas:

a) possui apenas 3 camadas, cada uma realizando operações que se tornam progressivamente mais próximas do conjunto de instruções da máquina.

b) tem, na camada mais interior, os componentes que implementam as operações de interface com o usuário.

c) pode ser combinada com uma arquitetura centrada nos dados em muitas aplicações de bancos de dados.

d) tem, como camada intermediária, o depósito de dados, também chamado de repositório ou quadro-negro.

e) tem, na camada mais externa, os componentes que realizam a interface com o sistema operacional.

Comentários:

(a) Errado. Não necessariamente possui três camadas – pode possuir 2, 3, 4, 5, etc; (b) Errado. Interface com o usuário é, em geral, na camada mais externa; (c) Correto. Não há nada que impeça isso; (d) Errado. O depósito de dados, em geral, fica na camada mais interna; (e) Errado. Interface com o sistema operacional, em geral, fica na camada mais interna.

Gabarito: Letra C

---

7. **(FCC / TRF2 – 2012)** São aspectos que podem caracterizar uma arquitetura cliente-servidor, estabelecida logicamente em 4 camadas:

I. A camada Cliente contém um navegador de Internet, caracterizado como cliente universal.

II. A camada de Lógica do Negócio se quebra em duas: camada de Aplicação e camada Web, em que o servidor Web representa a camada de Apresentação.

III. Na camada de Lógica do Negócio, o servidor de aplicação passa a utilizar middleware, para suporte a thin clients (PDA, celulares, smart cards, etc) e soluções baseadas em componentes, tais como, J2EE e .Net.

IV. Se, de um lado, a camada de Aplicação estabelece uma interface com a camada de Dados, do outro o faz com a camada Web e com os thin clients da camada Cliente.

Está correto o que consta em:

a) I e II, apenas.
b) III e IV, apenas.
c) I, II e III, apenas.
d) II, III e IV, apenas.
e) I, II, III e IV.

Comentários:

(I) Correto. Na Arquitetura em 4 Camadas, a Camada Cliente contém um navegador, caracterizado como cliente universal; (II) Errado. Camada de Lógica de Negócio é a Camada de Aplicação e a Camada Web não faz parte dela. Eu nunca ouvi falar nisso, mas a FCC disse que é correto! A diferença entre a Arquitetura em 4 Camadas e 3 Camadas é que a Apresentação não fica mais no cliente e, sim, na Camada Web. (III) Errado. Imagino que o middleware à que a questão se refere é

o Servidor Web, mas ele ficaria na Camada Web fazendo o meio-campo para suportar thin clients e, não, na Camada de Lógica de Negócio – a FCC discorda! (IV) Correto. A Camada de Aplicação fica entre a Camada de Dados e a Camada Web.

Gabarito: Letra E

---

8. **(FCC / TST – 2012)** No padrão MVC é possível definir grupos de componentes principais: o Model (Modelo), o View (Apresentação) e o Controller (Controle). Deve fazer parte do componente:

a) Controller, uma classe que contém um método com a finalidade de calcular o reajuste de salário dos funcionários.

b) View, uma classe que contém um método para persistir o salário reajustado de um funcionário.

c) Controller, as animações desenvolvidas em Flash.

d) View, as validações necessárias ao sistema, geralmente definidas através de um conjunto de comparações.

e) Model, as classes com métodos conhecidos como setters e getters e que representam tabelas do banco de dados.

Comentários:

(a) Errado, isso deve fazer parte do Model; (b) Errado, isso deve fazer parte do Model; (c) Errado, isso deve fazer parte da View; (d) Errado, isso deve fazer parte da Model; (e) Correto, esses métodos de fato devem estar na Model.

Gabarito: Letra E

---

9. **(FCC / MPE-AP – 2012)** Em uma Aplicação Web desenvolvida utilizando o design pattern MVC, as páginas HTML e as classes com métodos que acessam o banco de dados e executam instruções SQL são representadas, respectivamente, nos componentes:

a) Presentation e Business.
b) View e Model.
c) Controller e Model.
d) Model e Business.
e) View e Business.

Comentários:

Páginas HTML são representadas na View; classes com métodos que acessam o banco de dados e executam instruções SQL são representadas na Model.

Gabarito: Letra B

---

10. (FCC / MPE-PE – 2012) O padrão de projeto utilizado em aplicações WEB que permite separar as páginas e classes da aplicação em três grupos (muitas vezes chamados de camadas) conhecidos como Apresentação, Controle e Modelo é denominado de:

a) 3-tier.
b) DAO.
c) MVC.
d) DTO.
e) DBO.

Comentários:

Só pode ser... MVC!

Gabarito: Letra C

---

11. (FCC / TRT-PE – 2012) O padrão de arquitetura MVC é um modelo de camadas que divide a aplicação em três componentes: Model (modelo), View (visualizador) e Controller (controlador). As funções de cada um destes três componentes são apresentadas abaixo:

I. interpreta eventos de entrada e envia requisições para o modelo de dados; em seguida, processa os dados carregados a partir do modelo e envia para o visualizador.

II. encapsula o acesso aos dados e funções básicas da aplicação, fornecendo ao usuário procedimentos que executam tarefas específicas.

III. exibe para o usuário os dados fornecidos pelo controle e estabelece uma interface para interação entre o usuário e a aplicação.

A associação correta do componente do padrão MVC com sua função está expressa, respectivamente, em

a) (I) Controller; (II) Model; (III) View;
b) (I) Model; (II) Controller; (III) View;
c) (I) View; (II) Model; (III) Controller;
d) (I) Controller; (II) View; (III) Model;
e) (I) Model; (II) View; (III) Controller;

Comentários:

(I) Quem faz essa orquestração de requisições é a Camada de Controle; (II) Acesso aos dados e funções básicas da aplicação são de responsabilidade da Camada de Modelo; (III) Exibição é sempre Camada de Visão!

Gabarito: Letra A

---

12. (FCC / TJ-PE – 2012) Com relação à arquitetura MVC, considere:

I. O MODEL representa os dados da empresa e as regras de negócio que governam o acesso e atualização destes dados.

II. O VIEW acessa os dados da empresa através do MODEL e especifica como esses dados devem ser apresentados. É de responsabilidade do VIEW manter a consistência em sua apresentação, quando o MODEL é alterado.

III. O CONTROLLER traduz as interações do VIEW em ações a serem executadas pelo MODEL. Com base na interação do usuário e no resultado das ações do MODEL, o CONTROLLER responde selecionando uma VIEW adequada.

IV. Permite uma única VIEW para compartilhar o mesmo modelo de dados corporativos em um fluxo de comunicação sequencial.

Está correto o que se afirma em:

a) I, II, III e IV.
b) I, II e III, apenas.
c) II e III, apenas.
d) II, III e IV, apenas.
e) I e II, apenas.

Comentários:

(I) Correto, ele representa os dados e as regras de negócio; (II) Correto, a View acessa os dados por meio do Model. Essa parte gera dúvida, mas é responsabilidade da View – após ser notificada – atualizar a apresentação quando há modificação na Model; (III) Correto, essa é a função do Controller, ele orquestra as requisições entre Model e View; (IV) Errado, na verdade permite quantas views forem necessárias para um mesmo Model;

Gabarito: Letra B

---

13. (FCC / MPE-PE – 2012) O componente Controller do MVC:

a) Define o comportamento da aplicação, as ações do usuário para atualizar os componentes de dados e seleciona os componentes para exibir respostas de requisições.

b) Envia requisições do usuário para o controlador e recebe dados atualizados dos componentes de acesso a dados.

c) Responde às solicitações de queries e encapsula o estado da aplicação.

d) Notifica os componentes de apresentação das mudanças efetuadas nos dados e expõe a funcionalidade da aplicação.

e) É onde são concentradas todas as regras de negócio da aplicação e o acesso aos dados.

Comentários:

(a) Correto, ele define o comportamento da aplicação, as ações do usuário para atualizar os componentes de dados e seleciona os componentes para exibir respostas de requisições; (b) Errado, essa responsabilidade é da View; (c) Errado, essa responsabilidade é do Model; (d) Errado, essa responsabilidade é do Model; (e) Errado, essa responsabilidade é do Model;

Gabarito: Letra A

---

14. (FCC / TRT-MT – 2011) No projeto de arquitetura modelo-visão-controle (MVC), o controlador:

a) renderiza a interface de usuário a partir da visão, o modelo encapsula funcionalidades e objetos de conteúdo e a visão processa e responde a eventos e invoca alterações ao controlador.

b) encapsula funcionalidades e objetos de conteúdo, o modelo processa e responde a eventos e invoca alterações ao controlador e a visão renderiza a interface de usuário a partir do modelo.

c) encapsula funcionalidades e objetos de conteúdo, o modelo renderiza a interface de usuário a partir da visão e a visão processa e responde a eventos e invoca alterações ao controlador.

d) processa e responde a eventos e invoca alterações ao modelo, o modelo encapsula funcionalidades e objetos de conteúdo e a visão renderiza a interface de usuário a partir do modelo.

e) processa e responde a eventos e invoca alterações ao modelo, o modelo renderiza a interface de usuário a partir da visão e a visão encapsula funcionalidades e objetos de conteúdo.

Comentários:

O Controlador processa e responde a eventos e invoca alterações ao modelo, o modelo encapsula funcionalidades e objetos de conteúdo e a visão renderiza a interface de usuário a partir do modelo.

Gabarito: Letra D

---

15. (FCC / TRT-SE – 2010) No desenvolvimento de sistemas, no âmbito das relações intermodulares entre as classes, diz-se que o programa está bem estruturado quando há:

a) maior coesão e maior acoplamento.
b) menor coesão e maior acoplamento.
c) menor coesão e menor acoplamento.
d) maior coesão e menor acoplamento.
e) apenas coesão ou apenas acoplamento.

Comentários:

Nosso mantra: alta coesão e baixo acoplamento!

Gabarito: Letra D

---

16. (FCC / TCM-PA – 2010) Extensão natural do conceito de ocultação de informações, que diz: "um módulo deve executar uma única tarefa dentro do procedimento de software, exigindo pouca interação com procedimentos que são executados em outras partes de um programa", é o conceito de:

a) coesão.
b) enfileiramento.
c) acoplamento.
d) visibilidade.
e) recursividade.

Comentários:

Módulo deve executar uma única tarefa? Divisão de responsabilidades, isto é, coesão.

Gabarito: Letra A

---

17. (FCC / TRT-SE – 2010) A arquitetura multicamadas divide-se em três camadas lógicas. São elas:

a) Apresentação, Negócio e Acesso a Dados.
b) Apresentação, Natureza e Acesso a Dados.
c) Apresentação, Negócio e Alteração.
d) Manipulação, Natureza e Acesso a Dados.

e) Manipulação, Negócio e Acesso a Dados.

Comentários:

Easy! Apresentação, Negócio e Acesso a Dados.

Gabarito: Letra A

---

18. (FCC / METRÔ-SP – 2010) A arquitetura multicamadas divide-se em três camadas lógicas. São elas:

a) Apresentação, Negócio e Alteração.
b) Manipulação, Natureza e Acesso a Dados.
c) Manipulação, Negócio e Acesso a Dados.
d) Apresentação, Natureza e Acesso a Dados.
e) Apresentação, Negócio e Acesso a Dados.

Comentários:

Opa... Apresentação, Negócio e Acesso a Dados. Vejam que as bancas se autocopiam – essa questão é quase idêntica à anterior.

▪

Gabarito: Letra E

---

19. (FCC / AL-SP – 2010) Sobre as camadas do modelo de arquitetura MVC (Model- View-Controller) usado no desenvolvimento web é correto afirmar:

a) Todos os dados e a lógica do negócio para processá-los devem ser representados na camada Controller.

b) A camada Model pode interagir com a camada View para converter as ações do cliente em ações que são compreendidas e executadas na camada Controller.

c) A camada View é a camada responsável por exibir os dados ao usuário. Em todos os casos essa camada somente pode acessar a camada Model por meio da camada Controller.

d) A camada Controller geralmente possui um componente controlador padrão criado para atender a todas as requisições do cliente.

e) Em aplicações web desenvolvidas com Java as servlets são representadas na camada Model.

Comentários:

(a) Errado, fica na Camada de Modelo; (b) Errado, quase tudo correto, mas as ações são executadas na Camada de Modelo; (c) Errado, trata-se de uma arquitetura triangular, logo pode ser acessada diretamente; (d) Correto, geralmente há um controlador padrão, sim (Ex: Servlets); (e) Errado, as Servlets são representadas na Camada de Controle.

Gabarito: Letra D

---

20. (FCC / TRT3 – 2009) Considerando o conjunto de tarefas que se relacionam em um módulo e o espectro de medidas da força funcional relativa dos módulos (coesão), a respectiva sequência, da pior para a melhor, é:

a) sequencial, temporal e lógica.
b) procedimental, coincidental e funcional.
c) temporal, lógica e sequencial.
d) temporal, comunicacional e sequencial.
e) procedimental, funcional e lógica.

Comentários:

Indesejáveis: Coincidental, Lógica e Temporal; **Intermediárias:** Procedural, Comunicacional e Sequencial; **Desejável:** Funcional. Então, do pior para a melhor, temos: Temporal, Comunicacional e Sequencial.

Gabarito: Letra D

---

21. (FCC / TJ-SE – 2009) No modelo de três camadas MVC para web services, o responsável pela apresentação que também recebe os dados de entrada do usuário é a camada:

a) View.
b) Application.
c) Controller.
d) Data.
e) Model.

Comentários:

Apresentação? Camada de Visão (View).

Gabarito: Letra A

---

22. (FCC / TRT-MA – 2009) Considere as funções:

I. Seleção do comportamento do modelo.

II. Encapsulamento dos objetos de conteúdo.
III. Requisição das atualizações do modelo.

Na arquitetura Model-View-Control - MVC, essas funções correspondem, respectivamente, a:

a) Model, View e Control.
b) Control, View e Model.
c) View, Model e Control.
d) Control, Model e View.
e) View, Control e Model.

Comentários:

A seleção do comportamento do modelo é feita pela Camada de Controle; Encapsulamento dos objetos de conteúdo é feito pela Camada de Modelo; Requisição das atualizações do modelo são feitas pela Camada de Visão.

Gabarito: Letra D

---

23. (FCC / TRT-GO – 2008) Visando obter maior independência funcional, é adequado que o esforço seja direcionado ao projeto de módulos:

a) que não usem estruturas de seleção.
b) cujas tarefas tenham elevada coesão.
c) cujas tarefas tenham coesão procedimental.
d) que não usem estruturas de repetição.
e) cujas tarefas tenham coesão lógica.

Comentários:

Maior independência funcional ocorre com alta coesão!

Gabarito: Letra B

---

24. (FCC / TRF5 – 2008) Via de regra as divisões da arquitetura de software em três camadas orientam para níveis que especificam:

a) os casos de uso, a estrutura dos dados e os processos de manutenção.
b) a apresentação, as regras de negócio e os testes.
c) a apresentação, os processos operacionais e a seqüência de execução.
d) a apresentação, os componentes virtuais e a seqüência de execução.
e) a apresentação, as regras de negócio e o armazenamento de dados.

Comentários:

Tranquila também, só mudaram as palavras! Apresentação, Regras de Negócio e Armazenamento de Dados.

Gabarito: Letra E

QUESTÕES COMENTADAS – DIVERSAS BANCAS

1. **(CS-UFG / SANEGO-GO – 2018)** Dentro dos bons princípios de projeto e construção de software, a Lei de Démeter diz que "um método deve enviar mensagens somente para objetos a que ele tem acesso direto". Essa lei tem como objetivo:

   a) aumentar a coesão.
   b) diminuir o acoplamento.
   c) facilitar a criação de dependência entre as classes.
   d) aumentar a quantidade de casos de teste.

Comentários:

Como a Lei de Démeter diz que o objeto pode enviar mensagens apenas para objetos a que ele tem acesso direto, a intenção é diminuir o acoplamento, ou seja, diminuir o nível de dependência entre as classes.

Gabarito: Letra B

---

2. **(UFG / SANEAGO – 2017)** O emprego de boas práticas de projeto (design) de software visa resultar em um código:

   a) altamente acoplado e altamente coeso.
   b) altamente acoplado e fracamente coeso.
   c) fracamente acoplado e altamente coeso.
   d) fracamente acoplado e fracamente coeso.

Comentários:

A regra de ouro de uma arquitetura de software: alta/forte coesão e baixo/fraco acoplamento!

