



PHP COM MYSQL

TRILHA DO ALUNO

SUMÁRIO

MÓDULO 1 – CONFIGURANDO O AMBIENTE DE DESENVOLVIMENTO	3
[Introdução] Configurando o ambiente de desenvolvimento	3
[Mídia 1] Introdução ao PHP	3
[Mídia 2] Introdução ao MySQL	8
MÓDULO 2 – DEFINIÇÃO E MANIPULAÇÃO DOS DADOS EM MYSQL	16
[Introdução] Definição e manipulação dos dados em MySQL	16
[Mídia 1] O servidor de banco de dados MySQL	16
[Mídia 2] SQL – Linguagem de consulta estruturada	26
[Síntese] Módulo 2	32
MÓDULO 3 – REALIZAÇÃO E INTEGRAÇÃO DOS DADOS	33
[Introdução] Realização e integração dos dados	33
[Mídia 1] Estrutura MVC	34
[MINIQUIZ] Variáveis no PHP	44
[MÍDIA 2] Conexão com o MySQL	44
[FÓRUM] Layout final	60
[Fechamento] Módulo 3	61

MÓDULO I – CONFIGURANDO O AMBIENTE DE DESENVOLVIMENTO

[INTRODUÇÃO] CONFIGURANDO O AMBIENTE DE DESENVOLVIMENTO

Bem-vindo (a) ao **Módulo 1 - Configurando o ambiente de desenvolvimento**, conforme as funcionalidades e características do sistema web a ser codificado, do curso de PHP com MySQL. Esse módulo é composto por 2 mídias interativas, compostas por podcast, tutorial e outros recursos que visam deixar o conteúdo ainda mais interessante e didático para você.

[Mídia 1] Introdução ao PHP

[Mídia 2] Introdução ao MySQL

Boa jornada de estudos!

[MÍDIA 1] INTRODUÇÃO AO PHP

Durante o curso iremos desenvolver a competência “Programar sistemas web utilizando a linguagem PHP, com a integração ao banco de dados utilizando o servidor MySQL”, porém nesse módulo e nessa mídia, iniciaremos pela introdução à linguagem PHP.

Bons estudos!

Vamos lá, só clicar em [Mídia 1] Introdução ao PHP

Olá, Aluno!

Bem-vindo (a) à mídia do Módulo 1 – Introdução ao PHP, como o próprio nome já sugere o assunto que abordaremos aqui, vamos ao que interessa não é mesmo?!

Bons estudos!

Palavra do autor

Caso queira ler a transcrição da fala do autor, clique no arquivo de PDF abaixo:

Transcrição do Podcast

Autor: Vitor Gabriel Martines Weinfortner

Olá aluno, meu nome é Vitor Gabriel Martines Weinfortner e eu sou o autor do seu curso de PHP com MySQL, estou passando por aqui para desejar as boas-vindas ao Módulo 1 e apresentar um breve panorama do que veremos por aqui.

Seja bem-vindo ao curso de PHP com MySQL, neste curso iremos desenvolver a competência de programar sistemas web utilizando a linguagem PHP, com a integração ao banco de dados utilizando o servidor MySQL.

Você conhece ou já trabalhou com alguma dessas tecnologias?

Se não, não tem problema! Você está no lugar certo!

Estamos aqui para ensiná-lo passo a passo de como iniciar a trabalhar com essas tecnologias e desenvolver um sistema web desde a análise até a persistência dos dados.

Para entendermos melhor como o PHP funciona, vamos começar aprendendo um pouco da sua história. A linguagem de programação PHP foi criada em 1994 por Rasmus Lerdorf que, em meados de 1995 criou um sistema para ter estatísticas sobre acessos em seu currículo on-line, liberou os códigos para o público, quando foi aprimorando com novas funcionalidades. Ela é uma linguagem de script Open Source, o que significa que pode ser acessada abertamente pelo público, podendo modificá-lo e distribuí-lo conforme suas necessidades.

O PHP é utilizado principalmente no ambiente web, instalado em servidores, os quais resultam em páginas com conteúdo dinâmico, mas também pode ser executado e interpretado de outras maneiras, como IOT (Internet of Things, ou “Internet das Coisas”), linha de comando, aplicativos desktop e muitos outros.

É também amplamente utilizado no desenvolvimento web, pois permite criar sites dinâmicos, possibilitando uma interação com o usuário através de formulários, parâmetros da URL e links. Outra característica importante é que é uma linguagem interpretada, isso significa que, quando alteramos o código, não precisamos compilá-lo novamente para que ele seja legível, porém significa também que algum programa precisa ler o nosso arquivo de texto, interpretar aquelas instruções e fazer as execuções no sistema operacional.

Desta maneira o servidor interpreta as informações e retorna somente o HTML puro, não expondo o código fonte para o cliente. Isso pode ser útil quando o programa está lidando com senhas ou qualquer tipo de informação confidencial.

Espero que goste do curso, bons estudos!

Após ouvir a palavra do autor Vitor Gabriel Martines Weinfortner e dar uma refletida sobre o assunto, você consegue perceber o quanto essa linguagem de programação possibilitou e possibilita até hoje avanços tecnológicos e, conseqüentemente, otimização de processos cotidianos?!

Mas saiba que nem sempre foi assim...

O início ...

No início era tudo “mato” como diriam...

Brincadeiras à parte, quando a computação dava seus primeiros passos, era algo de se pensar muito distante do que conhecemos hoje, em muitos aspectos.

Quando chegaram nas máquinas, eram usadas apenas para processos que consideramos simples hoje em dia, como cálculos por exemplo.

Após a criação dos computadores pessoais, o uso para edição de texto, armazenamento e acesso a dados eram as principais funções utilizadas.

Com a constante evolução do meio digital, viu-se a necessidade de se conectar com informações e, principalmente, com pessoas.

E os avanços tecnológicos vem proporcionando até mais que isso ... internet das coisas, robótica, big data...

Consequentemente, as linguagens também se aprimoraram muito até chegarmos na complexidade de códigos que temos atualmente.

Então, vamos ao que interessa não é mesmo... dar os primeiros passos para sair programando em PHP!

Se concentre e mãos à obra!

Nota importante!

A maioria dos conteúdos sobre tecnologia, inclusive sobre PHP com MySQL, está na língua inglesa, pois é por meio dela que a comunidade de tecnologia se comunica e produz a maior parte dos conteúdos.

É importante que você se aproxime desse universo!

Como apoio você pode utilizar o Google Tradutor <<https://translate.google.com.br/?hl=pt-BR>> ou outros aplicativos de tradução de texto.

Como funciona a arquitetura cliente/servidor

Para que você possa programar sistemas web você precisará dar o primeiro passo, que é:

Configurar o ambiente de desenvolvimento, conforme as funcionalidades e características do sistema WEB a ser codificado.

É nesse ambiente de desenvolvimento configurado que você vai organizar e programar os requisitos/parâmetros necessários para chegar ao objetivo do projeto, que pode ser qualquer sistema web.

Para iniciar nosso desenvolvimento precisamos entender um pouco melhor o funcionamento da arquitetura cliente/servidor, que é de grande importância para o desenvolvimento WEB, é no servidor que o PHP vai ser interpretado pelo sistema operacional para executar as ações que necessitamos.

Para tal, além de aprender o conceito, vamos configurar nosso servidor necessário para a programação utilizando o PHP com MySQL.

No ambiente WEB, a arquitetura mais comum para acesso e manutenção de informações é a arquitetura Cliente/Servidor, como citado acima. Esse modelo é constituído por dois ou mais computadores, interagindo de modo que um oferece os serviços aos outros, permitindo que os recursos sejam compartilhados aos usuários para que acessem informações e serviços ao mesmo tempo, de qualquer lugar com acesso ao servidor necessário.



É no servidor que fica toda nossa regra de negócio e validações necessárias para as aplicações web.

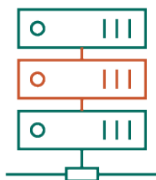
Cliente e Servidor

A arquitetura cliente/servidor é constituída por duas partes independentes, que interagem entre si:



CLIENTE: FRONT-END

O cliente, o que chamamos na programação de Front-end, pode ser qualquer computador conectado ao sistema através da rede, ele é o responsável por iniciar a comunicação. Corresponde ao usuário que está em busca de realizar uma ação ou acessar alguma informação e solicita ao servidor que execute um processo específico por meio de uma requisição. Tem como principais características ser a parte ativa da comunicação e não compartilhar nenhuma informação com o servidor. É a interface de comunicação com o usuário final.



SERVIDOR: BACK-END

O servidor, conhecido também por back-end, tem como principais características ser a parte reativa da comunicação, apenas reagindo a demanda solicitada. É o provedor de acesso e uma central de dados responsável pela manutenção e segurança da informação, atende a diversos clientes simultaneamente ficando disponível para todo cliente que necessita. É no servidor que as regras da aplicação ficam localizadas.

Para fazer uma analogia com uma situação real, um exemplo pode ser um atendente de alguma instituição bancária, que fica parado aguardando algum cliente chegar com alguma solicitação específica, que pode ser um saque, uma consulta de extrato bancário etc. Somente após o cliente expor sua necessidade é que o atendente, vai executar uma ação, assim como num servidor, que nesse caso o servidor é o atendente.

Protocolo HTTP

No contexto de desenvolvimento WEB, o cliente inicia a comunicação através de uma requisição WEB com o protocolo HTTP, como por exemplo o GET que busca alguma informação e o POST que é utilizado para gravar ou alterar alguma informação no servidor, ao qual vamos entender melhor mais para frente, fazendo com que a informação chegue ao servidor.

Esse protocolo garante principalmente a segurança da aplicação, seguindo nosso exemplo, um cliente não poderá sacar o valor desejado no banco se for com o cartão errado, ou se não for liberado para acessar aquela conta. Vamos utilizar essas requisições no decorrer do nosso projeto, para realizar consultas e persistir os dados no nosso servidor.

[MOD 1] Atividade: Lógica de Programação Cliente/Servidor

De acordo com seu conhecimento atual em lógica de programação, juntamente com o conceito de cliente/servidor, organize as sentenças a seguir na ordem correta, por meio do exemplo dado do atendente do banco com o cliente.

- () Solicitar o cartão identificando conta a ser utilizada e receber a opção da operação a ser processada.
- () Validar dados da conta e solicitar senha de liberação.
- () Receber através da interface o valor que o cliente deseja sacar.
- () Processar o valor solicitado validando de acordo com o saldo em conta e informar ao usuário o resultado da validação. (Sucesso ou Falha)
- () Liberar o valor solicitado se a operação for válida.

Segurança, acessibilidade e centralização

Atualmente, a arquitetura cliente/servidor é indispensável, em um mundo onde o trabalho remoto é cada vez mais forte principalmente em grandes empresas, é de extrema importância se conectar com outras pessoas ou informações de maneira rápida e segura. Entre as principais vantagens dessa arquitetura temos:

Segurança

O servidor deve garantir a segurança das informações, permitindo acesso somente para quem for conveniente e garantindo que as informações não sejam perdidas durante o processo. Se o disposi-

tivo usado pelo cliente apresentar algum problema, as informações continuam disponíveis no servidor.

Acessibilidade

As tarefas podem ser executadas sem a monopolização dos recursos, ou seja, o acesso pode ser feito de vários pontos ao mesmo tempo.

Centralização

Sabendo que os sistemas de informação crescem e evoluem com o passar do tempo, sendo necessário atualizações e manutenções, o servidor de acesso possibilita de maneira muito mais fácil modernizá-lo quando necessário. Para atualizar as aplicações locais, é necessário que todos os usuários interessados façam o download da aplicação e executem dispositivo por dispositivo, podendo ocorrer situações em que o sistema local fica desatualizado em relação aos demais. O servidor traz uma grande vantagem nessa questão, ao invés de atualizarmos diversos dispositivos, podemos atualizar e alterar apenas o servidor, garantindo que os usuários mesmo não sendo afetados continuem a acessar o sistema na versão desejada.

[MOD 1] Atividade: Função das camadas Cliente e Servidor

Agora, com base nos conhecimentos que construiu até aqui, vamos refletir um pouco mais sobre a arquitetura cliente-servidor? Lembre-se que esses conhecimentos são de extrema relevância para que você configure o ambiente de desenvolvimento que culminará na programação de sistemas web em linguagem PHP.

[MOD 1] Atividade: Função das camadas Cliente e Servidor

Assinale as alternativas verdadeiras de acordo com a função e responsabilidade de cada camada (cliente e servidor) nas aplicações WEB.

- ☐ () Na nossa arquitetura de desenvolvimento, o cliente é o responsável por responder uma solicitação recebida e validar se quem solicitou tem a permissão necessária para acessar esses recursos.
- ☐ () O cliente possui somente a interface de comunicação com o usuário.
- ☐ () O servidor armazena as regras de negócio da aplicação.
- ☐ () O servidor, na maioria das vezes, inicia a comunicação solicitando a informação necessária para o cliente.

[MÍDIA 2] INTRODUÇÃO AO MYSQL

Depois de entender como funciona a arquitetura cliente/servidor, vamos para a próxima etapa da nossa aplicação WEB e uma das mais importantes pois é nela que todas as informações do nosso sistema são salvas, o servidor de banco de dados.

Vamos lá, só clicar em [Mídia 2] **Introdução ao MySQL**

Bem-vindo (a) a nossa segunda mídia do **Módulo 1 - Configurando o ambiente de desenvolvimento, conforme as funcionalidades e características do sistema web a ser codificado.**

Bons estudos!

O Programa MySQL

Você aluno, conhece ou já trabalhou com alguma dessas tecnologias?

No nosso caso, iremos utilizar o servidor de banco de dados MySQL, muito utilizado no desenvolvimento WEB, por ser um servidor completo e ao mesmo tempo fácil de usar.

O programa MySQL é de licença dupla, ou seja, os usuários podem escolher entre utilizar a versão gratuita e de código livre, assim como o PHP, ou podem comprar a versão comercial, que é mais completa e conta com o suporte dos mantenedores do software.

Mas como seria isso?

Vou utilizar a mesma referência do atendente do banco para exemplificar. Imagine quando o cliente vai abrir sua conta no banco, para utilizar os serviços do banco ele precisa criar um cadastro e depositar algum valor em dinheiro para utilizar novamente quando necessário, certo?

E onde essas informações ficam guardadas?

Quando o cliente for usar o valor depositado, o banco precisa validar diversas informações antes de liberar a quantia solicitada, pois essas informações ficam salvas no banco de dados, para estarem disponíveis para alterações e consultas. Para cada solicitação, há um registro salvo na camada de banco de dados, organizados em várias tabelas, que representam os nossos objetos na lógica de programação.

Complicou? Ficou confuso?

Calma, vamos entender melhor como funciona nessa mídia e no futuro próximo isso vai ser muito simples para você.

PK (Primary Key) e FK (Foreing Key)

PK (Primary Key): chave primária

Um banco de dados pode ser entendido como uma coleção de dados, estruturados em tabelas contendo as informações que necessitamos salvar para correta execução do sistema, no nosso exemplo, existe uma tabela de cliente, que salva informações pessoais como:

- Nome;
- Documentos pessoais
- e estado civil por exemplo.

Lembrando que toda tabela deve ter sua PK (Primary Key) ou chave primária, que é o que identifica o registro na tabela, para melhor entendermos podemos pensar como o nome do registro, o que permite acessá-lo mais fácil.

FK (Foreing Key): chave estrangeira

Em outra tabela separada, podemos ter informações da conta, como:

- Número;
- Agência;
- Saldo, etc.

E o mais importante, uma referência para o cliente que possui a conta, o que chamamos de FK (Foreing Key) ou chave estrangeira, é a partir dela que sabemos a qual cliente essa conta pertence. Observe a imagem:

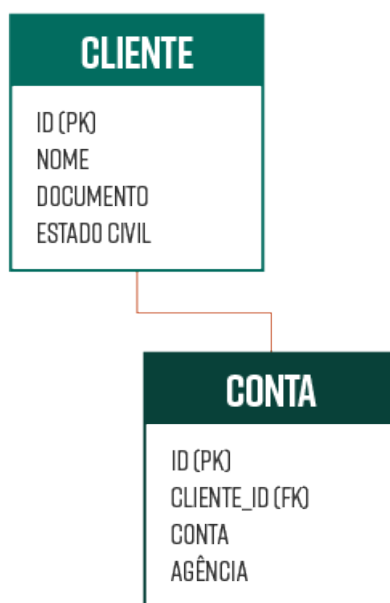


Figura 1: Modelo relacional de banco de dados, criado em <https://app.creately.com>

Para adicionar, acessar e processar dados armazenados em um banco de dados de um computador, você necessita de um sistema de gerenciamento de bancos de dados como o Servidor MySQL, o qual vamos utilizar, é através desse gerenciador que vamos gravar os nossos clientes, informações de conta, transações e utilizá-las conforme a nossa necessidade.

Configurações iniciais

Agora, vamos começar a configurar nosso ambiente de desenvolvimento?

No nosso caso, iremos utilizar o MySQL de código livre juntamente com o PHP, que nessa parte do curso será usado para testar o nosso servidor.

Como dito anteriormente, o PHP é uma linguagem interpretada, ou seja, precisa de algum programa para ler e interpretar o código. Para isso, vamos instalar o Xampp, que é um utilitário que cuida da

instalação e configuração inicial do nosso servidor, utilizando o servidor Apache que fica responsável por interpretar o PHP, podemos entender o Apache como um entregador de encomendas virtuais, que quando acessado pela barra de endereço do navegador entrega os arquivos solicitados.

Vamos utilizar também o PHPMyAdmin, que é uma interface para a interação com o MySQL, ao qual podemos criar e acessar nosso banco de dados.

