ANÁLISE COMPARATIVA ENTRE PHP E LARAVEL: DESEMPENHO E ESTRUTURA NO DESENVOLVIMENTO WEB

João Pedro Warmling de Oliveira[1], Diorgines Mattos Machado[2]

Resumo: O presente trabalho tem como objetivo analisar comparativamente as tecnologias PHP e o framework Laravel, considerando aspectos de estrutura, desempenho, escalabilidade, segurança e compatibilidade com bancos de dados no desenvolvimento web. A pesquisa foi de natureza aplicada e caráter experimental, envolvendo o desenvolvimento de duas aplicações equivalentes, uma em PHP puro e outra com o framework Laravel, além de uma revisão bibliográfica sobre o uso e a evolução dessas tecnologias. Os resultados demonstraram que o Laravel apresentou desempenho superior em praticamente todos os critérios avaliados, destacando-se pela aplicação do padrão MVC, pela automação de tarefas com o Artisan CLI e pelo uso de recursos como Eloquent ORM e migrations, que aumentaram a produtividade e a segurança. O PHP puro, embora mais rápido em tarefas simples, mostrou limitações quanto à padronização, manutenção e escalabilidade. Constatou-se, portanto, que o framework Laravel oferece uma solução mais robusta e eficiente para sistemas de médio e grande porte, enquanto o PHP se mantém adequado para aplicações menores e de execução direta.

Palavras-chave: desenvolvimento web; PHP; Laravel; desempenho; frameworks.

[1]Curso de Ciência da Computação, Universidade do Extremo Sul Catarinense (Unesc),j.pedro.wo973@gmail.com

[2]Orientador.Curso de Ciência da Computação, Universidade do Extremo Sul Catarinense (Unesc), diorginesmattos@unesc.net

ABSTRACT: This study aims to perform a comparative analysis between the PHP language and the Laravel framework, considering aspects such as structure, performance, scalability, security, and database compatibility in web development. The research is applied in nature and experimental in character, involving the development of two equivalent applications, one using pure PHP and the other developed with the Laravel framework, in addition to a literature review on the use and evolution of these technologies. The results showed that Laravel achieved superior performance in almost all evaluated criteria, standing out for its implementation of the MVC pattern, task automation through Artisan CLI, and the use of resources such as Eloquent ORM and migrations, which enhanced productivity and security. Pure PHP, although faster in simple tasks, presented limitations regarding standardization, maintainability, and scalability. Therefore, it was found that Laravel provides a more robust and efficient solution for medium and large-scale systems, while pure PHP remains suitable for smaller and more direct applications.

Keywords: web development; PHP; Laravel; performance; frameworks.

1 INTRODUÇÃO

Desde sua chegada ao Brasil em 1995, a internet cresceu continuamente, tornando-se parte essencial do cotidiano da sociedade. Esse aumento constante no uso trouxe consigo uma demanda crescente por sistemas web que não sejam apenas funcionais, mas também rápidos, seguros e capazes de lidar com um grande volume de acessos simultâneos, Com a escolha da linguagem de programação e do *framework* adequado tornou-se estratégica para desenvolvedores e empresas, entre as alternativas disponíveis, o PHP e o *framework* Laravel se destacam como soluções amplamente utilizadas, cada uma com abordagens específicas para atender às demandas de desenvolvimento (Khan; Khanam, 2023)

O PHP foi inicialmente concebido como uma linguagem de *script* para sites dinâmicos e, ao longo dos anos, passou por uma evolução significativa, é amplamente utilizado no backend de plataformas web de diferentes portes, pela sua sintaxe simples, a facilidade de integração com HTML e a compatibilidade com uma ampla variedade de bancos de dados o tornam uma escolha popular entre desenvolvedores que buscam agilidade, no entanto projetos de maior escala e complexidade podem revelar algumas limitações do PHP, especialmente no que diz respeito à organização do

código e à sua manutenção em cenários mais robustos (Saroni; Mulyanti, 2020).

O Laravel é um framework que amplia as possibilidades do PHP, oferecendo uma estrutura organizada e um conjunto de ferramentas prontas para uso, com base na arquitetura MVC (*Model-View-Controller*), que promove separação clara entre as responsabilidades do sistema, e também oferece ferramentas como roteamento simplificado, suporte a migrações de banco de dados e um sistema de autenticação integrado, tornando-se uma alternativa atraente para projetos que demandam produtividade e escalabilidade (Adam; Andolo, 2019).

A adoção de um *framework* pode aumentar a complexidade do ambiente de desenvolvimento e exigir um conhecimento mais avançado dos programadores, os frameworks como o Laravel tendem a ser mais exigentes em termos de recursos do servidor, o que pode impactar o desempenho em situações de alta demanda (Abutaleb; Tamimi; Alrawashdeh, 2021).

Em um mercado de desenvolvimento web cada vez mais competitivo, as empresas precisam oferecer serviços rápidos e eficientes. Os sistemas que não acompanham esse ritmo correm o risco de perder usuários e prejudicar a imagem da marca. Torna-se, portanto, essencial encontrar formas de melhorar o desempenho em sistemas desenvolvidos tanto com PHP quanto com Laravel, sem comprometer a segurança e a escalabilidade (Cardoso; Bispo, 2019).