Gabarito: Letra C

---

3. **(UFG / SANEAGO – 2017)** Dentro dos padrões arquiteturais de software, a arquitetura Model-View-ViewModel (MVVM) é próxima da arquitetura Model-View-Presenter (MVP), porém diferencia-se desta pelo fato de:

   a) ser desprovida de um componente controlador como existe no Model-View-Controller (MVC).

   b) implementar o padrão de projeto Observer na ligação entre dados (ViewModel) e tela (view).

   c) ligar diretamente as classes de tela (view) e dados (Model) dentro da estrutura do projeto.

d) vincular a realização de atualizações de tela (view) à atualização de dados (ViewModel).

Comentários:

Perfeito! É a aplicação do Padrão de Projeto Observer!

Gabarito: Letra B

---

4. **(IBFC / EBSERH – 2017)** O modelo de três camadas físicas (3-tier), especificado nas alternativas, divide um aplicativo de modo que a lógica de negócio resida no meio das três camadas, foi adaptado como uma arquitetura para as aplicações Web em todas as linguagens de programação maiores. Muitos frameworks de aplicação comerciais e não comerciais foram criados tendo como base a arquitetura:

a) MVC (Model-View-Controller)
b) MDB (Model-Data-Business)
c) UDC (User-Data-Controller)
d) MDC (Model-Data-Controller)
e) UVB (User-View-Business).

Comentários:

A divisão do aplicativo que separa em três camadas é a Arquitetura MVC (Model, View e Control).

Gabarito: Letra A

---

5. **(CESGRANRIO / CEFET-RJ – 2014)** No contexto da Arquitetura de Sistemas, o MVC (model – view – controller) é um estilo arquitetural:

a) interativo
b) estrutural
c) distribuído
d) adaptável
e) monolítico

Comentários:

É um estilo arquitetural interativo, no sentido de que é um estilo para interface de usuário, fornecendo diversas visões diferentes para um mesmo modelo de dados.

Gabarito: Letra A

---

6. **(IBFC / TRE-AM – 2014)** Na arquitetura cliente-servidor, além dos dois principais componentes Cliente e o Servidor, existe um terceiro elemento intermediando os dois. Esse componente é chamado tecnicamente de:

a) coreware.
b) middleware.
c) mainware.
d) centerware.

Comentários:

O conceito de Middleware é amplamente utilizado em diversos contextos da computação. Como o próprio nome remete, ele é um "software do meio" ou "software intermediário". Possui diversas aplicações, como: intermediar a comunicação entre cliente e servidor (muito utilizado em ambientes distribuídos); intermediar a comunicação entre Softwares que utilizam protocolos ou plataformas diferentes; Intermediar a comunicação entre Sistema Operacional e Aplicações.

Gabarito: Letra B

---

7. **(IBFC / TRE-AM – 2014)** No desenvolvimento de sistemas dentro do conceito da arquitetura cliente-servidor de três camadas, temos as seguintes camadas:

1. Camada de Dados.
2. Camada de Apresentação.
3. Camada de Aplicações.
4. Camada de Negócio.

Estão corretas as afirmativas:

a) somente 1, 2 e 4.
b) somente 2, 3 e 4.
c) somente 1, 3 e 4.
d) somente 1, 2 e 3.

Comentários:

A arquitetura cliente-servidor se divide em Camada de Dados, Camada de Apresentação e Camada de Negócio. Logo, somente 1,2 e 4.

Gabarito: Letra A

---

8. **(ESAF / CGU – 2012)** A definição de que um sistema deve ser desenvolvido em três níveis é feita pelo padrão de projeto:

a) MVC (Model View Controller).
b) MVC-Dev (Model Value Constructive Development).
c) TMS (Time Milestones Setting).
d) PMC (Project Main Controller).
e) MCA (Model Classes Assignment).

Comentários:

A questão trata do MVC (Model View Controller).

Gabarito: Letra A

---

9. **(ESAF / CVM – 2010)** Modelo MVC significa:

a) Modo-View-Construtor.
b) Modelo-View-Controlador.
c) Modelo-Versão-Case.
d) Módulo-Verificador-Controlador.
e) Medida-Virtual-Concepção.

Comentários:

A questão trata do Modelo-View-Controlador.

Gabarito: Letra B

LISTA DE QUESTÕES – CESPE

1. **(CESPE / CAU-BR – 2024)** Ao se migrar para uma arquitetura cliente/servidor multinível, a mesma aplicação pode assumir simultaneamente as funções de cliente e de servidor.

2. **(CESPE / FINEP – 2024)** O principal objetivo de se reduzir o acoplamento entre módulos em um sistema de software é:

   a) minimizar a necessidade de documentação.
   b) melhorar a eficiência de comunicação entre módulos.
   c) diminuir o tamanho dos módulos.
   d) aumentar a coesão entre módulos.
   e) facilitar a integração de novos módulos.

3. **(CESPE / BNB – 2022)** No padrão MVC, o componente de modelo gerencia as requisições dos usuários.

4. **(CESPE / BNB – 2022)** Na arquitetura em camadas, os componentes da camada mais interna opera o sistema operacional, ao passo que os da camada mais externa interagem com o usuário.

5. **(CESPE / Petrobrás - 2022)** Enquanto a arquitetura é responsável pela infraestrutura de alto nível do software, o design é responsável pelo software a nível de código, como, por exemplo, o que cada módulo está fazendo, o escopo das classes e os objetivos das funções.

6. **(CESPE / TJ-RJ - 2021)** Na arquitetura MVC (Model-View-Controller), as funcionalidades de cada segmento são mais bem descritas como:

   a) o modelo encapsula as funcionalidades; o view gerencia as requisições dos usuários; o controlador prepara dados do modelo.

   b) o modelo encapsula objetos de conteúdo; a visão solicita atualizações do modelo; o controlador seleciona o comportamento do modelo.

   c) o modelo incorpora todos os estados; o view gerencia as requisições dos usuários; o controlador encapsula objetos de conteúdo.

   d) o modelo encapsula objetos de conteúdo; o view seleciona o comportamento do modelo; o controlador solicita atualizações do modelo.

   e) o modelo seleciona a resposta da visão; a visão apresenta a visão selecionada pelo controlador; o controlador encapsula objetos de conteúdo.

7. **(CESPE / TJ-RJ - 2021)** Em um ambiente cliente/servidor, a arquitetura que permite a mesma aplicação assumir tanto o papel de cliente quanto o de servidor é conhecida como arquitetura C/S:

a) simples.
b) de dois níveis.
c) multinível.
d) de três camadas.
e) par-par.

8. **(CESPE / TELEBRÁS - 2021)** Na arquitetura de software, a arquitetura cliente/servidor tem como vantagem uma maior facilidade de manutenção e segurança dos dados, e como desvantagens possíveis bloqueios no tráfego da rede, além de problemas de atualização da interface de aplicação.

9. **(CESPE / TELEBRÁS - 2021)** Por se tratar de uma arquitetura distribuída, o modelo cliente-servidor pressupõe facilidades para atualizar os servidores de forma transparente, sem que isso afete outras partes do sistema.

10. **(CESPE / TRE-BA – 2017)** Com referência às arquiteturas multicamadas de aplicações para o ambiente web, assinale a opção correta.

a) Se, na camada de dados, for realizada uma alteração no banco de dados, o restante das camadas também será afetado.

b) O modelo de três camadas recebe essa denominação caso um sistema cliente-servidor seja desenvolvido mantendo-se a camada de negócio do lado do cliente e as camadas de apresentação e dados no lado do servidor.

c) Cada camada é normalmente mantida em um servidor específico para tornar-se o mais escalonável e independente possível em relação a outras camadas, estando entre as suas principais características o eficiente armazenamento e a reutilização de recursos.

d) O objetivo das arquiteturas multicamadas consiste na junção de responsabilidades entre os componentes das aplicações web, de modo a atender aos requisitos funcionais e não funcionais esperados pela aplicação.

e) Na arquitetura de duas camadas — apresentação e armazenamento —, o computador que contiver a base de dados terá de ficar junto com os computadores que executarem as aplicações.

11. **(CESPE / STJ – 2015)** Na arquitetura em camadas MVC (modelo-visão-controlador), o modelo encapsula o estado de aplicação, a visão solicita atualização do modelo e o controlador gerencia a lógica de negócios.

12. **(CESPE / MEC – 2015)** O controlador gerencia as requisições dos usuários encapsulando as funcionalidades e prepara dados do modelo.

13. **(CESPE / MEC – 2015)** A visão encapsula objetos de conteúdo, solicita atualizações do modelo e seleciona o comportamento do modelo.

14. **(CESPE / STJ – 2015)** No padrão em camadas modelo-visão-controle (MVC), o controle é responsável por mudanças de estado da visão.

15. **(CESPE / ANTAQ – 2014)** O modelo MVC é um padrão de arquitetura que consiste na definição de camadas para a construção de softwares.

16. **(CESPE / ANTAQ – 2014)** O controller tem a responsabilidade de armazenar e buscar os dados que deverão ser exibidos pelo view.

17. **(CESPE / INPI – 2013)** De acordo com os princípios da engenharia de software relacionados à independência funcional, os algoritmos devem ser construídos por módulos visando unicamente ao alto acoplamento e à baixa coesão, caso a interface entre os módulos dê-se pela passagem de dados.

18. **(CESPE / STF – 2013)** Quanto maior for o número de camadas, menor será o desempenho do software como um todo.

19. **(CESPE / STF – 2013)** Cada camada tem comunicação (interface) com todas as demais camadas, tanto inferiores quanto superiores.

20. **(CESPE / STF – 2013)** Em uma arquitetura em camadas, a camada de persistência é responsável por armazenar dados gerados pelas camadas superiores e pode utilizar um sistema gerenciador de banco de dados para evitar, entre outros aspectos, anomalias de acesso concorrente dos dados e problemas de integridade de dados.

21. **(CESPE / FUB – 2013)** Aplicações cliente-servidor multicamadas são usualmente organizadas em três camadas principais: apresentação, lógica e periférico.

22. **(CESPE / FUB – 2013)** Entre as desvantagens de se executar todas as camadas de uma aplicação cliente-servidor no lado do servidor se destaca a dificuldade de atualização e correção da aplicação.

23. **(CESPE / BACEN – 2013)** MVC (Model-View-Controller) é um modelo de arquitetura de software que separa, de um lado, a representação da informação e, de outro, a interação do usuário com a informação.

24. **(CESPE / TCE-ES – 2013)** No Padrão MVC, as regras do negócio que definem a forma de acesso e modificação dos dados são geridas pelo controlador.

25. **(CESPE / Banco da Amazônia – 2012)** De acordo com o princípio da coesão de classes, cada classe deve representar uma única entidade bem definida no domínio do problema. O grau de coesão diminui com o aumento contínuo de código de manutenção nas classes.

26. **(CESPE / Banco da Amazônia – 2012)** O acoplamento de métodos expressa o fato de que qualquer método deve ser responsável somente por uma tarefa bem definida.

27. **(CESPE / CET – 2011)** No padrão de desenvolvimento modelo-visualização-controlador (MVC), o controlador é o elemento responsável pela interpretação dos dados de entrada e pela manipulação do modelo, de acordo com esses dados.

28. **(CESPE / MEC – 2011)** O modelo MVC pode ser usado para construir a arquitetura do software a partir de três elementos: modelo, visão e controle, sendo definidas no controle as regras de negócio que controlam o comportamento do software a partir de restrições do mundo real.

29. **(CESPE / MEC – 2011)** O controlador, no modelo MVC, realiza a comunicação entre as camadas de visão e modelo.

30. **(CESPE / MEC – 2011)** No MVC, o modelo representa o estado geral do sistema.

31. **(CESPE / MEC – 2011)** Apesar do seu amplo emprego em aplicações web, o MVC deve ser utilizado apenas em interfaces gráficas em função de sua arquitetura de componentes.

32. **(CESPE / MEC – 2011)** No MVC, é o modelo que permite apresentar, de diversas formas diferentes, os dados para o usuário.

33. **(CESPE / MEC – 2011)** O controlador é o responsável pelas regras de negócio e pelos dados de uma aplicação no MVC.

34. **(CESPE / MEC – 2011)** A independência dos componentes é um dos atributos que reflete a qualidade do projeto. O grau de independência pode ser medido a partir dos conceitos de acoplamento e coesão, os quais, idealmente, devem ser alto e baixo, respectivamente.

35. **(CESPE / MEC – 2011)** O termo cliente é usado para designar uma parte distinta de um sistema de computador que gerencia um conjunto de recursos relacionados e apresenta sua funcionalidade para usuários e aplicativos.

36. **(CESPE / MEC – 2011)** A arquitetura cliente/servidor proporciona a sincronização entre duas aplicações: uma aplicação permanece em estado de espera até que outra aplicação efetue uma solicitação de serviço.

37. **(CESPE / MEC – 2011)** A arquitetura cliente/servidor enseja o desenvolvimento de um sistema com, no máximo, duas camadas, quais sejam, cliente e servidor.

38. **(CESPE / ABIN – 2010)** Em sistemas de grande porte, um único requisito pode ser implementado por diversos componentes; cada componente, por sua vez, pode incluir elementos de vários requisitos, o que facilita o seu reúso, pois os componentes implementam, normalmente, uma única abstração do sistema.

39. **(CESPE / INMETRO – 2010)** A coesão e o acoplamento são formas de se avaliar se a segmentação de um sistema em módulos ou em componentes foi eficiente. Acerca da aplicação desses princípios, assinale a opção correta.

a) O baixo acoplamento pode melhorar a manutebilidade dos sistemas, pois ele está associado à criação de módulos como se fossem caixas-pretas.

b) Os componentes ou os módulos devem apresentar baixa coesão e um alto grau de acoplamento.

c) Os componentes ou os módulos devem ser fortemente coesos e fracamente acoplados.

d) Um benefício da alta coesão é permitir realizar a manutenção em um módulo sem se preocupar com os detalhes internos dos demais módulos.

e) A modularização do programa em partes especializadas pode aumentar a qualidade desses componentes, mas pode prejudicar o seu reaproveitamento em outros programas.

40. **(CESPE / BASA – 2010)** Nessa arquitetura (arquitetura multicamadas), quando são consideradas três camadas, a primeira camada deve ser implementada por meio do servidor de aplicação.

41. **(CESPE / BASA – 2010)** Em arquitetura multicamadas, o servidor de aplicação nada mais é do que um programa que fica entre o aplicativo cliente e o sistema de gerenciamento de banco de dados.

42. **(CESPE / BASA – 2010)** Uma desvantagem dessa arquitetura (arquitetura multicamadas) é o aumento na manutenção da aplicação, pois alterações na camada de dados, por exemplo, acarretam mudanças em todas as demais camadas.

43. **(CESPE / BASA – 2010)** Em uma arquitetura cliente-servidor, os clientes compartilham dos recursos gerenciados pelos servidores, os quais também podem, por sua vez, ser clientes de outros servidores.

44. **(CESPE / BASA – 2010)** A Internet baseia-se na arquitetura cliente-servidor, na qual a parte cliente, executada no host local, solicita serviços de um programa aplicativo denominado servidor, que é executado em um host remoto.

45. **(CESPE / BASA – 2010)** A arquitetura cliente-servidor viabiliza o uso simultâneo de diferentes dispositivos computacionais, do seguinte modo: cada um deles realiza a tarefa para a qual é mais capacitado, havendo a possibilidade de uma máquina ser cliente em uma tarefa e servidor em outra.

46. **(CESPE / EMBASA – 2010)** O MVC promove a estrita separação de responsabilidades entre os componentes de uma interface.

47. **(CESPE / EMBASA – 2010)** No MVC, a visão é responsável pela manutenção do estado da aplicação.

48. **(CESPE / EMBASA – 2010)** O modelo no MVC tem como atribuição exibir a parte que é responsável pela manutenção da aplicação para o usuário.

49. **(CESPE / EMBASA – 2010)** O controlador é responsável pela coordenação entre atualizações no modelo e interações com o usuário.

50. **(CESPE / EMBASA – 2010)** Por meio do MVC, é possível o desenvolvimento de aplicações em 3 camadas para a Web.

51. **(CESPE / UNIPAMPA – 2009)** Normalmente, a arquitetura em três camadas conta com as camadas de apresentação, de aplicação e de dados.

52. **(CESPE / UNIPAMPA – 2009)** Em uma arquitetura em três camadas, na camada de aplicação, usualmente está um servidor de banco de dados que gerencia o conjunto de requisições.

53. **(CESPE / UNIPAMPA – 2009)** O uso de middlewares é comum em aplicações de n camadas.

54. **(CESPE / UNIPAMPA – 2009)** Na camada de persistência dos dados em aplicações n camadas, podem ser utilizados o banco de dados orientado a objetos e o banco de dados relacionais.

55. **(CESPE / UNIPAMPA – 2009)** Nas aplicações cliente-servidor, em duas camadas, é simples acessar fontes de dados heterogêneas porque o legado de base de dados não precisa de drivers de conexões diferentes.