Disponível para download, onde conseguimos escolher a versão de acordo com o sistema operacional que você está utilizando.

Instalando o Xampp

Agora, assista ao vídeo com o passo a passo para instalação do Xampp.

Caso queira ler a transcrição do vídeo, clique no arquivo de PDF abaixo:

Transcrição do vídeo

Autor: *Vitor Gabriel Martines Weinfortner*

Olá aluno!

Nesse vídeo, iremos mostrar a primeira parte para a configuração do nosso ambiente de desenvolvimento, a instalação do software Xampp. Ao acessar o link que disponibilizamos no módulo 1, você deve escolher a versão de acordo com o sistema operacional que você está utilizando. O tutorial exemplificado na sequência foi feito com base no sistema operacional Windows.

Após executar o arquivo que você baixou de acordo com seu sistema operacional, a tela de boas-vindas do Xampp será exibida. Podemos selecionar o botão next para prosseguir com a instalação.

A segunda tela exibida é a dos componentes que serão instalados juntamente com o Xampp. Na parte dos servidores temos o Apache para o back-end, o MySQL para o banco de dados entre outros. Nas linguagens de programação PHP e Pearl, podemos pressionar o botão next para seguir com a instalação.

Na terceira tela exibida podemos selecionar a pasta em que o Xampp será instalado. Por padrão, o caminho é **C:\xampp**, sugiro que mantenhamos esse padrão, pois ele será usado no decorrer do curso.

Para prosseguir a instalação pressionamos o botão **next**.

Na próxima tela selecionamos a linguagem de instalação e execução do Xampp, temos duas opções por enquanto o inglês e o alemão. Selecione a linguagem de sua preferência e podemos apertar o botão next para prosseguir a instalação.

A próxima tela exibida é apenas de informação do Xampp, vou desmarcar a opção **learn more**, para evitar que a página Web do Xampp seja exibida.

Para prosseguir na estação apertamos o botão **Finish**. Assim que finalizarmos as configurações iniciais, a página de informação do Xampp é exibida, avisando que ele será instalado no seu computador. Para aceitar a instalação apertamos o botão **next**.

É isso aí!

Agora é só aguardar a instalação ficar pronta. Assim que terminar a instalação dos arquivos, a página avisando que a instalação está completa será exibida, com a função de iniciar automaticamente o painel de controle ou não, no nosso caso, vamos iniciar o painel de controle e apertar o botão **Finish**.

Após aparecer o painel de controle do Xampp, é possível verificar os serviços disponíveis, bem como iniciá-los por meio do botão **start**, iniciar em inglês. Para parar basta apertar o botão **stop**.

Temos também as opções de configuração para cada serviço, no nosso caso vamos apenas iniciar serviço Apache para o back-end, utilizando o PHP e o serviço MySQL para o banco de dados.

Podemos verificar na tela de log a mudança de status dos serviços, agora estão com o status running, que significa executando em inglês.

[MOD 1] Atividade: Instalando e configurando ambiente de programação

Agora que o ambiente está configurado, podemos começar a desenvolver, mas antes vamos recapitular os processos de instalação e configuração realizando a atividade a seguir:

[MOD 1] Atividade: Instalando e configurando ambiente de programação

Imagine que você trabalha numa empresa de Tecnologia da Informação e está recebendo um estagiário que participa de um programa de estudos e incentivo à formação de profissionais na área de Tecnologia da Informação.

Seu gestor sabe de sua experiência e habilidades, por isso sugeriu que você ensinasse ao seu colega como instalar e configurar o ambiente de desenvolvimento. Lembre-se que a colaboração e o compartilhamento de conhecimentos são habilidades muito relevantes para as empresas.

Com base na situação apresentada, ordene o passo a passo para orientar seu colega adequadamente.

- () Explicar para o colega a importância de ter um ambiente de programação instalado e configurado.
- () Clicar no link que disponibiliza o Xampp.
- () Baixar o arquivo de acordo com o sistema operacional utilizado na empresa.
- () Executar o arquivo baixado para instalação.
- () Indicar o local que o arquivo será salvo.
- () Finalizar a instalação.
- () Abrir o painel de controle do Xampp.
- () Iniciar os serviços de Apache e MySQL.

Criar e editar arquivos em PHP

Antes de fecharmos esse módulo e prosseguirmos para o próximo, uma última ação: baixar um edi-

tor de texto para PHP. Para criar e editar os arquivos em PHP, será utilizado o Sublime, que é um editor de texto gratuito, que reconhece nosso código e apresenta de maneira fácil de se compreender, porém fique à vontade para utilizar o editor de sua preferência.

Fazendo o download do Sublime

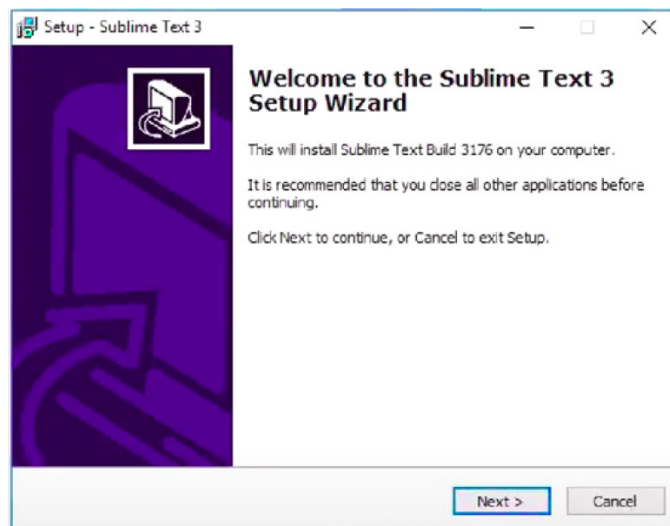
Atenção! Mantenha o Xampp aberto e com o Apache e MySQL funcionando!

Passo 1

Clique no link <https://www.sublimetext.com> para baixar a versão de acordo com o seu sistema operacional. O utilizado nesse curso é o sistema operacional Windows.

Passo 2

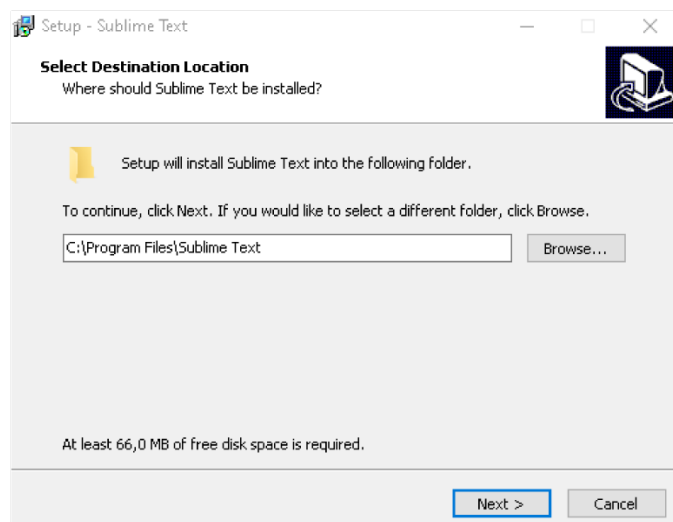
Execute o arquivo baixado e confirme a instalação através do botão **Next**, como na imagem a seguir.



Tela inicial da instalação do sublime.

Passo 3

Após essa etapa, será exibido a tela para escolha do local onde o Sublime será instalado, escolha o local de sua preferência e selecione a opção **Next** para prosseguir com a instalação.



Passo 4

Clique em **Install** e depois em **Finish** que confirma nossa instalação.

Passo 5

Acesse o local onde salvou o arquivo baixado, instale e execute-o.

Passo 6

Ao executar o Sublime, uma tela vazia será exibida, onde vamos digitar o código como na imagem abaixo:

Arquivo inicial, formulario.php

No arquivo criado, podemos verificar alguns detalhes importantes, como as tags de abertura e fechamento do bloco de código em PHP, utilizamos a tag `<?php` para iniciar e `?>` para finalizar o nosso bloco, verificamos também a função **echo**, que envia para o cliente qualquer informação, nesse caso exibe na tela o texto "Olá Mundo!".

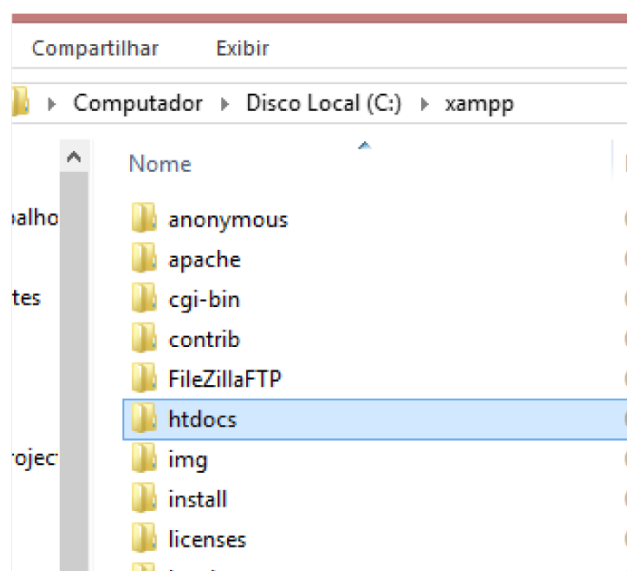
```
<!DOCTYPE html>
<html>
  <body>
    <?php
      echo "Olá Mundo";
    ?>
  </body>
</html>
```

Para salvar – Passo 6.1

Selecionar a opção **Arquivo (file)** -> **Salvar como (save as)**

Para salvar - Passo 6.2

Selecionar a pasta **C:\xampp** e encontrar a pasta htdocs (veja na imagem abaixo)



Pasta **htdocs**, disponível na pasta de instalação do Xampp.

Para salvar – Passo 6.3

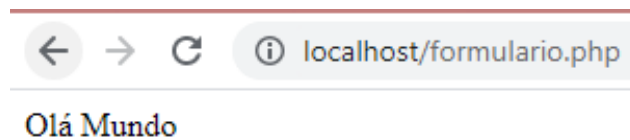
Salve o arquivo com o nome **formulario.php**.

Para salvar – Passo 6.5

Após salvar o arquivo, pode acessá-lo, já interpretado, digitando o link **http://localhost/formulario.php** no seu navegador de internet.

Atenção! Caso apareça erros referente ao certificado de segurança tente acessar de uma guia anônima.

Após seguir todos os passos apresentados, você visualizará a imagem abaixo!



Arquivo PHP após ser interpretado pelo servidor.

É isso aí aluno, seu servidor já está configurado e pronto para as próximas etapas do curso, simples, não é?

Até o Módulo 2!

Nosso projeto será criado dentro da pasta htdocs do xampp, para que possamos acessá-lo por meio do navegador por meio do link indicado.

MÓDULO 2 – DEFINIÇÃO E MANIPULAÇÃO DOS DADOS EM MYSQL

[INTRODUÇÃO] DEFINIÇÃO E MANIPULAÇÃO DOS DADOS EM MYSQL

Olá, Aluno!

Bem-vindo (a) ao **Módulo 2 – Definição e manipulação dos dados em MySQL**, de acordo com as instruções SQL e as funcionalidades do sistema web a ser desenvolvido, do curso de PHP com MySQL. Nesse módulo aprofundaremos mais os conhecimentos acerca do conceito de arquitetura Cliente/Servidor visto no módulo anterior, bem como características, instalação e configuração além de consulta e transação de dados e muito mais. O módulo é composto por 2 mídias interativas:

[Mídia 1] O servidor de banco de dados MySQL

[Mídia 2] SQL - Linguagem de consulta estruturada

Boa jornada de estudos!

[MÍDIA 1] O SERVIDOR DE BANCO DE DADOS MYSQL

Nessa mídia, construiremos conhecimentos acerca da definição e manipulação dos dados em MySQL, de acordo com as instruções SQL e as funcionalidades do sistema web a ser desenvolvido.

Vamos lá, só clicar em **[Mídia 1] O servidor de banco de dados MySQL**

O servidor de banco de dados MySQL

Olá aluno! Seja bem-vindo a nossa primeira mídia do Módulo 2 sobre definição e manipulação dos dados em MySQL.

Vamos lá!

Introdução

Nessa etapa do curso você definirá e manipulará os dados em MySQL, de acordo com as instruções SQL e as funcionalidades do sistema web a ser desenvolvido. Para que essa ação seja possível, vamos entender melhor como funciona o servidor de banco de dados MySQL, junto com a linguagem de consulta SQL, usados para a persistência dos dados da nossa aplicação, ou seja, a partir dessas ferramentas que vamos criar nosso banco de dados, salvar, editar e excluir informações utilizadas no nosso projeto.

Para isso, vamos iniciar a estrutura básica de um e-commerce de produtos variados. Dentro dessa estrutura básica alguns parâmetros precisam ser seguidos e existem requisitos necessários.

Acesse a atividade abaixo e assinale quais funcionalidades/requisitos serão necessários para iniciar a estrutura básica de um e-commerce.

[MOD 1] Atividade: Estrutura básica de um sistema de vendas on-line

Hoje em dia é muito comum comprarmos e vendermos itens on-line, onde temos acesso a uma enorme quantidade de anúncios e compradores para os produtos de nosso interesse. Imaginando esse cenário de vendas on-line, podemos visualizar que é necessário manter a consistência das informações que são recebidas e processadas. E para manter o negócio ativo e funcional, afinal, não queremos que seja enviado o produto errado para o comprador, ou que o vendedor receba um valor diferente do que foi acordado, não é mesmo?!

Com base nesse raciocínio, realize a atividade a seguir:

[MOD 2] Atividade: Estrutura básica de um sistema de vendas on-line

De acordo com a situação apresentada, para a elaboração de uma estrutura básica de e-commerce de produtos variáveis torna-se importante que:

- I – Deve conter funcionalidades para criar, editar e excluir usuários e produtos.
- II – Ter controle de estoque, para evitar a venda de um produto sem estoque suficiente, por exemplo.
- III – Informações de endereço e entrega.
- IV – Limite máximo de compras para cada usuário.

Assinale abaixo a alternativa correta:

- a) Estão corretas as assertivas I, III e IV.
- b) Estão corretas as assertivas I, II e III.
- c) Estão corretas todas as assertivas.
- d) Estão corretas as assertivas I, III.

O servidor de banco de dados MySQL

Você se recorda quando falamos sobre o caso do banco?

Retomando... um cliente faz uma solicitação (extrato, por exemplo) a um servidor e essa relação cliente-servidor é que vai gerar o resultado, ou melhor a informação ou produto solicitado. Mas...

Onde essas informações são armazenadas? Como se sabe de qual conta o extrato deve ser gerado ou então de qual pessoa?

Para isso existe uma ferramenta muito importante e se chama: **Banco de dados!**

Banco de dados

Essa mesma ferramenta torna-se relevante para o correto funcionamento do nosso e-commerce, trabalhado na lâmina anterior, lembra? Pois então, precisamos de alguma ferramenta que fique res-

responsável por guardar as informações recebidas e processadas pelo nosso site, essas informações devem ficar disponíveis sempre que o site estiver on-line.

Vamos imaginar uma venda sem um banco de dados, nessa situação a cada cadastro ou venda o vendedor anotaria em um caderno o registro da operação, ao comprar um produto, ele anotaria no caderno o nome do produto, valor e quantidade, na venda a mesma situação: anotaria no caderno o valor de venda e a quantidade vendida. Porém, antes de realizar a venda, o vendedor teria que percorrer essa grande quantidade de informações escritas no caderno, para validar se o produto ainda está disponível para comercialização por exemplo, correndo o risco de vender produtos que não foram comprados ou por um valor abaixo do que comprou, gerando prejuízo ao invés de receita para a loja on-line. Com isso, provavelmente o negócio não duraria muito tempo com todo esse trabalho e incerteza não é mesmo?!

Sistema de manutenção de registros

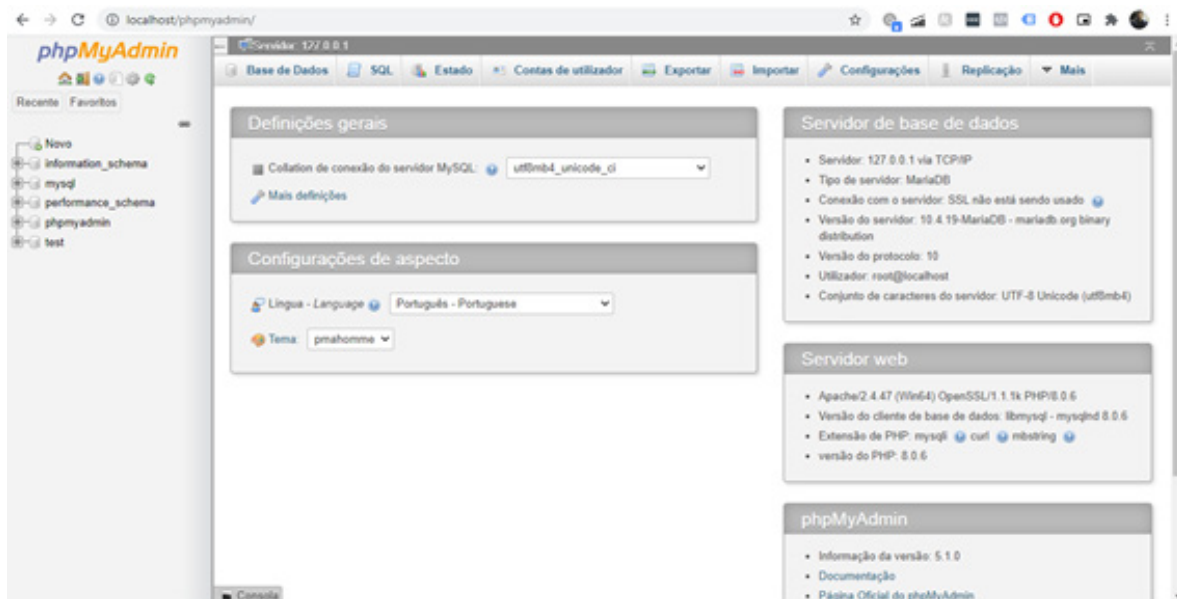
Para resolver desafios como os apresentados anteriormente e tantos outros, surgiu o banco de dados, que nada mais é do que um sistema de manutenção de registros. Podemos entender o banco de dados como uma representação digital de um caderno de anotações e registros, com os dados organizados de uma maneira fácil e rápida de se manipular. A ferramenta responsável por manter esse banco de dados é o servidor de banco de dados MySQL, é através dele que podemos criar novos registros, acrescentar dados a registros existentes, buscar, editar e excluir do nosso banco de dados.



