Estratégia Concursos

Aula 00

DPE-RO (Analista -Redes e Comunicação de Dados) Desenvolvimento de software - 2021 (Pós-Edital)

Autor:
Diego Carvalho, Equipe Informática e TI, Thiago Rodrigues Cavalcanti

12 de Outubro de 2021

Índice

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.

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 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.

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.

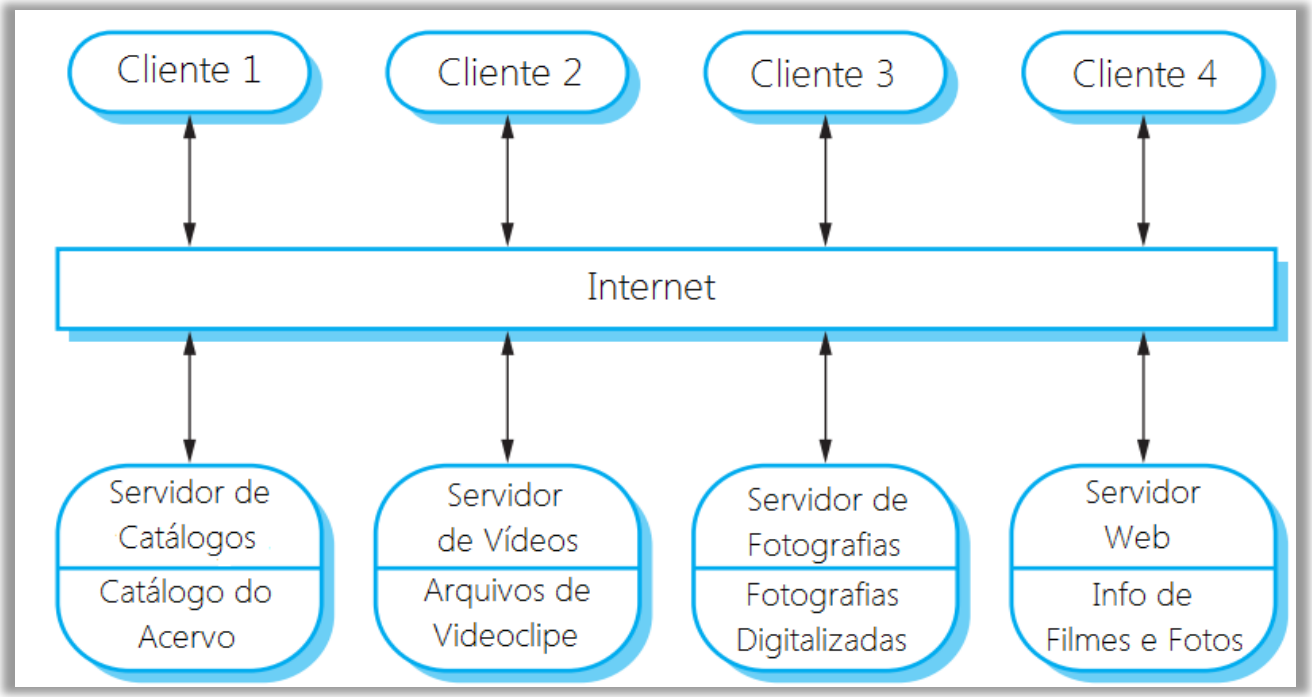
O processamento da informação é dividido em módulo ou processos distintos, sendo uma abordagem de computação que separa os processos em plataformas independentes que interagem, **permitindo que os recursos sejam compartilhados enquanto se obtém o máximo de benefício de cada dispositivo diferente**. Os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

[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?* ;)

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 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 DE MODELO

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[2]), 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 DE CONTROLE

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

[2] 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.

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*).