56. **(CESPE / ANTAQ – 2009)** Os principais componentes da arquitetura cliente-servidor, que é um modelo de arquitetura para sistemas distribuídos, são o conjunto de servidores que oferecem serviços para outros subsistemas, como servidores de impressão e servidores de arquivos, o

conjunto de clientes que solicitam os serviços oferecidos por servidores, e a rede que permite aos clientes acessarem esses serviços.

57. **(CESPE / BASA – 2009)** Em arquiteturas cliente-servidor multicamadas, na maior parte das aplicações, o browser é adotado como cliente universal.

58. **(CESPE / ANAC – 2009)** O framework modelo visão controlador (MVC – model view controller) é muito utilizado para projeto da GUI (graphical user interface) de programas orientados a objetos.

59. **(CESPE / TCU – 2009)** No MVC (model-view-controller), um padrão recomendado para aplicações interativas, uma aplicação é organizada em três módulos separados. Um para o modelo de aplicação com a representação de dados e lógica do negócio, o segundo com visões que fornecem apresentação dos dados e input do usuário e o terceiro para um controlador que despacha pedidos e controle de fluxo.

60. **(CESPE / ANATEL – 2009)** Uma das vantagens da arquitetura distribuída é o compartilhamento de recursos, que permite que sistemas, aplicativos e dispositivos periféricos, como discos, impressoras, arquivos, estejam associados a diferentes computadores em uma rede. Uma segunda vantagem é a concorrência, uma vez que vários processos podem operar ao mesmo tempo em diferentes computadores na rede. E, por fim, uma terceira vantagem é a proteção, pois o acesso é feito de forma centralizada.

61. **(CESPE / ANTAQ – 2009)** Uma das desvantagens da arquitetura distribuída é sua complexidade, uma vez que é mais difícil compreender as propriedades emergentes dos sistemas que as dos sistemas centralizados.

62. **(CESPE / INMETRO – 2009)** Em uma arquitetura distribuída, middleware é definido como uma camada de software cujo objetivo é mascarar a heterogeneidade e fornecer um modelo de programação conveniente para os programadores de aplicativos. Como exemplos de middlewares é correto citar: Sun RPC, CORBA, RMI Java e DCOM da Microsoft.

63. **(CESPE / IPEA – 2008)** Na arquitetura cliente-servidor com três camadas (three tier), a camada de apresentação, a camada de aplicação e o gerenciamento de dados ocorrem em diferentes máquinas. A camada de apresentação provê a interface do usuário e interage com o usuário, sendo máquinas clientes responsáveis pela sua execução. A camada de aplicação é responsável pela lógica da aplicação, sendo executada em servidores de aplicação. Essa camada pode interagir com um ou mais bancos de dados ou fontes de dados. Finalmente, o gerenciamento de dados ocorre em servidores de banco de dados, que processam as consultas da camada de aplicação e enviam os resultados.

64. **(CESPE / STJ – 2008)** A arquitetura de um sistema de software pode se basear em determinado estilo de arquitetura. Um estilo de arquitetura é um padrão de organização. No estilo cliente-servidor, o sistema é organizado como um conjunto de serviços, servidores e clientes associados

que acessam e usam os serviços. Os principais componentes desse estilo são servidores que oferecem serviços e clientes que solicitam os serviços.

65. **(CESPE / TJ-CE – 2008)** A arquitetura MVC fornece uma maneira de dividir a funcionalidade envolvida na manutenção e apresentação dos dados de uma aplicação web.

66. **(CESPE / TJ-CE – 2008)** A arquitetura MVC foi desenvolvida recentemente para mapear as tarefas complexas de saída do sistema do usuário.

67. **(CESPE / TJ-CE – 2008)** Na arquitetura MVC, um controlador define o comportamento da aplicação, já que este é o responsável por interpretar as ações do usuário e as relaciona com as chamadas do modelo.

68. **(CESPE / TJ-CE – 2008)** A arquitetura MVC não separa a informação de sua apresentação, porque, em sistemas web, informação e apresentação estão na mesma camada.

69. **(CESPE / TJ-CE – 2008)** O desenvolvimento de sistemas web ocorre tipicamente em três camadas. A arquitetura MVC aumenta o escopo do desenvolvimento para, no máximo, quatro camadas, sendo a quarta camada o processamento dos dados do usuário.

70. **(CESPE / IPEA – 2008)** A arquitetura distribuída é caracterizada pelo compartilhamento de recursos computacionais e serviços por meio da comunicação direta e descentralizada entre os sistemas envolvidos e inclui, entre outras coisas, a troca de informações, ciclos de processamento e espaço de armazenamento em disco.

71. **(CESPE / SERPRO – 2008)** Uma arquitetura distribuída permite a divisão de uma mesma tarefa em diferentes processadores em uma mesma CPU. Essa característica aumenta a velocidade de processamento de uma informação.

72. **(CESPE / TCU – 2007)** A arquitetura cliente-servidor tem por motivação sincronizar a execução de dois processos que devem cooperar um com outro. Assim, dadas duas entidades que queiram comunicar-se, uma deve iniciar a comunicação enquanto a outra aguarda pela requisição da entidade que inicia a comunicação.

73. **(CESPE / DATAPREV – 2006)** Uma arquitetura cliente/servidor caracteriza-se pela separação do cliente, o usuário que acessa ou demanda informações, do servidor. Um exemplo típico é um navegador que acessa páginas na Internet. É uma arquitetura que permite o acesso a serviços remotos através de rede de computadores, e que tem como principal deficiência a falta de escalabilidade.

74. **(CESPE / DATAPREV – 2006)** Arquiteturas cliente/servidor podem ser decompostas em mais de duas camadas. Uma configuração muito utilizada é aquela em que os clientes acessam informações por meio de servidores de aplicação, que por sua vez acessam servidores de banco de dados. Este tipo de arquitetura é conhecida como arquitetura em 3 camadas, ou three-tier.

75. **(CESPE / CENSIPAM – 2006)** O padrão MVC organiza um software em modelo, visão e controle. O modelo encapsula as principais funcionalidades e dados. As visões apresentam os dados aos usuários. Uma visão obtém os dados do modelo via funções disponibilizadas pelo modelo; só há uma visão para um modelo. Usuários interagem via controladoras que traduzem os eventos em solicitações ao modelo ou à visão; podem existir várias controladoras associadas a uma mesma visão.

76. **(CESPE / CENSIPAM – 2006)** No padrão MVC, se um usuário modifica o modelo, as visões que dependem desse modelo refletem essas modificações, pois o modelo notifica as visões quando ocorre uma modificação nos seus dados. Portanto, é usado um mecanismo para propagação de modificações que mantém um registro dos componentes que dependem do modelo.

77. **(CESPE / SEAD-PA – 2004)** Em um modelo cliente-servidor em que o processamento é concentrado nos clientes e o armazenamento concentrado no servidor, observa-se uma baixa carga de tráfego na rede.

78. **(CESPE / SERPRO – 2004)** Uma das vantagens da arquitetura cliente-servidor é que parte da carga de processamento é retirada do servidor e colocada nos vários clientes.

79. **(CESPE / STJ – 2004)** As camadas da arquitetura cliente-servidor de três camadas são: camada de interface de usuário, camada de regras de negócio e camada de acesso ao banco de dados.

80. **(CESPE / STJ – 2004)** Na arquitetura cliente-servidor multicamadas, uma alteração na camada de acesso aos dados não afeta a camada de interface de usuário, desde que essas camadas estejam na mesma máquina.

81. **(CESPE / STJ – 2004)** A arquitetura cliente-servidor multicamadas possui a vantagem de que a camada de interface de usuário pode se comunicar diretamente com qualquer outra camada, ou seja, não existe hierarquia entre camadas.

82. **(CESPE / TRE-RS – 2003)** Aplicações com arquitetura cliente-servidor são assimétricas, no sentido de que o cliente e o servidor possuem papéis diferentes na arquitetura de comunicações.

83. **(CESPE / TRE-RS – 2003)** O servidor, por possuir normalmente um hardware mais robusto, sempre deve executar a parte mais pesada do processamento.

84. **(CESPE / TRE-RS – 2003)** Do ponto de vista das funcionalidades de usuários, o servidor não precisa necessariamente de uma interface de usuário.

GABARITO

1. CORRETO
2. LETRA E
3. ERRADO
4. CORRETO
5. CORRETO
6. LETRA B
7. LETRA C
8. CORRETO
9. CORRETO
10. LETRA C
11. ERRADO
12. ERRADO
13. ERRADO
14. ERRADO
15. ERRADA
16. ERRADO
17. ERRADO
18. CORRETO
19. ERRADO
20. CORRETO
21. ERRADO
22. ERRADO
23. CORRETO
24. ERRADO
25. CORRETO
26. ERRADO
27. CORRETO
28. ERRADO
29. CORRETO
30. CORRETO
31. ERRADO
32. ERRADO
33. ERRADO
34. ERRADO
35. ERRADO
36. CORRETO
37. ERRADO
38. ERRADO
39. CORRETO
40. ERRADO
41. CORRETO
42. ERRADO
43. CORRETO
44. CORRETO
45. CORRETO
46. ANULADA
47. ERRADO
48. ERRADO
49. CORRETO
50. CORRETO
51. CORRETO
52. ERRADO
53. CORRETO
54. CORRETO
55. ERRADO
56. CORRETO
57. CORRETO
58. CORRETO
59. CORRETO
60. ERRADO
61. CORRETO
62. CORRETO
63. CORRETO
64. CORRETO
65. CORRETO
66. ERRADO
67. CORRETO
68. ERRADO
69. ERRADO
70. CORRETO
71. ERRADO
72. CORRETO
73. ERRADO
74. CORRETO
75. ERRADO
76. CORRETO
77. CORRETO
78. CORRETO
79. CORRETO
80. ERRADO
81. ERRADO
82. CORRETO
83. ERRADO
84. CORRETO

LISTA DE QUESTÕES – FCC

1. **(FCC / TRF3 – 2019)** Os conceitos alta coesão e baixo acoplamento, utilizados no processo de desenvolvimento de software, são princípios essenciais de:

   a) Abstração.
   b) Modularidade.
   c) Incrementação.
   d) Separação de Interesses.
   e) Generalidade.

2. **(FCC / DPE-AM – 2018)**

Trecho 1:
```
public int pensaoAlimenticia(){
  return Util.getFuncoes.getFuncoesData.calculaPensao(processo);
}
```

Trecho 2:
```
public int pensaoAlimenticia(){
  return Util.calculaPensao(processo);
}
```

Em um sistema Orientado a Objetos bem desenvolvido, os princípios relativos a acoplamento e coesão devem ser respeitados. O código Java apresentando no trecho 1 mostra um exemplo de

   a) baixo acoplamento e o trecho 2 o corrige para alto acoplamento.
   b) alta coesão e o trecho 2 o corrige para baixa coesão.
   c) alto acoplamento e o trecho 2 o corrige para baixo acoplamento.
   d) baixo acoplamento e o trecho 2 mostra um exemplo de baixa coesão.
   e) baixa coesão e o trecho 2 mostra um exemplo de alto acoplamento.

3. **(FCC / TCM-GO – 2015 – Adaptada)** Quanto à Arquitetura em 3 Camadas, é necessário um arranjo que possibilite a reutilização do código e facilite sua manutenção e seu aperfeiçoamento. Deve- se separar Apresentação, Regra de Negócio e Acesso a Dados. Busca-se a decomposição de funcionalidades de forma a permitir aos desenvolvedores concentrarem-se em diferentes partes da aplicação durante a implementação.

4. **(FCC / CNMP – 2015)** Há algumas variantes possíveis de arquitetura a serem utilizadas em um sistema de bancos de dados. Sobre essas variantes, é correto afirmar que:

   a) na arquitetura de 3 camadas, não há uma camada específica para a aplicação.

b) a camada de apresentação da arquitetura de 2 camadas situa-se, usualmente, no servidor de banco de dados.

c) na arquitetura de 3 camadas, a camada de servidor de banco de dados é denominada cliente.

d) a arquitetura de 3 camadas é composta pelas camadas cliente, aplicação e servidor de banco de dados.

e) na arquitetura de 2 camadas não há necessidade de uso de um sistema gerenciador de bancos de dados.

5. **(FCC / TJ-AP – 2014)** Uma arquitetura muito comum em aplicações web é o Modelo Arquitetural 3 Camadas:

I. Camada de Persistência.
II. Camada de Lógica de Negócio.
III. Camada de Apresentação.

Neste modelo, a correta associação dos componentes com as camadas é:

a) I-Servidor de Banco de Dados - II-Servidor de Aplicação - III-Máquina Cliente.
b) I-Servidor Web - II-Servidor Cliente - III-Servidor de Aplicação.
c) I-Servidor Web - II-Servidor de Banco de Dados - III-Máquina Cliente.
d) I-Servidor de Banco de Dados - II-Máquina Cliente - III-Servidor de Aplicação.
e) I-Máquina Cliente - II-Servidor de Banco de Dados - III-Servidor Web.

6. **(FCC / TST – 2012)** Uma arquitetura em camadas:

a) possui apenas 3 camadas, cada uma realizando operações que se tornam progressivamente mais próximas do conjunto de instruções da máquina.

b) tem, na camada mais interior, os componentes que implementam as operações de interface com o usuário.

c) pode ser combinada com uma arquitetura centrada nos dados em muitas aplicações de bancos de dados.

d) tem, como camada intermediária, o depósito de dados, também chamado de repositório ou quadro-negro.

e) tem, na camada mais externa, os componentes que realizam a interface com o sistema operacional.

7. **(FCC / TRF2 – 2012)** São aspectos que podem caracterizar uma arquitetura cliente-servidor, estabelecida logicamente em 4 camadas:

I. A camada Cliente contém um navegador de Internet, caracterizado como cliente universal.

II. A camada de Lógica do Negócio se quebra em duas: camada de Aplicação e camada Web, em que o servidor Web representa a camada de Apresentação.

III. Na camada de Lógica do Negócio, o servidor de aplicação passa a utilizar middleware, para suporte a thin clients (PDA, celulares, smart cards, etc) e soluções baseadas em componentes, tais como, J2EE e .Net.

IV. Se, de um lado, a camada de Aplicação estabelece uma interface com a camada de Dados, do outro o faz com a camada Web e com os thin clients da camada Cliente.

Está correto o que consta em:

a) I e II, apenas.
b) III e IV, apenas.
c) I, II e III, apenas.
d) II, III e IV, apenas.
e) I, II, III e IV.

8. **(FCC / TST – 2012)** No padrão MVC é possível definir grupos de componentes principais: o Model (Modelo), o View (Apresentação) e o Controller (Controle). Deve fazer parte do componente:

a) Controller, uma classe que contém um método com a finalidade de calcular o reajuste de salário dos funcionários.

b) View, uma classe que contém um método para persistir o salário reajustado de um funcionário.

c) Controller, as animações desenvolvidas em Flash.

d) View, as validações necessárias ao sistema, geralmente definidas através de um conjunto de comparações.

e) Model, as classes com métodos conhecidos como setters e getters e que representam tabelas do banco de dados.

9. **(FCC / MPE-AP – 2012)** Em uma Aplicação Web desenvolvida utilizando o design pattern MVC, as páginas HTML e as classes com métodos que acessam o banco de dados e executam instruções SQL são representadas, respectivamente, nos componentes:

a) Presentation e Business.
b) View e Model.
c) Controller e Model.
d) Model e Business.
e) View e Business.

10. **(FCC / MPE-PE – 2012)** O padrão de projeto utilizado em aplicações WEB que permite separar as páginas e classes da aplicação em três grupos (muitas vezes chamados de camadas) conhecidos como Apresentação, Controle e Modelo é denominado de:

a) 3-tier.
b) DAO.
c) MVC.
d) DTO.
e) DBO.

11. **(FCC / TRT-PE – 2012)** O padrão de arquitetura MVC é um modelo de camadas que divide a aplicação em três componentes: Model (modelo), View (visualizador) e Controller (controlador). As funções de cada um destes três componentes são apresentadas abaixo:

I. interpreta eventos de entrada e envia requisições para o modelo de dados; em seguida, processa os dados carregados a partir do modelo e envia para o visualizador.

II. encapsula o acesso aos dados e funções básicas da aplicação, fornecendo ao usuário procedimentos que executam tarefas específicas.

III. exibe para o usuário os dados fornecidos pelo controle e estabelece uma interface para interação entre o usuário e a aplicação.

