



APRENDIZAGEM EM FOCO

DESENVOLVIMENTO JAVA WEB



APRESENTAÇÃO DA DISCIPLINA

Autoria: Ariel da Silva Dias

Leitura crítica: Rogério Colpani

Olá, estudante! Seja bem-vindo à disciplina *Desenvolvimento Java Web*. Primeiramente, gostaria de contar um fato interessante: você sabia que a primeira página web foi lançada em agosto de 1991? E ela descrevia os planos para a *World Wide Web*. Foi também nesse ano que o HTML nasceu e a primeira descrição de suas *tags* foi lançada; inclusive, algumas delas ainda estão em uso atualmente, como as *tags* h1-h6, de parágrafo e de âncora. Só para curiosidade, essa primeira página ainda está ativa e você pode acessar pelo [w3.org](http://info.cern.ch/hypertext/WWW/InfoServer/WWW1.html).

De 1991 para cá, diversas tecnologias surgiram e modificaram como nós, desenvolvedores, construímos aplicações para que os usuários possam interagir. Dessa forma, nesta disciplina, conheceremos as principais tecnologias para desenvolver suas aplicações web, tendo como pano de fundo a linguagem de programação Java, uma das mais populares e versáteis linguagens de programação do mundo.

Iniciaremos nossos estudos conhecendo o *Java Enterprise Edition* e as principais tecnologias Java que podem ser adotadas em um projeto, como *Servlets*, *Java Server Pages*, *Java Server Pages*, *Hibernate*, entre outra. Além disso, dialogaremos sobre outras tecnologias envolvidas no desenvolvimento de aplicações web com a linguagem Java.

Este material foi preparado com toda a atenção para que você domine o assunto. Para isso, é muito importante que você execute os exemplos presentes no decorrer dos Temas e nas obras recomendadas, pois, somente assim, você continuará a evoluir nos estudos.

Bons estudos!

INTRODUÇÃO

Olá, aluno (a)! A *Aprendizagem em Foco* visa destacar, de maneira direta e assertiva, os principais conceitos inerentes à temática abordada na disciplina. Além disso, também pretende provocar reflexões que estimulem a aplicação da teoria na prática profissional. Vem conosco!



TEMA 1

Introdução à Programação Web

Autoria: Ariel da Silva Dias

Leitura crítica: Rogério Colpani



DIRETO AO PONTO

No desenvolvimento web, dois conceitos são fundamentais: *front-end* e *back-end*. O primeiro representa tudo aquilo que o usuário pode ver e com que pode interagir, como uma página desenvolvida em HTML com CSS e JavaScript. Já o segundo é mais complexo, pois nele fica toda a lógica de negócios da empresa. Desse modo, toda a programação em Java EE será executada no lado do servidor.

Em um olhar mais detalhado, para o desenvolvimento de uma aplicação web, podemos dividir esses dois conceitos em uma arquitetura de três camadas, que é um tipo de arquitetura de software composta de três lógicas de computação (Apresentação, Aplicação e Dados). Essas arquiteturas são frequentemente usadas em aplicativos com um tipo específico de sistema cliente-servidor e oferecem muitos benefícios para os ambientes de produção e desenvolvimento, modularizando a interface do usuário, a lógica de negócios e as camadas de armazenamento de dados.

Camada de Apresentação

A camada de apresentação é a camada frontal no sistema de três camadas, e é por meio dela que o usuário interage com o sistema (*front-end* ou *front-side*). Essa interface do usuário geralmente é gráfica, acessível por meio de um navegador da Web ou aplicativo baseado na Web e exibe conteúdo e informações úteis para o usuário final. Essa camada geralmente é criada em tecnologias da web, como HTML5, JavaScript e CSS, ou por meio de outras estruturas populares de desenvolvimento da web e se comunica com outras camadas por meio de chamadas de APIs.

Camada de Aplicação

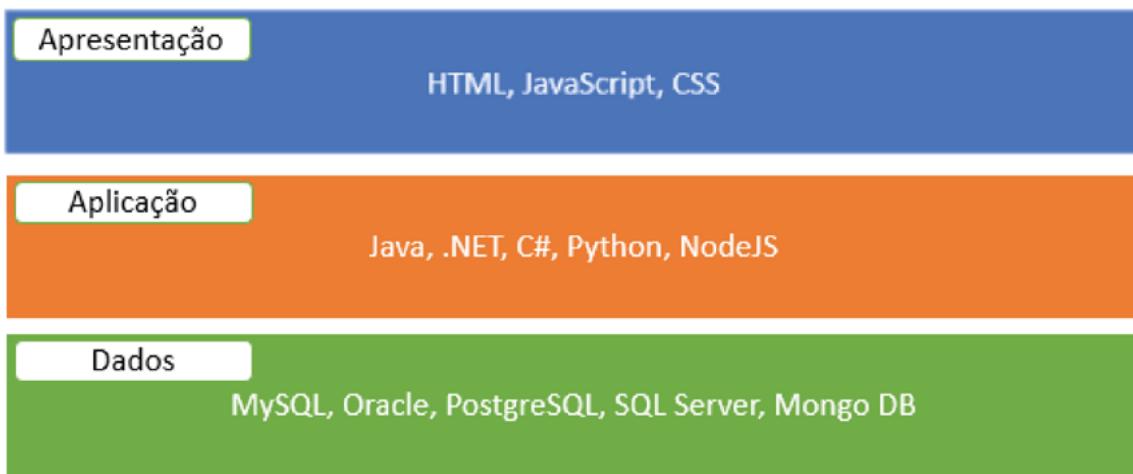
A camada de aplicação ou de negócio contém a lógica funcional de negócios, a qual impulsiona as capacidades essenciais de um aplicativo. É frequentemente escrita em Java, .NET, C #, Python, C ++, PHP, entre outros. Ela está no *back-end*, ou seja, é transparente para o usuário, uma vez que ele não interage diretamente com ela. É importante destacar aqui que a programação utilizando a plataforma Java EE ocorre exatamente na camada de aplicação.

Camada de Dados

Esta camada, também chamada de camada de persistência, compreende a base de dados ou sistema de armazenamento de dados e de acesso aos dados. Exemplos de tais sistemas são MySQL, Oracle, PostgreSQL, Microsoft SQL Server, MongoDB, entre outros. Os dados são acessados pela camada de aplicação por meio de chamadas de APIs. Ela também está no *back-end*.

A arquitetura de três camadas está resumida na Figura 1, sendo elas: camada de apresentação, camada de aplicação e camada de dados.

Figura 1 – Camadas de apresentação, de aplicação e de dados



Fonte: elaborada pelo autor.

Essa arquitetura em três camadas geralmente é ideal para incorporar e integrar softwares de terceiros a um aplicativo existente. Considere, por exemplo, um Sistema web que realiza vendas on-line. Nele, não seria necessário desenvolver uma aplicação para validar o CEP do usuário. Para isso, a equipe de desenvolvimento poderia utilizar um sistema externo (um microsserviço), que validaria o CEP digitado pelo usuário e retornaria para a aplicação.