Vários estudos recentes destacam o uso do framework Laravel em aplicações web, evidenciando sua eficiência, segurança e escalabilidade em diferentes contextos. Na pesquisa de Cajado (2024), realizada no Instituto Federal do Maranhão, foi desenvolvido um sistema de automação de laboratórios com integração via ESP8266, utilizando o Laravel como base para o backend. O autor demonstrou que o uso do padrão MVC, aliado a recursos como Eloquent ORM, migrations e o Artisan CLI, proporciona maior organização, rastreabilidade e produtividade, comprovando a viabilidade do framework em soluções integradas à Internet das Coisas (IoT).

Da mesma forma, Caldeira (2023), na Universidade Portucalense, aplicou o Laravel no desenvolvimento de um sistema de autenticação e assinatura digital em conformidade com o regulamento europeu eIDAS. O estudo explorou funcionalidades nativas de segurança, rotas protegidas, criptografia e integração com APIs externas. Seus resultados evidenciam que o framework oferece uma base confiável e escalável para aplicações

jurídicas digitais, reforçando seu papel em sistemas que exigem conformidade e integridade de dados.

Complementarmente, Lima (2025), na Universidade Federal do Ceará, realizou um estudo comparativo entre os frameworks Laravel, Django e Ruby on Rails, analisando critérios como produtividade, arquitetura e curva de aprendizado. O autor concluiu que o Laravel apresenta desempenho equilibrado, destacando-se pela modularidade, facilidade de manutenção e curva de aprendizado mais acessível, o que o torna uma ferramenta ideal para projetos de médio porte.

O objetivo deste estudo é analisar comparativamente as tecnologias PHP e o *framework* Laravel, considerando aspectos de estrutura, desempenho, escalabilidade e segurança no desenvolvimento web.

Para isso, o trabalho envolve uma revisão de artigos e publicações científicas sobre o uso dessas tecnologias e o desenvolvimento de uma aplicação implementada em ambas, uma versão em PHP puro e outra com o *framework* Laravel, com o foco de fazer uma avaliação prática e fundamentada, em que busca-se identificar as principais funcionalidades e limitações de cada abordagem, analisando seu impacto na organização do código, na velocidade de desenvolvimento, na segurança e na compatibilidade com bancos de dados, verificando ao final a eficiência e a escalabilidade de cada solução e apontando qual delas apresenta melhores resultados para diferentes tipos de sistemas web.

2 MATERIAIS E MÉTODOS

A pesquisa envolveu conhecimentos de disciplinas do curso de Ciência da Computação que compõem a base teórica e prática necessária ao desenvolvimento de sistemas, como Programação Web, Banco de Dados, Estrutura de Dados e Análise de Algoritmos.

O objetivo foi comparar o desenvolvimento de sistemas utilizando a linguagem PHP em sua forma pura (ou seja, sem o uso de *frameworks* ou bibliotecas externas) com outro sistema construído a partir do *framework* Laravel. O estudo buscou identificar diferenças em estrutura e desempenho e apresentar resultados para o aprimoramento do processo de desenvolvimento.

Esse estudo teve caráter experimental e exploratório, fundamentado em levantamento bibliográfico e desenvolvimento prático de *software*, seguindo o método comparativo.

Duas soluções equivalentes foram elaboradas e testadas sob o

mesmo banco de dados e ambiente de execução, permitindo uma análise técnica equilibrada entre as tecnologias. O estudo foi dividido em duas etapas principais:

I – Levantamento bibliográfico: foi realizada uma pesquisa teórica e técnica sobre as características estruturais, de desempenho e metodológicas das linguagens analisadas, com foco na avaliação comparativa entre uma linguagem pura (PHP) e outra baseada em *framework* (Laravel). Nesta etapa, também foi realizado o estudo de métodos de análise algorítmica, visando compreender como cada tecnologia organiza, processa e otimiza o fluxo de execução e o gerenciamento de dados;

II – Desenvolvimento do Sistema: consistiu na implementação de um sistema de controle de estoque em ambas as tecnologias, utilizando o mesmo banco de dados e conjunto de funcionalidades, esta abordagem possibilitou mensurar critérios como estrutura, desempenho, velocidade de desenvolvimento, escalabilidade, segurança e compatibilidade com bancos de dados, permitindo uma análise entre as duas ferramentas.

2.1 LEVANTAMENTO BIBLIOGRÁFICO

Na literatura, investigaram-se as características estruturais, de desempenho e metodológicas das linguagens analisadas, com foco na avaliação comparativa entre uma linguagem pura (PHP) e outra baseada em *framework* (Laravel).

Nessa comparação, os autores abordam aspectos essenciais, como a organização do código, o processamento de informações e a influência na eficiência de execução. A partir dessas contribuições, foram identificados seis pontos principais de análise, que serviram de base para o método de avaliação adotado nesta pesquisa:

a) estrutura: buscou-se identificar, com base em autores como (Vicente; Sirqueira, 2020) e (Almeida; outros, 2025), que o PHP puro tende a oferecer liberdade total na organização do código, exigindo maior disciplina do desenvolvedor, ao passo que o Laravel aplica o padrão MVC (*Model-View-Controller*), promovendo modularidade, separação de responsabilidades e melhor manutenção do sistema;

b) desempenho: foram analisados estudos, como os de (Saroni; Mulyanti, 2020) e (Laaziri; outros, 2019), que discutem a influência das camadas adicionais introduzidas pelos *frameworks* sobre o tempo de resposta e o consumo de recursos. Esses estudos destacam que o PHP puro tende a ser mais leve em tarefas simples, enquanto o Laravel compensa

com otimizações internas e mecanismos de cache integrados;