A associação correta do componente do padrão MVC com sua função está expressa, respectivamente, em

a) (I) Controller; (II) Model; (III) View;
b) (I) Model; (II) Controller; (III) View;
c) (I) View; (II) Model; (III) Controller;
d) (I) Controller; (II) View; (III) Model;
e) (I) Model; (II) View; (III) Controller;

12. **(FCC / TJ-PE – 2012)** Com relação à arquitetura MVC, considere:

I. O MODEL representa os dados da empresa e as regras de negócio que governam o acesso e atualização destes dados.

II. O VIEW acessa os dados da empresa através do MODEL e especifica como esses dados devem ser apresentados. É de responsabilidade do VIEW manter a consistência em sua apresentação, quando o MODEL é alterado.

III. O CONTROLLER traduz as interações do VIEW em ações a serem executadas pelo MODEL. Com base na interação do usuário e no resultado das ações do MODEL, o CONTROLLER responde selecionando uma VIEW adequada.

IV. Permite uma única VIEW para compartilhar o mesmo modelo de dados corporativos em um fluxo de comunicação sequencial.

Está correto o que se afirma em:

a) I, II, III e IV.
b) I, II e III, apenas.
c) II e III, apenas.
d) II, III e IV, apenas.
e) I e II, apenas.

13. (FCC / MPE-PE – 2012) O componente Controller do MVC:

a) Define o comportamento da aplicação, as ações do usuário para atualizar os componentes de dados e seleciona os componentes para exibir respostas de requisições.

b) Envia requisições do usuário para o controlador e recebe dados atualizados dos componentes de acesso a dados.

c) Responde às solicitações de queries e encapsula o estado da aplicação.

d) Notifica os componentes de apresentação das mudanças efetuadas nos dados e expõe a funcionalidade da aplicação.

e) É onde são concentradas todas as regras de negócio da aplicação e o acesso aos dados.

14. (FCC / TRT-MT – 2011) No projeto de arquitetura modelo-visão-controle (MVC), o controlador:

a) renderiza a interface de usuário a partir da visão, o modelo encapsula funcionalidades e objetos de conteúdo e a visão processa e responde a eventos e invoca alterações ao controlador.

b) encapsula funcionalidades e objetos de conteúdo, o modelo processa e responde a eventos e invoca alterações ao controlador e a visão renderiza a interface de usuário a partir do modelo.

c) encapsula funcionalidades e objetos de conteúdo, o modelo renderiza a interface de usuário a partir da visão e a visão processa e responde a eventos e invoca alterações ao controlador.

d) processa e responde a eventos e invoca alterações ao modelo, o modelo encapsula funcionalidades e objetos de conteúdo e a visão renderiza a interface de usuário a partir do modelo.

e) processa e responde a eventos e invoca alterações ao modelo, o modelo renderiza a interface de usuário a partir da visão e a visão encapsula funcionalidades e objetos de conteúdo.

15. **(FCC / TRT-SE – 2010)** No desenvolvimento de sistemas, no âmbito das relações intermodulares entre as classes, diz-se que o programa está bem estruturado quando há:

a) maior coesão e maior acoplamento.
b) menor coesão e maior acoplamento.
c) menor coesão e menor acoplamento.
d) maior coesão e menor acoplamento.
e) apenas coesão ou apenas acoplamento.

16. **(FCC / TCM-PA – 2010)** Extensão natural do conceito de ocultação de informações, que diz: "um módulo deve executar uma única tarefa dentro do procedimento de software, exigindo pouca interação com procedimentos que são executados em outras partes de um programa", é o conceito de:

a) coesão.
b) enfileiramento.
c) acoplamento.
d) visibilidade.
e) recursividade.

17. **(FCC / TRT-SE – 2010)** A arquitetura multicamadas divide-se em três camadas lógicas. São elas:

a) Apresentação, Negócio e Acesso a Dados.
b) Apresentação, Natureza e Acesso a Dados.
c) Apresentação, Negócio e Alteração.
d) Manipulação, Natureza e Acesso a Dados.
e) Manipulação, Negócio e Acesso a Dados.

18. **(FCC / METRÔ-SP – 2010)** A arquitetura multicamadas divide-se em três camadas lógicas. São elas:

a) Apresentação, Negócio e Alteração.
b) Manipulação, Natureza e Acesso a Dados.
c) Manipulação, Negócio e Acesso a Dados.
d) Apresentação, Natureza e Acesso a Dados.
e) Apresentação, Negócio e Acesso a Dados.

19. **(FCC / AL-SP – 2010)** Sobre as camadas do modelo de arquitetura MVC (Model- View-Controller) usado no desenvolvimento web é correto afirmar:

a) Todos os dados e a lógica do negócio para processá-los devem ser representados na camada Controller.

b) A camada Model pode interagir com a camada View para converter as ações do cliente em ações que são compreendidas e executadas na camada Controller.

c) A camada View é a camada responsável por exibir os dados ao usuário. Em todos os casos essa camada somente pode acessar a camada Model por meio da camada Controller.

d) A camada Controller geralmente possui um componente controlador padrão criado para atender a todas as requisições do cliente.

e) Em aplicações web desenvolvidas com Java as servlets são representadas na camada Model.

20. **(FCC / TRT3 – 2009)** Considerando o conjunto de tarefas que se relacionam em um módulo e o espectro de medidas da força funcional relativa dos módulos (coesão), a respectiva sequência, da pior para a melhor, é:

a) sequencial, temporal e lógica.
b) procedimental, coincidental e funcional.
c) temporal, lógica e sequencial.
d) temporal, comunicacional e sequencial.
e) procedimental, funcional e lógica.

21. **(FCC / TJ-SE – 2009)** No modelo de três camadas MVC para web services, o responsável pela apresentação que também recebe os dados de entrada do usuário é a camada:

a) View.
b) Application.
c) Controller.
d) Data.
e) Model.

22. **(FCC / TRT-MA – 2009)** Considere as funções:

I. Seleção do comportamento do modelo.
II. Encapsulamento dos objetos de conteúdo.
III. Requisição das atualizações do modelo.

Na arquitetura Model-View-Control - MVC, essas funções correspondem, respectivamente, a:

a) Model, View e Control.
b) Control, View e Model.
c) View, Model e Control.
d) Control, Model e View.
e) View, Control e Model.

23. **(FCC / TRT-GO – 2008)** Visando obter maior independência funcional, é adequado que o esforço seja direcionado ao projeto de módulos:

a) que não usem estruturas de seleção.
b) cujas tarefas tenham elevada coesão.
c) cujas tarefas tenham coesão procedimental.
d) que não usem estruturas de repetição.
e) cujas tarefas tenham coesão lógica.

24. **(FCC / TRF5 – 2008)** Via de regra as divisões da arquitetura de software em três camadas orientam para níveis que especificam:

a) os casos de uso, a estrutura dos dados e os processos de manutenção.
b) a apresentação, as regras de negócio e os testes.
c) a apresentação, os processos operacionais e a seqüência de execução.
d) a apresentação, os componentes virtuais e a seqüência de execução.
e) a apresentação, as regras de negócio e o armazenamento de dados.

GABARITO

1.	LETRA B	9.	LETRA B	17.	LETRA A
2.	LETRA C	10.	LETRA C	18.	LETRA E
3.	LETRA C	11.	LETRA A	19.	LETRA D
4.	LETRA D	12.	LETRA B	20.	LETRA D
5.	LETRA A	13.	LETRA A	21.	LETRA A
6.	LETRA C	14.	LETRA D	22.	LETRA D
7.	LETRA E	15.	LETRA D	23.	LETRA B
8.	LETRA E	16.	LETRA A	24.	LETRA E

LISTA DE QUESTÕES – DIVERSAS BANCAS

1. **(CS-UFG / SANEGO-GO – 2018)** Dentro dos bons princípios de projeto e construção de software, a Lei de Démeter diz que "um método deve enviar mensagens somente para objetos a que ele tem acesso direto". Essa lei tem como objetivo:

   a) aumentar a coesão.
   b) diminuir o acoplamento.
   c) facilitar a criação de dependência entre as classes.
   d) aumentar a quantidade de casos de teste.

2. **(UFG / SANEAGO – 2017)** O emprego de boas práticas de projeto (design) de software visa resultar em um código:

   a) altamente acoplado e altamente coeso.
   b) altamente acoplado e fracamente coeso.
   c) fracamente acoplado e altamente coeso.
   d) fracamente acoplado e fracamente coeso.

3. **(UFG / SANEAGO – 2017)** Dentro dos padrões arquiteturais de software, a arquitetura Model-View-ViewModel (MVVM) é próxima da arquitetura Model-View-Presenter (MVP), porém diferencia-se desta pelo fato de:

   a) ser desprovida de um componente controlador como existe no Model-View-Controller (MVC).

   b) implementar o padrão de projeto Observer na ligação entre dados (ViewModel) e tela (view).

   c) ligar diretamente as classes de tela (view) e dados (Model) dentro da estrutura do projeto.

   d) vincular a realização de atualizações de tela (view) à atualização de dados (ViewModel).

4. **(IBFC / EBSERH – 2017)** O modelo de três camadas físicas (3-tier), especificado nas alternativas, divide um aplicativo de modo que a lógica de negócio resida no meio das três camadas, foi adaptado como uma arquitetura para as aplicações Web em todas as linguagens de programação maiores. Muitos frameworks de aplicação comerciais e não comerciais foram criados tendo como base a arquitetura:

   a) MVC (Model-View-Controller)
   b) MDB (Model-Data-Business)
   c) UDC (User-Data-Controller)
   d) MDC (Model-Data-Controller)
   e) UVB (User-View-Business).

5. **(CESGRANRIO / CEFET-RJ – 2014)** No contexto da Arquitetura de Sistemas, o MVC (model – view – controller) é um estilo arquitetural:

a) interativo
b) estrutural
c) distribuído
d) adaptável
e) monolítico

6. **(IBFC / TRE-AM – 2014)** Na arquitetura cliente-servidor, além dos dois principais componentes Cliente e o Servidor, existe um terceiro elemento intermediando os dois. Esse componente é chamado tecnicamente de:

a) coreware.
b) middleware.
c) mainware.
d) centerware.

7. **(IBFC / TRE-AM – 2014)** No desenvolvimento de sistemas dentro do conceito da arquitetura cliente-servidor de três camadas, temos as seguintes camadas:

1. Camada de Dados.
2. Camada de Apresentação.
3. Camada de Aplicações.
4. Camada de Negócio.

Estão corretas as afirmativas:

a) somente 1, 2 e 4.
b) somente 2, 3 e 4.
c) somente 1, 3 e 4.
d) somente 1, 2 e 3.

8. **(ESAF / CGU – 2012)** A definição de que um sistema deve ser desenvolvido em três níveis é feita pelo padrão de projeto:

a) MVC (Model View Controller).
b) MVC-Dev (Model Value Constructive Development).
c) TMS (Time Milestones Setting).
d) PMC (Project Main Controller).
e) MCA (Model Classes Assignment).

9. **(ESAF / CVM – 2010)** Modelo MVC significa:

a) Modo-View-Construtor.
b) Modelo-View-Controlador.
c) Modelo-Versão-Case.
d) Módulo-Verificador-Controlador.
e) Medida-Virtual-Concepção.

GABARITO

1. LETRA B
2. LETRA C
3. LETRA B
4. LETRA A
5. LETRA A
6. LETRA B
7. LETRA A
8. LETRA A
9. LETRA B

ARQUITETURA DE MICROSSERVIÇOS

Conceitos Básicos

INCIDÊNCIA EM PROVA: BAIXÍSSIMA

Imagine que você é JK e está planejando cidade de Brasília/DF. Cada prédio é um serviço que essa cidade precisa oferecer. Você tem um prédio para o serviço de transporte, outro para o serviço de saúde, outro para o serviço de educação, e assim por diante. Em vez de construir um único prédio enorme que abriga todos esses serviços, você decide construir vários prédios menores e interconectá-los com ruas.

Cada prédio é um microsserviço que executa uma tarefa específica, como o serviço de transporte, e pode se comunicar com outros microsserviços para fornecer uma experiência mais completa ao cidadão. Assim como na arquitetura de microsserviços, cada microsserviço é autônomo e independente, executando uma tarefa específica e se comunicando com outros microsserviços por meio de APIs ou outros protocolos de comunicação.

Cada microsserviço pode ser implantado e escalado independentemente, tornando o sistema mais flexível e fácil de manter.

A Arquitetura de Microsserviços é uma abordagem para o desenvolvimento de uma aplicação como um conjunto de pequenos serviços, cada um executando em seu próprio processo e se comunicando por meio de mecanismos leves. **Estes serviços são construídos em torno das capacidades do negócio e independentemente implantados por máquinas de implantação totalmente automatizadas**.

Há um mínimo de gerenciamento centralizado destes serviços, que podem ser escritos em diferentes linguagens de programação e utilizarem diferentes tecnologias de armazenamento de dados. **Para começar a entender mais sobre esse estilo arquitetônico, vamos compará-lo com uma arquitetura monolítica**. *O que é isso, professor?* É uma aplicação que é construída como uma única unidade lógica.

Aplicativos Corporativos são, muitas vezes, construídos em três partes principais: a interface de usuário do lado do cliente (Ex: Páginas HTML com CSS e JavaScript), um banco de dados (Ex: Tabelas inseridas em um SGBD Relacional) e um Servidor de Aplicação (Ex: JBoss EAP). A aplicação do servidor irá lidar com Requisições HTTP, executar a lógica de domínio, recuperar e atualizar dados do banco de dados e selecionar as Visões HTML para serem enviadas ao browser.

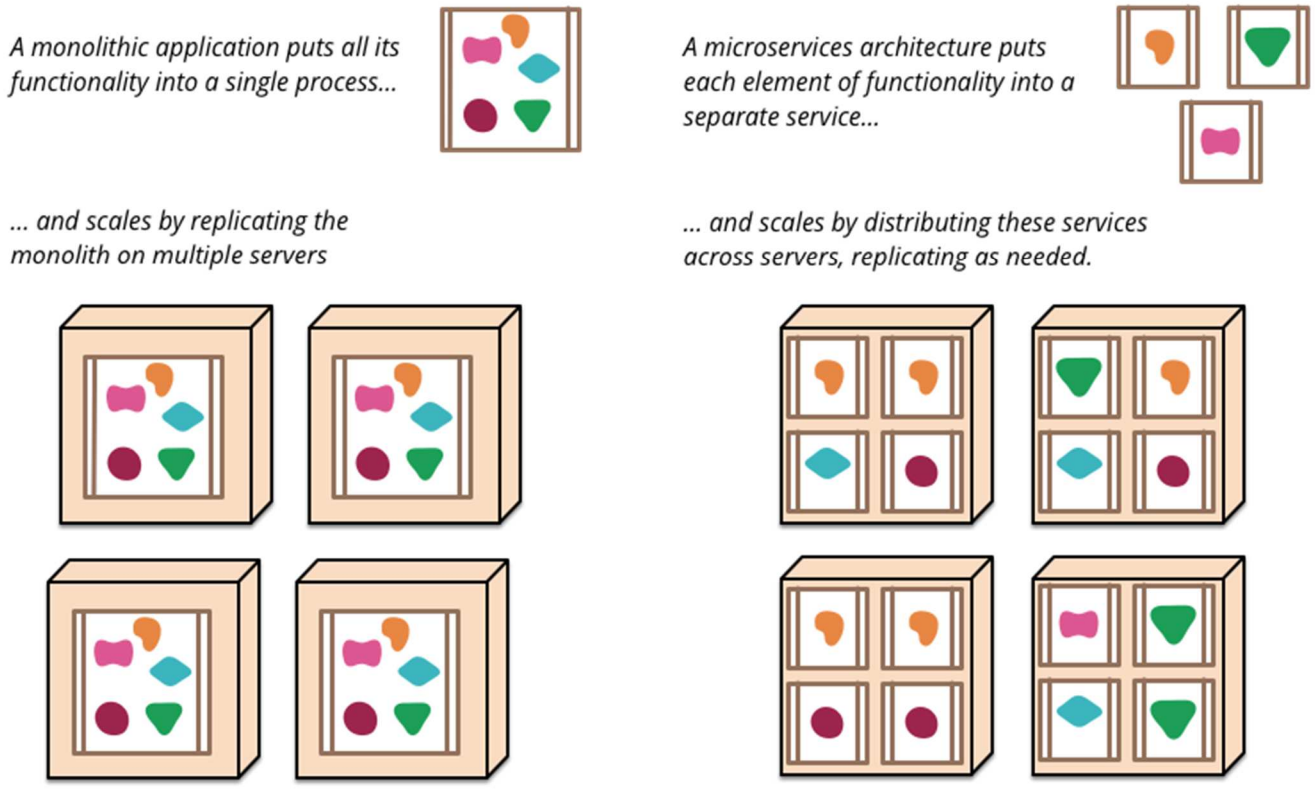
Esta aplicação é monolítica, isto é, existe um único executável lógico. Quaisquer mudanças no sistema envolvem criação e implantação de uma nova versão do aplicativo do lado do servidor. Um servidor monolítico é uma abordagem natural de construir um sistema. Toda a sua lógica para tratar

uma requisição é executada em um único processo, o que lhe permite usar os recursos básicos de sua linguagem para dividir a aplicação em classes, funções e namespaces.