Essa flexibilidade de integração também o torna ideal para incorporar software de análise em aplicativos preexistentes, sendo frequentemente usada por fornecedores de análise, como Google Analytics e Google Adsense. Arquiteturas de três camadas são frequentemente usadas em aplicativos baseados em nuvem ou local e em aplicativos de Software como Serviço (SaaS).

Referências bibliográficas

SOMMERVILLE, I. **Engenharia de Software**. São Paulo: Pearson, 2011.



PARA SABER MAIS

Dependendo do tipo de servidor que você está tentando configurar para seu projeto de aplicativo web, há uma variedade para escolher. À medida que você explora e começa a conhecer a vasta gama de opções, pode se perguntar qual seria a diferença entre um servidor web e um servidor de aplicativos, por exemplo.

Um servidor web é projetado para servir o conteúdo de um site a um navegador web que está em um desktop ou dispositivo móvel, enquanto um servidor web é alimentado por um software, como *Apache*, *Nginx* ou *Internet Information Services* (IIS) da Microsoft.

Todos eles, por padrão, entregam conteúdo pela porta 80 (conteúdo não criptografado) ou pela porta 553 (conteúdo criptografado por SSL). Para entregar o conteúdo estático, é utilizado o protocolo HTTP (*Hypertext Transfer Protocol*). Quando você navega em um site, seu navegador entra em contato com o servidor web desse site e solicita o conteúdo estático, que nada mais é do que os componentes básicos, como as páginas HTML, arquivos de imagem, arquivos de vídeo e outros tipos de arquivos do site.

Um servidor de aplicativo ou servidor de aplicações, como o nome sugere, refere-se a um servidor cuja principal função é a entrega de conteúdo e ativos para um aplicativo móvel, da web ou de desktop. Sendo assim, ele será o computador que fornecerá quaisquer ativos remotos solicitados pelo usuário. Enquanto o servidor web serve conteúdo apenas utilizando o protocolo HTTP ou HTTPS, o servidor de aplicativos pode usar qualquer variedade de protocolos para o envio de dados.

O principal benefício de um servidor de aplicativos é que ele pode servir conteúdo a uma ampla variedade de usuários e dispositivos e responder em tempo real de acordo com a lógica de negócios. Quando você acessa, por exemplo, uma loja on-line que exibe

informações de preço e disponibilidade de um item em tempo real, o servidor de aplicativos verifica os dados desse item e entrega essas informações para você. Entre os principais servidores de aplicação, destacam-se o Apache TomEE e o GlassFish.

Por fim, vale ressaltar que cada site necessita de um serviço web para entregar os arquivos HTML, afinal estes são a base de todo site. Desse modo, o servidor de aplicativos nunca poderá substituir a função do servidor web. Podemos concluir então que, em vez de serem tecnologias concorrentes, os servidores web e os servidores de aplicativos trabalham juntos para oferecer uma ótima experiência de navegação ao usuário.

Referências bibliográficas

APACHE. **Apache Tomcat 8. 2021.** Disponível em: <http://tomcat.apache.org/tomcat-8.5-doc/index.html>. Acesso em: 14 jan. 2021.

TANENBAUM, A. **Redes de Computadores.** 5. ed. Rio de Janeiro: Elsevier, 2011.

TEORIA EM PRÁTICA

Uma arquitetura de três camadas é um tipo de arquitetura de software composta por três camadas lógicas, sendo frequentemente usada em aplicativos do tipo cliente-servidor. Ela oferece muitos benefícios para os ambientes de produção e desenvolvimento, modularizando a interface do usuário, a lógica de negócios e as camadas de armazenamento de dados. Isso proporciona maior flexibilidade às equipes de desenvolvimento, permitindo que elas atualizem uma parte específica de um aplicativo independentemente das outras partes.

De acordo com essas informações, reflita sobre cada uma das três camadas e as tecnologias envolvidas. Considerando uma aplicação

de *streaming* em que o usuário precisa efetuar log in para assistir a um vídeo, como poderia ser implementada uma arquitetura em três camadas para essa aplicação?

Para conhecer a resolução comentada proposta pelo professor, acesse a videoaula deste *Teoria em Prática* no ambiente de aprendizagem.

LEITURA FUNDAMENTAL

Indicações de leitura

Indicação 1

Para que você compreenda os conceitos de *back-end* e *front-end* em uma arquitetura cliente-servidor, no Capítulo 15 do livro indicado, o autor apresenta a criação de um sistema cliente-servidor, explicando o conceito de soquetes de cliente e de servidor. Ao chegar ao final da leitura, você terá um cliente de bate-papo totalmente funcional.

Para realizar a leitura, acesse a nossa plataforma Biblioteca Virtual e busque pelo título da obra no parceiro Biblioteca Senac.

SIERRA, Kathy; BATES, Bert. **Use a cabeça!: Java.** Rio de Janeiro: Alta Books, 2009. p. 330-364.

Indicação 2

O artigo indicado apresenta o processo de desenvolvimento de uma aplicação utilizando o Java EE, bem como uma explicação sobre as tecnologias empregadas no desenvolvimento.

Para realizar a leitura, acesse a nossa plataforma Biblioteca Virtual e busque pelo título da obra no parceiro Biblioteca Virtual 3.0 – Pearson.

ALLIAN, A. P.; OLIVEIRA JUNIOR, E. A. Uma experiência de adoção do Java 6 Web Profile no desenvolvimento de um sistema para gerenciamento de uma clínica de psicologia. **Revista Tecnológica**, [s.l.], v. 24, n. 1, p. 25-39, 2015.

QUIZ

Prezado aluno, as questões do Quiz têm como propósito a verificação de leitura dos itens *Direto ao Ponto, Para Saber Mais, Teoria em Prática e Leitura Fundamental*, presentes neste Aprendizagem em Foco.

Para as avaliações virtuais e presenciais, as questões serão elaboradas a partir de todos os itens do Aprendizagem em Foco e dos slides usados para a gravação das videoaulas, além de questões de interpretação com embasamento no cabeçalho da questão.

1. O desenvolvimento *back-end* é qualquer tipo de desenvolvimento que não envolva a criação de código que produza uma interface com o usuário. Mesmo que o desenvolvimento de *front-end* obtenha grande parte da glória por ser visível, a maior parte do código que existe no mundo é de *back-end*. De acordo com essas informações, assinale a alternativa correta.
 - a. Um desenvolvedor de *front-end* precisa saber muito mais sobre a arquitetura de aplicativos do que um desenvolvedor *back-end*.