c) velocidade de desenvolvimento: de acordo com z(Adam; Andolo, 2019) e (Ferreira, 2021), o Laravel reduz o esforço manual ao oferecer ferramentas prontas, como o Artisan CLI, as migrações e o Eloquent ORM, que automatizam tarefas comuns no desenvolvimento. Isso contrasta com o PHP puro, no qual é necessária a criação manual dessas funcionalidades, o que aumenta o tempo de desenvolvimento e manutenção;

d) escalabilidade: segundo (Costa, 2025) e (Sunardi et al., 2019), o Laravel se destaca por incluir ferramentas nativas para o gerenciamento de filas, cache e variáveis de ambiente, o que facilita o crescimento das aplicações e o trabalho em ambientes distribuídos, em contrapartida, o PHP, embora flexível, requer configurações adicionais e adaptações manuais para alcançar o mesmo nível de escalabilidade;

e) segurança: os estudos de (Paula; Oliveira, 2024) e (Oliveira, 2023) evidenciam que o Laravel oferece mecanismos automáticos de proteção contra vulnerabilidades comuns, como injeção em SQL, XSS (*cross-site scripting*) e CSRF (*cross-site request forgery*), enquanto no PHP puro essas medidas devem ser implementadas manualmente pelo desenvolvedor, demandando maior cuidado na codificação;

f) compatibilidade com bancos de dados: de acordo com (Souza, 2021) e (Camilo, 2019), ambas as tecnologias apresentam boa integração com o MySQL e outros sistemas de gerenciamento de banco de dados.

No entanto, o Laravel oferece um controle mais automatizado e seguro por meio de migrações, sementes e transações nativas, enquanto o PHP puro depende de *scripts* manuais para criar e manipular estruturas de dados.

Com base nesse levantamento, foi definido o método de avaliação comparativa, que consistiu no desenvolvimento de dois sistemas idênticos: um em PHP puro e outro em Laravel. Sobre eles, foram aplicados testes específicos para cada critério citado. Esses testes possibilitaram observar, na prática, como as diferenças teóricas se manifestam no comportamento real das aplicações.

Para sistematizar as informações obtidas durante o levantamento bibliográfico e evidenciar a forma como cada autor avaliou as tecnologias estudadas, foi elaborada a Tabela 1.Ela sintetiza as características analisadas, os métodos práticos de avaliação identificados nos estudos e as respectivas referências da pesquisa.

Tabela 1 - Relação entre características analisadas

CARACTERÍSTICA	MÉTODO PRÁTICO DE AVALIAÇÃO	REFERÊNCIA
Estrutura	Avaliação da organização do código e da separação de responsabilidades entre PHP puro e Laravel, com base na adoção do padrão MVC e modularidade das pastas.	(Vicente; Sirqueira, 2020); (Almeida; outros, 2025)
Desempenho	Medição do tempo de resposta e consumo de recursos em aplicações equivalentes; comparação de inserções e requisições entre PHP puro e Laravel.	(Saroni; Mulyanti, 2020); (Lazaziri; outros, 2019)
Velocidade de desenvolvimento	Mensuração do tempo e da quantidade de arquivos criados na implementação de um CRUD, avaliando o uso de ferramentas como Artisan CLI, migrations e ORM.	(Adam; Andolo, 2019); (Ferreira, 2021)
Escalabilidade	Simulação de aumento de usuários e requisições simultâneas, observando a capacidade de adaptação de cada tecnologia e o uso de filas, cache e variáveis de ambiente.	(Costa, 2025); (Sunardi et al., 2019)
Segurança	Testes práticos de resistência contra ataques simulados, como SQL Injection, XSS e CSRF, verificando proteções automáticas e manuais de cada abordagem.	(Paula; Oliveira, 2024)
Compatibilidade com bancos de dados	Análise da integração prática com MySQL e do uso de migrations, seeders e transações; verificação da automação e versionamento do banco.	(Souza, 2021); (Camilo, 2019)

Fonte: do autor.

2.1.1 Programação em PHP e Laravel do Sistema

O sistema de controle de estoque foi projetado e desenvolvido para manter a equivalência total de funcionalidades entre as versões em PHP puro e Laravel, assegurando uma comparação justa e coerente entre as duas tecnologias. A estrutura e o fluxo de navegação foram planejados de maneira idêntica para que as diferenças observadas se concentrassem exclusivamente nos aspectos técnicos e nas diferenças de implementação do *backend*.

As telas e funcionalidades desenvolvidas nas duas versões foram: i) Tela de login: responsável pela autenticação de usuários e pelo controle de acesso ao sistema; ii) Cadastro de usuário: possibilita o registro de novos usuários e a definição de suas credenciais de acesso; iii) Página inicial: apresenta o painel principal com atalhos e resumos das operações disponíveis; iv) Listagem de produtos: exibe, de forma organizada, os produtos cadastrados, com opções de busca e filtragem; v) Cadastro de produtos: permite incluir novos itens no estoque com informações de nome, categoria, preço e quantidade; vi) Edição de produtos: oferece a possibilidade de alterar dados existentes, assegurando a atualização contínua do inventário.