Com algum cuidado, você pode executar e testar o aplicativo no laptop de um desenvolvedor. Dessa forma, pode usar um Deployment Pipeline para garantir que as mudanças sejam devidamente testadas e implantadas em produção. Se desejar, pode também executar diversas instâncias por trás de um balanceador de carga. Logo, aplicações monolíticas podem ser bem-sucedidas, mas cada vez mais as pessoas estão sentindo frustradas com elas.

Ciclos de mudanças são altamente acoplados – uma mudança em uma parte da aplicação requer que um novo build e um novo deploy da aplicação inteira. Com o tempo, muitas vezes é difícil manter uma boa estrutura modular, tornando complicado manter as alterações que deveriam afetar apenas um módulo dentro de outro módulo. Estas frustrações levaram à arquitetura de microsserviços, isto é, construção de aplicações como conjuntos de serviços.

Esses microsserviços são independentemente implantáveis e escaláveis, sendo que cada serviço fornece um escopo bem definido, mesmo permitindo que diferentes serviços sejam escritos em diferentes linguagens de programação. **Eles também podem ser gerenciados por equipas diferentes**. Vamos dar uma olhada na imagem seguinte: observem que que cada cor representa uma funcionalidade.



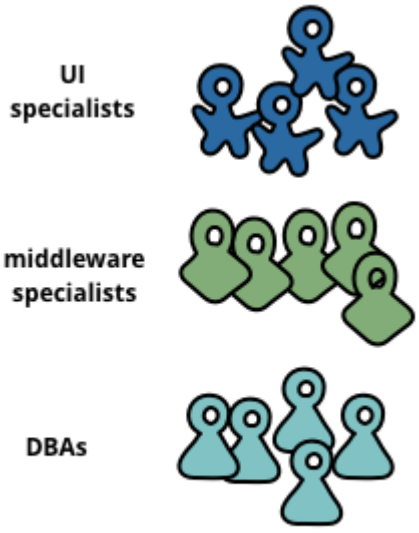
Na arquitetura monolítica, nós modularizamos diversas funcionalidades; na arquitetura de microsserviços, nós modularizamos cada funcionalidade. Dessa forma, podemos compor os

módulos de diferentes formas de acordo com as suas necessidades. Observem que os módulos à direita são todos diferentes. Martin Fowler afirma que não se pode dizer que há uma definição formal para a arquitetura de microsserviços.

No entanto, nós podemos tentar descrever o que vemos como características comuns para arquiteturas que se encaixam nesse perfil. Como acontece com qualquer definição que descreve características comuns, nem todas arquiteturas de microsserviços tem todas as características. Vejamos as principais: **(1) Componentização via serviços** – trata-se de uma divisão do software em serviços (e, não, bibliotecas, etc). *Por que, Diego?*

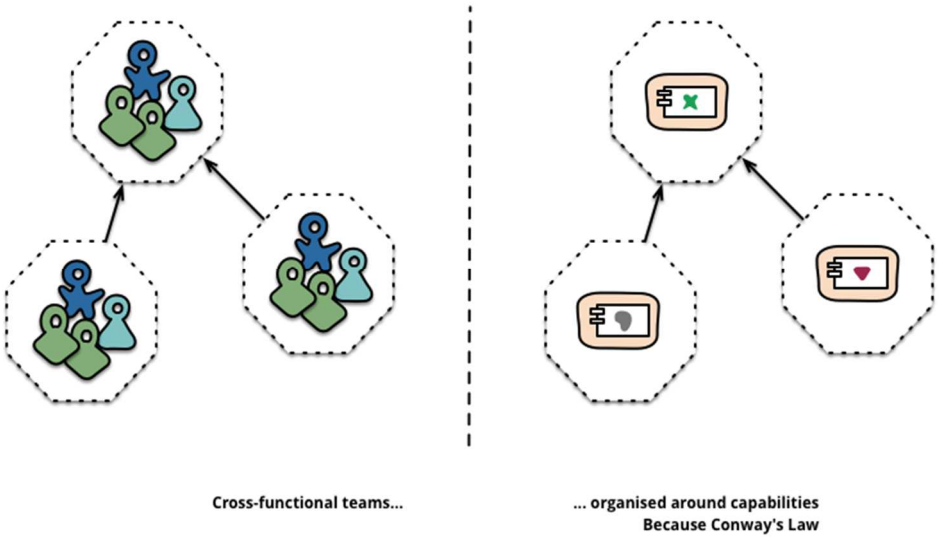
Dentre várias razões, serviços podem ser implantados independentemente. Se uma aplicação consiste em um conjunto de bibliotecas em um único processo, uma mudança em apenas um componente resultará no redeploy da aplicação inteira. Utilizando serviços como componentes, em geral, você precisará fazer o deploy apenas do serviço que foi modificado. *Vocês têm noção do tanto que isso é absurdamente eficiente?* Pois é...

(2) Organização em torno das capacidades do negócio – trata-se da organização dos recursos em torno das capacidades e especialidades da organização e não da tecnologia. Em uma empresa, a área de TI geralmente está organizada de acordo com as especialidades tecnológicas das pessoas. Logo, temos especialistas em front-end, back-end e DBAs – são equipes organizadas de acordo com sua função. A arquitetura de microsserviços funciona de outra forma:



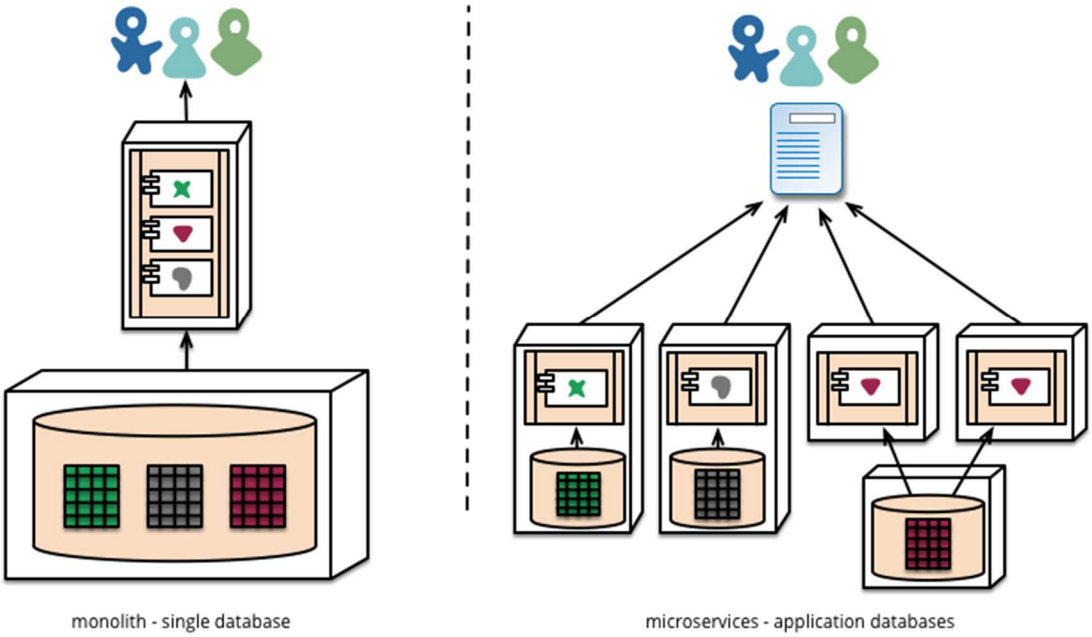
Siloed functional teams...

Ela divide os serviços de acordo com as capacidades de negócio e as equipes de acordo com os serviços. Logo, a equipe não será focada em uma tecnologia, mas em um serviço. Uma equipe pode ser formada por um cara do back-end, dois do front-end e três DBAs. **Equipes multidisciplinares agregam muito valor na construção de um serviço, como mostra a imagem a seguir**.



Produtos e, não, projetos – a equipe é totalmente responsável pelo produto e, não, apenas pelo projeto. A maioria das aplicações de desenvolvimento seguem a forma: crie um projeto, desenvolva um software e termine o projeto. **A partir daí, a equipe se desfaz qualquer correção ou manutenção é feita por outra equipe**. A ideia aqui é que uma equipe deve ser 'dona' do produto por todo seu ciclo de vida.

É a ideia de '*You build, you run it*'. Em outras palavras, a equipe de desenvolvimento tem total responsabilidade sobre o software já em produção. **Isso cria um contato diário dos desenvolvedores sobre como seus softwares se comportam quando já estão em produção e também aumentam o contato com os usuários**. Vejam que há também uma ligação com a ideia de capacidades de negócio:



Entre outras características da arquitetura de microsserviços, há também o gerenciamento de dados descentralizado, na medida em que eu posso ter um serviço de banco de dados para um ou mais serviços, como mostra a imagem anterior; há também a governança descentralizada, isto é, evita-se uma padronização para todos os serviços, como se eles não tivessem particularidades inerentes.

Podemos citar também a automatização da infraestrutura, isto é, entrega, integração e publicações contínuas da aplicação em diversos ambientes automaticamente (Produção, Desenvolvimento, Testes, etc). **Por fim, uma característica importante da arquitetura de microsserviços é que se trata de um design preparado para falhas, isto é, as aplicações precisam ser desenhadas de modo que tolerem falhas de serviços.**

(STF – 2013) A arquitetura de microsserviços considera todo o trabalho como um conjunto de requisições encadeadas executadas que forma uma unidade lógica de processamento no banco de dados. A referida arquitetura deve incluir um conjunto mínimo de operações atômicas para ser executada completamente.

_____

Comentários: a questão trata de uma arquitetura monolítica. (Errado).

(STJ – 2015) A arquitetura de microsserviços, abordagem em que o aplicativo é desenvolvido em uma única unidade contendo pequenos serviços, dependentes entre si, que se comunicam com um ente central denominado biblioteca de componentes, propicia o gerenciamento centralizado desses serviços para automatizar a segurança.

_____

Comentários: a abordagem em que o aplicativo é desenvolvido em uma única unidade é a abordagem monolítica (Errado).

(BANRISUL – 2022) Microsserviços representam a fragmentação de uma API em operações menores, o que facilita a comunicação e otimiza o desenvolvimento de interfaces.

_____

Comentários: microsserviços representam uma aplicação construída como um conjunto de serviços menores e independentes que funcionam juntos. Cada serviço executa uma única tarefa e é responsável por uma parte menor da aplicação. São esses serviços que utilizam uma API para se comunicar. Logo, microsserviços representam a fragmentação de uma aplicação como um conjunto de serviços menores que utilizam uma API (Errado).

ARQUITETURA HEXAGONAL

Conceitos Básicos

ARQUITETURA HEXAGONAL

Também chamada de Ports and Adapters, a arquitetura hexagonal é uma forma de organizar o código em camadas, cada qual com a sua responsabilidade, tendo como objetivo isolar totalmente a lógica da aplicação do mundo externo. Este isolamento é feito por meio de portas e adaptadores, onde as portas são as interfaces que as camadas de baixo nível expõe e adaptadores são as implementações para as interfaces em questão.

Uma das principais ideias da arquitetura hexagonal é separar o código-fonte de negócio do código-fonte de tecnologia. Ainda assim, não apenas isso, também devemos garantir que o lado da tecnologia dependa do lado do negócio para que este possa evoluir sem nenhuma preocupação sobre qual tecnologia é usada para cumprir os objetivos de negócios. E também devemos ser capazes de alterar o código da tecnologia sem causar danos à sua contraparte comercial.

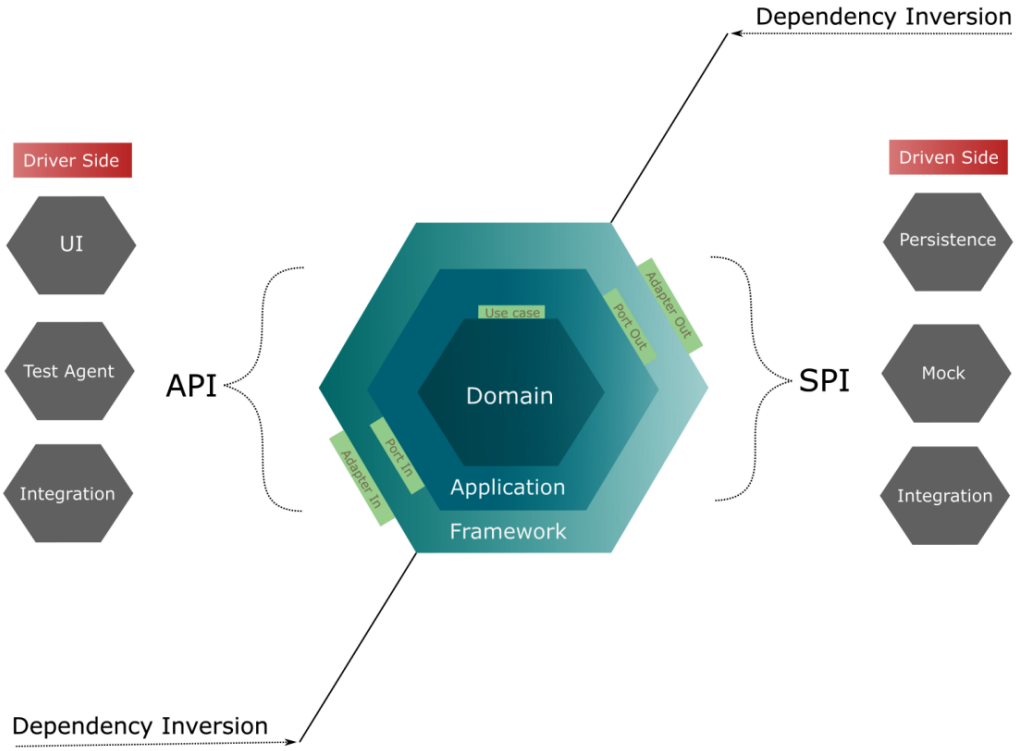
Para atingir esses objetivos, devemos determinar um local onde o código de negócios existirá, isolado e protegido de quaisquer preocupações de tecnologia. Isso dará origem à criação do nosso primeiro hexágono: **Hexágono de Domínio**.

No Hexágono de Domínio, reunimos os elementos responsáveis por descrever os problemas centrais que queremos que nosso software resolva. Entidades e objetos de valor são os principais elementos utilizados nesse hexágono. As entidades representam coisas às quais podemos atribuir uma identidade e os objetos de valor são componentes imutáveis que podemos usar para compor nossas entidades. Tudo isso é bastante baseado no DDD (*Domain Driven Design*).

Também precisamos de maneiras de usar, processar e orquestrar as regras de negócios provenientes do hexágono de domínio – é isso que faz o **Hexágono de Aplicação**. Ele fica entre os lados de negócios e tecnologia, servindo como intermediário para interagir com ambas as partes. O Hexágono de Aplicação utiliza portas e casos de uso para executar suas funções.

Já o **Hexágono de Framework** fornece a interface do mundo externo. Esse é o lugar onde temos a oportunidade de determinar como expor os recursos da aplicação – é aqui que definimos endpoints REST/gRPC, por exemplo. E para consumir coisas de fontes externas, usamos o Hexágono de Framework para especificar os mecanismos que buscam dados de bancos de dados, agentes de mensagens ou qualquer outro sistema.

Na arquitetura hexagonal, materializamos as decisões de tecnologia por meio de adaptadores. O diagrama a seguir fornece uma visão de alto nível da arquitetura:



A Arquitetura Hexagonal – também conhecida como Arquitetura de Portas e Adaptadores – é apresentada na imagem anterior. Vamos tentar entendê-la melhor:

DOMÍNIO	Este é o núcleo da arquitetura e contém toda a lógica de negócios da aplicação. É a parte mais interna e mais importante do sistema, pois representa as regras e conceitos centrais do domínio que o software está modelando.
APLICAÇÃO	Esta camada orquestra as operações do domínio, lidando com a lógica de aplicação (por exemplo, coordenação de casos de uso). Ela não contém lógica de negócios, mas sim a lógica de aplicação que dita a maneira como as interações com o domínio devem ocorrer.
FRAMEWORK	Essa camada lida com aspectos de infraestrutura e funcionalidades de suporte, como frameworks de persistência, integração com APIs externas, e quaisquer outros aspectos técnicos que não fazem parte do domínio de negócios.
API	Representa as portas de entrada do sistema. Esses pontos de entrada são as interfaces pelas quais outros sistemas ou componentes externos (por exemplo, Interface do Usuário, Agentes de Teste, Integração) interagem com o núcleo da aplicação. A ideia é que o mundo exterior interaja com o sistema através dessas portas, sem acessar diretamente o domínio ou a lógica de aplicação.
SPI	Representa as portas de saída do sistema, onde o núcleo do sistema se conecta com outros componentes externos, como serviços de persistência de dados (bancos de dados), mockings, e outras integrações externas. A interação com esses sistemas externos é feita através de interfaces, permitindo a inversão de dependência e facilitando o teste e substituição de implementações.
DRIVER SIDE	Refere-se aos componentes externos que dirigem as operações no sistema. Isso inclui a interface do usuário (UI), agentes de teste, e outros mecanismos de integração que iniciam uma ação no sistema.