Imaginando agora a situação da venda utilizando o banco de dados, ao realizar a compra do produto o mesmo é cadastrado no site e ao realizar a venda, o back-end do nosso sistema, que como foi visto no módulo 1, também é chamado de Servidor onde ficam as regras de negócio, valida em questão de milésimos de segundos os registros do banco de dados, fazendo validações como: O produto tem estoque suficiente? Qual é o valor de venda do produto? Esse cliente está liberado para fazer compras na loja? Para essas e outras funções, o banco de dados é tão importante.

Agora que entendemos a importância do banco de dados nos sistemas de informação, vamos começar a manipular os dados do nosso banco. A ferramenta que vamos utilizar para a manipulação dos dados é o phpMyAdmin, que como dito anteriormente, é a interface/gerenciador desenvolvida em PHP que disponibiliza acesso ao servidor de banco de dados MySQL, ao qual instalamos anteriormente junto com o Xampp.

O nosso gerenciador de banco de dados fica disponível através do link <http://localhost/phpmyadmin>, conforme imagem a seguir sempre que estiver em execução no Xampp.



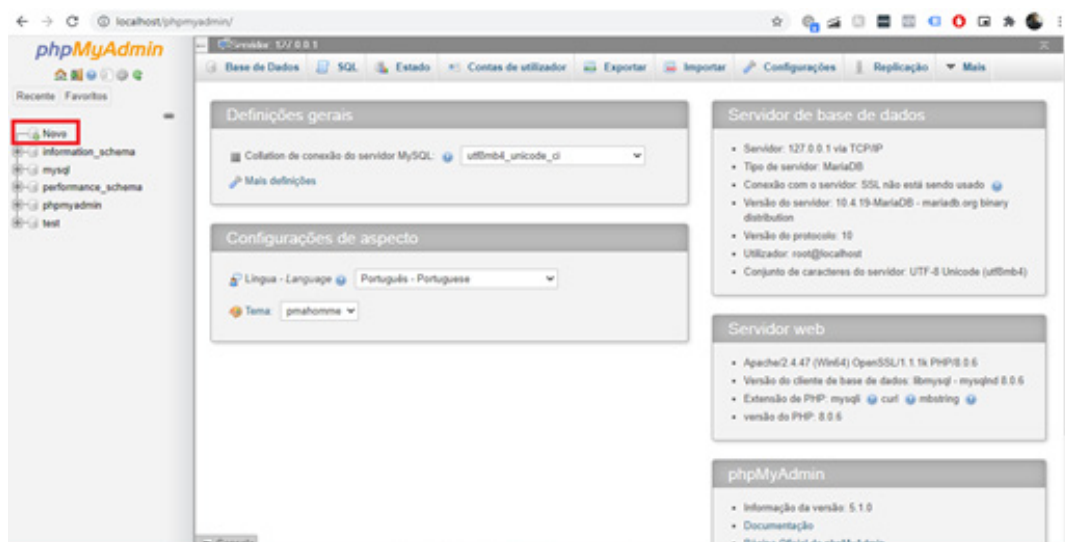
Painel administrativo do phpMyAdmin.

A imagem acima mostra o painel administrativo do phpMyAdmin, na barra lateral esquerda são listados os bancos de dados, por padrão, alguns bancos já vem criados, agora é a hora de iniciarmos o nosso banco de dados.

Iniciando o banco de dados



Para iniciar o banco de dados, siga os 2 passos a seguir.






Passo 1 - Clicar no botão **Novo**, conforme imagem no verso.



Passo 2 - Escolher o nome do banco de dados, no nosso caso e-commerce e clicar no botão **Criar**, conforme imagem no verso.

Base de Dados

 Criar base de dados 

Base de Dados	Agrupamento (Collation)	Acções
<input type="checkbox"/> information_schema	utf8_general_ci	 Verificar Privilégios
<input type="checkbox"/> mysql	utf8mb4_general_ci	 Verificar Privilégios
<input type="checkbox"/> performance_schema	utf8_general_ci	 Verificar Privilégios
<input type="checkbox"/> phpmyadmin	utf8_bin	 Verificar Privilégios
<input type="checkbox"/> test	latin1_swedish_ci	 Verificar Privilégios
Total: 5		

Pronto!

A nossa base de dados foi criada e já pode ser acessada, porém ela ainda está vazia não é mesmo? O que o nosso banco de dados deve conter para funcionar corretamente?

Na nossa primeira atividade definimos as seguintes funcionalidades:

- I – Deve conter funcionalidades para criar, editar e excluir usuários e produtos.
- II – Ter controle de estoque, para evitar a venda de um produto sem estoque suficiente, por exemplo.
- III – Informações de endereço e entrega.

Mas, será que outras funcionalidades são importantes? Como estruturamos esse banco de dados?

Modelagem do banco de dados

Modelagem do banco de dados é o primeiro passo para a construção do banco de dados. Agora, para a manutenção das nossas informações, vamos começar a construir nosso banco de dados que vai ser usado para a aplicação que iremos criar. Para modelá-lo devemos pensar sobre o nosso problema, que no cenário do curso é **desenvolver um e-commerce para vender um produto fictício**, de uma maneira que seja possível mapear a solução.

Primeiramente, pensando em um e-commerce, já imaginamos o nosso objetivo que é vender produtos, certo?

Nessa afirmação já temos dois dados importantes para o nosso mapeamento, a venda e o produto, esses dados abstraídos da nossa solução são os chamados **OBJETOS**, que você aprendeu nas aulas de algoritmos e lógica de programação. Cada um desses objetos, se transforma em uma tabela na nossa estrutura do banco de dados.

E qual informação nossas tabelas devem conter para o correto funcionamento do e-commerce?

Essas informações são também as colunas das nossas tabelas do banco de dados, essa relação fica mais clara na imagem de exemplo a seguir, que traz a tabela de usuários representando o objeto usuário. Temos nessa tabela as informações ID, nome, e-mail, senha e endereço que representam nossas colunas.

Tabela Usuarios				
ID	Nome	Email	Senha	Endereco
1	João	joao@email.com	123	São Paulo, centro, 000
2	Maria	maria@email.com	123456	Curitiba, Paraná, 000
3	Pedro	pedro@email.com	887766	Rio de Janeiro, 000

Clique [AQUI](#) para relembrar o conceito de objetos

A orientação a objetos é um paradigma de análise, modelagem e programação de sistemas de software que consiste na interação de objetos. Os objetos são entidades manipuladas pelo software, de maneira que representem uma informação da vida real de maneira simplificada, como por exemplo, um produto. Cada objeto contém suas propriedades específicas, que representam o objeto trabalhado, como por exemplo o objeto produto, pode possuir as propriedades de nome, cor, tamanho, etc. Dessa maneira, dentro do nosso software, teremos o produto 1, com o nome camiseta, de cor amarela e tamanho M, o produto 2, com o nome tênis, de cor azul e tamanho 40 por exemplo.

Lembrete importante do Módulo 1

Um banco de dados pode ser entendido como uma coleção de dados, estruturados em tabelas contendo as informações que precisamos salvar, para correta execução do sistema, no nosso exemplo, existe uma tabela de cliente, que salva informações pessoais como nome, documento e estado civil, lembrando que toda tabela deve ter sua PK(Primary Key) ou chave primária, que é o que identifica o registro na tabela. Em outra tabela separada, podemos ter informações da conta, como número, agência, saldo, etc e o mais importante, uma referência para o cliente que possui a conta, o que chamamos de FK (Foreign Key) ou chave estrangeira, é a partir dessa chave estrangeira que sabemos a qual cliente essa conta pertence.

Desenvolvimento do projeto

Vamos ao nosso objetivo: desenvolver um e-commerce para vender um produto fictício.

Para o nosso exemplo, vamos começar com a organização de dois objetos que serão inseridos no banco de dados: os produtos e os pedidos. Vamos iniciar criando a tabela de pedidos, conforme imagem a seguir.

PEDIDOS	
ID (PK)	INT
VALOR	DOUBLE
STATUS	VARCHAR

Na imagem acima, podemos visualizar de forma mais clara a nossa tabela e as suas colunas, essas informações são necessárias para o funcionamento correto do nosso site, são elas que serão manipuladas pela linguagem PHP com a intenção de resolver o problema - desenvolver um e-commerce para venda de produtos variados. Para a tabela de pedidos, os dados necessários nesse momento são:

ID – Número de identificação do nosso registro, é do tipo INTEIRO, ou seja, aceita apenas números inteiros, como aprendido na aula de algoritmos. É gerado automaticamente pelo banco de dados.

VALOR – Guarda a informação do valor do produto, é do tipo DOUBLE, que no banco de dados aceita números inteiros e fracionários.

STATUS – Campo com o status do nosso pedido, é do tipo VARCHAR, que corresponde no banco de dados ao tipo STRING no PHP, são dados de texto.

Para a criação das tabelas, devemos nos atentar aos tipos de dados que cada coluna aceita, para que as informações tenham a consistência necessária. Essa tipagem evita que o usuário cadastre o valor da venda como texto por exemplo, se no valor da venda fosse salva a informação “Venda realizada”, geraria erros na nossa aplicação não é mesmo?

Verificamos os três principais tipos na tabela, o tipo INTEIRO, que aceita números inteiros, o tipo DOUBLE, que aceita números inteiros e fracionários e o tipo VARCHAR, que corresponde no banco de dados às informações do tipo texto.

[MOD 2] Atividade: Modelagem do banco de dados – Tabela produtos

Seguindo a mesma lógica da Tabela pedidos, realize a atividade criando a Tabela de produtos, conforme as informações solicitadas.

1. Acesse o link <https://app.creately.com/>.
2. Faça o login.
3. Acesse o modelo Database.
4. Abrirá um modelo inicial e é nele que você poderá iniciar a elaboração das tabelas.

Orientações para realização da atividade:

- Crie dois objetos e posicione as tabelas lado a lado: produto e pedidos.

Para a criação da tabela de pedidos, siga o modelo anterior. Para a criação da tabela produtos, inclua as informações a seguir, juntamente com o tipo dos dados que iremos utilizar.

Informações a serem incluídas:

- **ID** – Número de identificação do nosso registro, é a PK do tipo INTEIRO e gerado automaticamente pelo banco de dados.
- **DESCRICAO** – Campo com a descrição do produto, é do tipo VARCHAR.
- **VALOR** – Guarda a informação do valor do produto, é do tipo DOUBLE, que no banco de dados aceita números inteiros e fracionários.
- **CATEGORIA** – Campo com informação da categoria do produto, é do tipo VARCHAR.
- **QUANTIDADE** – Campo com a quantidade disponível para o produto, é do tipo DOUBLE.

Ao finalizar suas tabelas clique para continuar!

Feedback da atividade

Caro aluno, veja como as tabelas ficam ao serem organizadas. Analise e compare com as que você elaborou e realize ajustes, se for necessário.

Modelo das tabelas de produtos e pedidos criados em <https://app.creately.com/> pelo autor.

PRODUTOS		PEDIDOS	
ID (PK)	INT	ID (PK)	INT
DESCRICAO	VARCHAR	VALOR	DOUBLE
VALOR	DOUBLE	STATUS	VARCHAR
CATEGORIA	VARCHAR		
QUANTIDADE	DOUBLE		

Voltando a refletir sobre nosso e-commerce a ser desenvolvido, uma informação muito importante que temos que ter também é o controle de usuários, onde eles irão se cadastrar e realizar o login para que consigam comprar no site. Ou seja, o objeto usuários passa a ser uma tabela no nosso banco de dados.

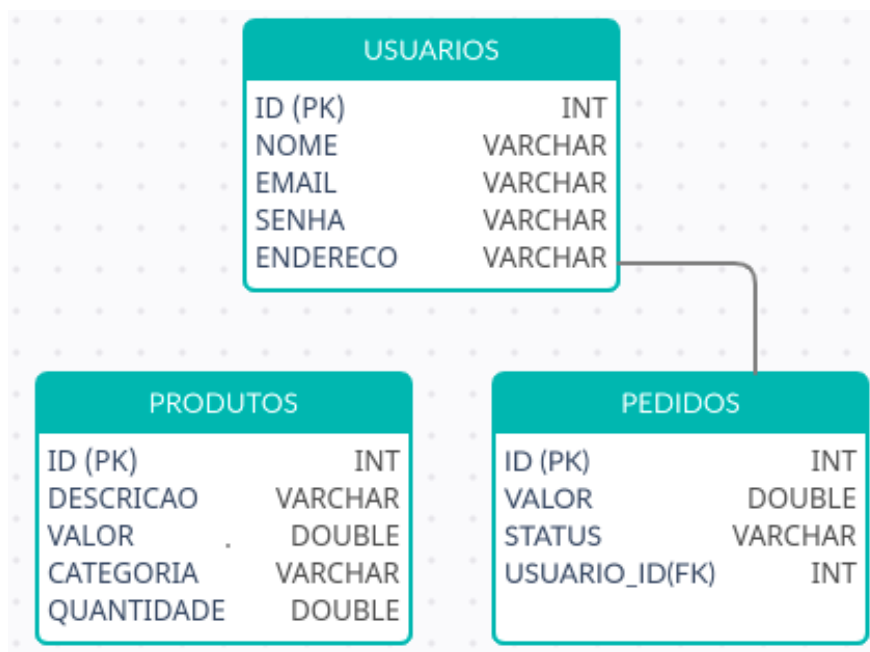
Vamos complementar a atividade feita acima? Acesse o site <<https://app.creately.com/>> e crie a tabela do objeto e inclua os seguintes requisitos: ID, nome, e-mail, senha e endereço.

Feito? Se sim, siga para a próxima lâmina!

Usuários, produtos e pedidos

Agora temos 3 tabelas, que representam nossos 3 objetos: USUÁRIOS, PRODUTOS E PEDIDOS. Mas, como vamos saber qual usuário que fez o pedido? E o endereço de entrega? Para esse tipo de associação, onde o pedido tem apenas um usuário, podemos referenciar o usuário no nosso pedido, através das chaves estrangeiras (FK), que convencionalmente são o número de identificação (ID) das tabelas relacionadas.

Observe como a modelagem desse banco de dados fica e realize os ajustes necessários no seu arquivo:



Relacionamento de um para muitos, das tabelas de usuários e pedidos, criado com <https://app.createely.com/>

Nota importante!

A relação é de um para muitos, ou seja, um usuário pode ter vários pedidos, mas um pedido pode ter apenas um usuário. Dessa maneira, devemos criar a referência na tabela de pedidos, que se relaciona somente com um usuário, se fizermos na tabela de usuários, o usuário não vai conseguir manter mais de um pedido ao mesmo tempo. Essa relação fica mais clara na imagem a seguir.

Verificamos no nosso modelo relacional acima, que o ID (identificador) do usuário, que é a PK (chave primária) da tabela, se transforma na FK (chave estrangeira) **USUARIO_ID**, da tabela pedidos, é através dessa informação que vamos saber qual usuário que fez algum pedido específico.

Nossa tabela de usuários deve conter as seguintes informações:

ID – Número de identificação do nosso registro, é do tipo INTEIRO e gerado automaticamente pelo banco de dados.

NOME – Guarda o nome informado pelo usuário, é do tipo VARCHAR.

EMAIL – Campo com o e-mail informado pelo usuário, é do tipo VARCHAR.

SENHA – Campo com a senha informada pelo usuário, é do tipo VARCHAR.

ENDEREÇO – Campo com o endereço informado pelo usuário, é do tipo VARCHAR.

Outro detalhe que temos que resolver, é que os pedidos contêm produtos, certo? E como vamos saber qual produto temos no pedido?

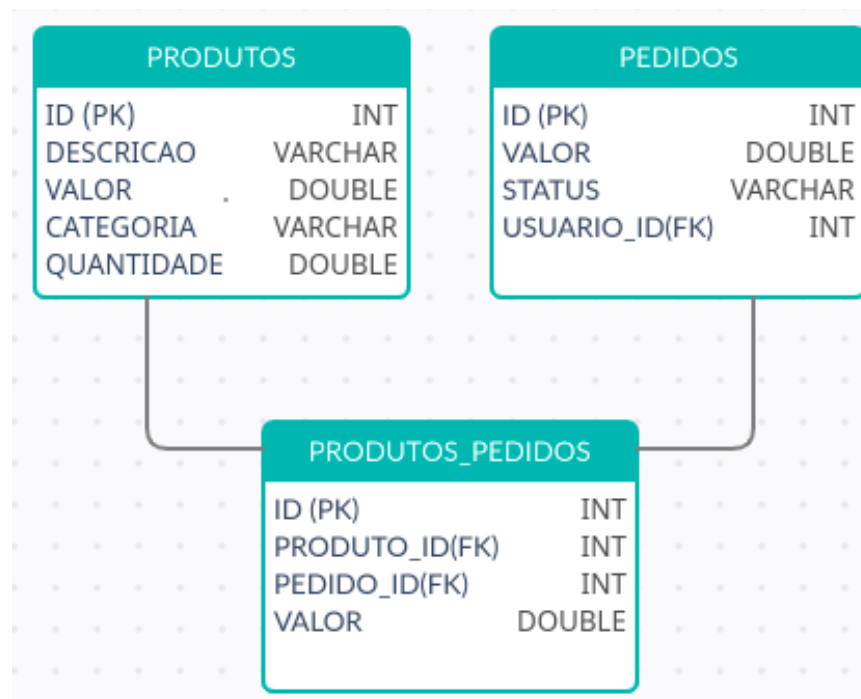
Essa associação é um pouco diferente, o pedido pode conter muitos produtos e o produto pode ser vendido em vários pedidos, para resolver esse relacionamento de muitos para muitos entre produ-

tos e pedidos, devemos criar uma terceira tabela que irá relacionar as duas já existentes, através das chaves estrangeiras (FK), que como dito anteriormente, é o número de identificação (ID) das tabelas relacionadas.