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 DE VISÃO

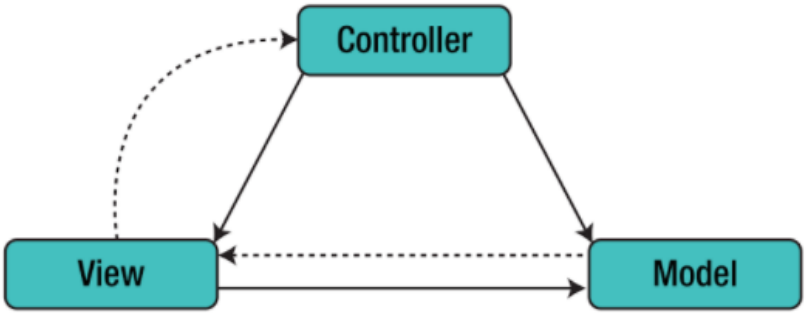
Essa é a camada responsável pela interação com o usuário, sendo responsável apenas pela exibição de dados. Trata-se de uma representação visual do modelo. Ela permite apresentar, de diversas formas diferentes, os dados para o usuário. A visão não sabe nada sobre o que a aplicação está fazendo atualmente,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

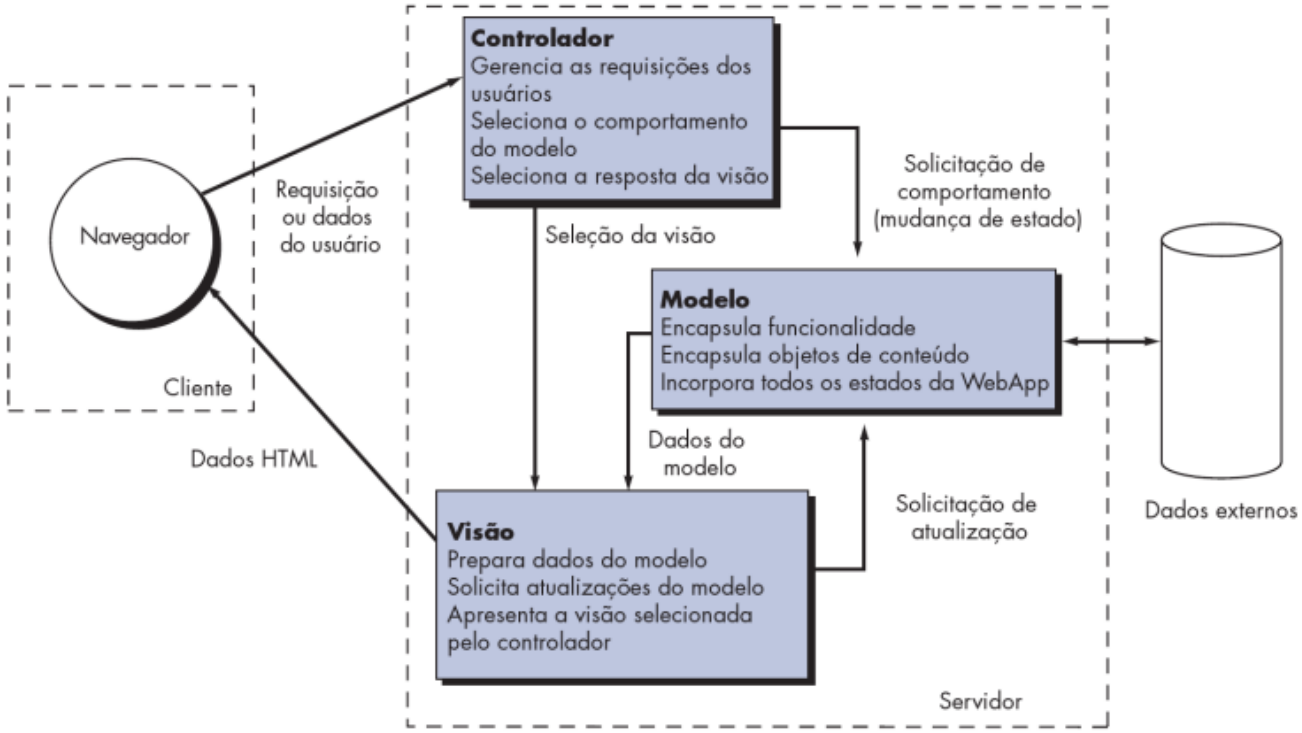
Na Arquitetura em Três-Camadas, a comunicação entre camadas é rigidamente linear, isto é, a Camada de Apresentação e a Camada de Dados só se conversam bidirecionalmente com a Camada de Lógica, mas nunca entre si. Já no MVC, a comunicação é triangular – **existem diversas implementações diferentes dessa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