- b. A linguagem Java EE e a linguagem PHP são executadas tanto na camada de apresentação quanto na camada de aplicação.
 - c. O desenvolvedor *back-end* precisa saber sobre tecnologias web, como Java EE, PHP, Ruby ou ASP.NET.
 - d. Um desenvolvedor de *back-end* se concentra no HTML e CSS, enquanto o desenvolvedor *front-end* no layout da aplicação.
 - e. O desenvolvedor de *back-end* concentra seu trabalho nas camadas de apresentação e de aplicação no modelo de três camadas.
2. O *back-end* de um site é uma combinação de tecnologia e _____ que alimenta um site. Consiste em três partes que um usuário nunca vê: um _____, um aplicativo e um _____.
- Os desenvolvedores de *back-end* desempenham um papel crítico nas equipes de desenvolvimento da web e garantem que os _____ ou _____ solicitados pelo sistema ou software de *front-end* sejam fornecidos.

- Assinale a alternativa que completa adequadamente as lacunas.
- a. Programação; servidor; banco de dados; dados; serviços.
 - b. APIs; layout; servidor; serviços; requisições.
 - c. Recursos; servidor; layout; dado; dados; requisições.
 - d. Programação; layout; serviço; recursos; requisições.
 - e. APIs; servidor; dado; dados; serviços.



GABARITO

Questão 1 - Resposta C

Resolução: A arquitetura de aplicativos web é uma preocupação e foco do desenvolvedor *back-end*, no qual está a camada de aplicação, em que é criada a lógica de negócio utilizando linguagens como Java EE e PHP. O desenvolvedor *back-end* (e não o de *front-end*) precisa conhecer tecnologias como Java EE e outras linguagens de programação, além de banco de dados, pois estes fazem parte de sua rotina de trabalho. Por outro lado, é o desenvolvedor *front-end* que concentra seus esforços trabalhando com HTML e CSS na camada de apresentação.

Questão 2 - Resposta A

Resolução: Um sistema web pode ser dividido em *back-end* e *front-end*. O *front-end* é o dispositivo por onde o usuário interage. Por outro lado, o *back-end* é o servidor e nele está toda a lógica do negócio, a qual é implementada em uma linguagem de programação, sendo baseada muitas vezes na consulta de um banco de dados. Desse modo, quando o cliente (*front-end*) realiza uma requisição, o *back-end* é responsável por garantir que os dados ou serviços requisitados sejam retornados a ele.



TEMA 2

Servlets e Java Server Pages

Autoria: Ariel da Silva Dias

Leitura crítica: Rogério Colpani



DIRETO AO PONTO

O *JavaServer Pages* (JSP) é uma linguagem de script que fornece um modo mais prático de acessar e executar *servlets*. Assim como outras linguagens, como o PHP e o ASP, os JSP permitem ao desenvolvedor misturar Java com HTML e colocar o Java entre tags especiais, como <% e%>.

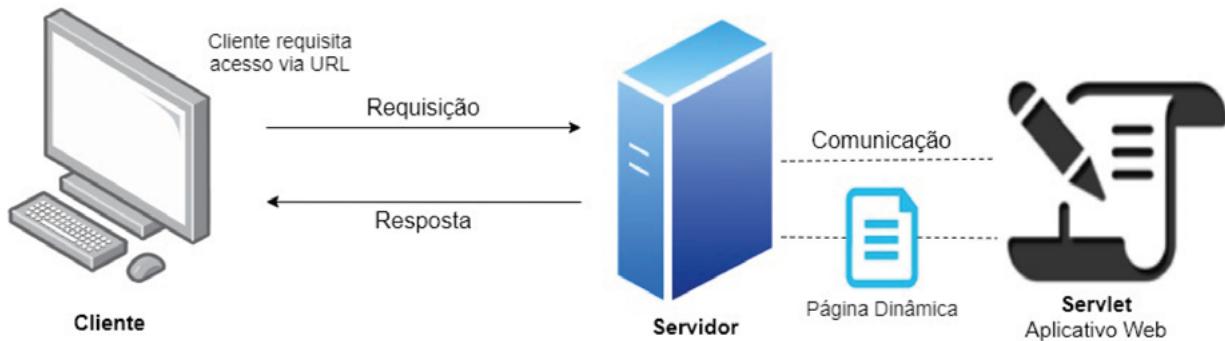
Um componente JSP é um tipo de *servlet* Java projetado para cumprir a função de uma interface de usuário para um aplicativo da web. Os desenvolvedores web escrevem JSP como arquivos de texto que combinam código HTML, elementos XML e ações e comandos JSP integrados.

Usando JSP, você pode coletar informações de usuários por meio de formulários de página da Web, apresentar registros de um banco de dados ou de outra fonte e criar páginas da Web dinamicamente. As *tags* JSP podem ser usadas para uma variedade de propósitos, como recuperar informações de um banco de dados ou registrar preferências do usuário; acessar componentes JavaBeans; passar o controle entre páginas; compartilhar informações entre solicitações, páginas etc.

Desse modo, podemos afirmar que um arquivo JSP é simplesmente uma página HTML com algum código Java espalhado e basicamente fornece um conteúdo dinâmico que você pode incluir em sua página. É um modelo para uma página Web que usa código Java para gerar um documento HTML dinamicamente. Os JSPs são executados em um componente do lado do servidor conhecido como contêiner JSP, que os converte em *servlets* Java. Essa comunicação entre cliente e servidor é realizada utilizando o protocolo HTTP (*Hypertext Transfer Protocol*).

O HTTP é um protocolo cliente-servidor de resposta a pedidos assimétrico, conforme ilustrado na Figura 1. Um cliente HTTP envia uma mensagem de solicitação para um servidor HTTP, e o servidor, por sua vez, retorna uma mensagem de resposta. Em outras palavras, o HTTP é um protocolo *pull*, e o cliente extrai informações do servidor (em vez de o servidor enviar as informações para o cliente).

Figura 1 – Arquitetura cliente-servidor realizando comunicação HTTP



Fonte: elaborada pelo autor.

Para sites e páginas, o navegador atua como cliente e um servidor da Web, como Apache, atua como servidor. O servidor hospeda os arquivos (como html, áudio, arquivos de vídeo, entre outros) e responde às solicitações do cliente com os dados. Dependendo da solicitação, uma resposta contém o status da solicitação.

O HTTP é um protocolo sem estado, ou seja, a solicitação atual não sabe o que foi feito nas solicitações anteriores. Acrescenta-se ainda que ele permite negociar o tipo e a representação dos dados, de modo a permitir a construção de sistemas de forma independente dos dados que estão sendo transferidos.

Sempre que você emite uma URL (*Uniform Resource Locator*) do seu navegador para obter um recurso da Web usando HTTP, por exemplo `http://www.google.com.br`, o navegador transforma a URL em uma mensagem de requisição e a envia ao servidor HTTP. O servidor HTTP, então, interpreta a mensagem de requisição e retorna uma mensagem de resposta apropriada, que pode ser o recurso solicitado ou uma mensagem de erro.

O HTTP é um protocolo no nível de aplicação na arquitetura cliente-servidor. Geralmente, é executado em uma conexão TCP/IP (*Transmission Control Protocol / Internet Protocol*), mas ele não precisa ser executado no TCP/IP. Como ele pressupõe apenas um transporte confiável, qualquer protocolo de transporte que forneça essas garantias pode ser usado.