DRIVEN SIDE	Refere-se aos componentes externos que são acionados pelo sistema, como a persistência de dados (bancos de dados), mocks, e outros serviços de integração. O sistema dirige essas operações.
INVERSÃO DE DEPENDÊNCIA	A imagem enfatiza a ideia de inversão de dependência, um princípio fundamental no design orientado a objetos. Isso significa que tanto o Driver Side quanto o Driven Side dependem de abstrações (interfaces), e não de implementações concretas. Isso torna o sistema flexível, testável e desacoplado das implementações de infraestrutura.
PORTA DE ENTRADA	São as portas de entrada que permitem a interação com o sistema, frequentemente associadas à API.
PORTA DE SAÍDA	São as portas de saída usadas pelo sistema para interagir com outros serviços, frequentemente associadas ao SPI.

Antes de seguir, vamos falar sobre tipos de portas e adaptadores:

TIPOS DE PORTAS	DESCRIÇÃO
PRIMÁRIAS	Interfaces que expõem as funcionalidades principais da aplicação, permitindo que os usuários ou sistemas externos interajam com ela (ex.: Interface gráfica, CLI, APIs).
SECUNDÁRIAS	Interfaces através das quais a aplicação interage com sistemas externos (ex.: bancos de dados, serviços externos).

TIPOS DE ADAPTADORES	DESCRIÇÃO
PRIMÁRIOS	Implementações das portas primárias que conectam a aplicação a interfaces de usuário ou APIs.
SECUNDÁRIOS	Implementações das portas secundárias que conectam a aplicação a recursos externos, como bancos de dados ou outros serviços.

Hexágono de Domínio

O **Hexágono de Domínio** representa um esforço para entender e modelar um problema do mundo real. Suponha que você esteja em um projeto que precise criar um inventário de rede e topologia para uma empresa de telecomunicações. O principal objetivo deste inventário é fornecer uma visão abrangente de todos os recursos que compõem a rede. Dentre esses recursos, temos roteadores, switches, racks, estantes e outros tipos de equipamentos.

Nosso objetivo aqui é usar o Hexágono de Domínio para modelar o conhecimento necessário para identificar, categorizar e correlacionar esses elementos de rede e topologia em código-fonte, bem como fornecer uma visão lúcida e organizada do inventário desejado. Esse conhecimento deve ser, tanto quanto possível, representado de forma agnóstica em relação à tecnologia.

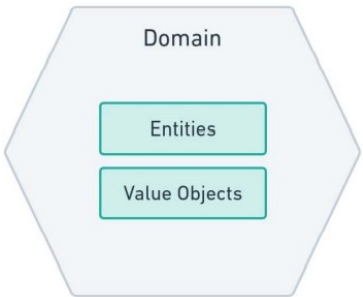
Esta busca não é trivial. Os desenvolvedores envolvidos podem não conhecer empresas de telecomunicações e deixar de lado esse negócio de inventário. É necessário consultar especialistas de domínio ou outros desenvolvedores que já conheçam o problema do domínio. Se nenhum deles estiver disponível, você deve tentar preencher a lacuna de conhecimento consultando livros ou qualquer outro material que ensine sobre o domínio do problema.

Dentro do Hexágono de Domínio, temos entidades correspondentes a dados e regras críticas de negócios. Eles são críticos porque representam um modelo do problema real. Esse modelo leva algum tempo para evoluir e refletir consistentemente o problema que estamos tentando modelar. Esse é o caso de novos projetos de software em que nem os desenvolvedores nem os especialistas do domínio têm uma visão clara do objetivo do sistema em seus estágios iniciais.

Em tais cenários, que são particularmente recorrentes em ambientes de *startups*, é normal e previsível ter um modelo de domínio inicial desajeitado que evolui apenas conforme as ideias de negócios evoluem e são validadas por usuários e especialistas do domínio. É uma situação curiosa onde o modelo de domínio é desconhecido, mesmo para os chamados especialistas de domínio.

Por outro lado, em cenários onde o domínio do problema existe e está claro nas mentes dos especialistas do domínio, se não conseguirmos compreender esse domínio do problema e como ele se traduz em entidades e outros objetos do domínio, como objetos de valor, construiremos nosso software baseado em suposições fracas ou erradas. Isso pode ser considerado um dos motivos pelos quais qualquer software começa simples e, à medida que sua base de código cresce, acumula dívida técnica e se torna mais difícil de manter.

Essas suposições fracas podem levar a um código frágil e inexpressivo que pode inicialmente resolver problemas de negócios, mas não está pronto para acomodar mudanças de maneira coesa. Lembre-se de que o hexágono do domínio é composto por qualquer tipo de categoria de objeto que você considere adequado para representar o domínio do problema. Aqui está uma representação baseada apenas em Entidades e Objetos de Valor:



Hexágono de Aplicação

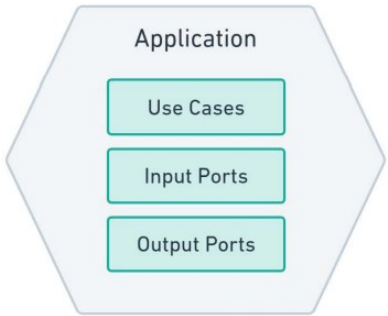
Até agora, discutimos como o hexágono Domínio encapsula regras de negócios com entidades e objetos de valor. Mas há situações em que o software não precisa operar diretamente no nível do Domínio. Algumas operações existem apenas para permitir a automação fornecida pelo software.

Essas operações – embora suportem regras de negócios – não existiriam fora do contexto do software. Estamos falando de operações específicas da aplicação.

O Hexágono de Aplicação é onde lidamos abstratamente com tarefas específicas do aplicativo. Quero dizer abstrato porque ainda não estamos lidando diretamente com questões de tecnologia. Esse hexágono expressa a intenção e os recursos do usuário do software com base nas regras de negócios do Hexágono de Domínio.

Com base na mesma topologia e cenário de rede de inventário descritos anteriormente, suponha que você precise de uma maneira de consultar roteadores do mesmo tipo. Seria necessário algum tratamento de dados para produzir tais resultados. Seu software precisaria capturar alguma entrada do usuário para consultar os tipos de roteador. Você pode querer usar uma regra de negócios específica para validar a entrada do usuário e outra regra de negócios para verificar os dados obtidos de fontes externas.

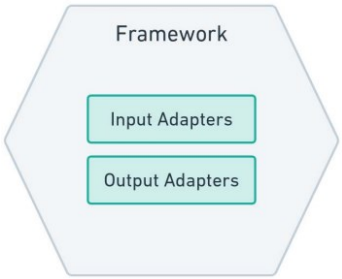
Se nenhuma restrição for violada, seu software fornecerá alguns dados mostrando uma lista de roteadores do mesmo tipo. Podemos agrupar todas essas diferentes tarefas em um caso de uso. O diagrama a seguir descreve a estrutura de alto nível do hexágono do aplicativo com base em casos de uso, portas de entrada e portas de saída:



Hexágono de Framework

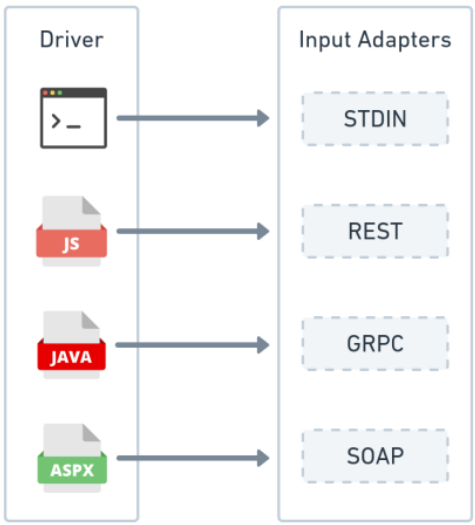
As coisas parecem bem organizadas com nossas regras críticas de negócios restritas ao Hexágono de Domínio, seguido pelo Hexágono de Aplicação que lida com algumas operações específicas da aplicação por meio de casos de uso, portas de entrada e portas de saída. Agora chega o momento em que precisamos decidir quais tecnologias devem ter permissão para se comunicar com nosso software.

Essa comunicação pode ocorrer de duas formas, uma conhecida como *Driver* e outra conhecida como *Driven*. Para o lado do *Driver*, usamos adaptadores de entrada, e para o lado do *Driven*, usamos adaptadores de saída, conforme o diagrama a seguir:



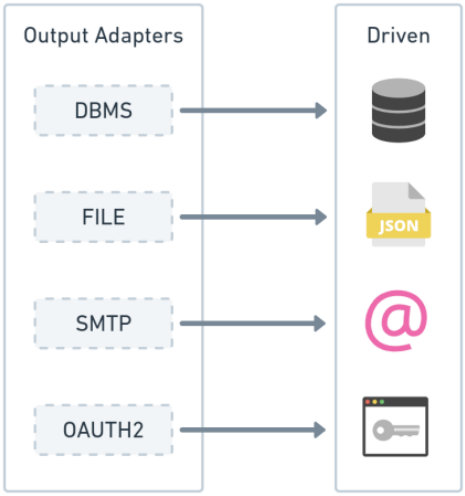
As operações de *Driver* são as que solicitam ações ao software. Pode ser um usuário com um cliente de linha de comando ou um aplicativo front-end em nome do usuário, por exemplo. Pode haver alguns conjuntos de teste verificando a exatidão das coisas expostas pelo seu software. Ou podem ser apenas outros aplicativos em um grande ecossistema que precisam interagir com alguns recursos de software expostos.

Essa comunicação ocorre por meio de uma API (*Application Programming Interface*) construída sobre os adaptadores de entrada. Essa API define como as entidades externas irão interagir com seu sistema e, em seguida, traduzir sua solicitação para o aplicativo do seu domínio. O termo Driver é usado porque essas entidades externas estão dirigindo o comportamento do sistema. Os adaptadores de entrada podem definir os protocolos de comunicação suportados pelo aplicativo, conforme mostrado aqui:



Do outro lado da moeda, nós temos as Operações *Driven*. Essas operações são acionadas a partir da sua aplicação e vão para o mundo externo para obter dados a fim de atender às necessidades do software. Uma Operação *Driven* geralmente ocorre em resposta a alguma condução. Como você pode imaginar, a forma como definimos o lado acionado é através de adaptadores de saída. Esses adaptadores devem estar em conformidade com nossas portas de saída ao implementá-los.

Lembre-se que uma porta de saída nos diz que tipo de dados ela precisa para executar algumas tarefas específicas da aplicação. Cabe ao adaptador de saída descrever como obterá os dados. Vejam um diagrama de adaptadores de saída e Operações *Driven*:



Principais Vantagens

Se você busca um padrão que o ajude a padronizar a forma como o software é desenvolvido em sua empresa ou até mesmo em projetos pessoais, a arquitetura hexagonal pode ser utilizada como base para criar essa padronização, influenciando como classes, pacotes e a estrutura de código-fonte como um todo estão organizados. A arquitetura hexagonal ajuda as organizações a estabelecer os princípios fundamentais nos quais o software é estruturado. Vamos ver as principais vantagens...

Tolerância a Mudanças

As mudanças tecnológicas estão acontecendo em um ritmo acelerado. Novas linguagens de programação e uma infinidade de ferramentas sofisticadas surgem todos os dias. Para vencer a concorrência, muitas vezes, não basta apenas ficar com tecnologias bem estabelecidas e testadas pelo tempo.

O uso de tecnologia de ponta deixa de ser uma escolha e passa a ser uma necessidade, e se o software não estiver preparado para acomodar tais mudanças, a empresa corre o risco de perder dinheiro e tempo em grandes refatorações porque a arquitetura do software não é tolerante a mudanças.

Dessa forma, a natureza das portas e adaptadores da arquitetura hexagonal nos dá uma grande vantagem ao fornecer os princípios arquitetônicos para criar aplicativos prontos para incorporar mudanças tecnológicas com menos atrito.

Manutenibilidade

Se for necessário alterar alguma regra de negócio, você sabe que a única coisa que deve ser alterada é o Hexágono de Domínio. Por outro lado, se precisarmos permitir que um recurso existente seja acionado por um cliente que usa uma determinada tecnologia ou protocolo que ainda não é

suportado pela aplicação, basta criar um novo adaptador, o que só podemos fazer no Hexágono de Framework.

Essa separação de preocupações parece simples, mas quando aplicada como um princípio de arquitetura, ela concede um grau de previsibilidade suficiente para diminuir a sobrecarga mental de compreender as estruturas básicas de software antes de mergulhar profundamente em suas complexidades. O tempo sempre foi um recurso escasso e, se houver uma chance de economizá-lo por meio de uma abordagem de arquitetura que remova algumas barreiras mentais, deve-se ao menos tentar.

Testabilidade

Um dos objetivos finais da arquitetura hexagonal é permitir que os desenvolvedores testem a aplicação quando suas dependências externas não estiverem presentes, como sua interface de usuário e bancos de dados. Isso não significa, entretanto, que essa arquitetura ignore os testes de integração. Em vez disso, permite uma abordagem de integração mais contínua, dando-nos a flexibilidade necessária para testar a parte mais crítica do código, mesmo na ausência de dependências de tecnologia.

Avaliando cada um dos elementos que compõem a arquitetura hexagonal e sabendo das vantagens que ela pode trazer aos nossos projetos, estamos agora munidos dos fundamentos para desenvolver aplicações hexagonais.

(BANRISUL – 2022) Em uma arquitetura hexagonal, as classes de domínio independem das classes de infraestrutura, tecnologias e sistemas externos.

_____

Comentários: a arquitetura hexagonal é uma abordagem de design de software projetada para promover o encapsulamento de dependências. Ela separa os componentes de um sistema em três hexágonos: o mais interno, que contém funcionalidades do domínio; e outros dois, que contém funcionalidades de infraestrutura (aplicação e framework). O objetivo dessa arquitetura é isolar os requisitos de negócio da infraestrutura de software, permitindo assim que o código de domínio possa ser reutilizado em diferentes ambientes, como aplicações web, serviços da web, aplicativos móveis ou outras plataformas. Ela também torna mais fácil a substituição de infraestruturas ou tecnologias, pois não há dependências entre o domínio e a infraestrutura, resultando em um código de domínio de alta qualidade e portabilidade (Correto).

RMI (Remote Method Invocation)

Conceitos Básicos

INCIDÊNCIA EM PROVA: BAIXA

Imagine que você precisa pedir ajuda para um amigo que está muito distante de você, em outra cidade. Para isso, você poderia telefonar para ele e pedir ajuda, certo?

O RMI (Remote Method Invocation) funciona de maneira semelhante, mas ao invés de telefonar, você invoca (chama) um método (função) de um objeto que está em outro computador, em uma rede distante.

Assim como você precisa de um número de telefone para ligar para seu amigo, o RMI utiliza uma referência (identificador) para o objeto remoto, permitindo que você invoque métodos desse objeto como se ele estivesse sendo executado em seu próprio computador. Dessa forma, é possível acessar recursos e executar operações em outros computadores da rede, sem precisar estar fisicamente presente nesses locais.

O RMI é uma tecnologia baseada no protocolo JRMP (Java Remote Method Protocol) utilizada em sistemas distribuídos para permitir que objetos em diferentes máquinas possam interagir uns com os outros como se estivessem na mesma máquina. O funcionamento do RMI se dá através de uma arquitetura cliente/servidor por meio da criação de uma interface remota que define os métodos que serão acessados remotamente.

Essa interface é implementada por uma classe que contém a lógica do método, e é registrada em um servidor RMI, que se torna responsável por gerenciar as conexões e comunicações entre as diferentes máquinas. Quando um cliente solicita a execução de um método remoto, ele invoca um objeto stub, que é uma representação local da interface remota no cliente. Esse stub é responsável por fazer a comunicação com o servidor RMI, passando as informações necessárias para a execução do método remoto, e retornando o resultado para o cliente.

Entre as vantagens do uso de RMI, estão a **simplicidade de implementação**, a **transparência de rede** (ou seja, o cliente não precisa saber em qual máquina está o objeto remoto), a **segurança de dados** (porque o RMI permite o uso de autenticação e criptografia de dados), e a **facilidade de manutenção**.