[MOD 2] Atividade: Inserindo o item Carrinho

Relacionamento das tabelas de produtos e pedidos

O relacionamento entre as tabelas citado na lâmina anterior fica mais claro na imagem a seguir.



Relacionamento das tabelas de produtos e pedidos, criado com <https://app.createely.com/>

Importante!

A tabela Usuário continua vinculada às tabelas Pedidos e Produtos, porém, foi recortada na imagem acima para que o foco fique na relação entre as tabelas Pedidos, Produtos e Produtos_Pedidos

No exemplo podemos ver a maneira em que as tabelas se relacionam, onde um pedido pode conter um ou mais registros na tabela PRODUTOS_PEDIDOS. Ao realizarmos uma venda no nosso sistema, serão criados registros na tabela **PRODUTOS_PEDIDOS** com as informações dele e pedido correspondentes, é nela que iremos consultar quais são os produtos de cada pedido. As colunas da nossa tabela serão:

- **ID** – Numero de identificação do nosso registro, do tipo INTEIRO, e gerado automaticamente pelo banco de dados.
- **PRODUTO_ID** – Numero de identificação do nosso produto, correspondente ao ID do registro da tabela **PRODUTOS**.
- **PEDIDO_ID** – Numero de identificação do nosso pedido, correspondente ao ID do registro da tabela **PEDIDOS**.

Antes de seguir para o próximo passo da criação do nosso banco de dados, vamos adicionar mais um detalhe ao nosso modelo. Vamos criar uma tabela **CARRINHOS**, para que o usuário veja todos os produtos selecionados antes de realizar a compra, permitindo também que escolha os produtos desejados e realize a compra em outro momento.

[MOD 2] Atividade: Inserindo o item Carrinho

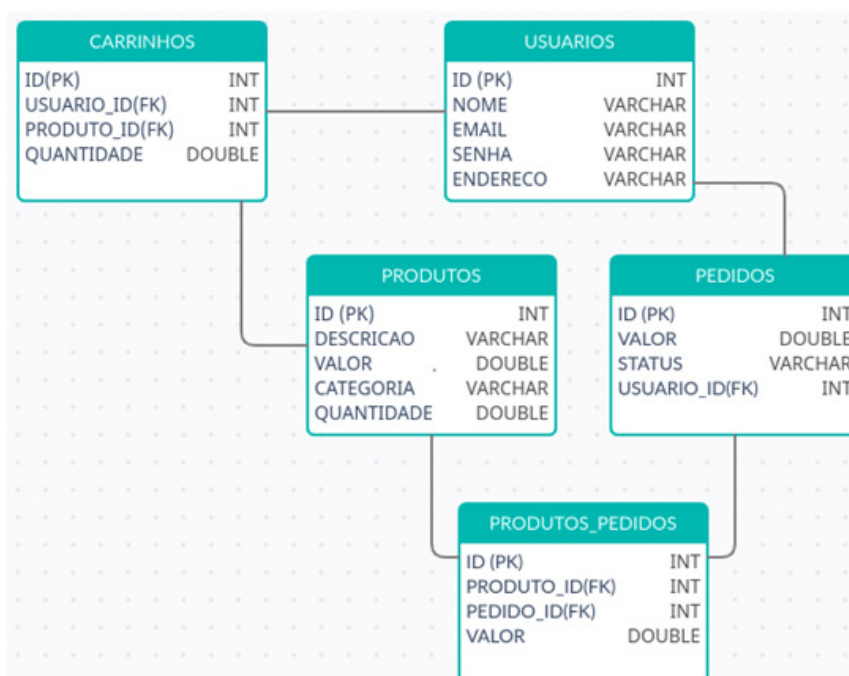
Insira na sua modelagem de banco de dados a tabela carrinhos, a qual deve conter as seguintes informações:

- **ID** – Número de identificação do nosso registro, é do tipo INTEIRO, e gerado automaticamente pelo banco de dados.
- **USUARIO_ID** – FK com a informação do ID do usuário, é do tipo INTEIRO.
- **PRODUTO_ID** – FK com a informação do ID do produto, é do tipo INTEIRO.
- **USUARIO_ID** – Campo com a quantidade do produto adicionado ao carrinho, é do tipo DOUBLE.

Relacione a tabela carrinhos com a tabela usuários e produtos. Finalize essa atividade e clique no feedback abaixo para verificar se precisa realizar ajustes.

Feedback da atividade

Ótimo avanço Aluno, agora nos sabemos como os relacionamentos básicos do banco de dados funcionam. Nosso modelo fica conforme a imagem a seguir e bora criar o nosso banco de dados na prática!



[MÍDIA 2] SQL – LINGUAGEM DE CONSULTA ESTRUTURADA

Nessa mídia, você vai compreender melhor o que significa SQL - “*Structured Query Language*”, ou “Linguagem de consulta estruturada”, em português. Resumidamente, é uma linguagem de programação para lidar com banco de dados relacional (baseado em tabelas).

Vamos lá, só clicar em [Mídia 2] SQL – Linguagem de consulta estruturada

SQL – Linguagem de consulta estruturada

A Linguagem de consulta estruturada foi criada para que vários desenvolvedores pudessem acessar e modificar dados de uma empresa simultaneamente, de maneira descomplicada e unificada.

Bons estudos!

Fala autor!

Ouçã o que autor tem para adiantar a você sobre esse módulo. Só clicar no PLAY!

Caso queira ler o que foi apresentado no podcast, basta acessar o PDF abaixo com a transcrição do áudio!

Transcrição do podcast

Autor: *Vitor Gabriel Martines Weinfortner*

E aí Aluno, tudo bem até aqui? Espero muito que sim...

Como você sabe, eu sou o Vitor Gabriel Martines Weinfortner autor desse curso e estou passando para saber o que você está achando do curso até aqui? Você pode responder logo abaixo desse PODCAST. Mas já adianto que se estiver achando complicado, fique tranquilo, que lendo o conteúdo, realizando as atividades e estando atento às orientações que preparei você vai tirar de letra.

E aproveite, pois, agora é que vai começar a parte mais divertida: a parte prática. A partir da linguagem de consulta SQL é que vamos criar as nossas tabelas com suas respectivas colunas.

SQL - significa Linguagem de Consulta Estruturada, é usada para manipulação de dados dentro do banco de dados, criar, excluir e editar as informações que precisamos. As consultas utilizam palavras em inglês, como select para selecionar, create para criar e assim por diante, então quem conhece a língua já sai na frente, mas se você não conhece sem problema algum, vamos aprender o passo a passo para realizar as principais consultas via SQL.

É muito importante aprendermos essa linguagem de consulta, pois é usada em todos os sistemas desenvolvidos que utilizam o banco de dados, para criar, para editar, excluir esses dados, como por exemplo, em um relatório de vendas, que por meio do comando select, podemos selecionar as informações que queremos consultar.

Pensando em um cenário real, se desejarmos saber o valor total de vendas por exemplo, é possível consultar por meio dos comandos SQL.

Vamos lá?

Execução dos comandos

Como observamos na primeira mídia desse módulo, vamos utilizar a interface phpMyAdmin para executar os comandos relacionados ao banco de dados. Relembre que construímos a modelagem

do banco de dados estruturando os objetos e os requisitos e, agora, vamos manusear esse banco de dados dentro da interface que instalamos anteriormente.

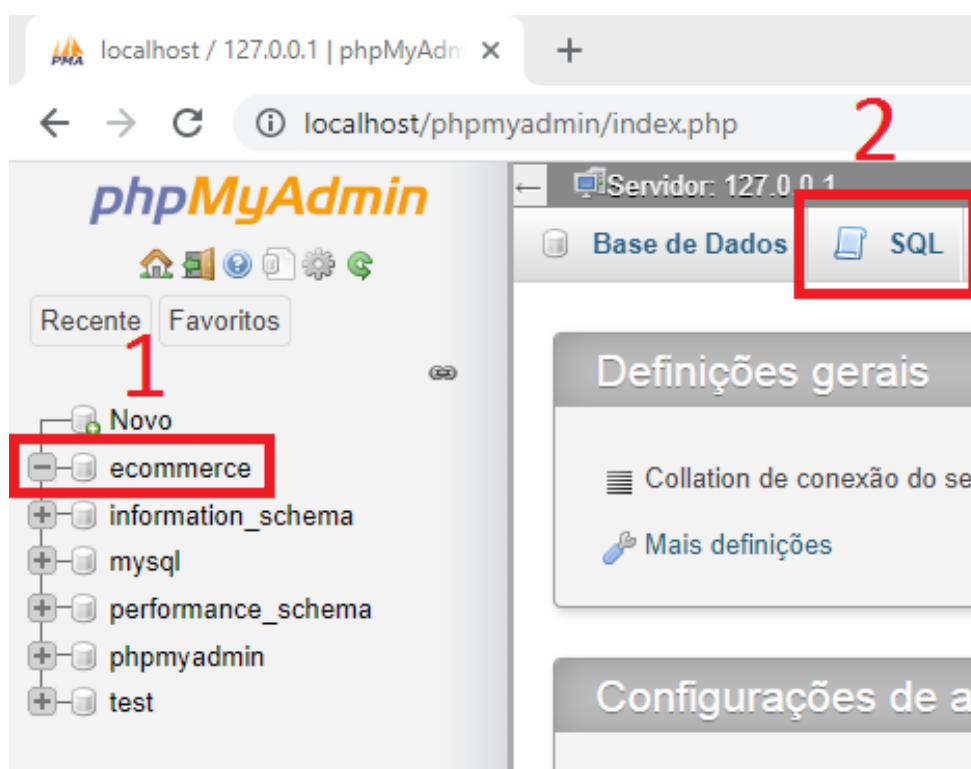
Para isso, volte ao painel inicial do phpMyAdmin (<http://localhost/phpmyadmin>) e, na sequência siga os passos a seguir:

- **PASSO 1**

Selecione o banco que quer trabalhar, nesse caso é o banco E-commerce, criado anteriormente, conforme imagem 1.

- **PASSO 2**

Depois de selecionar o banco, clique no menu SQL (imagem 2) e o painel para execução das operações será exibido.



Nesse painel que iremos digitar os comandos em SQL para que realize a ação proposta, para criar a estrutura do banco, utilizamos a palavra **CREATE**, que é criar em inglês, junto com o que queremos criar. O banco de dados foi criado anteriormente pela interface, mas posso criá-lo também através do SQL com o comando **CREATE DATABASE ecommerce**, após o **CREATE** descrevemos o que queremos criar, no caso **DATABASE**, que é banco de dados em inglês seguido pelo nome do banco.

Criando tabelas

Como nosso banco já está criado, vamos seguir com a criação das tabelas, para criá-las utilizamos o comando aprendido anteriormente, o **CREATE**, junto com o que queremos criar, nessa caso o **TABLE**, iniciando pela tabela de usuário, vamos ter o seguinte comando:

```
1 CREATE TABLE Usuarios (  
2     ID int PRIMARY KEY NOT NULL AUTO_INCREMENT,  
3     Nome varchar(100) NOT NULL,  
4     Email varchar(100) NOT NULL,  
5     Senha varchar(100) NOT NULL,  
6     Endereco varchar(100) NOT NULL  
7 );
```

Verificamos nesse comando que devemos iniciar com **CREATE TABLE**, seguido do nome da nossa tabela:

- Após o nome devemos **abrir parênteses** indicando que entre esses parênteses estão as nossas colunas.
- Para a criação das colunas, devemos iniciar com o **nome da coluna, seguido do tipo de dado que será salvo**, como na imagem anterior.

Lembrando que a criação das tabelas segue o modelo criado anteriormente, importante atentar-se ao tipo de dado de cada coluna, para evitar possíveis erros no sistema que iremos criar. Como tentar cadastrar um texto em um campo que só aceita números por exemplo?!... se ficar em dúvida sobre quais os dados e seus respectivos tipos, volte ao modelo criado anteriormente.

Outro detalhe importante é na coluna ID, os dados vão ser do tipo **int** (inteiro), sendo que esse é um dado obrigatório na nossa tabela, afinal, sem o número de identificação fica muito mais complicado de encontrar o registro.

Para criar o ID como PK

Para criar o ID como PK da nossa tabela, utilizamos:

- O comando **PRIMARY KEY**, para criar a validação que evita que algum usuário seja salvo sem o ID,
- O comando **NOT NULL**, indicando que esse campo não pode ser nulo.
- O comando **AUTO_INCREMENT**, para que o campo seja incrementado automaticamente pelo nosso banco de dados, ou seja, ao cadastrarmos o primeiro usuário, o ID vai ser criado como 1, o segundo será o ID 2 e assim por diante.

As demais colunas também serão criadas com o **NOT NULL**, de maneira que não permitam ser salvas sem essas informações. Podemos verificar na imagem acima que nos dados do tipo **VARCHAR**, devemos adicionar o tamanho máximo do texto permitido, no nosso caso vão ser 100 caracteres, porém só vamos tratar essa validação no nosso back-end (servidor) futuramente.

Nota importante!

A linguagem SQL segue o padrão de pontuação em que no final de cada consulta devemos adicionar o caractere ";". As informações dentro do parênteses devem ser separadas pelo caractere ",", com exceção da última informação.

Resultado da operação

Selecionamos o botão Executar, que fica a baixo da caixa de texto ao qual escrevemos o nosso comando SQL. O **phpMyAdmin** nos mostra que o resultado da operação deve ser de sucesso, exibindo também a operação feita e a quantidade de registros retornados que, no nosso caso como é apenas um comando de criação, não retornou nenhum, conforme a imagem a seguir.

```
✓ MySQL não retornou nenhum registro. (A consulta demorou 0,3326 segundos.)

CREATE TABLE Usuarios ( ID int PRIMARY KEY NOT NULL AUTO_INCREMENT, Nome varchar(100) NOT NULL, Email varchar(100) NOT NULL, Senha varchar(100) NOT NULL, Endereco varchar(100) NOT NULL )

[ Editar em linha ] [ Editar ] [ Criar código PHP ]
```

Nota importante!

Se a mensagem de erro “Nenhum banco de dados foi selecionado” for exibida, conforme imagem de exemplo a baixo, você pode selecionar novamente o banco de dados através da interface, ou executar o comando SQL USE, seguido com o nome do banco, no nosso caso, da seguinte maneira: **USE ecommerce**. Esse comando deve ser executado no painel SQL do phpMyAdmin, como vimos anteriormente.

Mensagens do MySQL : ?

#1046 - Nenhum banco de dados foi selecionado

Criando as tabelas de produtos e pedidos

Da mesma maneira que criamos a tabela de usuários, no painel SQL do phpMyAdmin, vamos criar a de produtos, conforme imagem a seguir.

```
1 CREATE TABLE Produtos (
2     ID int PRIMARY KEY NOT NULL AUTO_INCREMENT,
3     Descricao varchar(250) NOT NULL,
4     Valor double NOT NULL,
5     Categoria varchar(100),
6     Quantidade double NOT NULL
7 );
```

Nesse caso, a quantidade de caracteres permitidos vai mudar para o campo descrição, permitindo que seja adicionada uma descrição de no máximo 250 caracteres. Nessa tabela vão aparecer também os dados do tipo **DOUBLE**, para o valor e quantidade. Não se esqueça que para executar a instrução e criar a tabela no banco de dados devemos selecionar o botão Executar, que fica a baixo da caixa de texto.

Criando a tabela de pedidos

Para a criação da tabela de pedidos, devemos nos atentar a um detalhe, o campo **Usuario_id** é uma

referência ao usuário que fez o pedido, ou seja, ao campo ID, do registro que corresponde ao usuário que fez a compra. **E como vamos criar essa referência pelo SQL?**

- Crie essa referência através do comando **FOREIGN KEY (Usuario_id) REFERENCES Usuarios (ID)**;
- Passe, após o comando **FOREING KEY** (do inglês chave estrangeira), o nome da coluna que criamos;
- Depois do comando **REFERENCES**, informe a tabela a qual está referenciando e a coluna da tabela entre parênteses, conforme imagem a seguir:

```
1 CREATE TABLE Pedidos (  
2     ID int PRIMARY KEY NOT NULL AUTO_INCREMENT,  
3     Valor double NOT NULL,  
4     Status varchar(100),  
5     Usuario_id int NOT NULL,  
6     FOREIGN KEY (Usuario_id) REFERENCES Usuarios(id)  
7 );  
8
```

[MOD 2] Atividade: Criação de tabelas usando SQL

Nosso banco de dados está quase pronto, agora que aprendemos a criar as tabelas e suas respectivas colunas, o que acha de praticarmos um pouco?