O TCP/IP, por sua vez, é um conjunto de protocolos da camada de transporte e da camada de rede, respectivamente, para que as máquinas se comuniquem entre si pela rede. O IP lida com endereçamento e roteamento de rede. Em uma rede IP, cada máquina recebe um endereço IP exclusivo (por exemplo, 192.168.0.1), sendo o software IP responsável por rotear uma mensagem do IP de origem para o IP de destino.

No IPv4, o endereço IP consiste em 4 bytes e cada um varia de 0 a 255, separados por pontos. O IPv6 é o mais recente e suporta mais endereços.

Como a memorização do número é difícil para a maioria das pessoas, um nome de domínio semelhante ao inglês, como `www.nomedsite.com.br`, é usado em seu lugar. O DNS (*Domain Name Service*), então, converte o nome do domínio no endereço IP (por meio de tabelas de pesquisa distribuídas). Um endereço IP especial 127.0.0.1 sempre se refere à sua própria máquina (chamado de *localhost*) e pode ser usado para teste local de *loopback*.

Referências bibliográficas

KURNIAWAN, Budi; DECK, Paul. **Servlet, JSP and Spring MVC.** Quebec: Brainy Software Inc, 2015.

PARA SABER MAIS

O JSP (*JavaServer Pages*) é programado utilizando a linguagem Java. Logo, a sintaxe utilizada ao programar aplicativos Java será a mesma a ser utilizada durante o desenvolvimento em JSP. Para que você esteja preparado, estão destacadas a seguir algumas das principais propriedades da linguagem.

Dados numéricos

Assim como em outras linguagens, os números podem ser inteiros, reais, positivos, decimais, octais e hexadecimais.

- `int idade = 25; // atribuição de valor inteiro na base decimal.`
- `double peso = 72.4; // atribuição de valor real com uma casa decimal.`

Dados alfanuméricos

Os dados alfanuméricos, ou *strings*, são declarações de sequência de caracteres alfanuméricos, que são delimitados por aspas duplas.

- `String cidade = "Belo Horizonte"; // atribuição de uma cadeia de caracteres.`

Outra característica é podermos utilizar caracteres controladores, como:

- \n – para criar uma nova linha.
- \" – para inserir no texto o caractere " (aspas)
- \r – para retorno do carro.
- \t – para tabulação.
- \\$ – para inserir o caractere \$ (cifrão).
- \\ – para inserir o caractere \ (barra invertida).

Representação de variáveis em JSP

Em JSP, o nome da variável deve sempre respeitar a seguinte estrutura: **[uma ou mais letras][letras ou números]**. Seguindo essa estrutura, veja a seguir nomes válidos e inválidos de variáveis:

- **Variáveis válidas:** nota1, v20arte, minha_variavel, _variavel_permitida98.
- **Variáveis inválidas:** 100valor, 22, inv@lido, t&este.

Acrescenta-se ainda que o JSP é case *sensitive*, ou seja, uma variável chamada Nota não é a mesma coisa que nota e NOTA. Então, por padrão, use sempre variáveis com a primeira letra minúscula.

Por fim, vale ressaltar que estruturas condicionais, como *if*, *if-else* e *switch-case*, bem como as estruturas de repetição, como *while*, *do-while* e *for*, possuem a mesma lógica e sintaxe da linguagem Java.

Referências bibliográficas

DEITEL, Paul J.; DEITEL, Harvey M. **Java: Como Programar.** 10. ed. São Paulo: Pearson, 2016.

KURNIAWAN, Budi; DECK, Paul. **Servlet, JSP and Spring MVC.** Quebec: Brainy Software Inc, 2015.

TEORIA EM PRÁTICA

Você foi contratado por uma empresa para criar um aplicativo web de comércio eletrônico. Ela informou que nesse aplicativo o cliente poderá realizar a pesquisa do produto, selecioná-lo e realizar a compra. Além disso, o processo de validação do pagamento será realizado por um sistema de terceiros: você enviará os dados, o sistema externo os processará e lhe retornará os parâmetros para a finalização da venda. Por fim, o sistema apenas informa o cliente que o pedido foi feito com sucesso e lhe apresenta um protocolo.

Você escolheu implementar esse aplicativo utilizando o JSP, uma vez que considera ser mais simples de implementar, se comparado ao *servlet*. De posse dessas informações, e considerando as particularidades de cada tecnologia, você acredita que esta é uma escolha certeira? Sabendo que você pode trabalhar com as duas tecnologias no mesmo projeto, então quando utilizar JSP e quando utilizar *servlets*?

Para conhecer a resolução comentada proposta pelo professor, acesse a videoaula deste *Teoria em Prática* no ambiente de aprendizagem.



LEITURA FUNDAMENTAL

Indicações de leitura

Indicação 1

No Capítulo 26 do livro indicado, o autor apresenta uma introdução à tecnologia de *Servlets*, conceituando a arquitetura bem como a configuração de um servidor Apache. Ele ainda apresenta diversos exemplos de tratamento de solicitações HTTP por *get* e *post*.

Para realizar a leitura, acesse a nossa plataforma Biblioteca Virtual e busque pelo título da obra no parceiro Biblioteca Virtual.

DEITEL, P.; DEITEL, H. **Java: Como Programar.** 6. ed. São Paulo: Pearson, 2006. p. 928-955.

Indicação 2

No Capítulo 27 do livro indicado, você encontrará os fundamentos necessários para o desenvolvimento de uma aplicação Java utilizando JSP. O autor apresenta diversos exemplos, bem como o estudo de um caso de livro de visita.

Para realizar a leitura, acesse a nossa plataforma Biblioteca Virtual e busque pelo título da obra no parceiro Biblioteca Virtual.

DEITEL, P.; DEITEL, H. **Java: Como Programar.** 6. ed. São Paulo: Pearson, 2006. p. 959-989.



QUIZ

Prezado aluno, as questões do Quiz têm como propósito a verificação de leitura dos itens *Direto ao Ponto, Para Saber Mais, Teoria em Prática e Leitura Fundamental*, presentes neste Aprendizagem em Foco.

Para as avaliações virtuais e presenciais, as questões serão elaboradas a partir de todos os itens do Aprendizagem em Foco e dos slides usados para a gravação das videoaulas, além de questões de interpretação com embasamento no cabeçalho da questão.

1. A internet possui uma vasta gama de recursos hospedados em diferentes servidores. Para você acessar esses recursos, seu navegador precisa enviar uma _____ aos servidores e exibir os recursos para você. O _____ é o protocolo usado para estruturar as _____ e as _____ para uma comunicação eficaz entre um cliente e um servido.
 - a. Requisição; HTTP; requisições; respostas.
 - b. Comunicação; HTML; páginas web; requisições.
 - c. Requisição; HTML; páginas web; respostas.
 - d. Comunicação; HTTP; páginas web; respostas.
 - e. Comunicação; HTML; requisições; respostas.