Por outro lado, as desvantagens do uso de RMI incluem a **dependência de uma conexão de rede estável e confiável** para a execução dos métodos remotos, o que pode impactar na performance e escalabilidade do sistema, e a **necessidade de implementar a interface remota corretamente** para garantir a compatibilidade entre as diferentes versões do sistema.

Uma interface remota é uma interface Java que define um conjunto de métodos que podem ser invocados remotamente por clientes que estão em outras máquinas. Essa interface é

implementada por uma classe que contém a lógica de negócio do método. Para criar uma interface remota, é necessário utilizar a interface **java.rmi.Remote** como superclasse da interface. Além disso, cada método definido na interface deve lançar a exceção **java.rmi.RemoteException**.

Por exemplo, suponha que você deseje criar uma interface remota para um objeto que permite adicionar e recuperar valores de uma lista. A interface poderia ser definida da seguinte forma:

```
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.List;

public interface RemoteList extends Remote {
    public void add(String value) throws RemoteException;
    public List<String> getValues() throws RemoteException;
}
```

Para registrar uma interface remota no servidor RMI, é necessário criar uma instância da classe que implementa a interface e passá-la para o método **java.rmi.Naming.rebind()**. Esse método registra o objeto no servidor RMI e torna-o disponível para ser acessado remotamente pelos clientes. Por exemplo, suponha que você tenha implementado a classe que representa a lista remota e deseja registrar essa lista no servidor RMI com o nome "RemoteList". O código para registrar o objeto seria:

```
import java.rmi.Naming;

public class RemoteListServer {
    public static void main(String[] args) {
        try {
            RemoteList list = new RemoteListImpl();
            Naming.rebind("RemoteList", list);
            System.out.println("RemoteList registered");
        } catch (Exception e) {
            System.err.println("RemoteList registration failed: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

O código acima cria uma instância da classe **RemoteListImpl**, que implementa a interface **RemoteList**, e a registra no servidor RMI com o nome "**RemoteList**".

Por fim, é importante mencionar que os servidores RMI são responsáveis por gerenciar as conexões e comunicações entre as diferentes máquinas. Para iniciar um servidor RMI, é necessário executar o utilitário **rmiregistry**, que cria um registro de nomes para objetos remotos. Depois disso, é possível executar o servidor RMI propriamente dito, que será responsável por fornecer os objetos remotos aos clientes que os solicitarem.

Para criar um cliente RMI, é necessário primeiro procurar o objeto remoto no registro RMI utilizando o método **java.rmi.Naming.lookup()**. Esse método recebe o nome do objeto remoto como parâmetro e retorna uma referência ao objeto remoto. Por exemplo, suponha que você deseje criar

um cliente para a lista remota criada no exemplo anterior, que está registrada no servidor RMI com o nome "RemoteList". O código para procurar a lista no servidor RMI seria:

```
import java.rmi.Naming;

public class RemoteListClient {
    public static void main(String[] args) {
        try {
            RemoteList list = (RemoteList) Naming.lookup("RemoteList");
            System.out.println("RemoteList found");
            // Use a lista remota aqui
        } catch (Exception e) {
            System.err.println("RemoteList lookup failed: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

O código acima procura a lista remota no servidor RMI e armazena uma referência a ela na variável "list". Para invocar métodos remotos em objetos remotos, basta chamar os métodos normalmente como se eles estivessem sendo chamados em um objeto local. No entanto, é importante lembrar que esses métodos podem lançar a exceção **java.rmi.RemoteException**, que deve ser tratada adequadamente.

Por exemplo, suponha que você deseje adicionar um valor à lista remota. O código para invocar o método "**add**" seria o seguinte:

```
try {
    list.add("Value 1");
} catch (RemoteException e) {
    System.err.println("Remote method invocation failed: " + e.getMessage());
    e.printStackTrace();
}
```

Por fim, a comunicação entre objetos remotos é feita através de chamadas de método remoto. **Quando um método é chamado em um objeto remoto, uma mensagem é enviada ao servidor RMI contendo as informações necessárias para a execução do método.** Depois que o método é executado, o resultado é enviado de volta ao cliente na forma de uma resposta.

É importante lembrar que a comunicação entre objetos remotos **pode ser lenta e sujeita a falhas de rede**, por isso é importante tomar medidas para garantir a confiabilidade e desempenho do sistema, como por exemplo o uso de autenticação e criptografia de dados.

Os objetos remotos se comunicam através de chamadas de métodos remotos. Quando um cliente chama um método em um objeto remoto, uma mensagem contendo as informações necessárias para a execução do método é enviada ao servidor RMI, onde o método é executado. Depois que o método é executado, o resultado é enviado de volta ao cliente na forma de uma resposta.

Para passar parâmetros entre objetos remotos, é necessário que esses parâmetros sejam serializáveis. **A serialização é o processo de converter um objeto em uma sequência de bytes que pode ser enviada pela rede.** Todos os objetos que são passados como parâmetros para métodos remotos devem implementar a interface **Serializable**.

Para retornar valores de métodos remotos, é necessário que esses valores também sejam serializáveis. O valor retornado é serializado e enviado de volta ao cliente como parte da resposta da chamada de método remoto. Quando se trata de segurança em RMI, é importante tomar medidas para garantir que apenas clientes autorizados possam se comunicar com o servidor e que os dados transmitidos sejam seguros contra acesso não autorizado. Algumas das medidas de segurança que podem ser tomadas em RMI incluem:

- **Autenticação:** os clientes devem se autenticar para acessar o servidor RMI, geralmente fornecendo um nome de usuário e senha ou algum outro tipo de credencial. O servidor pode então verificar se essas credenciais são válidas antes de permitir que o cliente acesse os objetos remotos.

- **Criptografia:** a comunicação entre o cliente e o servidor pode ser criptografada para proteger os dados transmitidos contra acesso não autorizado. Isso pode ser feito usando protocolos criptográficos como SSL/TLS.

- **Controle de acesso:** o servidor RMI pode ser configurado para permitir ou negar o acesso a determinados objetos remotos com base em um conjunto de regras de controle de acesso. Essas regras podem ser usadas para limitar o acesso a determinados usuários ou grupos de usuários, por exemplo.

- **Verificação de código:** o servidor RMI pode verificar o código dos objetos remotos antes de permitir que eles sejam executados. Isso ajuda a prevenir a execução de código malicioso em sistemas vulneráveis.

- **Monitoramento e auditoria:** é importante monitorar a atividade do servidor RMI e auditar os logs para detectar possíveis atividades maliciosas ou violações de segurança.

(TCE/CE – 2008) Recurso Java que permite que uma thread invoque um método em um objeto remoto (semelhante à RPC) denomina-se:

a) RMI.
b) getPriority().
c) matcher.
d) FocusRequester.
e) RemoteException.

_____

Comentários: o recurso Java que permite que uma thread invoque um método em um objeto remoto é denominado RMI (Remote Method Invocation), que é uma tecnologia de comunicação entre processos em Java. O RMI permite que um programa

Java em execução em uma máquina virtual Java (JVM) chame um método de um objeto em outra JVM em uma máquina diferente, como se o objeto estivesse sendo executado localmente na mesma JVM. Isso torna possível a criação de aplicativos distribuídos em Java (Letra A).

(INMETRO – 2010 – Letra E) A RMI (Remote Method Invocation) envolve a serialização e a desserialização de objetos durante a chamada de procedimentos remotos em sistemas orientados a objeto.

_____

Comentários: quando um objeto é passado como um parâmetro de um método remoto em Java, ele deve ser serializado em um formato que possa ser transmitido pela rede para a JVM remota. Isso significa que o objeto é convertido em uma sequência de bytes que podem ser transmitidos pela rede. A desserialização é o processo de converter essa sequência de bytes de volta em um objeto Java na JVM remota.

A serialização e desserialização são automatizadas pela RMI em Java. Para garantir que os objetos sejam serializados e desserializados corretamente, a classe correspondente deve implementar a interface Serializable. A serialização e desserialização são transparentes para o programador Java, tornando a RMI uma solução conveniente e eficiente para a criação de aplicativos distribuídos orientados a objetos em Java (Correto).

(IF/TO – 2021) Invocação Remota de Métodos (RMI) é uma forma de efetuar processamento distribuído utilizando objetos ou componentes remotos. O RMI abstrai a camada de transporte e permite a comunicação entre sistemas operacionais, máquinas virtuais e/ou hardwares distintos. Acerca do RMI identifique a opção incorreta:

a) O RMI permite a execução de chamadas remotas desenvolvidas em Java.
b) A arquitetura utilizada pelo RMI é a cliente servidor
c) O RMI necessita de compilador da linguagem C++ e ambiente UNIX para poder ser implementado.
d) O RMI utiliza o mecanismo de stub e skeleton para se comunicar com objetos remotos.
e) A utilização do stub e skeleton é transparente ao desenvolvedor.

_____

Comentários: (a) Correto. Trata-se de uma tecnologia que permite a execução de chamadas remotas desenvolvidas em Java; (b) Correto. Ele é baseado em uma arquitetura cliente-servidor, em que o cliente pode chamar métodos em objetos remotos que estão sendo executados em um servidor; (c) Errado. Trata-se de uma tecnologia Java que não requer compilador da linguagem C++ e nem ambiente UNIX para ser implementado. Na verdade, o RMI é compatível com diversos sistemas operacionais e abstrai a camada de transporte, permitindo a comunicação entre diferentes plataformas; (d) Correto. O RMI utiliza o mecanismo de stub e skeleton para se comunicar com objetos remotos. O stub é um objeto local que representa o objeto remoto e é responsável por serializar as chamadas de método, transmiti-las pela rede e desserializar as respostas. O skeleton é um objeto remoto que é responsável por receber as chamadas de método serializadas, desserializá-las e encaminhá-las para o objeto remoto real; (e) Correto. O desenvolvedor não precisa se preocupar com o mecanismo de stub e skeleton, pois o RMI provê essa infraestrutura de forma transparente para o desenvolvedor, permitindo que a comunicação com objetos remotos seja realizada de forma simples e intuitiva (Letra C).

(SERPRO – 2013) RMI (Remote Method Invocation) é o protocolo de programação que, utilizando WAP (Wireless Application Protocol), permite a construção de interface homem-máquina em dispositivos móveis.

_____

Comentários: RMI e WAP são tecnologias diferentes que têm finalidades diferentes. O RMI é uma tecnologia para objetos distribuídos em Java, enquanto o WAP é um protocolo de comunicação para dispositivos móveis que permite o acesso à Internet e a serviços baseados na Web (Errado).

(SUFRAMA – 2014) A tecnologia Java RMI (Remote Method Invocation), embasada no CORBA (Common Object Request Brokerage Architecture), é utilizada para a incorporação de objetos distribuídos, ou seja, objetos que interagem em diferentes plataformas por meio de uma rede.

_____

Comentários: RMI é baseado em JRMP (Java Remote Method Protocol) e, não, CORBA (Errado).

ARQUITETURA ORIENTADA A EVENTOS

Conceitos Básicos

INCIDÊNCIA EM PROVA: BAIXA

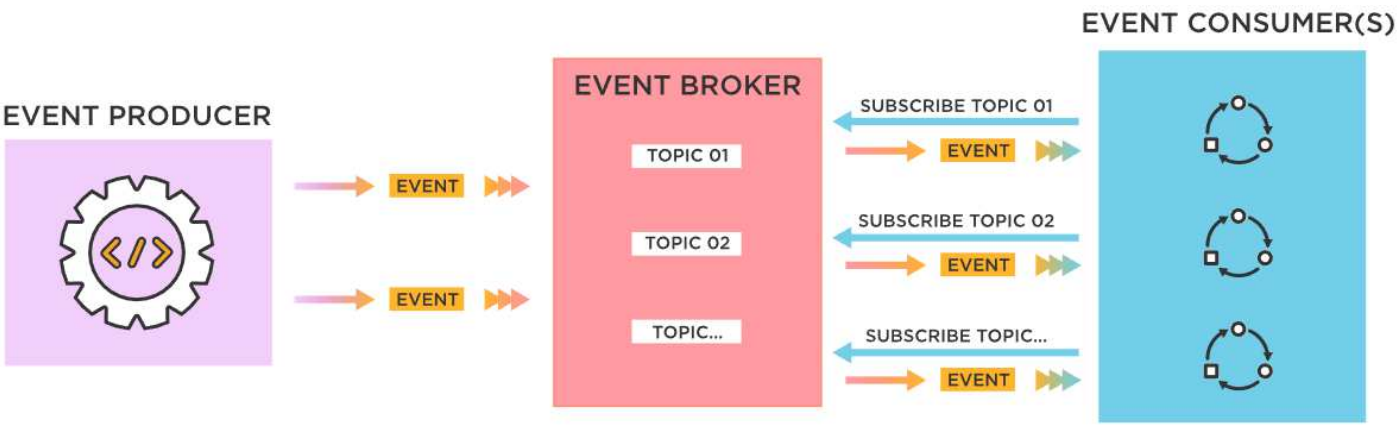
Antes de falarmos sobre a arquitetura orientada a eventos, vamos ver primeiro o que é um evento: uma primeira definição afirma que um evento é **qualquer ocorrência ou mudança de estado significativa no software ou hardware de um sistema**; uma segunda definição afirma que um evento representa uma **ação ou ocorrência significativa que ocorreu em um sistema ou ambiente e precisam ser monitorados ou processados**.

Não confundam a notificação de um evento com o evento em si: uma notificação de evento é uma mensagem enviada pelo sistema para avisar outra parte do mesmo sistema que algo ocorreu, mas a notificação não é o evento. Exemplos de eventos podem incluir a criação de um novo usuário, a atualização de um registro de banco de dados, a chegada de um pedido, a queda de uma conexão de rede, entre outros – lembrando que a origem de um evento pode ser interna ou externa.

E o que seria uma arquitetura orientada a eventos? Trata-se de um modelo de arquitetura de software para o projeto de aplicações. **Em um sistema orientado a eventos, os componentes de captura, comunicação, processamento e persistência de eventos formam a estrutura básica da solução.** Podemos dizer que a arquitetura orientada a eventos é uma abordagem arquitetural que se concentra na troca de eventos entre diferentes componentes.

Nessa arquitetura, os eventos são o centro da comunicação entre diferentes serviços ou sistemas, permitindo a comunicação assíncrona e a escalabilidade horizontal. Ela é uma alternativa às arquiteturas mais tradicionais (Ex: SOA e Arquitetura Monolítica). Ao contrário dessas abordagens, ela permite uma maior modularidade e flexibilidade, pois os serviços podem ser adicionados ou removidos facilmente sem afetar o sistema como um todo.

Uma arquitetura orientada a eventos geralmente composta por três componentes principais: produtores de eventos, consumidores de eventos e broker de eventos:



COMPONENTES	DESCRIÇÃO
PRODUTOR DE EVENTOS	É o componente responsável por gerar e enviar eventos para o broker de eventos. Os produtores podem ser sensores, dispositivos IoT, aplicativos ou qualquer outro componente do sistema que possa gerar eventos.
CONSUMIDOR DE EVENTOS	É o componente que recebe e processa os eventos enviados pelo broker de eventos. Os consumidores podem ser aplicativos, serviços ou outros componentes do sistema que precisam ser notificados sobre mudanças ou eventos que ocorrem em outros componentes.
BROKER DE EVENTOS	É o componente central que gerencia a comunicação entre produtores e consumidores de eventos. O broker recebe os eventos dos produtores e os encaminha para os consumidores apropriados. Ele também pode filtrar e transformar eventos antes de entregá-los aos consumidores.

O produtor de eventos é responsável por gerar e enviar eventos para o broker de eventos. O consumidor de eventos é responsável por receber e processar eventos do broker de eventos. O broker de eventos atua como intermediário entre o produtor e o consumidor de eventos, recebendo eventos do produtor e encaminhando-os para os consumidores correspondentes. **Os serviços de eventos se comunicam com outros serviços na arquitetura por meio do broker de eventos.**

Quando um serviço de evento produz um evento, ele o envia para o broker de eventos, que o encaminha para os serviços que se inscreveram naquele evento específico. O serviço de evento consumidor pode então processar o evento de acordo com suas necessidades e enviar a resposta de volta ao broker de eventos, que a encaminha de volta ao serviço produtor, se necessário. *Entendido?* Bacana!