Para quem quiser conhecer melhor, recomendo os seguintes artigos:

- ***http://www.cfgigolo.com/2008/01/mvc-model-view-controller-e-os-tres-macacos***
- ***https://r.je/views-are-not-templates.html***
- ***http://tableless.com.br/mvc-afinal-e-o-que***

Por fim, não podemos confundir MVC com MVP (Model-View-Presenter). O O MVP é uma evolução do MVC que se comunica bidirecionalmente com as outras camadas, evitando que o Model tenha que se comunicar diretamente com a View sem passar pelo Controller e este último é fundamental para a interação com o usuário. O MVP desacopla as funções e torna a arquitetura ainda mais modular.

A camada Presenter é ciente de tudo o que ocorre nas outras duas camadas e deixa-as cientes do que ela está fazendo; a interação com usuário é feita primariamente pela View, e esta pode delegar ao Presenter determinadas tarefas; há uma relação um-para-um entre estas camadas. **Nesta relação, há uma referência do View para o Presenter mas não o oposto.**

Não devemos confundir também com o MVVM (Model-View-ViewModel). O MVVM é uma pequena evolução do MVP em um lado e um retrocesso em outro. Nele o ViewModel não está ciente do que ocorre no View, **mas este está ciente do que ocorre no ViewModel** (como no Padrão de Projeto Observer). No caso do Model, ambos estão cientes do que ocorre em cada um.

O nome se dá porque ele adiciona propriedades e operações ao Model para atender as necessidades do View, **portanto ele cria um novo modelo para a visualização**. É possível associar várias Views para um ModelView. É comum que as Views sejam definidas de forma declarativa (HTML/CSS, XAML, etc.). O Data Binding é feito entre a View e o ViewModel.

Com esse padrão é possível reduzir a quantidade de código para manter. Algumas automações são possíveis por ter todas as informações necessárias no ViewModel. ViewModel – é só um modelo mais adequado para uma visão específica (ou mais que uma). Pessoal, acredito que isso basta em relação a esses assuntos que só recentemente começaram a ser cobrados.

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**

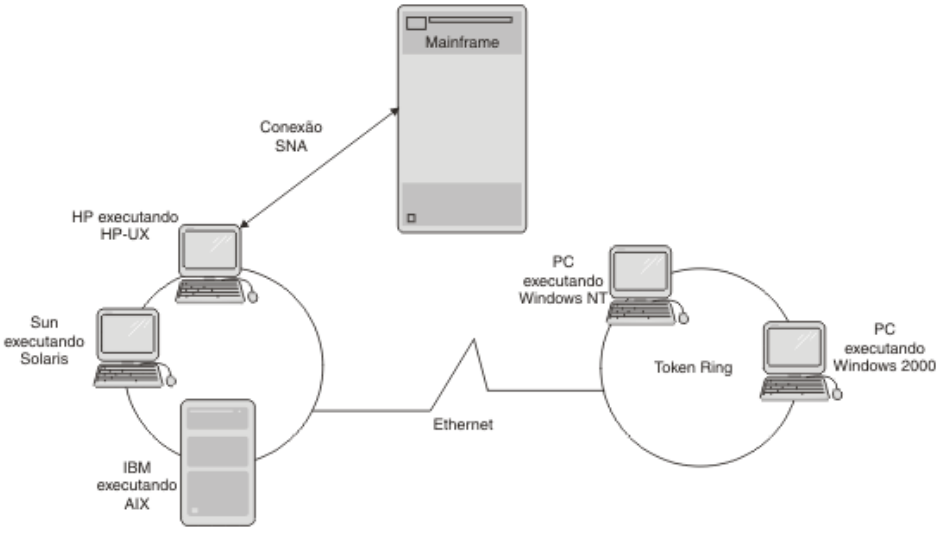
dispersado através da organização e seus desktops e servidores. *Professor, quais são as vantagens dessa abordagem?* Cara, há ganhos de responsividade, escalabilidade, redundância, disponibilidade, compartilhamento de recursos, controle e envolvimento do usuário, etc.