2. O _____ é útil para construir sites _____ e acessar informações de banco de dados em um _____. Embora as páginas _____ possam ter Java intercalado com HTML, todo o código Java é analisado no servidor. Portanto, assim que a página chega ao navegador, é apenas HTML.
- a. JSP; dinâmicos; servidor web; JSP.
 - b. *Servlet*; web; servidor; web.
 - c. *Servlet*; dinâmicos; servidor web; web.
 - d. JSP; dinâmicos; repositório; web.
 - e. *Servlet*; web; repositório; JSP.



GABARITO

Questão 1 - Resposta A

Resolução: O protocolo HTTP é o formato que permite a comunicação entre cliente e servidor, podendo realizar requisições e receber respostas no formato de mensagens. Ele possui métodos como o GET, que é usado para solicitar dados de um recurso específico, e o POST, que é usado para enviar dados para um servidor.

Questão 2 - Resposta A

Resolução: Os JSPs, como *servlets*, são programas do lado do servidor executados dentro de um servidor HTTP compatível com Java. Eles foram desenvolvidos para criarmos conteúdo web dinâmicos, como acessar um banco de dados no servidor e retorná-lo ao cliente. Tornam a criação e a manutenção de páginas dinâmicas HTML muito mais fácil do que o *servlet*, sendo, por isso, mais conveniente para lidar com a apresentação.



TEMA 3

JSF, Arquiteturas MVC e Introdução à persistência de dados

Autoria: Ariel da Silva Dias

Leitura crítica: Rogério Colpani



DIRETO AO PONTO

Uma arquitetura de três camadas é um tipo de arquitetura de software composta de três lógicas de computação (apresentação, aplicação e dados), frequentemente usadas em aplicativos como um tipo específico de sistema cliente-servidor.

A camada de apresentação está no *front-end* (HTML, JavaScript, CSS), e as camadas de aplicação (Java, .NET, Python) e dados (MySQL, SQL Server, Oracle) estão localizadas no servidor.

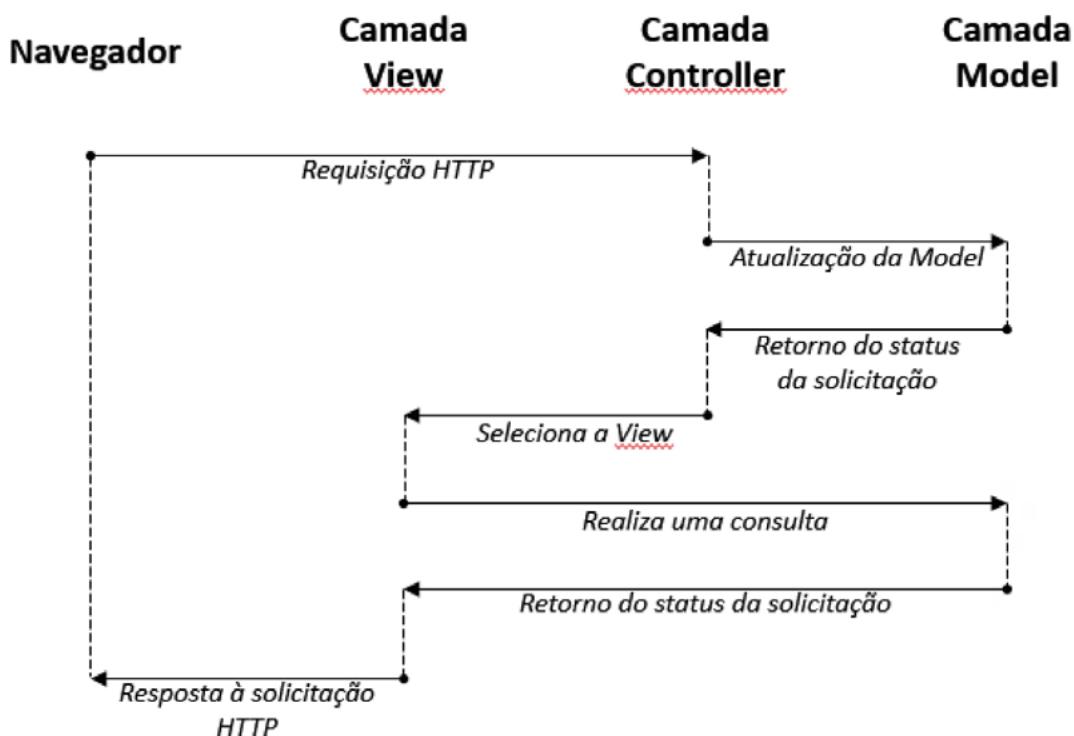
Apesar de aparentarem certa semelhança, cabe salientar que existem diferenças entre elas. O MVC é um padrão usado para facilitar a manutenção e teste do código da interface do usuário. Quando o padrão MVC é usado, uma parte maior do código da interface do usuário pode ser testada em unidade.

A arquitetura de três camadas é um padrão usado por um motivo completamente diferente, pois separa o aplicativo inteiro em grupos significativos: interface do usuário, lógica de negócios, armazenamento de dados.

Portanto, é possível dizer que uma aplicação web de três camadas refere-se a todo o código do aplicativo. O padrão MVC é um padrão usado na camada da *User Interface* (UI) ou interface do usuário.

Criar um aplicativo MVC pode resultar em mais classes e objetos do que um sistema que não utiliza o modelo MVC. Isso significa mais design no início. Por outro lado, um sistema MVC bem projetado será mais fácil de adaptar e expandir, pois o código será melhor separado.

Figura 1 - Diagrama de sequência



Fonte: elaborada pela autora.

Observe, na Figura 1, que a camada *Controller* não é responsável em realizar a comunicação entre as camadas *View* e *Model*. Desse modo, as *views* realizam requisições diretas à camada *Model*.

Uma vez que a camada *Controller* é a responsável pelo tratamento das solicitações e pela seleção da página apropriada (visualização), não há acoplamento imediato entre a solicitação feita pelo usuário e a página resultante.

Isso acaba sendo muito útil se o fluxo de página no aplicativo for complexo, mas mesmo para aplicativos simples, a camada *Controller* é um bom lugar para lidar com ações comuns, como, por exemplo, realizar autenticação e gerenciamento de sessão.

Conforme a tecnologia continua a evoluir, o mesmo ocorre com a arquitetura de aplicativos web. Uma dessas tendências é representada pelo uso e criação de arquitetura orientada a serviços.

É aqui, que a maior parte do código de todo o aplicativo existe como serviços. Além disso, cada um tem sua própria API HTTP. Como resultado, uma parte de um código realiza uma solicitação para outra parte de outro código, que é executada em um servidor diferente.

Microsserviços são serviços pequenos e leves que executam uma única funcionalidade. A estrutura da arquitetura de microsserviços possui várias vantagens que permitem que os desenvolvedores não apenas aumentem a produtividade, mas também acelerem todo o processo de implantação.

Os componentes que compõem uma compilação de aplicativos, usando a arquitetura de microsserviços, não dependem diretamente um do outro. Como tal, não precisam ser construídos usando a mesma linguagem de programação.