2.1.2 Tela de Login

Na versão em PHP puro, a autenticação foi implementada na página login.php, que continha um formulário HTML com campos para e-mail e senha. A validação dos dados era feita manualmente e as credenciais eram enviadas ao script autentica.php, que

era responsável por verificar o usuário no banco de dados por meio da PDO.O gerenciamento de sessões foi realizado por meio das funções nativas do PHP (`session_start()` e `$_SESSION`). , sem o uso de bibliotecas externas. Isso exigiu um controle manual maior sobre a segurança e a persistência da sessão. Na versão desenvolvida em Laravel, o processo de login é implementado pela rota /login, controlada pelo AuthController. O *framework* utiliza o método $request->validate() para garantir que os campos obrigatórios estejam preenchidos corretamente antes da autenticação. O *hash* da senha é aplicado automaticamente com bcrypt e a autenticação é tratada pelo sistema Auth nativo do Laravel, proporcionando assim maior segurança e reduzindo significativamente o volume de código necessário.

As views foram criadas utilizando o Blade Template Engine, que permite a separação entre a camada visual e a lógica de controle e oferece uma estrutura mais organizada e reutilizável. Essa abordagem evidencia a diferença entre o desenvolvimento manual em PHP puro e o uso de abstrações e automações oferecidas pelo Laravel. A Figura 1 apresenta o comparativo entre os trechos de código responsáveis pela autenticação nas duas tecnologias, enquanto a Figura 2 exibe a interface da tela de login desenvolvida nas duas versões, demonstrando a padronização visual e funcional proposta para manter as mesmas condições de análise entre os ambientes.

Figura 1 - Comparativo entre o código de autenticação em PHP e Laravel

LARAVEL

```php
public function login(Request $request)
    {
        $request->validate([
            'email' => 'required|email',
            'password' => 'required|min:6',
        ]);
        if (Auth::guard('web')->attempt([
            'email' => $request->email,
            'password' => $request->password
        ])) {
            $request->session()->regenerate();
            return redirect()->route('home')->with('sucesso', 'Login realizado com sucesso!');
        }
        return back()->withErrors([
            'email' => 'Email ou senha inválidos.'
        ])->withInput();
    }
```

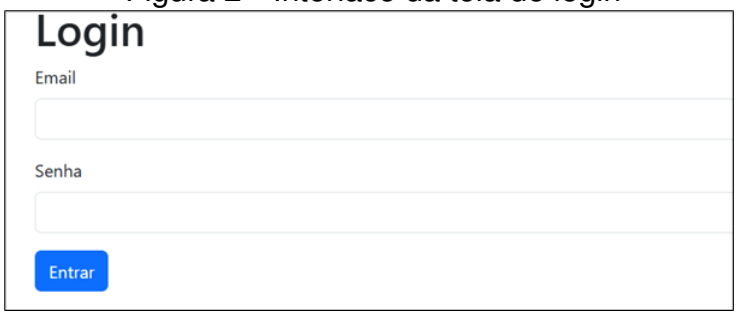
PHP

```php
$email = $_POST['email'] ?? null;
$senha = $_POST['senha'] ?? null;
if($email && $senha) {
    $stmt = mysqli_prepare($conexao, "SELECT id, nome, senha FROM usuarios WHERE email = ?");
    mysqli_stmt_bind_param($stmt, "s", $email);
    mysqli_stmt_execute($stmt);
    mysqli_stmt_bind_result($stmt, $id, $nome, $senhaHash);
    mysqli_stmt_fetch($stmt);
    mysqli_stmt_close($stmt);

    if(isset($senhaHash) && password_verify($senha, $senhaHash)) {
        $_SESSION['usuario_id'] = $id;
        $_SESSION['usuario_nome'] = $nome;
        header("Location: index.php");
        exit();
    } else {
        echo "<div class='alert alert-danger'>Email ou senha incorretos.</div>";
    }
} else {
    echo "<div class='alert alert-warning'>Preencha todos os campos.</div>";
}
```

Fonte: Do Autor.

Figura 2 - Interface da tela de login



Fonte: Do Autor.

2.1.3 Cadastro de Usuário

No sistema desenvolvido em PHP puro, o cadastro de usuários foi implementado por meio da página usuário-formulário.php, que é responsável por coletar os dados e enviá-los, via método POST, ao *script* cadastra-usuário.php. A inserção no banco de dados era feita manualmente com instruções SQL INSERT INTO, e o gerenciamento de erros e as verificações básicas eram feitos diretamente no código PHP. Como não havia validação automatizada, era necessária maior atenção do desenvolvedor para prevenir inconsistências e falhas de segurança.

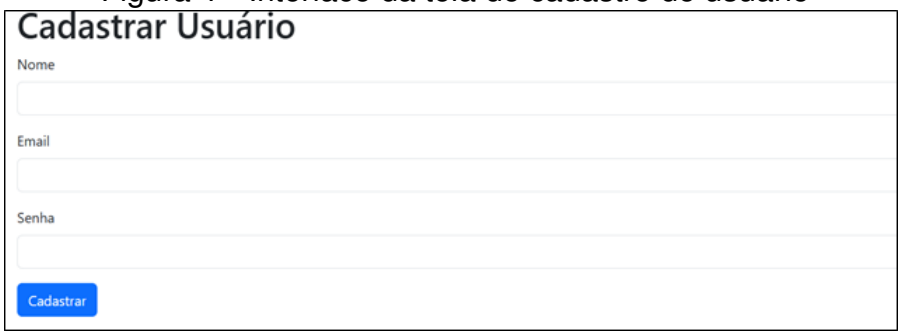
No Laravel, o processo de cadastro foi desenvolvido por meio da rota /*register*, controlada pelo UsuarioController, que utiliza o método $request->validate() para verificar os campos obrigatórios e garantir a integridade dos dados antes do armazenamento. A persistência foi tratada de forma automatizada pelo Eloquent ORM por meio do método User::create(), que abstrai a manipulação direta do SQL e proporciona maior segurança.