> *Imagine uma grande feira de negócios, onde várias empresas estão expondo seus produtos e serviços para potenciais clientes. Cada empresa é um sistema independente, com sua própria equipe de vendas, produtos e materiais de marketing. Porém, para atrair mais clientes e fazer negócios, elas precisam se comunicar e interagir umas com as outras.*
>
> *Nesse contexto, podemos imaginar que a arquitetura orientada a eventos funciona como um "correio elegante" entre as empresas. Cada empresa é um "produtor" que cria um cartão com informações sobre o que está oferecendo, e coloca esse cartão em uma caixa central (o "broker"). As outras empresas são os "consumidores" que estão interessados em receber esses cartões para ver se há algo que possam aproveitar.*
>
> *Assim, cada vez que uma empresa cria um novo cartão e o coloca na caixa central, o "broker" envia uma cópia desse cartão para todas as outras empresas que estão interessadas em recebê-lo. Dessa forma, as empresas não precisam se comunicar diretamente umas com as outras, nem precisam saber exatamente quem está interessado em seus produtos e serviços. Elas simplesmente produzem informações (eventos) que são distribuídas para todos os consumidores que possam estar interessados, de forma assíncrona e sem acoplamento direto entre as partes.*

Galera, a arquitetura orientada a serviços pode ser utilizada em diferentes setores de forma bastante eficiente como podemos ver a seguir:

- **Setor Financeiro:** a arquitetura orientada a eventos é amplamente utilizada no setor financeiro para construir sistemas de negociação de alta frequência e processamento de dados em tempo real. Esses sistemas usam eventos para capturar dados de mercado e tomadas de decisão em tempo real, permitindo que as empresas tomem decisões comerciais rapidamente com base nas informações mais recentes.

- **Setor de Saúde:** a arquitetura orientada a eventos é usada no setor de saúde para construir sistemas de monitoramento e análise de dados em tempo real. Os dados de saúde podem ser gerados a partir de diferentes fontes, como sensores de monitoramento, registros médicos eletrônicos e outros dispositivos médicos. A arquitetura orientada a eventos ajuda a gerenciar esses dados em tempo real e fornece informações valiosas aos profissionais de saúde.

- **Comércio eletrônico:** a arquitetura orientada a eventos é usada em comércio eletrônico para construir sistemas de gerenciamento de pedidos e estoque em tempo real. Quando um cliente faz um pedido, um evento é gerado e enviado para diferentes sistemas que precisam saber sobre o pedido. Esses sistemas podem atualizar o estoque, enviar informações de envio e atualizar o sistema de contabilidade, tudo em tempo real.

- **Internet das Coisas (IoT):** a arquitetura orientada a eventos é usada em sistemas IoT para gerenciar dados de sensores e atuadores em tempo real. Os sensores podem gerar eventos que disparam ações em outros dispositivos, permitindo que os sistemas IoT funcionem de forma autônoma e eficiente.

- **Sistemas de recomendação:** a arquitetura orientada a eventos é usada em sistemas de recomendação para capturar informações sobre as preferências do usuário e gerar recomendações personalizadas. Quando um usuário visualiza um produto ou faz uma compra, um evento é gerado e usado para alimentar os algoritmos de recomendação.

Vantagens e Desvantagens

INCIDÊNCIA EM PROVA: BAIXA

A arquitetura orientada a eventos oferece uma série de vantagens e desvantagens em relação a outras abordagens de arquitetura de software. Vejamos:

VANTAGENS	DESCRIÇÃO
ESCALABILIDADE	A arquitetura orientada a eventos permite que um sistema seja dimensionado facilmente para atender a demandas crescentes, adicionando ou removendo produtores ou consumidores de eventos conforme necessário.
DESEMPENHO	A arquitetura orientada a eventos permite que os sistemas processem eventos de forma assíncrona, o que pode melhorar significativamente o desempenho do sistema.
FLEXIBILIDADE	Os sistemas orientados a eventos são altamente flexíveis, permitindo que diferentes componentes sejam adicionados, removidos ou atualizados sem interromper o sistema como um todo.
RESILIÊNCIA	A arquitetura orientada a eventos é altamente tolerante a falhas, pois os eventos podem ser armazenados em buffer e reprocessados quando ocorrer uma falha.
ADAPTABILIDADE	A arquitetura orientada a eventos permite que um sistema se adapte rapidamente a novos requisitos de negócios, adicionando ou removendo consumidores de eventos conforme necessário.

DESVANTAGENS	DESCRIÇÃO
COMPLEXIDADE	A arquitetura orientada a eventos pode ser mais complexa do que outras abordagens de arquitetura, pois requer a implementação de um broker de eventos e a configuração de consumidores e produtores de eventos.
LATÊNCIA	A comunicação assíncrona entre produtores e consumidores de eventos pode causar uma certa latência no processamento de eventos.
GERENCIAMENTO DE ESTADO	A arquitetura orientada a eventos pode tornar mais difícil o gerenciamento do estado do sistema, uma vez que cada componente é notificado de mudanças em outros componentes por meio de eventos.
DIFICULDADE DE DEPURAÇÃO	O rastreamento de problemas em sistemas orientados a eventos pode ser mais difícil do que em sistemas mais tradicionais, pois os eventos podem ser gerados e processados em diferentes partes do sistema.

Galera, a arquitetura orientada a eventos é como um lego, no sentido de poder adicionar ou remover peças facilmente para aumentar ou diminuir a capacidade do sistema – além de ser super flexível e adaptável. Por outro lado, ela é bem mais complexa e, em casos de necessidade de depurar o código, é como um quebra-cabeças: **é bem mais difícil de encontrar um problema ou um erro do que em outras arquiteturas.**

Principais Modelos

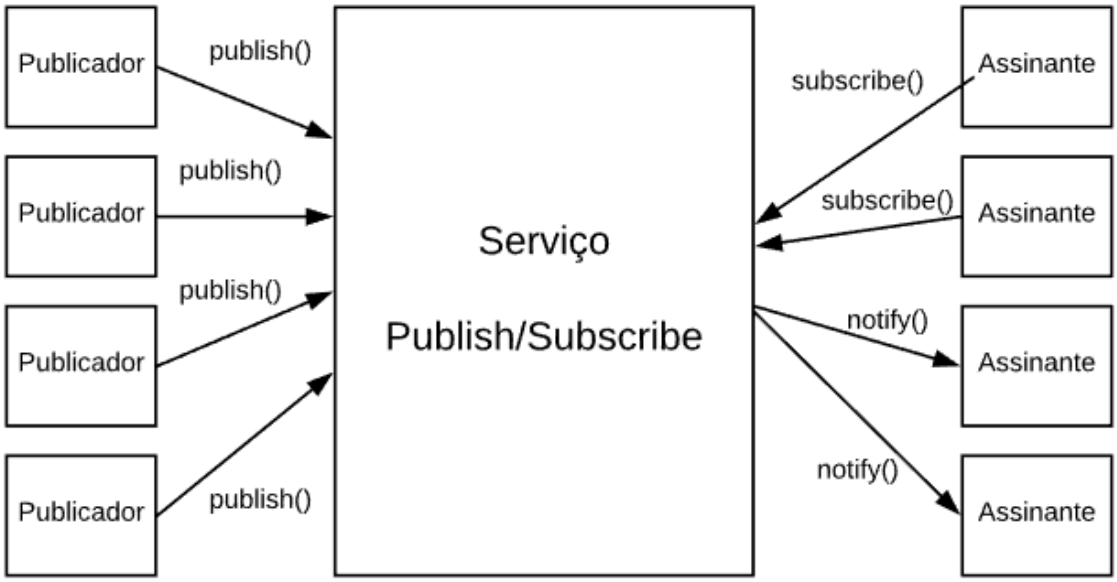
INCIDÊNCIA EM PROVA: BAIXA

A arquitetura orientada a eventos pode ser baseada em um modelo de publicação/subscrição (pub/sub) ou de transmissão de eventos. Vejamos...

Modelo de Pub/Sub

A arquitetura orientada a eventos pode ser baseada em um modelo de publicação/subscrição (pub/sub) ou de transmissão de eventos. De acordo com *Marco Tulio Valente* em *Engenharia de Software Moderna*, em arquiteturas publish/subscribe, as mensagens são chamadas de eventos. Os componentes da arquitetura são chamados de publicadores (publishers) e assinantes (subscribers) de eventos.

Publicadores produzem eventos e os publicam no serviço de publish/subscribe, que normalmente executa em uma máquina separada. Assinantes devem previamente assinar eventos de seu interesse. Quando um evento é publicado, os seus assinantes são notificados. **Assim como ocorre quando se usa filas de mensagens, arquiteturas publish/subscribe também oferecem desacoplamento no espaço e no tempo.**



No entanto, existem duas diferenças principais entre publish/subscribe e sistemas baseados em filas de mensagens:

- **Em publish/subscribe, um evento gera notificações em todos os seus assinantes.** Por outro lado, em filas de mensagens, as mensagens são sempre consumidas — isto é, retiradas da fila — por um único servidor. Portanto, em publish/subscribe temos um estilo de comunicação de 1 para n, também conhecido como comunicação em grupo. Já em filas de mensagens, a comunicação é 1 para 1, também chamada de comunicação ponto-a-ponto.

- **Em publish/subscribe, os assinantes são notificados assincronamente.** Primeiro, eles assinam certos eventos e, então, continuam seu processamento. Quando o evento de interesse ocorre, eles são notificados por meio da execução de um determinado método. Por outro lado, quando se usa uma fila de mensagens, os servidores — isto é, os consumidores das mensagens — têm que puxar (pull) as mensagens da fila.

Em alguns sistemas publish/subscribe, eventos são organizados em tópicos, que funcionam como categorias de eventos. Quando um publicador produz um evento, ele deve informar seu tópico. Assim, clientes não precisam assinar todos os eventos que ocorrem no sistema, mas apenas eventos de um determinado tópico. **Arquiteturas publish/subscribe são, às vezes, chamadas de arquiteturas orientadas a eventos.**

> *O serviço de publish/subscribe, às vezes, é chamado também de broker de eventos, pois ele funciona como um barramento por onde devem trafegar todos os eventos.*

Vamos fazer uma analogia: imagine um jornal de grande publicação. O publicador seria o jornalista que escreve a notícia e envia para a redação (broker), que é como um intermediário que cuida da distribuição das notícias do jornal. Em vez de ligar para cada leitor individualmente, o jornal é distribuído para vários pontos de venda (canais) e quem estiver interessado na notícia pode comprá-lo e lê-lo.

No geral, o modelo pub/sub é uma maneira eficiente de enviar informações para muitos interessados ao mesmo tempo, sem que o publicador precise saber quem são eles ou como acessar suas informações diretamente. **É como enviar uma mensagem de uma só vez para vários destinatários que estão interessados em receber suas novidades.** Vamos ver o segundo modelo de arquitetura orientada a eventos...

Modelo de Transmissão de Eventos

No modelo de transmissão, os eventos são gravados em um log. Os consumidores não precisam se inscrever em uma transmissão de evento. Na verdade, eles podem ler a partir de qualquer parte da transmissão e ingressar nela a qualquer momento. **Neste modelo, os produtores de eventos enviam informações para um broker, que é um intermediário que armazena e distribui essas informações para os canais de destino.**

Os consumidores se conectam a um ou mais canais para receber as informações que desejam. É como se os produtores estivessem enviando mensagens para uma estação de rádio ou TV, e os consumidores estivessem sintonizando essa estação para receber as informações. **Há três tipos principais de transmissão de eventos: processamento do fluxo do evento, processamento de eventos simples e processamento de evento complexo.**

TIPOS	DESCRIÇÃO

PROCESSAMENTO DO FLUXO DO EVENTO	Usa uma plataforma de transmissão, como o Apache Kafka, para ingerir e processar eventos ou transformar seu fluxo. Esse método pode ser usado para detectar padrões significativos em fluxos de eventos.
PROCESSAMENTO DE EVENTO SIMPLES	É quando um evento aciona imediatamente uma ação no consumidor.
PROCESSAMENTO DE EVENTO COMPLEXO	Requer que um consumidor processe uma série de eventos para detectar padrões.

Vários designs de aplicações modernos são direcionados para eventos, como os frameworks de mecanismo do cliente, que devem utilizar dados do cliente em tempo real. É possível criar aplicações orientadas a eventos em qualquer linguagem porque esse tipo de arquitetura é uma abordagem de programação. Por meio dela, você usa uma quantidade mínima de acoplamentos, o que a torna uma boa opção para arquiteturas de aplicações distribuídas modernas.

(SERPRO – 2013) Na arquitetura distribuída, os sistemas orientados a eventos possuem processos fortemente acoplados.

_____

Comentários: sistemas orientados a eventos em arquiteturas distribuídas geralmente são projetados para serem fracamente acoplados. Isso significa que cada componente ou serviço é independente e pode funcionar de forma autônoma, sem depender de outros componentes (Errado).

Em uma arquitetura orientada a eventos, os serviços se comunicam assincronamente por meio de eventos, e não há necessidade de conexão direta entre os serviços. Cada serviço se concentra em um conjunto específico de eventos que ele produz ou consome e pode se inscrever ou cancelar a inscrição em eventos específicos sem afetar outros serviços. **Isso permite que os serviços sejam adicionados, removidos ou substituídos sem afetar a funcionalidade do sistema como um todo.**

A arquitetura orientada a eventos distribuída também é escalável horizontalmente, o que significa que os serviços podem ser replicados e dimensionados conforme necessário. Isso é especialmente útil em sistemas de grande escala, onde a demanda pode variar ao longo do tempo. A replicação de serviços garante que o sistema possa lidar com picos de tráfego sem afetar o desempenho. *Vamos ver outra questão parecida?*

(SERPRO – 2013) Na arquitetura distribuída, a definição de um modelo arquitetônico com base em eventos é caracterizada por processos fracamente acoplados, que precisam se referir explicitamente uns aos outros. Esses processos são denominados desacoplados no espaço ou referencialmente desacoplados.

_____

Comentários: os processos são fracamente acoplados, logo não precisam se referir explicitamente uns aos outros (Errado).

Essa abordagem de desacoplamento permite que os componentes do sistema funcionem independentemente uns dos outros e sejam escalados de forma independente, sem afetar a funcionalidade geral do sistema. **Cada componente se concentra em um conjunto específico de eventos que ele produz ou consome e pode se inscrever ou cancelar a inscrição em eventos específicos sem afetar outros componentes.**

Essa separação permite que diferentes equipes trabalhem em diferentes componentes do sistema sem interferir no trabalho de outras equipes. Cada equipe pode se concentrar em seu próprio conjunto de eventos e não precisa se preocupar com os detalhes de como os outros componentes funcionam. Além disso, como os eventos são propagados de forma assíncrona, os componentes podem funcionar em diferentes velocidades e serem escalados independentemente um do outro.

O desacoplamento no espaço também significa que os componentes podem ser distribuídos geograficamente, permitindo que o sistema funcione em uma ampla área geográfica. Cada componente pode ser executado em um servidor diferente ou em diferentes data centers, e ainda assim, o sistema como um todo funcionará de maneira coesa e consistente. Acho que é isso que tínhamos para ver sobre arquitetura orientada a eventos...

(BNB – 2022) Em uma arquitetura orientada a eventos, o produtor identifica um evento que chega ao consumidor de forma assíncrona.

_____

Comentários: questão bastante esquisita - o produtor não tem conhecimento direto sobre quais consumidores estão recebendo os eventos, nem quando esses eventos são consumidos. Eu discordo do gabarito definitivo (Correto).

ESSA LEI TODO MUNDO CONHECE:
PIRATARIA É CRIME.

Mas é sempre bom revisar o porquê e como você pode ser prejudicado com essa prática.

1. 

Professor investe seu tempo para elaborar os cursos e o site os coloca à venda.

2. 

Pirata divulga ilicitamente (grupos de rateio), utilizando-se do anonimato, nomes falsos ou laranjas (geralmente o pirata se anuncia como formador de "grupos solidários" de rateio que não visam lucro).

3. 

Pirata cria alunos fake praticando falsidade ideológica, comprando cursos do site em nome de pessoas aleatórias (usando nome, CPF, endereço e telefone de terceiros sem autorização).

4. 

Pirata compra, muitas vezes, clonando cartões de crédito (por vezes o sistema anti-fraude não consegue identificar o golpe a tempo).

5. 

Pirata fere os Termos de Uso, adultera as aulas e retira a identificação dos arquivos PDF (justamente porque a atividade é ilegal e ele não quer que seus fakes sejam identificados).

6. 

Pirata revende as aulas protegidas por direitos autorais, praticando concorrência desleal e em flagrante desrespeito à Lei de Direitos Autorais (Lei 9.610/98).

7. 

Concurseiro(a) desinformado participa de rateio, achando que nada disso está acontecendo e esperando se tornar servidor público para exigir o cumprimento das leis.

8. 

O professor que elaborou o curso não ganha nada, o site não recebe nada, e a pessoa que praticou todos os ilícitos anteriores (pirata) fica com o lucro.

Deixando de lado esse mar de sujeira, aproveitamos para agradecer a todos que adquirem os cursos honestamente e permitem que o site continue existindo.