Portanto, os desenvolvedores que trabalham com a arquitetura de microsserviços possuem livre acesso a uma pilha de tecnologia à sua escolha. Isso torna o desenvolvimento do aplicativo mais simples e rápido.

Referências bibliográficas

PRESSMAN, R. **Engenharia de software:** uma abordagem profissional. São Paulo: Bookman. 2016.

PARA SABER MAIS

Compreender o significado de persistência de dados é importante para avaliar diferentes sistemas de armazenamento de dados. De acordo com a importância do armazenamento de dados na

maioria dos aplicativos modernos, fazer uma escolha incorreta pode significar um tempo de inatividade substancial ou perda de dados.

No contexto do armazenamento de dados em um sistema de computador, isso significa que os dados sobrevivem após o término do processo com o qual foram criados. Em outras palavras, para um armazenamento de dados ser considerado persistente, deve gravar em um armazenamento não volátil.

A maneira mais prática de adicionar persistência é com os sistemas de gerenciamento de banco de dados (SGBD), como o SQL Server, Oracle, MySQL, SQLite, MongoDB, entre outros. Falaremos sobre um dos mais populares gerenciadores de banco de dados relacional, o MySQL.

O MySQL é um gerenciador de banco de dados que organiza os dados em um conjunto de dados relacionais estruturados ou em forma de tabelas, com colunas e linhas. Nesse modelo, as tabelas representam os objetos, colunas representam os campos e linhas representam os registros. É o sistema de gerenciamento de banco de dados relacional amplamente empregado, pois está disponível gratuitamente e como código aberto para qualquer pessoa utilizar (MySQL, 2021).

O MySQL possui suporte as consultas básicas da linguagem de programação *Structured Query Language* (SQL), ou- linguagem de consulta estruturada, usada para criar, atualizar, excluir e gerenciar as tabelas e seu conteúdo da base de dados. Desse modo, poderemos realizar consultas para recuperar dados de uma tabela (ou de várias delas), por meio de comandos. O MySQL pode ser usado para consultar, filtrar, classificar, agrupar e modificar dados, além de poder unir tabelas e realizar relacionamentos.

Os principais comandos MySQL são:

- **SELECT:** usado para recuperação de dados das tabelas.
Exemplo: `SELECT * FROM nometabela.`
- **WHERE:** usado para filtrar os dados com base em condições.
Exemplo: `SELECT * FROM nometabela WHERE condição.`
- **INSERT:** usado para inserir uma ou mais linhas na tabela.
Exemplo: `INSERT INTO nometabela (coluna1, coluna2, ...) VALUES (valor1, valor2, ...).`
- **UPDATE:** usado para atualizar a tabela em coluna específica para o registro específico. Exemplo: `UPDATE nometabela SET coluna1 = valor1 WHERE coluna2 = valor2.`
- **DELETE:** usado para excluir o registro da tabela para um valor específico. Exemplo: `DELETE FROM nometabela WHERE condição.`

Os comandos de consulta do SQL são explicados acima, usados, principalmente, para recuperar os dados do banco de dados, inserir e remover. Esses são os principais comandos para uso no MySQL. Existem outros, porém, como o objetivo deste material não é aprofundar a respeito desses comandos, são citados apenas esses principais.

Referências bibliográficas

MYSQL. **MySQL**. Disponível em: <https://www.mysql.com/>. Acesso em: 26 mar. 2021.

TEORIA EM PRÁTICA

O modelo arquitetônico MVC segue uma ideia elementar, devemos separar as responsabilidades em qualquer aplicativo em modelo,

visualização e controlador. A metodologia MVC desempenha um papel muito importante para projetar uma interação mais eficiente com os aplicativos de modelos de dados, sendo usado, principalmente, nas linguagens de programação Java, ASP .NET e Smalltalk.

Agora, reflita sobre um caso hipotético onde seu colega de projeto está em dúvida sobre como as camadas do modelo MVC se relacionam, e qual a diferença desta arquitetura para a arquitetura em três camadas. Como você responderia a essas dúvidas? Na escolha entre uma arquitetura ou outra, qual alternativa poderia ser oferecida a esse colega?

Para conhecer a resolução comentada proposta pelo professor, acesse a videoaula deste *Teoria em Prática* no ambiente de aprendizagem.

LEITURA FUNDAMENTAL

Indicações de leitura

Indicação 1

No livro *Engenharia de software: uma abordagem profissional*, entre as páginas 346 e 350, o autor apresenta o conceito de projeto arquitetural de aplicativo web. Dentre os conceitos apresentados, consta o modelo MVC.

Para realizar a leitura, acesse a plataforma Biblioteca Virtual e busque pelo título da obra no parceiro *Biblioteca Virtual 3.0_Pearson*.

PRESSMAN, R. **Engenharia de software:** uma abordagem profissional. p.346-350. São Paulo: Bookman, 201.

Indicação 2

No livro *Java: como programar*, entre as páginas 818 e 840, o autor apresenta tópicos importantes sobre o conceito de persistência de dados, mais especificamente sobre bancos de dados relacionais, como criar bancos de dados e realizar consultas.

Para realizar a leitura, acesse a plataforma Biblioteca Virtual e busque pelo título da obra no parceiro *Biblioteca Virtual 3.0_Pearson*.

DEITEL, P.; DEITEL, H. **Java: como programar**. 6. ed., p. 818-840. São Paulo: Pearson, 2006.

QUIZ

Prezado aluno, as questões do Quiz têm como propósito a verificação de leitura dos itens *Direto ao Ponto, Para Saber Mais, Teoria em Prática e Leitura Fundamental*, presentes neste Aprendizagem em Foco.

Para as avaliações virtuais e presenciais, as questões serão elaboradas a partir de todos os itens do Aprendizagem em Foco e dos slides usados para a gravação das videoaulas, além de questões de interpretação com embasamento no cabeçalho da questão.

1. O MVC é um modelo de _____ de aplicativo composto por três partes interconectadas, que incluem o modelo (_____), a visualização (_____) e o controlador (______). Esse modelo fornece os componentes fundamentais para o design de programas para desktop ou celular, além de aplicativos da web.

Assinale a alternativa que completa adequadamente as lacunas:

- a. Design; dados; interface; manipulação de entrada.
 - b. *Framework*; interface; dados; manipulação de entrada.
 - c. Estrutura; manipulação de entrada; interface; dados.
 - d. Design; interface; manipulação de entrada; dados.
 - e. *Framework*; manipulação de entrada; dados; interface.
2. Um *Managed Bean* contém a _____ de uma aplicação web. Dentre as principais propriedades do *Managed Bean*, no JSF, destaca-se que deve possuir a anotação _____, um construtor _____; e métodos públicos _____ para suas variáveis privadas.