A camada de apresentação foi construída com o Blade Template Engine, que permite a separação entre a lógica de controle e o *layout* visual, resultando em um código mais limpo, organizado e fácil de manter. Essa diferença estrutural pode ser observada na comparação de códigos apresentada na Figura 3, enquanto a Figura 4 ilustra a tela de cadastro de usuário desenvolvidas nas duas versões, evidenciando a equivalência funcional e a padronização visual entre os sistemas.

Figura 3 - Comparação entre códigos

Fonte: Do Autor.

Figura 4 - Interface da tela de cadastro de usuário



Fonte: Do Autor.

2.1.4 Página Inicial

Na aplicação em PHP puro, a página index.php atuava como o painel principal, acessado após a autenticação do usuário, eram exibidos *links* de navegação para cadastro e listagem de produtos, bem como atalhos para outras funcionalidades do sistema, como gerenciamento de categorias e usuários, a interface foi construída em HTML puro. Isso demandou um maior esforço manual para a organização e a padronização visual dos elementos da interface.

Na versão em Laravel, a página inicial foi configurada como uma *view* principal (home.blade.php), acessível apenas após autenticação por meio de *middleware*. O *layout* foi estruturado com Blade Components, o que possibilitou o reaproveitamento de elementos visuais e a manutenção centralizada do design.

A estilização foi gerenciada pelo Vite, ferramenta nativa do *framework* para compilação e versionamento de arquivos CSS, garantindo maior desempenho e padronização visual. A Figura 5 apresenta a comparação entre as implementações, mostrando o contraste entre os códigos das duas abordagens. A Figura 6 exibe a interface final da página inicial, evidenciando a equivalência funcional e as diferenças estruturais e visuais entre os projetos.

Figura 5 - Comparativo entre o código da página inicial em PHP e Laravel

LARAVEL

```
<div class="text-center mt-5 pt-5">
    <h1 class="display-4 mb-4">Bem-vindo ao Meu Projeto!</h1>
    <p class="lead mb-4">Gerencie seus produtos de forma simples e prática.</p>

    @auth
        <a href="{{ route('produtos.index') }}" class="btn btn-success btn-lg mx-2">Ver Produtos</a>
        <a href="{{ route('produtos.create') }}" class="btn btn-primary btn-lg mx-2">Adicionar Produto</a>
    @else
        <a href="{{ route('usuarios.create') }}" class="btn btn-primary btn-lg mx-2">Cadastrar-se</a>
        <a href="{{ route('login') }}" class="btn btn-secondary btn-lg mx-2">Login</a>
    @endauth
</div>
```

PHP

```
<div class="text-center mt-5 pt-5">
    <h1 class="display-4 mb-4">Bem-vindo ao Meu Projeto!</h1>
    <p class="lead mb-4">Gerencie seus produtos de forma simples e prática.</p>

    <?php if(isset($_SESSION['usuario_nome'])): ?>
        <a href="produto-lista.php" class="btn btn-success btn-lg mx-2">Ver Produtos</a>
        <a href="produto-formulario.php" class="btn btn-primary btn-lg mx-2">Adicionar Produto</a>
    <?php else: ?>
        <a href="usuario-formulario.php" class="btn btn-primary btn-lg mx-2">Cadastrar-se</a>
        <a href="login.php" class="btn btn-secondary btn-lg mx-2">Login</a>
    <?php endif; ?>
</div> "Todos os campos são obrigatórios.";
}
```

Fonte: Do Autor.

Figura 6 - Interface da página inicial

Fonte: Do Autor.

2.1.5 Listagem de Produtos

Na versão desenvolvida em PHP puro, a listagem de produtos foi implementada na página "produto-lista.php", responsável por realizar consultas diretas ao banco de dados utilizando PDO.

Os registros retornados eram renderizados manualmente em uma tabela HTML com colunas para nome, categoria, preço e quantidade.

Cada linha continha botões de ação para edição e exclusão dos itens, que redirecionavam para os s*cripts* altera-produto.php e remove-produto.php, respectivamente. Essa abordagem, embora funcional, demandava maior esforço na manutenção e na atualização do código, pois cada operação era gerenciada por arquivos independentes, sem controle centralizado. Na versão em Laravel, a listagem de produtos foi implementada na view produtos/index.blade.php e é alimentada pelo ProdutoController, que utiliza o método Produto::all() do Eloquent ORM para recuperar os registros.

A utilização do Blade Template Engine simplificou significativamente a renderização da tabela, pois permitiu a inclusão dediretivas (@foreach e @if) para a exibição dinâmica dos dados de maneira organizada, sem a duplicação de código. O Laravel utilizou rotas do tipo resource, que automatizaram as ações de edição e exclusão de registros, reduzindo a necessidade de criação de múltiplos *scripts* e promovendo uma estrutura mais modular e escalável.

Essa diferença entre as abordagens pode ser observada nas Figuras 7 e 8. A figura 7 apresenta a interface final da listagem de produtos enquanto a Figura 8 compara os trechos de código utilizados em ambas as versões demonstrando a equivalência funcional e a superioridade estrutural do *framework* em termos de organização e manutenção.

Figura 7 - Interface da tela de listagem de produtos

Fonte: Do Autor.

Figura 8 - Comparativo entre o código da listagem de produtos em PHP e Laravel

Fonte: Do Autor.