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.

QUESTÕES COMENTADAS – DIVERSAS BANCAS

1. **(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

---

2. **(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: Correto

---

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. **(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

---

6. **(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

---

7. **(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:

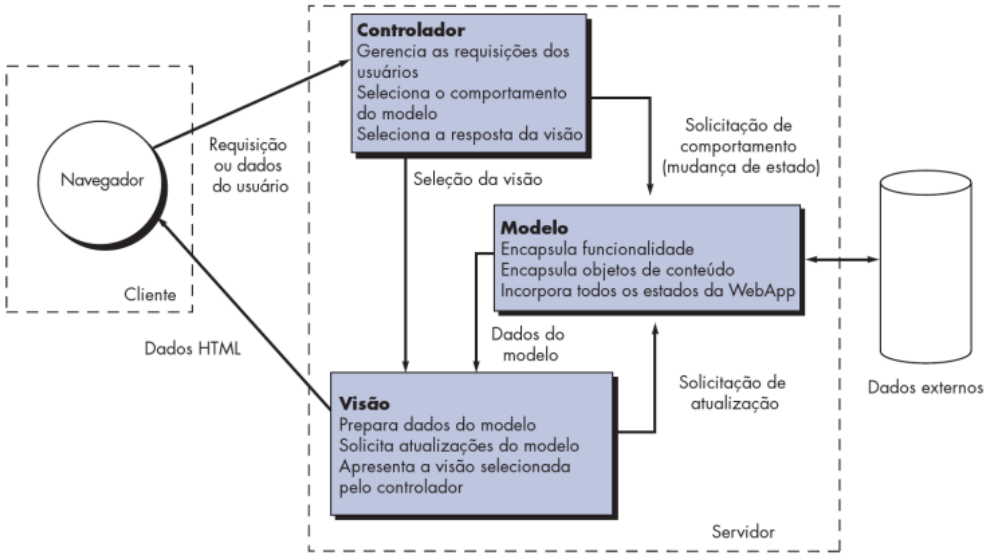
A questão está errada! *O modelo encapsula o estado da aplicação?* Sim, isso é verdade. *A visão solicita atualização do modelo?* Sim, vimos que ela faz isso por meio de eventos que notificam o controlador. *O Controlador gerencia a lógica de negócios?* Não, quem gerencia a lógica de negócios é o modelo.

Gabarito: Errado

---

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

Comentários:



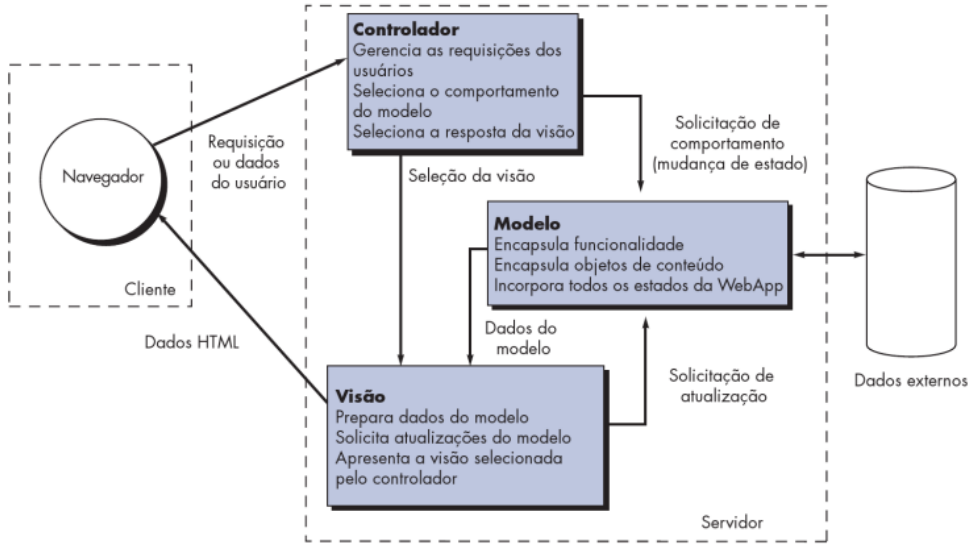
O controlador gerencia as requisições dos usuários, o modelo encapsula funcionalidades e a visão prepara dados do modelo.