Para iniciar os exercícios você deve criar as tabelas Carrinhos e Produto_pedidos. Utilize os conhecimentos que construiu até aqui. Lembre-se de criar e executar o script SQL também no phpMyAdmin, para que as alterações sejam feitas no banco de dados.

Ao finalizar a criação das tabelas Carrinhos e Produto_Pedidos salve as informações para consultar quando necessário e, depois, realize as atividades a seguir.

[MOD 2] Atividade: Criação de tabelas usando SQL

Prosseguindo com a criação das nossas tabelas, selecione a sequência correta de comandos e caracteres que, sendo adicionados respectivamente a imagem, façam com que a tabela Carrinhos seja criada corretamente, conforme imagem:

```
CREATE TABLE Carrinhos 1  
    ID int 2 NOT NULL AUTO_INCREMENT,  
    Usuario_id int NOT NULL,  
    Produto_id int NOT NULL,  
    FOREIGN KEY 3 REFERENCES Usuarios(id) ,  
    FOREIGN KEY 4 REFERENCES Produtos(id)  
    ;
```

- 1- (/ PRIMARY KEY / (Usuario_id) / (Produto_id).
- 2- USE / PRIMARY KEY / Usuarios / (Produto_id).
- 3- ; / FOREIGN KEY / (id) / (id).
- 4- (/ PRIMARY KEY / (Usuario_id) / Produtos.
- 5- (/ PRIMARY KEY / Usuarios / (Produto_id).

[MOD 2] Atividade: Criação de tabelas usando SQL

Selecione a sequência correta de comandos e caracteres que, sendo adicionados respectivamente à imagem, façam com que a tabela produtos_pedidos seja criada conforme referência apresentada:

```
1 2 produtos_pedidos (  
    ID int PRIMARY KEY NOT NULL 3,  
    Produto_id 4 NOT NULL,  
    Pedido_id 5 NOT NULL,  
    Valor 6 NOT NULL,  
    FOREIGN KEY (Produto_id) REFERENCES Produtos(id) ,  
    FOREIGN KEY (Pedido_id) REFERENCES Pedidos(id)  
-);
```

- 1- CREATE / DATABASE / REFERENCES / Produtos / Pedidos / double.
- 2- CREATE / TABLE / AUTO_INCREMENT / (Produto_id) / (Pedido_id) / double.
- 3- CREATE / TABLE / int / int / int / double.
- 4- CREATE / TABLE / AUTO_INCREMENT / int / int / double.
- 5- USE / TABLE / AUTO_INCREMENT / int / int / double.

[SÍNTESE] MÓDULO 2

Parabéns pela finalização desse módulo!

Até aqui, já realizou um grande avanço no desenvolvimento do projeto de E-commerce, pois se seguiu os passos e orientações, realizou as atividades adequadamente, já tem a estrutura do banco de dados preparada para aplicação.

O próximo passo é construirmos conhecimentos juntos acerca da linguagem PHP, para manipular os dados dessas tabelas.

Nos vemos no Módulo 3!

MÓDULO 3 – REALIZAÇÃO E INTEGRAÇÃO DOS DADOS

[INTRODUÇÃO] REALIZAÇÃO E INTEGRAÇÃO DOS DADOS

Bem-vindo (a) ao **Módulo 3** - *Realizar a integração dos dados ao sistema utilizando os comandos, conforme as linguagens PHP e SQL.*

Agora, ouça as boas-vindas do autor clicando no PLAY!

Caso não tenha conseguido ouvir, leia a transcrição clicando AQUI!

Transcrição do Podcast

Autor: Vitor Gabriel Martines Weinfortner

Podcast de abertura

Olá aluno (a), seja bem-vindo (a) ao Módulo 3 do curso de PHP com MySQL!

Já finalizamos a instalação e configuração do nosso ambiente de desenvolvimento no Módulo 1 e aprendemos como funciona a modelagem das tabelas do banco de dados e a linguagem de consulta SQL no Módulo 2.

Chegou a hora de utilizar o banco de dados e desenvolver o nosso conhecimento na linguagem PHP, construindo o nosso E-commerce. Bora lá?

No nosso projeto iremos utilizar a estrutura MVC para a organização das pastas e dos arquivos, você conhece ou já trabalhou com esse padrão?

O MVC é um padrão de arquitetura de software, usado para organizar o nosso projeto, ele é dividido em três partes:

M – O “M” vem de *model* ou modelo em português, nessa pasta ficam os arquivos responsáveis por gerenciar os dados e estabelecer as regras de negócio do nosso sistema. É aqui que vamos criar os arquivos que representam nossos objetos.

V – O “V” vem de *view* ou visão em português, essa camada é a camada visual do nosso sistema. É importante que não tenham regras de negócio na *view* para correta organização do nosso projeto.

C – O “C” vem de *controller* ou controlador, corresponde a camada de controle do nosso sistema, ela fica responsável por receber os dados da *view* e repassá-los para o *model* correto, recebendo e processando esses dados.

Espero que goste de aprender uma das linguagens mais utilizadas no mundo, bons estudos!

Já finalizamos a instalação e configuração do nosso ambiente de desenvolvimento no Módulo 1 e aprendemos como definir e manipular os dados em MySQL, de acordo com as instruções SQL e as funcionalidades do sistema web a ser desenvolvido no Módulo 2. Chegou a hora de utilizar o banco de dados e desenvolver o nosso conhecimento na linguagem PHP, construindo o nosso e-commerce de produtos variados. Para isso, esse módulo é composto por 3 mídias interativas:

[Mídia 1] Estrutura MVC

[MINIQUIZ] Variáveis no PHP

[Mídia 2] Conexão com o MySQL

[MÍDIA 1] ESTRUTURA MVC

Nessa mídia, vamos iniciar nosso projeto utilizando a estrutura MVC para a organização das pastas e arquivos. Você já conhece ou trabalhou com esse padrão? Se sim, ou se não, acesse a mídia para saber mais.

Clique abaixo!

Bem-vindo (a) a nossa primeira mídia sobre *Realizar a integração dos dados ao sistema utilizando os comandos, conforme as linguagens PHP e SQL*.

Arquitetura MVC

A arquitetura MVC é um padrão de arquitetura de software, usado para organizar o nosso projeto. Ele é dividido em três partes, que ficarão mais claras ao decorrer do desenvolvimento do nosso sistema.

M

O “M” vem de Model, ou modelo em português. Essa é a pasta onde ficam os arquivos responsáveis por gerenciar os dados e estabelecer as regras de negócio do nosso sistema. É aqui que vamos criar os arquivos que representam nossos objetos.

V

O “V” vem de View, ou visão em português. Essa é a camada visual do nosso sistema. É importante que não haja regras de negócio na view, para correta organização do nosso projeto.

C

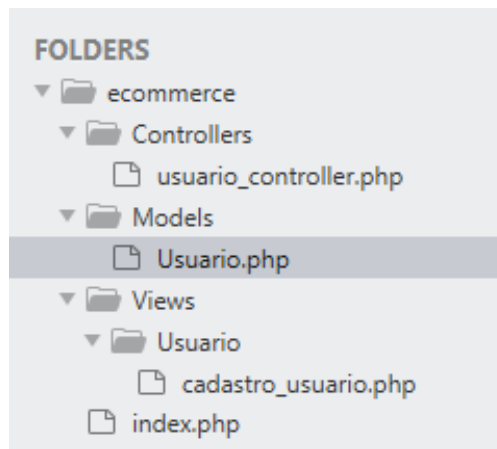
O “C” vem de Controller, ou controlador. É a camada de controle do nosso sistema. Ela fica responsável por receber os dados da view e repassá-los para o model correto, recebendo e processando esses dados.

Criação da estrutura

Para criar nossa estrutura de pastas e os nossos arquivos, vou utilizar o editor Sublime, mas fique à vontade para usar o editor de sua preferência. Como dito anteriormente, a pasta interpretada pelos servidores que instalamos juntamente com o Xampp é a htdocs, que fica no caminho **C:\xampp\htdocs**.

- Vamos criar dentro dessa pasta uma nova pasta com o nome ecommerce, que será o nome do nosso projeto.

- Dentro da pasta ecommerce vamos criar três pastas: Models, Views e Controllers.
- A estrutura fica como na imagem de exemplo a seguir.

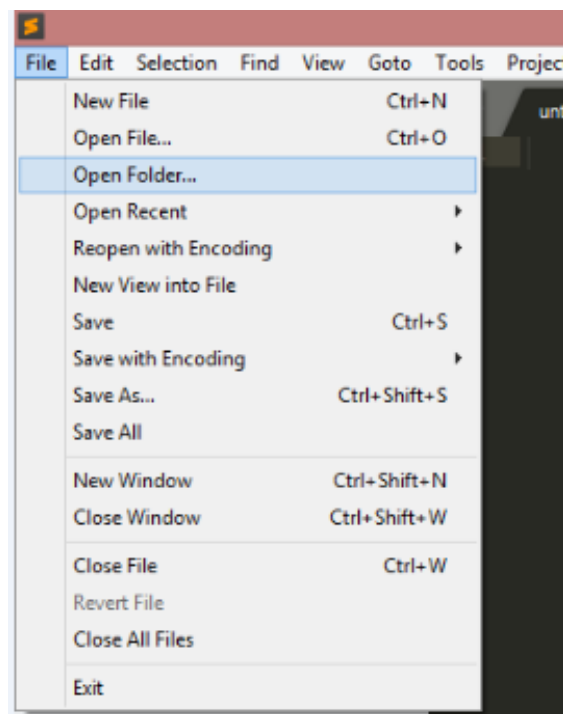


Arquivo inicial

Para abrir por meio do Sublime o projeto que criamos e criar nosso arquivo inicial, vamos seguir os seguintes passos:

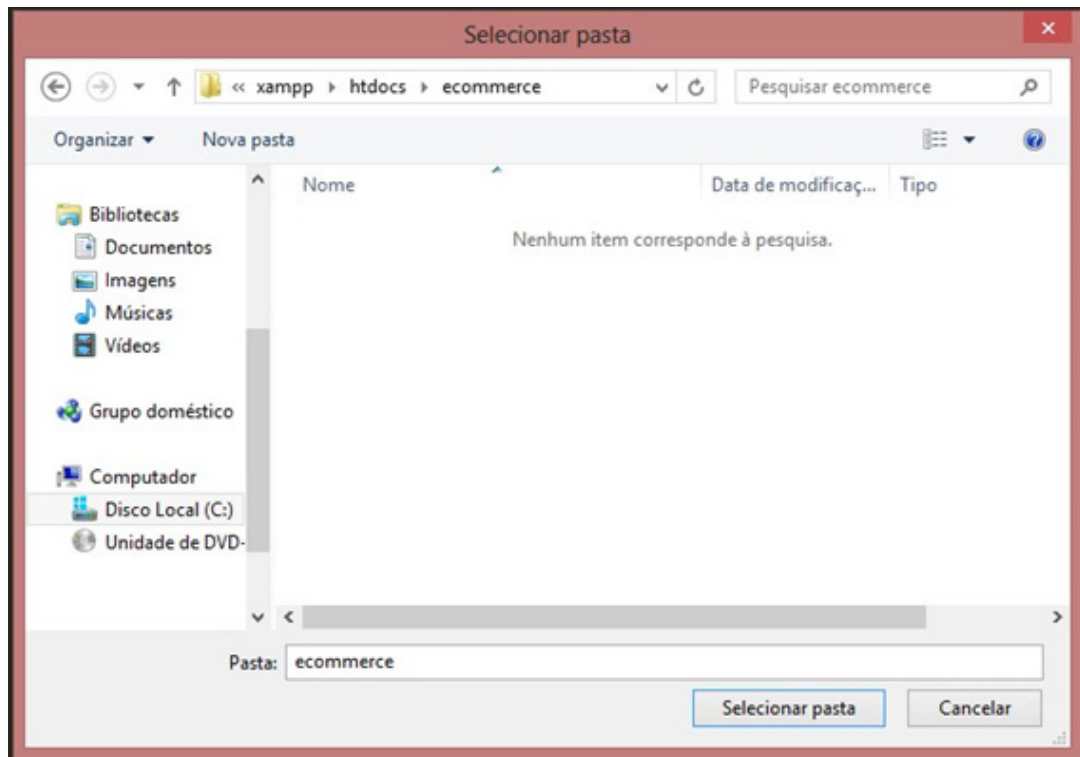
Passo 1

Acessar a opção File -> Open Folder.



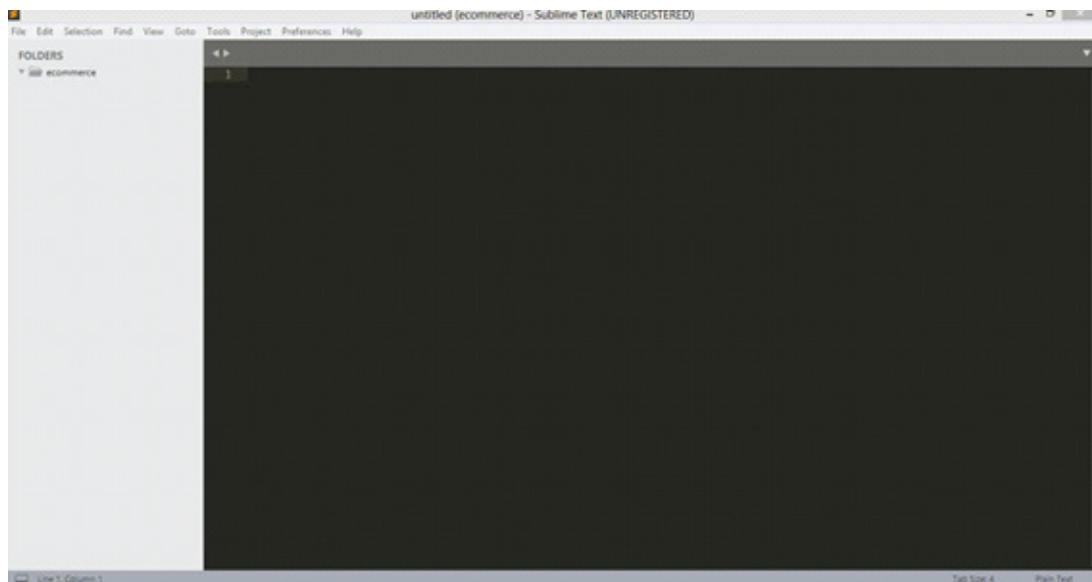
Passo 2

Acessar a pasta criada e clicar na opção Selecionar Pasta.



Passo 3

Acessar o menu File -> New File e depois File -> Save, para salvar nosso arquivo. O primeiro arquivo terá o nome de index.php. A extensão .php é muito importante, pois é ela que diz para o servidor o tipo do arquivo que será processado, fazendo com que o servidor reconheça nosso código PHP.



Pronto, nosso primeiro arquivo já está criado. Por padrão, na grande maioria dos servidores web, ao interpretar um projeto, o arquivo que será executado primeiro é aquele com o nome de index, ou seja, quando o servidor é acessado pelo navegador, ele reconhece e exibe primeiro a página index. Isso pode ser personalizado, mas para o nosso curso essa será a página inicial do nosso sistema e será responsável pelo nosso login.

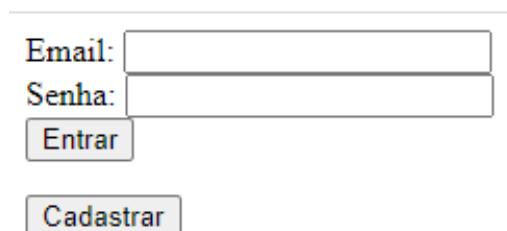
Criptografia de dados

Nos sistemas web, o login é feito de uma maneira mais complexa a fim de garantir a segurança dos dados e da aplicação, como por exemplo os dados de senha, que são criptografados antes de serem armazenados no banco de dados. Essa criptografia basicamente transforma a senha em um dado de texto com caracteres diferentes do inicial, evitando que a senha que o usuário informou seja acessada por usuários indevidos. Mas para o nosso aprendizado, a validação será feita de forma simples e rápida.

O vídeo a seguir trata sobre a criptografia de dados. O vídeo pode ser usado como seu material complementar, pois o assunto é de suma importância quando trabalhamos com dados confidenciais, como senhas de acesso, por exemplo. Dê um **PLAY** e saiba mais!

Página index

A página index terá apenas os campos de E-mail e Senha e os botões Entrar e Cadastrar, para que o usuário faça o login se possuir uma conta, ou para que ele possa se cadastrar, caso seja necessário. O conteúdo inicial do nosso arquivo fica como na imagem a seguir.



Como nosso foco de aprendizado é a linguagem PHP e o servidor MySQL, os arquivos serão criados com o layout simples, mas aproveite essa oportunidade para treinar as suas habilidades de HTML construídas anteriormente para deixar as páginas mais apresentáveis.

Página index

Crie seu arquivo index.php, adicione o seguinte código e estilize a página da maneira que preferir.

```
index.php
1  <html>
2  <body>
3      <form action="" method="post">
4          Email: <input type="text" name="email"><br>
5          Senha: <input type="text" name="senha"><br>
6          <button type="submit" name="logar">Entrar</button>
7      </form>
8      <button>Cadastrar</button>
9  </body>
10 </html>
```

Variáveis em PHP

Agora que temos os campos para o usuário informar o login e a senha no nosso sistema, precisamos receber e tratar esses dados, através das variáveis, validando se estão corretos e se o acesso será

liberado.

As variáveis em PHP possuem características importantes. Todas devem começar com o caractere cifrão (\$) seguido pelo nome da variável. São case sensitive, ou seja, as letras maiúsculas e minúsculas são reconhecidas como diferentes para as variáveis. Por exemplo, `$variavel`, `$Variavel` e `$VARIABEL` são variáveis diferentes e podem possuir valores diferentes.