2.1.6 Cadastro de Produtos

Na versão desenvolvida em PHP puro, o cadastro de novos produtos era feito por meio da página produto-formulario.php, que possuía um formulário HTML simples para inserção dos dados. As informações eram enviadas ao *script* adiciona-produto.php pelo método POST, que executava manualmente o comando SQL INSERT INTO produtos (...) VALUES (...) para registrar os itens no banco de dados. A ausência de um mecanismo de abstração exigia maior esforço na manipulação dos dados e cuidados adicionais na validação e na prevenção de erros durante a inserção.

No Laravel, o cadastro foi desenvolvido de maneira mais estruturada, utilizando o método store() do ProdutoController, vinculado à rota /produtos/create. O formulário, renderizado na view produtos/create.blade.php pela Blade Template Engine, empregou o método $request->validate() para a validação automática dos campos e o Eloquent ORM para a persistência dos dados por meio da função Produto::create(). Essa abordagem reduziu significativamente a quantidade de código, aumentou a segurança e melhorou a consistência da aplicação ao centralizar a lógica de cadastro e aplicar boas práticas de arquitetura MVC. As diferenças entre as duas implementações são apresentadas nas figuras 9 e 10. A figura 9 mostra o comparativo entre os códigos em PHP puro e Laravel. A figura 10 mostra as interfaces de cadastro de produtos desenvolvidas em ambas as versões, evidenciando a equivalência funcional e a melhoria de organização obtida com o uso do *framework*.

Figura 9 - Comparativo entre códigos na seção produtos

LARAVEL

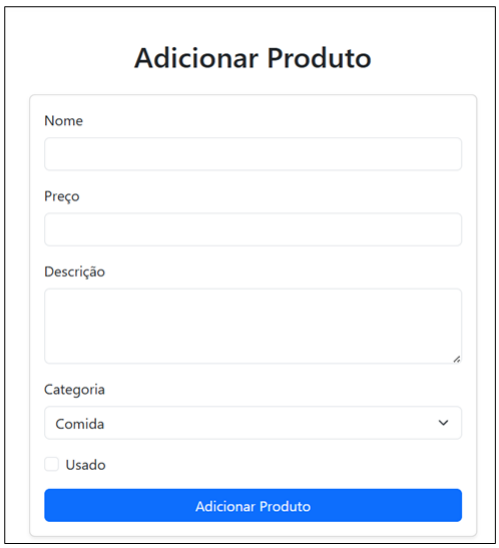
```
<form action="{{ route('produtos.store') }}" method="POST">
        @csrf
        <div class="mb-3">
            <label for="nome" class="form-label">Nome</label>
            <input type="text" name="nome" id="nome" class="form-control" required>
        </div>
        <div class="mb-3">
            <label for="preco" class="form-label">Preço</label>
            <input type="number" name="preco" id="preco" step="0.01" class="form-control" required>
        </div>
        <div class="mb-3">
            <label for="descricao" class="form-label">Descrição</label>
            <textarea name="descricao" id="descricao" class="form-control"></textarea>
        </div>
        <div class="mb-3">
            <label for="categoria_id" class="form-label">Categoria</label>
            <select name="categoria_id" id="categoria_id" class="form-select">
                <option value="">Selecione</option>
                @foreach($categorias as $categoria)
                    <option value="{{ $categoria->id }}">{{ $categoria->nome }}</option>
                @endforeach
            </select>
        </div>
        <div class="form-check mb-3">
            <input type="checkbox" name="usado" id="usado" value="1" class="form-check-input">
            <label for="usado" class="form-check-label">Usado</label>
        </div>
    <button type="submit" class="btn btn-success">Adicionar Produto</button>
</form>
```

PHP

```
$nome = $_POST['nome'] ?? '';
$preco = $_POST['preco'] ?? '';
$descricao = $_POST['descricao'] ?? '';
$categoria_id = $_POST['categoria_id'] ?? '';
$usado = isset($_POST['usado']) ? 1 : 0;
$inicio = microtime(true);
if (insereProduto($conexao, $nome, $preco, $descricao,
$categoria_id, $usado)) {
    $fim = microtime(true);
    $tempoExecucao = ($fim - $inicio) * 1000;
    $stmtTempo = $conexao->prepare("INSERT INTO atribuicoes_tempo
(operacao, tempo_execucao_ms) VALUES (?, ?)");
    $stmtTempo->bind_param("sd", $operacao, $tempoExecucao);
    $operacao = "Inserir Produto (PHP Puro)";
    $stmtTempo->execute();
    echo '<div class="alert alert-success text-center mt-5"
role="alert">';
    echo "Produto <strong>{$nome}</strong> adicionado com sucesso!
Preço: R$ " . number_format($preco, 2, ',', '.');
    echo "<br>Tempo de execução: " . number_format($tempoExecucao,
3) . " ms.";
    echo '</div>';
} else {
    echo '<div class="alert alert-danger text-center mt-5"
role="alert">';
    echo "Erro: Produto <strong>{$nome}</strong> não foi
adicionado.";
    echo '</div>';
}
```

Fonte: Do Autor.

Figura 10 - Interface da tela de cadastro de produtos



Fonte: Do Autor.

2.1.7 Edição de Produtos

Na aplicação em PHP puro, a edição de produtos foi implementada na página produto-altera-formulario.php, que é responsável por carregar os dados do item selecionado e exibi-los em um formulário para alteração. Após as modificações, as informações eram enviadas ao *script* altera-produto.php, que atualizava os registros no banco de dados por meio do comando SQL UPDATE. Todo o processo exigia o tratamento manual dos campos e a realização de verificações básicas de consistência, o que aumentava o risco de erros e de duplicidade de código.