Assinale a alternativa que completa adequadamente as lacunas:

- a. Lógica de negócios; @*ManagedBean*; sem argumentos; *get* e *set*.
- b. Base de dados; @*Override*; com retorno; com retorno.
- c. Estrutura de dados; @*ManagedBean*; com retorno; com retorno.
- d. Lógica de negócios; @*Override*; sem retorno; sem retorno.
- e. Base de dados; @*Override*; sem argumentos; *get* e *set*.



GABARITO

Questão 1 - Resposta A

Resolução: As três camadas do MVC estão interconectadas, onde a *view* é responsável pela interface e é por onde o usuário interage com a aplicação; a camada modelo está relacionada aos dados e permite a reutilização de um mesmo objeto em *views* diferentes; a camada do controlador contém a lógica do negócio, com as classes lógicas da aplicação.

Questão 2 - Resposta A

Resolução: Um *Managed Bean* é uma classe regular Java que possui a lógica de negócios da uma aplicação. Deve possuir uma marcação `@ManagedBean`, caso contrário, a classe apresentará um erro. Também deve possuir um método construtor com zero parâmetro e, por fim, métodos públicos *get* e *set* para as variáveis.



TEMA 4

Hibernate e Integração JSF e JPA

Autoria: Ariel da Silva Dias

Leitura crítica: Rogério Colpani



DIRETO AO PONTO

JPA é uma abreviatura que significa *Java Persistence API*. É uma especificação que faz parte do Java EE, e define uma API para mapeamentos relacionais de objetos e para gerenciar objetos persistentes.

O próprio JPA não fornece nenhuma classe de implementação. O JAR da API contém apenas um conjunto de interfaces que pode ser usado para implementar sua camada de persistência. Entretanto, não se pode usar JPA sozinho. É preciso um provedor JPA, que implemente a especificação. Existem várias opções acessíveis, sendo os mais populares: *Hibernate* e *EclipseLink*.

A especificação define a maioria dos recursos que se pode usar com todas as implementações JPA.

Veja alguns dos mais importantes:

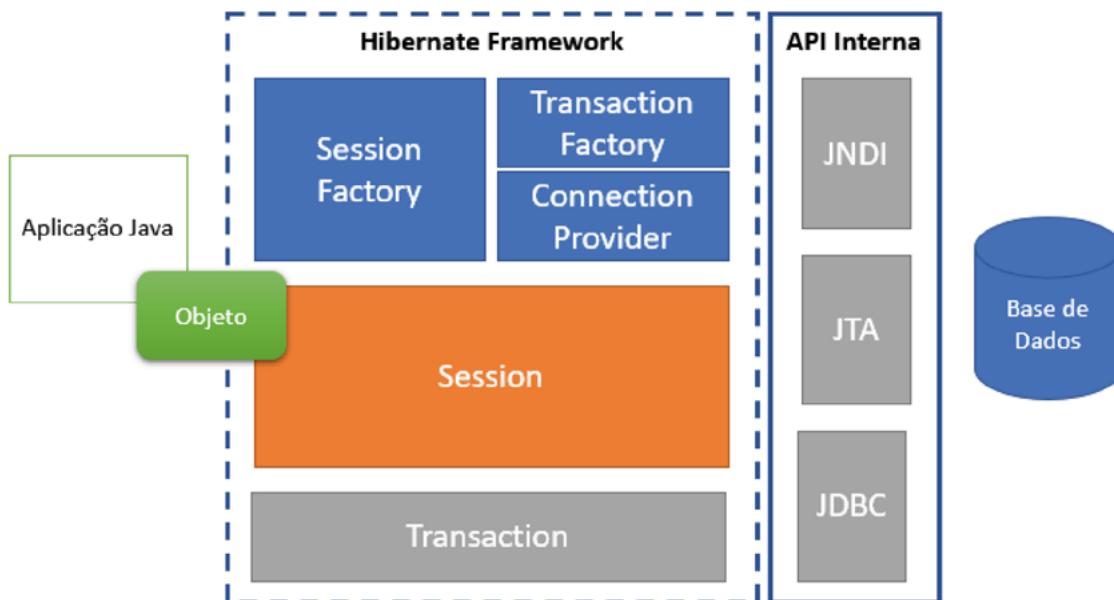
- **Bootstrapping e mapeamento básico de *entities*:** o primeiro passo antes de começar a usar o JPA, é adicioná-lo ao seu projeto, configurar uma unidade de persistência (*persistence unity*), mapear *entities* para suas tabelas de banco de dados e inicializá-lo.
- **Mapeando associações:** JPA permite mapear atributos das *entities* com colunas do banco de dados, entretanto, o JPA também permite que sejam mapeadas as associações entre tabelas de banco de dados. Isso torna seu modelo de *entity* muito prático de usar, uma vez que só é preciso chamar um método *get* em uma *entity* para carregar as *entities* associadas. Ao realizar essa solicitação, o provedor de persistência executa das operações no banco de dados para recuperar e gerenciar as associações.

- **JPQL e consultas nativas:** JPQL é a linguagem de consulta do JPA, muito semelhante ao SQL. Isso permite definir consultas com base no modelo de domínio mapeado, em vez do modelo de tabela do banco de dados. Para executar consulta com JPQL, é necessário chamar o método `createQuery` do `EntityManager`. Por outro lado, se deseja utilizar os comandos nativos do SQL, basta chamar o método `createNativeQuery` e passar a consulta SQL por parâmetro.

Essas especificações permitiram o desenvolvimento de implementações do JPA. Um exemplo é o *Hibernate*, uma ferramenta *Object Relational Mapping* (ORM), que reduz a dificuldade no desenvolvimento de aplicações.

O *Hibernate* utiliza muitos objetos, como mostra a figura a seguir.

Figura 1 - Arquitetura do *Hibernate*



Fonte: elaborada pelo autor.

Veja descrição de cada um dos elementos apresentados:

- **SessionFactory:** a interface `org.hibernate.SessionFactory` fornece o método para obter o objeto da sessão.

- **TransactionFactory**: responsável por criar transações.
- **ConnectionProvider**: abstrai o aplicativo de *DriverManager* ou *DataSource*.
- **Session**: esse elemento fornece uma interface entre o aplicativo e os dados armazenados no banco. Trata-se de um objeto de curta duração, que abre e encerra a conexão JDBC. Nele, encontramos um cache de nível um, que é obrigatório, além de fornecer métodos para inserir, atualizar e excluir o objeto.
- **Transaction**: a interface *org.hibernate.Transaction* fornece métodos para gerenciamento de transações.

Além desses, que são elementos do framework *Hibernate*, temos também as APIs Java como o JDBC (Java Database Connectivity), JTA (*Java Transaction API*) e a JNDI (*Java Naming Directory Interface*).

Referências bibliográficas

HIBERNATE. **Hibernate ORM**. Disponível em: <https://hibernate.org/orm>. Acesso em: 26 mar. 2021.

PARA SABER MAIS

Em um banco de dados relacional é comum o relacionamento entre duas tabelas, definido por chaves estrangeiras. Nesse caso, geralmente, uma tabela possui uma coluna que contém a chave primária de outra tabela, isso é o que caracteriza chave estrangeira e o relacionamento entre tabelas.