O PHP possui conversão automática de tipo, ou seja, não precisamos definir explicitamente o tipo de cada variável. O tipo é determinado pelo valor que atribuímos a ela. Ou seja, em um contexto com duas variáveis:

`$v1 = 0` e `$v2 = "texto"`

Atribui-se o valor de cada uma delas através do caractere igual (=), onde:

- `$v1` será do tipo inteiro, pois recebe um número inteiro;
- `$v2` será do tipo string, pois um valor entre aspas é interpretado pelo PHP como um dado do tipo texto.

Isso significa também que uma variável pode mudar de tipo durante a execução do sistema. Podemos realizar a seguinte operação:

`$v1 = "agora sou do tipo string"`, onde `$v1` vai receber um dado de texto e representar um valor do tipo string.

Operadores

Na imagem abaixo vemos um exemplo de soma no PHP, utilizando o **operador +**, que soma o valor das variáveis informadas.

```
<?php
    $v1 = "teste"; // $v1 é do tipo string
    $v2 = 2; // $v1 é do tipo inteiro

    echo $v1 + $v2; // erro

    $v1 = 1; // $v1 agora é do tipo inteiro

    echo $v1 + $v2; // Será exibido o resultado correto na tela (3)

?>
```

Porém, o PHP disponibiliza outros operadores matemáticos que podemos utilizar para resolver os nossos problemas do dia a dia. Esses operadores são:

- o **caractere - (menos)**, utilizado para uma subtração;
- o **caractere /**, usado para divisões;
- o **caractere ***, utilizado para multiplicação; e

- o **caractere %** (módulo), que retorna o resto da divisão.

Na imagem abaixo, um exemplo dos operadores matemáticos.

```
<?php
    $x = 11;
    $y = 2;

    echo $x + $y; // Resultado: 13
    echo $x - $y; // Resultado: 9
    echo $x / $y; // Resultado: 5.5
    echo $x * $y; // Resultado: 22
    echo $x % $y; // Resultado: 1
?>
```

Operadores de comparação:

```
<?php

    $x and $y //verdadeiro se $x e $y forem verdadeiros.
    $x or $y //verdadeiro se $x ou $y forem verdadeiros.
    $x xor $y //verdadeiro se $x ou $y forem verdadeiros, mas não ambos.
    !$x //verdadeiro se $x for falso.
    $x && $y //verdadeiro se $x e $y forem verdadeiros.
    $x || $y //verdadeiro se $x ou $y forem verdadeiros.

?>
```

E os operadores lógicos:

```
<?php

    $x == $y //igual;
    $x === $y //idêntico;
    $x != $y //diferente;
    $x > $y //maior que;
    $x < $y //menor que;
    $x >= $y //maior ou igual a;
    $x <= $y //menor ou igual a;

?>
```

Métodos de Requisição HTTP

Outro detalhe muito importante nos sistemas web são as requisições disponíveis para indicar a ação que vamos realizar. Essas requisições são definidas pelo protocolo de comunicação HTTP.

Vimos a descrição dos métodos de requisição HTTP, comentamos os principais métodos e por meio do documento a seguir serão apresentados os demais métodos e suas funções.

Requisição e método

Os dois principais métodos de requisição são o **GET** e o **POST**. O **GET** é usado para requisitar algum dado, como uma consulta no banco, por exemplo, a fim de receber a informação solicitada. Já o **POST** é responsável por criar e atualizar informações. Quando utilizamos o **POST** passamos informações nessa requisição a fim de realizar alguma ação, cadastrar um usuário por exemplo.

Método post

Na página index, criamos anteriormente um formulário com o atributo **action=""** e o **method="post"**. Isso significa que ao subtermos o formulário através do nosso botão de login ele será redirecionado para a página atual, utilizando o método post, ao qual podemos passar parâmetros na nossa requisição.

Para receber esses dados enviados pela requisição, utilizamos a variável **\$_POST** e o nome da nossa variável, definido pelo atributo **name** do nosso formulário, entre chaves e entre aspas, vamos adicionar as instruções no nosso arquivo inicial, que ficará da seguinte maneira:

```
<?php

$email = $_POST["email"];

$senha = $_POST["senha"];

?>
```

Na linguagem PHP, o bloco de código deve ser aberto com os caracteres **<?php** e fechado com **?>**. Todas as instruções devem ser separadas com ponto e vírgula (;) no final.

O que acha de testarmos nossa aplicação em PHP?

Vamos adicionar o método **"echo"** juntamente com nossas variáveis. Essa instrução exibe informações na tela para o usuário. Para testarmos nossa aplicação, vamos exibir o e-mail e a senha informada pelo usuário. Para isso, vamos usar o caractere ponto (.), que é utilizado para concatenar strings no PHP. Adicione a instrução no seu arquivo index seguindo o modelo do exemplo a seguir:

```
1 <html>
2   <body>
3     <form action="" method="post">
4       Email: <input type="text" name="email"><br>
5       Senha: <input type="text" name="senha"><br>
6       <button type="submit" name="login">Entrar</button>
7       <button>Cadastrar</button>
8     </form>
9   </body>
10 </html>
11 <?php
12
13   $email = $_POST["email"];
14   $senha = $_POST["senha"];
15
16   echo "Olá, o e-mail cadastrado foi ".$email." e a senha ".$senha;
17 ?>
```

Agora podemos acessar a nossa primeira página, através do link <http://localhost/ecommerce>, onde vamos obter o seguinte resultado:

Email:

Senha:

Warning: Undefined array key "email" in C:\xampp\htdocs\ecommerce\index.php on line 13

Warning: Undefined array key "senha" in C:\xampp\htdocs\ecommerce\index.php on line 14
Olá, o e-mail cadastrado foi e a senha

Opa, apareceram algumas mensagens indesejadas na nossa tela, não é? Esses são os erros ou alertas no PHP. Podemos analisar a mensagem recebida e visualizar o motivo do erro, **“Undefined array key”**, para o e-mail e senha, nas linhas 13 e 14 do nosso arquivo index.php. O que isso significa? Responda na atividade a seguir.

[MOD 3] Atividade: Erros e Alertas no PHP

O PHP exibe em tela os erros e alertas que ocorrem durante a execução do nosso sistema. Antes de responder essa atividade, analise a mensagem de erro a seguir e, utilizando seu conhecimento em algoritmos e lógica de programação, assinale as alternativas corretas.

“Warning: Undefined Array key “email” in C:\xampp\htdocs\ecommerce\index.php on line 13”

- a. A mensagem de erro não tem informações suficientes para reconhecermos o local em que o erro ocorreu.
- b. O erro ocorreu porque o tipo da variável está incorreto.
- c. O erro ocorreu na linha 13 do nosso arquivo index.php.
- d. O erro está na chave informada para acessar um item do array.
- e. A maneira com que acessamos a senha está incorreta.

Nota importante

Os arrays são utilizados para armazenar uma série de informações de tal forma que podemos acessar cada um deles por uma chave ou índice.

Para criar o array utilizamos o comando:

```
$lista = ["valor1","valor2"]
```

Para acessá-los:

```
echo $lista[0] – exibe “valor1” (o índice começa em 0)
```

```
echo $lista[1] – exibe “valor2”
```

Para adicionar itens:

```
array_push($lista, “novo valor”)
```

```
echo $lista[2] – exibe “novo valor”
```

Estamos tentando acessar as variáveis enviadas via método post, `$_POST["email"]` e `$_POST["senha"]`. Porém, como o usuário ainda não digitou as informações e submeteu o formulário através do botão Entrar, elas ainda não existem no nosso contexto, não é mesmo?

Nesse caso, é necessário criarmos uma validação para evitar que esse erro indesejado seja exibido para o nosso usuário final. Para realizar essa validação, vamos utilizar a estrutura de decisão **if**, que executa o trecho de código que informarmos dentro da estrutura apenas se a condição informada for atendida. A estrutura dessa funcionalidade é a seguinte:

```
if (condição) {  
    ação;  
} elseif (condição2) {  
    ação 2;  
} else {  
    ação 3;  
}
```

Podemos verificar na imagem anterior que iniciamos a estrutura com a instrução **if** seguida com a condição entre parênteses. A ação dentro das chaves ({ }) só é executada se a condição for atendida. Podemos usar também opcionalmente o **elseif**, que valida outra condição quando a primeira não foi atendida, podendo executar outra ação.

Para finalizar, podemos utilizar a instrução **else**, que é executada quando todas as condições anteriores não foram atendidas. Nessa instrução não passamos nenhuma condição.

Árvore de decisão

A seguir temos uma árvore de decisão que exemplifica de maneira mais clara como funciona essa estrutura de decisão.

Árvore de decisão if/else, criado com <https://app.creately.com/>

No nosso contexto, vamos exibir as variáveis apenas se o usuário apertar o botão Entrar, que foi criado com a propriedade **name="logar"**. Podemos acessar essa informação da mesma maneira que fizemos com o e-mail e a senha, através da variável `$_POST["logar"]`.

Outra função importante no PHP é o `isset($variavel)`, que retorna uma informação do tipo **boolean**, ou seja, verdadeiro ou falso. Se a variável que está entre parênteses foi iniciada, o retorno será **true**, verdadeiro em inglês, e se não foi iniciada ainda, ou seja, se tem o valor **null**, o retorno será **false**, ou falso em inglês.

Uma possível validação a ser feita é a seguinte:

```
1 <html>
2 <body>
3 <form action="" method="post">
4     Email: <input type="text" name="email"><br>
5     Senha: <input type="text" name="senha"><br>
6     <button type="submit" name="logar">Entrar</button>
7     <button>Cadastrar</button>
8 </form>
9 </body>
10 </html>
11 <?php
12
13 if (isset($_POST['logar'])){
14     $email = $_POST["email"];
15     $senha = $_POST["senha"];
16
17     echo "Olá, o e-mail cadastrado foi ".$email." e a senha ".$senha;
18 }
19 ?>
```

Na imagem acima podemos verificar que o nosso trecho de código que fica dentro da estrutura de decisão só vai ser executado se a nossa condição for atendida, ou seja, se a variável `$_POST["logar"]` já estiver iniciada. Acessando a página novamente o erro não é mais exibido. Se informarmos os dados e acessarmos a opção Entrar, a nossa mensagem será exibida corretamente.

Validação

Após atualizar seu código levando como base o da imagem acima, o que acha de testar nossa validação? Teste e verifique se o que aparece se relaciona com a imagem abaixo.

Email:

Senha:

Olá, o e-mail cadastrado foi email@teste.com.br e a senha senhateste

Nota importante

Podemos adicionar tags HTML na nossa instrução `echo`, a fim de formatar e estilizar nossas mensagens da maneira que preferirmos.

Assista ao vídeo descontraído sobre a linguagem de marcação HTML, como funciona e a maneira de usarmos.

[MINIQUIZ] VARIÁVEIS NO PHP

Antes de iniciar a próxima mídia, responda à atividade proposta sobre Variáveis no PHP.

Clique abaixo para realizar o Miniquiz!

Essa mídia corresponde a uma atividade intermediária entre os conhecimentos construídos anteriormente e os que virão na sequência.

Bom Miniquiz!

[MOD 3] Miniquiz: Variáveis no PHP

As variáveis são utilizadas no desenvolvimento de sistemas para armazenar e processar informações, na linguagem PHP. A criação delas segue algumas regras específicas. Com base nisso, assinale as alternativas corretas.

- ☐ a. As variáveis em PHP não são case sensitive, ou seja, `$variavel` e `$VARIABLE` são reconhecidas como a mesma variável.
- ☐ b. Todas as variáveis no PHP devem iniciar com o caractere `&`.
- ☐ c. O PHP possui conversão automática de tipo, ou seja, não precisamos definir explicitamente o tipo de cada variável, pois este é reconhecido pelo contexto do nosso sistema.
- ☐ d. Podemos concatenar variáveis do tipo string no PHP utilizando o caractere ponto (`.`).
- ☐ e. Para atribuir valor a uma variável, utilizamos o operador igual (`=`).

[MÍDIA 2] CONEXÃO COM O MYSQL

Bem-vindo (a) à última mídia do último módulo! Muita atenção e dedicação para esse fechamento. Se necessário, revise conteúdos anteriores para dar prosseguimento e ter um bom aproveitamento do curso.

Clique abaixo para acessar a mídia!

Estamos quase finalizando o curso, fique atento. Para aquecer, vamos dar início à mídia.

Bons estudos!

Classes e objetos

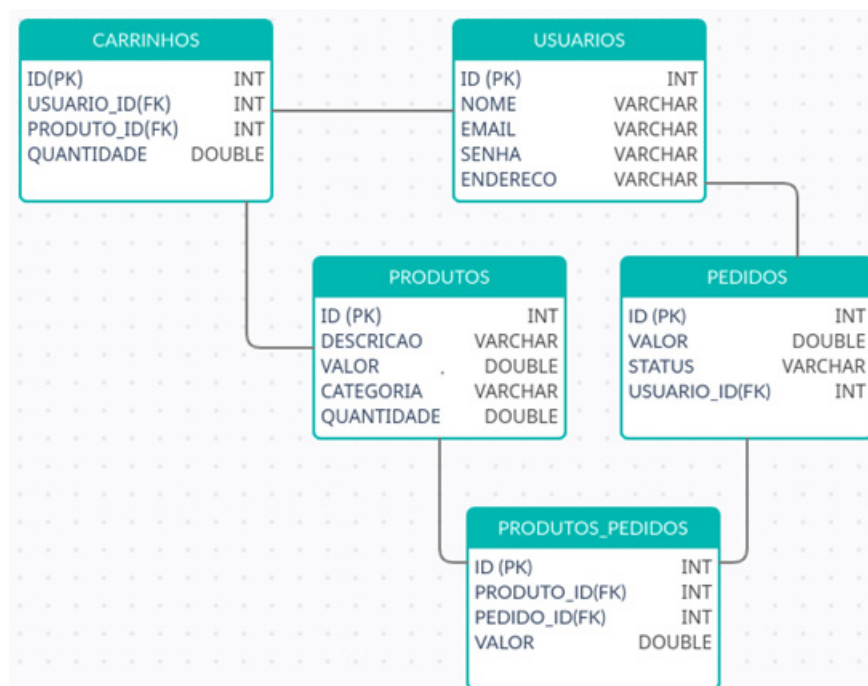
Agora que já entendemos como as variáveis funcionam, precisamos manipulá-las de acordo com o necessário para o funcionamento do nosso site. No padrão MVC que estamos utilizando, as regras de negócio ficam na camada Model, como vimos anteriormente na mídia 1.

Vamos criar nosso arquivo responsável pelos dados dos usuários. Para isso:

- crie o arquivo **Usuario.php** dentro da pasta Models do nosso sistema. Esse arquivo vai conter a classe Usuário, que é a abstração do nosso objeto usuário;
- para criarmos uma classe utilizamos a instrução **class**, seguida do nome da classe e as nossas instruções entre chaves (**{}**).

Você se recorda do nosso modelo de banco de dados?

Onde criamos o modelo das informações e o relacionamento entre elas? No modelo encontramos as informações que precisamos manter para cada usuário: **id**, **nome**, **e-mail**, **senha** e **endereço**. Vamos adicionar essas informações como variáveis na nossa classe.

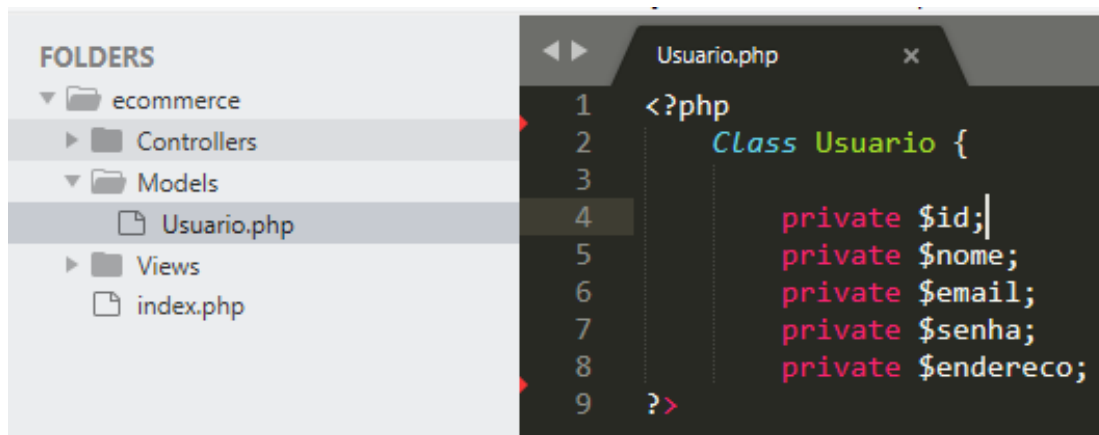


Modelo relacional de banco de dados, criado com <https://app.creately.com/>.

Instrução *private*

Um detalhe importante é que as variáveis da classe devem ser privadas, para garantir a segurança das nossas informações, evitando que sejam acessadas de fora da nossa classe. Fazemos isso através da instrução **private**.

Nosso arquivo inicialmente fica como na imagem a seguir:



Para acessar essas informações, devemos criar métodos públicos, que serão acessados através das nossas views e controllers. Esses são os chamados getters e setters, que facilitam o nosso trabalho com objetos. Os getters são usados para acessar e os setters para modificar as informações. Os métodos e as funções no PHP são criadas com a instrução `function`, seguida do nome da função e parênteses, com as chaves delimitando as instruções desse método ou função. Lembrando que eles serão acessados de outras partes do sistema, portanto devem ser públicos. Para isso, utilizamos a instrução `public` no início da nossa função.