Na versão em Laravel, a mesma funcionalidade foi implementada de maneira mais estruturada, por meio do método update() do ProdutoController, com rotas PUT e PATCH gerenciadas automaticamente pelo resource controller. O formulário de edição, localizado em produtos/edit.blade.php, herdou a estrutura do template principal e realizou as alterações com validação automática por meio de $request->validate() e persistência de dados por intermédio do Eloquent ORM.

Essa abordagem reduziu o número de linhas de código, centralizou a lógica de atualização e aumentou a confiabilidade do processo. A Figura 11 apresenta uma comparação entre as implementações em PHP e Laravel, enquanto a Figura 12 exibe a interface da tela de edição de produtos, demonstrando a padronização visual e a eficiência obtidas com o uso do *framework*.

Figura 11 - Comparação de códigos no editor de produtos

```
name="nome"
                        value="{{ old('nome', $produto->nome) }}" required>
                </div>

                <div class="mb-3">
                    <label for="preco" class="form-label">Preço</label>
                    <input type="number" class="form-control" id="preco"
name="preco" step="0.01"
                        value="{{ old('preco', $produto->preco) }}" required>
                </div>

                <div class="mb-3">
                    <label for="descricao" class="form-label">Descrição</label>
                    <textarea class="form-control" id="descricao"
name="descricao" rows="3">{{ old('descricao', $produto->descricao) }}</textarea>
                </div>

                <div class="mb-3">
                    <label for="categoria_id" class="form-
label">Categoria</label>
                    <select class="form-select" id="categoria_id"
name="categoria_id">
                        @foreach($categorias as $categoria)
                            <option value="{{ $categoria->id }}"
                                {{ (old('categoria_id', $produto->categoria_id)
== $categoria->id) ? 'selected' : '' }}>
                                {{ $categoria->nome }}
                            </option>
                        @endforeach
                    </select>
                </div>

                <div class="form-check mb-3">
                    <input class="form-check-input" type="checkbox" id="usado"
name="usado" value="1"
                        {{ old('usado', $produto->usado) ? 'checked' : '' }}>
                    <label class="form-check-label" for="usado">
                        Usado
                    </label>
```

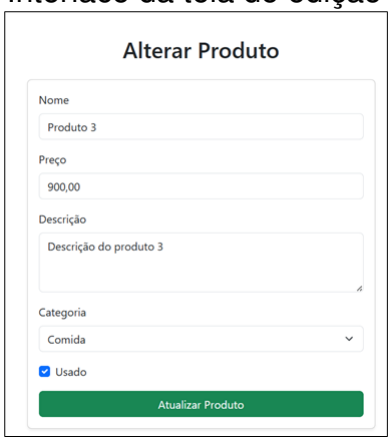
```
PHP

$id = $_POST['id'] ?? null;
$nome = $_POST['nome'] ?? '';
$preco = $_POST['preco'] ?? '';
$descricao = $_POST['descricao'] ?? '';
$categoria_id = $_POST['categoria_id'] ?? '';
$usado = isset($_POST['usado']) ? true : false;

if($id && alteraProduto($conexao, $id, $nome, $preco, $descricao,
$categoria_id, $usado)){
    header("Location: produto-lista.php?alterado=true");
    exit();
} else {
    echo '<div class="alert alert-danger text-center mt-5"
role="alert">';
    echo "Produto <strong>{$nome}</strong> não foi alterado.";
    echo '</div>';
    echo '<div class="text-center mt-3">';
    echo '<a href="produto-lista.php" class="btn btn-primary">Voltar
à lista</a>';
    echo '</div>';
}
```

Fonte: Do Autor.

Figura 12 - Interface da tela de edição de produtos



Fonte: Do Autor.

2.2 Criação do Banco de Dados

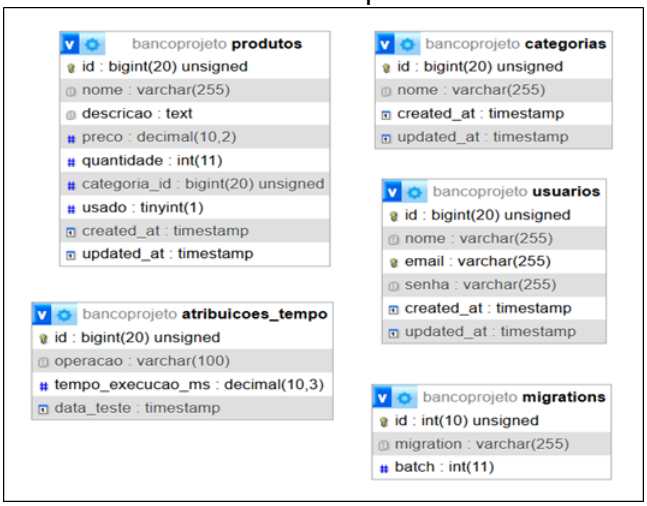
O banco de dados, denominado bancoprojeto, foi criado manualmente no phpMyAdmin utilizando o MySQL como sistema de gerenciamento. Sua estrutura, projetada com base no modelo relacional, contempla as principais tabelas e seus respectivos relacionamentos, de modo a atender às operações do sistema de controle de estoque. As tabelas incluem usuários, produtos, categorias e atribuições de tempo, todas interligadas por chaves primárias e estrangeiras, assegurando a integridade referencial dos dados.