Como em JPA, lidamos com objetos que são representações Java de tabelas de banco dados, precisamos de uma maneira diferente de estabelecer um relacionamento entre duas *entities*.

Os relacionamentos de *entities* em JPA definem como se referem uma às outras. Vejamos cada uma das relações possíveis.

Relacionamento um-para-um:

Essa relação é caracterizada pela anotação `@OneToOne`, de modo que permita relacionar duas *entities*.

Como exemplo, pense no caso de uma *entity* (tabela) **Funcionário**, que contém nome, cargo e salário. Entretanto, podemos desejar outras informações deste funcionário, como idade, sexo e peso. Nesse caso, podemos ter uma outra *entity* (tabela), chamada **Pessoa**, com esses campos.

Relacionamento um-para-muitos e muitos-para-um:

As anotações para essas relações são, respectivamente, `@OneToMany` e `@ManyToOne`.

Como exemplo, pense no caso de um livro que pode ter apenas um autor. Considere que o livro terá apenas o José da Silva Figueiredo como autor, observe que é uma regra de negócio. Entretanto, um autor pode ter muitos livros publicados. Isso significa que o currículo do José da Silva Figueiredo apresentará todos os livros que escreveu.

A partir desse exemplo, dizemos que a *entity* **Livro** possui um relacionamento `@ManyToOne` com a *entity* **Autor**. Por outro lado, a *entity* **Autor** possui um relacionamento `@OneToMany` com a *entity* **Livro**.

Relacionamento muitos-para-muitos:

Continuaremos utilizando o exemplo do livro mencionado anteriormente. Nesse caso, por regra de negócio, um livro pode ter muitos autores e muitos autores podem ter muitos livros. Aqui, utilizamos a anotação `@ManyToMany` para caracterizar essa relação entre a *entity Livro* e a *entity Autor*.

Referências bibliográficas

COELHO, H. **JPA Eficaz**: as melhores práticas de persistência de dados em Java. São Paulo: Casa do Código, 2014.

TEORIA EM PRÁTICA

Considere o seguinte caso: você foi contratado (a), recentemente, por uma empresa do ramo de desenvolvimento de aplicações web. A empresa está por iniciar um novo projeto de uma aplicação para fins comerciais. Trata-se de um aplicativo de loja virtual, que possui sua versão externa, ou seja, para o cliente interagir e realizar suas compras, bem como sua versão interna, com o qual a empresa poderá realizar cadastro de produtos, gerar relatórios, entre outras ações.

Você foi chamado (a) para realizar o levantamento dos requisitos não funcionais, logo, será necessário indicar quais as tecnologias de desenvolvimento web deverão fazer parte do dia a dia dos desenvolvedores, de modo que a entrega do produto ocorra dentro do prazo estabelecido pelo cliente contratante.

Considerando que a empresa utilizará Java Web no desenvolvimento da aplicação, reflita como escolheria a biblioteca ou *framework* de persistência de dados entre o JDBC, JPA e *Hibernate*. Desse

modo, aponte as características entre essas bibliotecas e como se relacionam.

Para conhecer a resolução comentada proposta pelo professor, acesse a videoaula deste *Teoria em Prática* no ambiente de aprendizagem.

LEITURA FUNDAMENTAL

Indicações de leitura

Indicação 1

No livro *Rich Internet Applications e Desenvolvimento Web para Programadores*, entre as páginas 651 e 659, o autor apresenta o processo para criação e execução de uma aplicação web, utilizando *Java Server Faces (JSF)* e persistência de dados.

Para realizar a leitura, acesse a plataforma Biblioteca Virtual e busque pelo título da obra no parceiro *Biblioteca Virtual*.

DEITEL, P. J.; DEITEL, H. M. A. *Rich Internet Applications e desenvolvimento Web para programadores*. São Paulo: Pearson Prentice Hall, 2009.

Indicação 2

No livro *Projeto de Banco de Dados*, entre as páginas 11 e 36, o autor apresenta o conceito de entidade relacionamento, bem como os conceitos de cardinalidade e exemplos de diagramas de entidade relacionamento.

Para realizar a leitura, acesse a plataforma Biblioteca Virtual e busque pelo título da obra no parceiro *Biblioteca Virtual 3.0_Pearson*.

HEUSER, C. A. **Projeto de banco de dados**. 6. ed., v. 4. Porto Alegre: Bookman, 2017.

QUIZ

Prezado aluno, as questões do Quiz têm como propósito a verificação de leitura dos itens *Direto ao Ponto, Para Saber Mais, Teoria em Prática e Leitura Fundamental*, presentes neste *Aprendizagem em Foco*.

Para as avaliações virtuais e presenciais, as questões serão elaboradas a partir de todos os itens do *Aprendizagem em Foco* e dos slides usados para a gravação das videoaulas, além de questões de interpretação com embasamento no cabeçalho da questão.

1. O _____ é uma especificação de baixo nível se comparada ao _____. A proposta do desenvolvimento do _____ é de ser uma API para interagir com um banco de dados usando SQL puro, enviando consultas e recuperando resultados. Ao usar o _____, cabe a você traduzir o conjunto de resultados vindos do banco em objetos Java.
 - a. JDBC; JPA; JDBC; JDBC.
 - b. JPA; JDBC; JPA; JPA.
 - c. JPA; JDBC; JDBC; JDBC.
 - d. JDBC; JPA; JPA; JDBC.
 - e. JDBC; JPA; JDBC; JPA.

2. O _____ é um *framework* Java que implementa _____ do _____ para persistência de dados. Trata-se de uma ferramenta _____ de código aberto e leve, cujo objetivo é de simplificar o desenvolvimento de aplicações Java para interagir com o banco de dados.

- a. *Hibernate*; as especificações; JPA; ORM.
- b. JDBC; os métodos; Java; Java.
- c. *Hibernate*; os métodos; Java; JPA.
- d. JDBC; as especificações; JPA; Java.
- e. JDBC; as especificações; Java; ORM.



GABARITO

Questão 1 - Resposta A

Resolução: JDBC é uma interface de programação de baixo nível entre um aplicativo Java e um banco de dados. Desse modo, usamos o JDBC para obter uma conexão com o banco de dados e, a partir dessa conexão, podemos emitir comandos SQL puro. Logo, para utilizar o JDBC, é importante que o desenvolvedor tenha um conhecimento de SQL. O JPA é uma especificação de alto nível, se comparada ao JDBC, uma vez que suas implementações são executadas acima do JDBC.

Questão 2 - Resposta A

Resolução: O JPA é um conjunto de especificações para persistência e acesso ao banco de dados. O JPA necessita de uma implementação de *Object Relational Mapping* (ORM) para trabalhar e persistir objetos em Java. Existem várias implementações do JPA e a principal delas é o *Hibernate*, um ORM que automatiza o mapeamento dos objetos da aplicação na base de dados.



BONS ESTUDOS!