Os getters são criados da seguinte maneira:

```
public function getNome() {
    return $this->nome;
}
```

Nota – getters

Acima podemos verificar a estrutura dos nossos métodos. Quando o método **getNome** for chamado, ele irá retornar o nome do usuário. Para isso, utilizamos a instrução **return**. A funcionalidade desse método vai ficar mais clara com o decorrer do nosso curso. Outro detalhe muito importante é que para acessar as variáveis privadas da nossa classe devemos utilizar a variável **\$this->**, seguida pelo nome da variável que desejamos acessar. Isso informa ao PHP que estamos acessando a propriedade do nosso objeto.

Os setters são criados da seguinte maneira:

```
public function setNome($nome) {
    return $this->nome = $nome;
}
```

Nota – setters

Já para os setters, temos como função alterar o valor da informação dentro do nosso objeto. Para isso, teremos entre parênteses uma variável. Isso significa que vamos receber uma variável e que podemos utilizá-la no nosso método. No exemplo acima, recebemos a variável `$nome` e alteramos o valor da propriedade `nome` do objeto com a instrução `$this->nome = $nome;`.

Para praticar e fixar as funções que vimos acima, realize a seguinte atividade em que vamos criar juntos os getters e os setters para as informações de e-mail e senha. Não esqueça de criar também no seu projeto os getters e setters de `id` e endereço, com exceção do setter do `id`, que não é necessário criarmos, pois como analisamos anteriormente ele será criado automaticamente pelo banco de dados.

[MOD 3] Atividade: Criação de métodos get e set

Para a criação dos métodos `get` e `set`, que irão manipular a informação do e-mail do nosso usuário, assinale a alternativa que preenche corretamente as lacunas, de acordo com cada método, para que os respectivos funcionem corretamente.

1. Método `GET`, assinale a sequência correta:

```
___ __ getEmail() {  
    ___ __;  
}
```

- a. `public / function / return / $this->email`
- b. `return / $this->email / public / function`
- c. `$this->email / public / return / function`

2. Método `SET`, assinale a sequência correta:

```
___ __ setEmail(___){  
    ___ __ = $email;  
}
```

- a. `public / function / $email / return / $this->email`
- b. `return / function / $email / public / $this->email`
- c. `public / return / $email / function / $this->email`

Relacionamento entre classes

Agora que temos a nossa classe usuário criada podemos utilizá-la no arquivo ***index.php***, a fim de melhorar o nosso código e torná-lo mais legível. Esse é um passo importante para o mercado de tra-

balho, imaginando um cenário em que o trabalho em grupo é necessário. É importante desenvolver o seu código da maneira mais clara possível, pois no futuro outro desenvolvedor pode precisar ler e interpretar o que seu código está fazendo.

O primeiro passo é importar a classe para o arquivo em que vamos utilizá-la. Fazemos isso através do comando:

```
include_once("arquivo.php")
```

Esse comando inclui o arquivo que informarmos entre parênteses e entre aspas, se ele ainda não foi incluído, tornando acessível a classe desenvolvida. Nossa importação fica então da seguinte forma:

```
include_once("C:/xampp/htdocs/ecommerce/Models/Usuario.php");
```

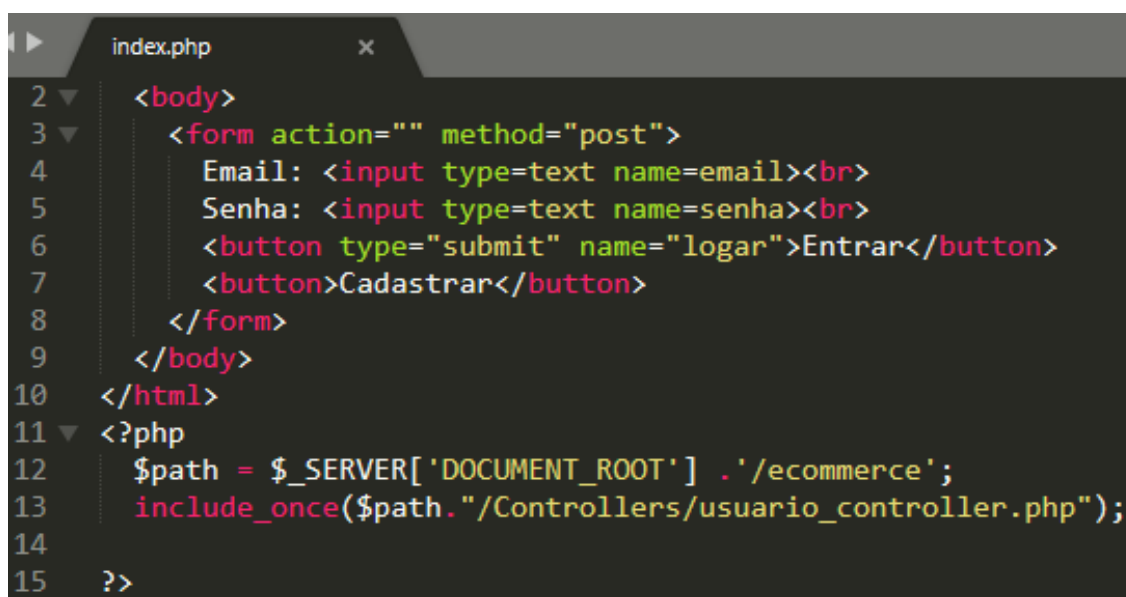
Podemos verificar que o caminho é o mesmo que usamos para acessar a pasta pelo nosso sistema operacional. Mas ainda não acabou. Podemos melhorar um pouco mais o nosso código. Vamos utilizar a variável:

```
$_SERVER['DOCUMENT_ROOT'].
```

Variável Server

A variável `$_SERVER` é um array que armazena alguns dados relevantes sobre o nosso servidor. Vamos utilizar o `DOCUMENT_ROOT` para acessar a informação da localização atual do arquivo no servidor. Podemos realizar um teste interessante com essa variável. Utilizando o conhecimento obtido até agora, altere o arquivo `index` com o objetivo de exibir na tela a variável `$_SERVER['DOCUMENT_ROOT']` e verifique o resultado.

A importação do nosso arquivo fica então como na instrução a seguir:



```
index.php
2 <body>
3   <form action="" method="post">
4     Email: <input type="text" name="email"><br>
5     Senha: <input type="text" name="senha"><br>
6     <button type="submit" name="login">Entrar</button>
7     <button>Cadastrar</button>
8   </form>
9 </body>
10 </html>
11 <?php
12   $path = $_SERVER['DOCUMENT_ROOT'] . '/ecommerce';
13   include_once($path . "/Controllers/usuario_controller.php");
14
15 ?>
```

A partir de agora, podemos trabalhar nossos dados utilizando a orientação a objeto, o que facilita muito a manipulação desses dados e o entendimento do código. Para isso, vamos criar o nosso objeto, que será do tipo `Usuário`, através da instrução `$objUsuario = new Usuario();` e receber as variáveis

com os métodos set, que criamos anteriormente. O código fica como na imagem a seguir:

```
index.php x
<body>
  <form action="" method="post">
    Email: <input type="text" name="email"><br>
    Senha: <input type="text" name="senha"><br>
    <button type="submit" name="login">Entrar</button>
    <button>Cadastrar</button>
  </form>
</body>
</html>
<?php
$path = $_SERVER['DOCUMENT_ROOT'] . '/ecommerce';
include_once($path."/Controllers/usuario_controller.php");

if (isset($_POST['login'])){
  $objUsuario = new Usuario();
  $objUsuario->setEmail($_POST["email"]);
  $objUsuario->setSenha($_POST["senha"]);

  echo "Olá, o e-mail cadastrado foi ".$objUsuario->getEmail()." e a senha ".$objUsuario->getSenha();
}
?>
```

Permissão de acesso

Agora que organizamos nosso código, o próximo passo é validar se o usuário terá permissão para acessar nosso site ou não. Para isso, vamos criar nosso arquivo controlador, que recebe e trata os dados recebidos da view e envia para o método correto no model.

- Crie o arquivo **usuario_controller.php** na nossa pasta Controllers.
- Crie também nesse arquivo a classe UsuarioController.
- Crie um método público no controller chamado **validaUsuario**, que irá receber um usuário, validar as informações e enviar a solicitação para o model.

Para isso, precisamos validar o e-mail e a senha informados, certo?

Seguindo o nosso modelo de banco de dados, desenvolvido no Módulo2, conforme imagem abaixo, podemos verificar que os campos de e-mail e senha são obrigatórios e com o tamanho máximo de 100 caracteres. É preciso validar se as informações seguem nossa regra de negócio, para evitar problemas ao salvá-las no banco de dados.

```
1 CREATE TABLE Usuarios (
2   ID int PRIMARY KEY NOT NULL AUTO_INCREMENT,
3   Nome varchar(100) NOT NULL,
4   Email varchar(100) NOT NULL,
5   Senha varchar(100) NOT NULL,
6   Endereco varchar(100) NOT NULL
7 );
```


Validação

Para validar nosso e-mail e senha, vamos utilizar novamente a estrutura de decisão if.

Podemos validar se o e-mail e senha foram informados por meio da instrução `$variavel == null`, onde o operador `==` valida se a variável é igual à condição, nesse caso, null. Se ela for nula, significa que ainda não foi instanciada, ou seja, o usuário não a informou corretamente.

Já para validar o tamanho do nosso e-mail e senha, podemos utilizar a instrução `strlen($variavel) > 100`, onde a função `strlen` retorna a quantidade de caracteres da variável informada, e o operador `>` (maior que) retorna se o tamanho é maior que 100 ou não.

Lembrando que, como vamos utilizar funções do nosso objeto usuário, precisamos importar a classe para o nosso controller. O código inicial fica como na imagem a seguir:

```
1 <?php
2 $path = $_SERVER['DOCUMENT_ROOT'] . '/ecommerce2';
3 include_once("$path/Models/Usuario.php");
4
5 class UsuarioController {
6
7     public function login($objUsuario){
8         echo $objUsuario->getEmail();
9         if ($objUsuario->getEmail() == null || strlen($objUsuario->getEmail()) > 100) {
10             return "E-mail inválido";
11         }
12
13         if ($objUsuario->getSenha() == null || strlen($objUsuario->getSenha()) > 100) {
14             return "Senha inválida";
15         }
16     }
17 }
18 ?>
```

Nota

Adicione as validações acima ao seu arquivo `usuario_controller.php`. Podemos verificar também na imagem que utilizamos o operador `||` (ou) para que não seja necessário criar vários *ifs* semelhantes. Seguindo essa lógica a ação será executada se uma ou outra condição for atendida; no caso, se a variável for nula ou possuir mais de 100 caracteres.

Agora que nosso controller foi criado, podemos utilizá-lo na nossa view, para validar o e-mail e senha. Para isso, vamos:

- criar a variável `$controllerUsuario`, que será uma instância da classe `UsuarioController` e será utilizada para chamarmos a função `login`, passando o nosso objeto por parâmetro, como na imagem a seguir.

Lembrando que, como vamos utilizar a classe, devemos primeiramente importá-la para o nosso arquivo.

```

<?php
    $path = $_SERVER['DOCUMENT_ROOT'] . '/ecommerce';
    include_once($path."/Controllers/usuario_controller.php");

    if (isset($_POST['logar'])){
        $objUsuario = new Usuario();
        $objUsuario->setEmail($_POST["email"]);
        $objUsuario->setSenha($_POST["senha"]);

        $controllerUsuario = new UsuarioController();

        $resposta = $controllerUsuario->login($objUsuario);

        echo $resposta;
    }
?>

```

Outro detalhe importante é que a classe *Usuario* já está sendo importada na classe *UsuarioController*, a qual importamos no nosso index, ou seja, podemos retirar a importação direta da classe *Usuario* do nosso index, já que este será importado indiretamente, deixando o nosso código mais limpo e legível.

Execute o site criado até aqui e teste a validação de usuário. Se ocorrer algum erro, faz parte. Uma das habilidades dessa área profissional é saber lidar e resolver problemas. Por isso, encare com tranquilidade. Se necessário, retorne aos passos anteriores e verifique onde está o erro para corrigi-lo antes de seguir para os próximos passos.

Conexão com o MySQL

Agora que criamos a estrutura básica para a validação de e-mail e senha, precisamos acessar o nosso banco de dados para verificar se o usuário está cadastrado ou não e se a senha corresponde com a senha cadastrada. Para isso:

- crie um arquivo chamado *Conexao.php* na pasta raiz do nosso projeto, ou seja, dentro da pasta *ecommerce*. Esse arquivo vai conter a classe ***Conexao***, e uma função chamada ***getConexao()***, com os dados de conexão, e será importado nos arquivos que precisarmos.

Quando utilizamos conexão com o banco de dados no PHP, precisamos iniciar essa conexão utilizando os dados de acesso do nosso banco de dados, como endereço, nome, usuário e senha. Nós criamos o banco de dados com o nome de *ecommerce* e, por padrão, os demais dados de conexão com o MySQL são:

Endereço = localhost.

Usuario = root.

Função responsável pela conexão

A função responsável por iniciar a conexão com o banco de dados é a ***mysqli(endereço, usuario, senha, nome)***, onde passamos por parâmetro os dados de conexão. Vamos realizar também uma validação

para verificar se a conexão foi iniciada corretamente. Podemos receber os erros de conexão através da função `mysql_error()`. Altere seu arquivo `conexao.php` seguindo o modelo a seguir:

```
1 <?php
2 Class Conexao {
3     public function getConexao() {
4         //Dados da conexão
5         $host = 'localhost';
6         $bd = 'ecommerce';
7         $usuariodb = 'root';
8         $senhadb = '';
9
10        //Criando a conexão
11        $conexao = new mysqli($host, $usuariodb, $senhadb, $bd);
12
13        //Verifica se foi possível se conectar com sucesso ao banco de dados
14        if (!$conexao) die ("Erro de conexão com localhost, o seguinte erro ocorreu -> ".mysql_error());
15
16        return $conexao;
17    }
18 }
19 ?>
```

Vamos utilizar essa conexão na classe **Usuario**. Como dito anteriormente, é no model que desenvolvemos as regras de negócio do nosso sistema. Vamos iniciar:

- Importe o arquivo conexão e instancie nosso objeto de conexão, `$objConexao = new Conexao()`.
- Inicie a conexão com o banco de dados através do método `getConexao`, criado na nossa classe **Conexao** através da instrução `$conexao = $objConexao->getConexao()`.

Instrução de consulta e validação

Você lembra de como acessar os dados do banco de dados por meio de instruções SQL?

Para selecionar a informação que queremos, utilizamos a instrução **SELECT** da linguagem de consulta SQL. Vamos criar então a nossa instrução de consulta para trazer o usuário que desejamos validar, e atribuir essa consulta à variável `$sql`.

```
$sql = "SELECT Email,Senha FROM Usuarios WHERE email = " . $this->getEmail() . "";
```

Podemos verificar nessa instrução SQL que vamos selecionar os campos e-mail e senha, da tabela **Usuarios**, e através da instrução **WHERE** dizemos que o e-mail deve ser igual ao e-mail do nosso usuário. Acessamos o e-mail através da função `$this->getEmail()`, como vimos anteriormente. Lembrando que, como é uma string, devemos executar a instrução com o e-mail entre aspas.

O comando utilizado para executar instruções SQL no banco de dados MySQL é:

```
query($sql)
```

E para receber nosso resultado, utilizamos a função:

```
fetch_assoc()
```

Nosso método login fica então como na imagem a seguir. Lembre-se de manter seu código atualizado conforme a imagem.

```
public function Login() {
    $objConexao = new Conexao();
    $conexao = $objConexao->getConexao();

    $sql = "SELECT Id, Nome, Email, Senha FROM Usuarios WHERE email = '" . $this->getEmail() . "'";

    $resposta = $conexao->query($sql);
    $usuario = $resposta->fetch_assoc();
}
```

Porém, ainda precisamos comparar os dados do banco de dados com os informados pelo usuário, para garantir que ele realmente tem acesso ao nosso site.

O resultado da nossa consulta foi armazenado na nossa variável **\$usuario**, e para acessar as informações desse usuário utilizamos a instrução **\$usuario["senha"]**, onde entre colchetes e entre aspas temos o nome da nossa coluna, que informamos após a instrução **SELECT** do nosso SQL.

Através da estrutura de decisão **if** vamos validar se o usuário existe e se a senha está correta. A validação se o usuário existe será feita da mesma maneira que fizemos anteriormente, verificando se ele informou o e-mail no login, através da instrução **\$usuario == null**, que verifica se a variável existe ou não.