Gabarito: Errado

---

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

Comentários:



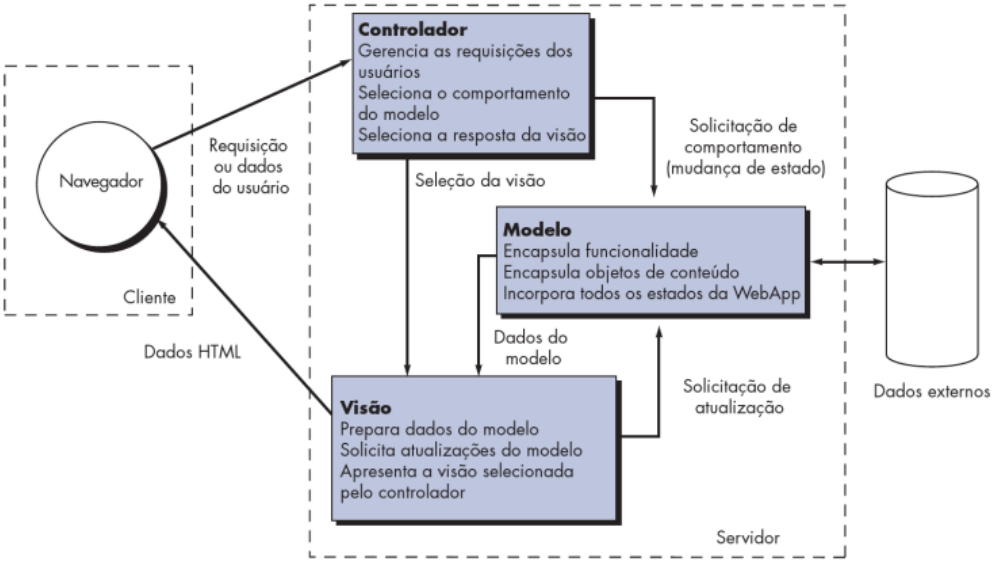
O modelo encapsula objetos de conteúdo, a visão solicita atualizações do modelo e o controle seleciona o comportamento do modelo.

Gabarito: Errado

---

10. **(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

---

11. **(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

---

12. **(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

---

13. **(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

---

14. **(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

---

15. **(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

---

16. **(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

---

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

LISTA DE QUESTÕES – DIVERSAS BANCAS

1. **(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.

2. **(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.

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. **(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.

6. **(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.

7. **(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.

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

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

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

11. **(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.

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

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

14. **(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

15. **(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.

16. **(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.

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.

GABARITO – DIVERSAS BANCAS

1. LETRA C
2. CORRETO
3. LETRA B
4. LETRA A
5. LETRA C
6. LETRA D
7. ERRADO
8. ERRADO
9. ERRADO
10. ERRADO
11. LETRA A
12. ERRADO
13. ERRADO
14. LETRA A
15. LETRA B
16. LETRA A
17. ERRADO
18. CORRETO
19. ERRADO
20. CORRETO
21. ERRADO
22. ERRADO
23. CORRETO
24. ERRADO
25. 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.