1. E para validar se a senha está correta, vamos comparar através do operador **!=** (diferente), que executa a ação se a variável informada for diferente da condição que passarmos. No caso, a ação será executada se a senha cadastrada no banco de dados for diferente da senha informada pelo usuário. Atualize seu código como na imagem abaixo.

```
public function Login() {
    $objConexao = new Conexao();
    $conexao = $objConexao->getConexao();

    $sql = "SELECT Id, Nome, Email, Senha FROM Usuarios WHERE email = '" . $this->getEmail() . "'";

    $resposta = $conexao->query($sql);
    $usuario = $resposta->fetch_assoc();
    if (!$usuario) {
        echo "Email não cadastrado";
    } elseif ($usuario['Senha'] != $this->getSenha()) {
        echo "Senha incorreta.";
    } else {
        echo "Sucesso";
    }

    mysqli_close($conexao);
}
```

2. Outro detalhe importante é o método **mysqli_close(\$conexao)**, que encerra a conexão com o banco de dados e deve ser utilizado após as operações realizadas no banco de dados. Em todos os métodos em que abrirmos a conexão, isso evita que um processo que não está mais sendo utilizado fique consumindo os recursos do nosso servidor.
3. A validação está pronta. Agora precisamos chamá-la no nosso controller e alterar nossa view para que o usuário consiga logar corretamente. Primeiro, vamos separar as validações do nosso método de login na classe **UsuarioController** em um novo método, deixando nosso código muito mais legível. Vamos fazer juntos primeiramente a validação de e-mail. O método **validaEmail** será criado. Ele vai receber o e-mail e retornar verdadeiro, se for válido, seguindo as regras que já desenvolvemos, ou falso, se algo estiver incorreto.
4. Após a criação do método, podemos utilizá-lo como condição no nosso **if**, para a execução da

ação login, do objeto `$objUsuario`. Atualize seu código como na imagem a seguir.

```
class UsuarioController {
    public function validaUsuario($objUsuario){
        if (validaEmail($objUsuario->getEmail())) {
            return $objUsuario->Login();
        }

        if ($objUsuario->getSenha() == null || strlen($objUsuario->getSenha()) > 100){
            return "Senha inválida";
        }
    }
}

function validaEmail($email){
    if ($email == null){
        echo "O e-mail é obrigatório.";
        return false;
    } elseif (strlen($email) > 100) {
        echo "O e-mail deve conter no máximo 100 caracteres.";
        return false;
    }
    return true;
}
```

5. Precisamos alterar também nossa view index.php, para realizar a ação de login quando recebe a resposta do controller. Para isso, vamos criar um if que redireciona se o retorno for a mensagem "Sucesso", para a listagem dos produtos do nosso sistema que vamos criar mais para frente através do comando header("Location: URL do arquivo"), onde informamos o endereço completo da nossa página web após a instrução location, como na imagem a seguir.

```
$resposta = $controllerUsuario->cadastraUsuario($objUsuario);

if ($resposta == "Sucesso"){
    header("Location: http://localhost/ecommerce/Views/inicio.php");
} else {
    echo $resposta;
}
```

Cadastro de usuário

Para o próximo passo vamos desenvolver a funcionalidade de cadastrar usuário. Para isso, vamos criar nosso método de cadastro de usuário na classe `Usuario`. Esse método terá algumas poucas diferenças se comparado com o método de login. Nossa instrução em SQL vai ficar da seguinte maneira:

```
$sql = "INSERT INTO Usuarios (Nome,Email,Senha) VALUES (".$this->nome.",".$this->email.",
".$this->senha.)";
```

Utilizando a instrução INSERT, inserimos os dados na nossa tabela do banco de dados chamada Usuarios, entre parênteses. Após o nome da tabela informamos os campos que iremos inserir e depois disso os valores já validados. Para executar a instrução, vamos usar agora a função `mysqli_query($conexao, $sql)`, que recebe o objeto de conexão e a instrução a ser executada, retornando verdadeiro se tudo foi executado corretamente e falso se algo deu errado. Nosso método de cadastro fica como na imagem a seguir:

```

public function Cadastrar() {
    $objConexao = new Conexao();
    $conexao = $objConexao->getConexao();

    $sql = "INSERT INTO Usuarios (Nome,Email,Senha)
            VALUES (".$this->nome.", " . $this->email.", " . $this->senha.)";

    if (mysqli_query($conexao, $sql)){
        return "Sucesso";
    } else {
        return "Erro";
    }

    mysqli_close($conexao);
}

```

Agora é a sua vez de praticar! Realize a seguinte atividade para criarmos os métodos que irão utilizar o método Cadastrar que criamos anteriormente. Se estiver com alguma dúvida, analise os métodos que já criamos, tomando-os como base para criar os novos.

[MOD 3] Atividade autoavaliativa: Criação dos métodos de cadastro

Crie um novo método para validar a senha informada, separando as validações de senha do nosso método de login do controller, e chame o método na condição do **if**, para que execute a ação somente se o e-mail e o login estiverem válidos.

Crie também a tela e os métodos necessários para cadastrarmos um novo usuário. Crie uma pasta chamada **Usuario**, dentro da pasta view, para separarmos todas as views relacionadas aos usuários nessa pasta. Crie também a view **cadastro_usuario.php**, dentro da pasta **Usuario**. Essa view será semelhante ao nosso index, só que nesse caso deve conter os campos nome, e-mail, senha e o botão cadastrar, que deverá ser criado com o atributo **name = "cadastar"**. Nosso formulário tem o atributo **action** vazio e o método **post**, para que ele seja redirecionado para a própria página. Logo abaixo da view vamos criar o código em PHP que irá importar a classe **UsuarioController**, e quando o botão cadastrar for pressionado, deve criar o objeto usuário e instanciar o controller de usuário. Deve também utilizar o controller instanciado para chamar o método **cadastroUsuario**, passando como parâmetro o objeto usuário criado.

Já na classe **UsuarioController**, crie outro método, chamado **cadastroUsuario**, que recebe como parâmetro um objeto usuário, valida o nome, e-mail e senha informados, e retorne a instrução **\$objUsuario->Cadastrar()**, que retorna o resultado do método cadastrar, que criamos no nosso modelo Usuário.

Realize as ações propostas e então siga para realizar sua autoavaliação.

Resultado

Se estiver com alguma dúvida, analise os métodos que já criamos, tomando-os como base para criar os novos.

View cadastro_usuario.php

Parabéns, aluno, tivemos um grande progresso até aqui. Como foi o desenvolvimento das classes e dos métodos do exercício anterior? Teve alguma dificuldade? Quais? Essas perguntas são importantes não só para você enquanto aluno, mas também como profissional. O tempo todo temos que nos questionar para buscar novas competências, novos conhecimentos, habilidades e atitudes.

É importante sempre testarmos nosso software e nos atentarmos aos detalhes, como os logs de erro por exemplo, que podem nos guiar a uma correção e melhoria para o nosso site.

Classe Produto

Agora que finalizamos nossos métodos para o usuário, vamos partir para os produtos. Já aprendemos como criar a view, o controller e o model anteriormente com o usuário. Essa mesma estrutura irá se repetir para o produto, então essa parte fica com você, ok? Lembre-se de criar os arquivos e os métodos necessários para cadastrar e validar um produto, com suas informações específicas, como vimos no modelo do banco de dados.

Nessa próxima etapa, vamos focar na edição e listagem dos produtos, começando por esta última:

- cadastre alguns no nosso site para que o nosso exemplo fique bem legal;
- na nossa classe Produto, no model produto, altere nosso SQL para **"SELECT * FROM Produtos"**, para selecionar todas as informações utilizando o caractere * da tabela Produtos;
- receba a resposta na variável **\$resposta**. Como estamos buscando vários produtos, a resposta será um array vindo do banco de dados, onde cada linha da nossa tabela, que corresponde a um produto, tem um espaço nesse array;
- acesse cada linha do array através da instrução **while**, disponível no PHP;
- dentro dos parênteses do **while**, utilize o método **mysqli_fetch_assoc(\$resultado)**, que retorna cada linha do array e atribui a linha à variável **\$produto**;
- a variável será adicionada no array **\$arrayProdutos** e retorna para o nosso controller;
- utilizando os conhecimentos construídos até agora, crie o método no controller de produtos que recebe a solicitação da view listagem_produtos e retorna o método **\$objProduto->listar()**.

Lembre-se de criar o objeto produto antes de utilizá-lo. Vemos nosso método de login na prática na imagem a seguir.

```
public function listar(){
    $objConexao = new Conexao();
    $conexao = $objConexao->getConexao();
    $arrayProdutos = [];
    $sql = "SELECT * FROM Produtos";

    $resposta = mysqli_query($conexao, $sql);
    while($produto = mysqli_fetch_assoc($resposta)){
        array_push($arrayProdutos, $produto);
    }
    mysqli_close($conexao);

    return $arrayProdutos;
}
```


Nota importante!

A estrutura de repetição **while**, enquanto em português, é uma das mais usadas na linguagem PHP. Sua função é executar um bloco de instruções enquanto a condição for atendida e segue a estrutura abaixo:

```
<?php
    $i = 0;
    while ($i < 10){
        echo $i;
        $i++;
    }
?>
```

Para exibir os produtos em tela, vamos receber o retorno do método **listaProdutos()** do nosso controller na variável **\$produtos** e criar uma tabela de produtos com ela. Para isso, vamos:

- utilizar o método **foreach()**, que entre parênteses irá receber o nosso array, e vai executar a ação proposta para cada produto desse array;
- para exibir as informações vamos criar um apelido que esses produtos irão receber. Fazemos isso através do comando **as**;
- para cada item, vamos criar uma linha da tabela em HTML e uma coluna para cada informação. Como aprendemos anteriormente, podemos utilizar marcações em HTML na função **echo** do PHP.

Atualize seu arquivo de listagem como o da imagem a seguir.

```
<?php
    foreach($produtos as $produto){
        echo "<tr>";
        echo "<td>";
            echo $produto['id'];
        echo "</td>";
        echo "<td>";
            echo $produto['Descricao'];
        echo "</td>";
        echo "<td>";
            echo $produto['Valor'];
        echo "</td>";
        echo "<td>";
            echo $produto['Categoria'];
        echo "</td>";
        echo "<td>";
            echo $produto['Quantidade'];
        echo "</td>";
        echo "</tr>";
    }
?>
```

Nota importante!

A estrutura de repetição foreach, para cada em português, tem como função executar um bloco de instruções para cada item que passarmos entre parênteses. Conforme exemplo da imagem a seguir, a ação será executada para cada item do array que passamos. No caso, será criada uma linha da tabela para cada produto. Acessamos esse item do array nomeando cada um após a instrução as.

```
<?php
foreach($produtos as $produto){
    echo "<tr>";
    echo "<td>";
    echo $produto['id'];
    echo "</td>";
    echo "<td>";
    echo $produto['Descricao'];
    echo "</td>";
    echo "<td>";
    echo $produto['Valor'];
    echo "</td>";
    echo "<td>";
    echo $produto['Categoria'];
    echo "</td>";
    echo "<td>";
    echo $produto['Quantidade'];
    echo "</td>";
    echo "</tr>";
}
?>
```

Edição e exclusão de informações

Estamos quase finalizando...

já aprendemos o relacionamento entre as classes, conexão com o banco de dados, cadastro e edição de informações, criação de novos métodos e vários métodos importantes do PHP, mas ainda falta aprendermos a editar e excluir as informações. Exercite seu conhecimento desenvolvendo as views, models e controllers que ainda não fizemos juntos. Lembre-se que a prática leva à perfeição.

Partindo do princípio de que já desenvolvemos a capacidade de criar as páginas e os métodos necessários para cadastrar e listar informações, vamos criar juntos os métodos necessários para a edição de usuários e exclusão de produtos do carrinho.

Edição

Para editar o usuário vamos utilizar:

- a variável `$_SESSION`, que salva as informações de que precisamos enquanto o usuário está no

nosso site. Isso permite que criemos uma variável em uma tela e utilizemos em outra sem precisar usar o método *post*. Um detalhe muito importante é que antes de utilizarmos essa variável devemos iniciar a sessão através da instrução *session_start()*. Vamos iniciar a sessão e criar a variável *\$_SESSION['usuario_id']* quando o usuário loga no nosso sistema.

Atualize seu arquivo *index* com as informações de seção como na imagem a seguir.

```
if ($resposta == 'Sucesso'){
    session_start();
    $_SESSION["usuario_id"] = $objUsuario->getId();
    header("Location: http://localhost/ecommerce/Views/Produto/listagem_produtos.php");
} else {
    echo "Acesso negado!<br>";
}
```

Pronto, nossa variável foi criada na sessão atual. Como visto na imagem, assim que realizado o login o usuário é redirecionado para a página de listagem dos produtos através do método *Header*. Nela foi criado um botão semelhante que redireciona para a página de *editar_usuario.php*.

Na página de edição, vamos iniciar a sessão e criar uma variável *\$usuario*, que vai receber um usuário do método *getUsuario(\$_SESSION['usuario_id'])*, que deve ser criado no seu respectivo controller. Esse método recebe como parâmetro o id do usuário, que armazenamos na variável *\$_SESSION*, seleciona o usuário através da instrução *"SELECT * FROM Usuarios WHERE id = \$id"* e retorna o resultado obtido. Nosso código na view fica como na imagem a seguir.

```
session_start();
$usuario = $controllerUsuario->getUsuario($_SESSION["usuario_id"]);

if (isset($_POST['editar'])){
    $objUsuario = new Usuario($_POST['nome'], $_POST['email'], $_POST['senha'], $_POST['endereco']);
    $controllerUsuario = new UsuarioController();
    $resposta = $controllerUsuario->editaUsuario($usuario['ID'],$objUsuario);

    if ($resposta == "Sucesso"){
        $usuario = $controllerUsuario->getUsuario($_SESSION["usuario_id"]);
        echo "Conta salva com sucesso!";
    } else {
        echo $resposta;
    }
}
```

Crie também o método de edição no model *Usuario*, de maneira muito semelhante ao método de cadastro. Alteramos apenas a instrução SQL, como na imagem a seguir.

```
public function Editar($id) {
    $objConexao = new Conexao();
    $conexao = $objConexao->getConexao();

    $sql = "UPDATE Usuarios set Nome = '". $this->nome."',
        Email = '". $this->email."',
        Senha = '". $this->senha."',
        Endereco = '". $this->endereco."'
        WHERE id = ".$id;

    if (mysqli_query($conexao, $sql)){
        return "Sucesso";
    } else {
        return "Erro";
    }

    mysqli_close($conexao);
}
```

Já o nosso formulário será um pouco diferente. Dessa vez, ele receberá também um valor utilizando o PHP, como na imagem a seguir. Lembre-se de atualizar o seu código.

```
<html>
<body>
  <form action="" method="post">
    <h3>Cadastro de Usuário</h3>
    Nome: <input type="text" name="nome" value="<?php echo $usuario['Nome'] ?>"><br>
    Email: <input type="text" name="email" value="<?php echo $usuario['Email'] ?>"><br>
    Senha: <input type="text" name="senha" value="<?php echo $usuario['Senha'] ?>"><br>
    Endereço: <input type="text" name="endereco" value="<?php echo $usuario['Endereco'] ?>"><br>
    <button type="submit" name="editar">Editar</button>
  </form>
  <a href="http://localhost/ecommerce2/Views/Produto/listagem_produtos.php"><button type="submit">Listagem de
  Produtos</button></a>
</body>
</html>
```

A edição é muito semelhante ao cadastro, não é mesmo? Só que nesse processo precisamos carregar as informações que vamos editar e passá-las para o model realizar a alteração no banco de dados.

Exclusão

Seguimos o mesmo princípio para a exclusão. O método deve receber o id do objeto que vamos excluir e realizar a exclusão no banco de dados através da instrução **DELETE FROM Carrinhos WHERE id = \$id**. Atualize seu código como na imagem de exemplo a seguir.

```
public function excluir($id){
    $objConexao = new Conexao();
    $conexao = $objConexao->getConexao();

    $sql = "DELETE FROM Carrinhos WHERE id = ".$id;

    $resposta = mysqli_query($conexao, $sql);
    if (mysqli_query($conexao, $sql)){
        return "Sucesso";
    } else {
        return "Erro";
    }
    mysqli_close($conexao);
}
```

[FÓRUM] LAYOUT FINAL

[FÓRUM] Layout Final

Parabéns pelo esforço e dedicação durante o curso. Agora chegamos na etapa final e até aqui você já aprendeu sobre o relacionamento entre classes e os métodos necessários para cadastrar, excluir e editar informações. A partir disso, desenvolva as funcionalidades do nosso e-commerce da maneira que preferir.

- Cadastro, edição e exclusão de usuários, produtos, carrinho e pedidos, com seus respectivos models, controllers e views.
- Views para listagem de produtos, carrinho, usuários e pedidos já realizados.

Clique no link abaixo para acessar o Fórum!

Você já aprendeu sobre o relacionamento entre classes e os métodos necessários para cadastrar, excluir e editar informações. A partir disso, desenvolva as funcionalidades do nosso e-commerce da maneira que preferir.

- Cadastro, edição e exclusão de usuários, produtos, carrinho e pedidos, com seus respectivos models, controllers e views.
- Views para listagem de produtos, carrinho, usuários e pedidos já realizados.

Ao finalizar a sua atividade, realize as seguintes ações:

- poste no fórum;
- comente na postagem de pelo menos um colega, trazendo contribuições de melhoria.

ATENÇÃO

Lembre-se que as regras de negócio e as operações com o banco de dados ficam localizadas no model. As views são responsáveis pela parte visual do software e os controllers validam os campos recebidos e encaminham a solicitação para o model. Se tiver dúvidas na criação, volte para os arquivos que criamos juntos e siga o mesmo padrão nas demais telas.

[FECHAMENTO] MÓDULO 3

Olá novamente, aluno (a)!

Ouça o que o autor tem a dizer e só clicar no PLAY!

Caso não tenha conseguido ouvir, leia a transcrição clicando AQUI!

Transcrição do Podcast

Autor: Vitor Gabriel Martines Weinfortner

Podcast de encerramento

Olá novamente aluno (a), parabéns por chegar até aqui e por todo o esforço realizado nesse processo!

Espero muito que tenha gostado e aproveitado ao máximo nosso curso e que pense em utilizar essa linguagem no seu futuro profissional.

Nunca se esqueça que a prática leva à perfeição, praticar muito e criar um portfólio legal é muito importante para o mercado de trabalho no ramo de desenvolvimento.

Bom trabalho e bons estudos.

Sucesso e até mais!

Esperamos muito que tenham gostado e aproveitado ao máximo nosso curso de PHP. Nunca se esqueça que a prática leva à perfeição e praticar muito e criar um portfólio diversificado e organizado é muito importante para o mercado de trabalho no ramo de desenvolvimento.

Sucesso!