Na versão desenvolvida em Laravel, essa estrutura foi gerada automaticamente por meio das migrações, que permitem definir campos, tipos de dados e relacio-

namentos diretamente no código. Esse processo garante rastreabilidade, padronização e reprodutibilidade do ambiente de dados, o que facilita a manutenção e a replicação do banco de dados em diferentes ambientes de desenvolvimento.

A Figura 13 apresenta o modelo relacional do banco de dados, evidenciando as relações entre as tabelas utilizadas em ambas as versões do sistema.

Figura 13 - Modelo relacional do banco de dados do sistema de controle de estoque



Fonte: Do Autor.

3 RESULTADOS E DISCUSSÃO

A análise comparativa entre as aplicações desenvolvidas em PHP puro e com o *framework* Laravel permitiu avaliar caracteristicas essenciais como estrutura, desempenho, velocidade de desenvolvimento, escalabilidade, segurança e compatibilidade com bancos de dados, Laravel aplica de forma consistente o padrão MVC, com uma organização modular das pastas *Controllers*, *Models*, *Middleware* e *Views*, além do uso de *migrations* que garantem o versionamento e a rastreabilidade da base de dados, o uso do Eloquent ORM e do Query Builder elimina a necessidade de escrever comandos SQL diretamente nos controladores, enquanto as validações são centralizadas por meio das classes FormRequest, assegurando a integridade e a padronização do código. Em contraste, o PHP puro apresentou uma estrutura linear e fragmentada, sem divisão em camadas, com funcionalidades isoladas em ficheiros independentes e com SQL presente diretamente nos *scripts*. A validação ocorre de forma manual e descentralizada, o que aumenta a redundância e o risco de erros.

De acordo com a Tabela 2, o Laravel obteve um desempenho estrutural superior em todos os critérios analisados padrão MVC, migrações, ausência de SQL nos controladores e validação centralizada ao passo que o PHP puro não apresentou nenhuma dessas funcionalidades automatizadas, estes resultados corroboram as afirmações de (Vicente; Sirqueira, 2020) e de(Almeida; outros, 2025) sobre a importância dos *frameworks* para a modularização e manutenção de sistemas.

Tabela 2 - Valores médios dos componentes

CARACTERÍSTICA	PHP PURO	LARAVEL
MVC (Model-View-Control)	não possui	possui
Migrations	0	5
SQL em Controller	possui	não possui
Validação Centralizada	não possui	possui

Fonte: do autor.

O teste de desempenho avaliou o tempo de inserção de 10 mil registros na tabela “Produtos”. O PHP puro apresentou um tempo de execução ligeiramente inferior, em razão da execução direta de comandos SQL por meio da biblioteca PDO. Por outro lado, o Laravel, ao utilizar o Eloquent ORM e mecanismos de verificação de integridade, obteve um tempo marginalmente superior, porém com maior estabilidade e controle de exceções. Conforme ilustrado na Figura 15, o PHP puro apresentou resposta mais rápida, enquanto o Laravel demonstrou maior robustez e rastreabilidade das operações. Esses resultados estão em conformidade com os achados de (Saroni; Mulyanti, 2020) e de (Laaziri; outros, 2019), que indicam que *frameworks* modernos tendem a sacrificar uma pequena fração do desempenho em troca de maior consistência e segurança. Assim, embora o PHP puro execute tarefas simples mais rapidamente, o Laravel oferece um equilíbrio mais eficiente entre desempenho, segurança e confiabilidade, características fundamentais para sistemas de médio e grande porte.

Figura 14 - Tabela atribuições tempo

	id	operacao	tempo_execucao_ms	data_teste
Editar Copiar Remover	1	Inserção de 10.000 produtos - PHP Puro	16941.472	2025-10-13 22:29:41
Editar Copiar Remover	2	Inserção de 10.000 produtos	19829.112	2025-10-13 22:32:29
Editar Copiar Remover	3	Inserção de 10.000 produtos	17207.344	2025-10-14 22:33:30
Editar Copiar Remover	4	Inserção de 10.000 produtos - PHP Puro	16484.425	2025-10-14 22:36:39

Fonte: Do Autor.

A análise da velocidade de desenvolvimento demonstrou que o Laravel requer menor esforço manual graças à automação de tarefas oferecida pelo Artisan CLI, que permite gerar controladores, modelos, migrações e rotas de forma automática. Em contrapartida, o PHP puro depende da criação manual de cada arquivo e da duplicação de código em diferentes páginas. Conforme apresentado na Tabela 3, o Laravel necessitou de menos arquivos para implementar o mesmo CRUD de produtos, além de centralizar as funções em controladores e modelos reutilizáveis.

Esses resultados confirmam as observações de (Laaziri; outros, 2019) e (Ferreira, 2021), que destacam a produtividade e a padronização proporcionadas pelos *frameworks* modernos. Assim, o Laravel mostra-se mais eficiente e produtivo, reduzindo o tempo de desenvolvimento e a probabilidade de erros, enquanto o PHP puro, embora mais simples, demanda maior esforço manual e apresenta menor escalabilidade.

Tabela 3 - Comparação de velocidade em Desenvolvimento

Tipo de Arquivo	PHP Puro	Laravel	Descrição
Model / Classe Produto	Não possui.	1 arquivo: `Produto.php`.	No Laravel o modelo é separado e reutilizável; no PHP o código de banco é misturado aos scripts.
Controller	Não possui controlador dedicado.	1arquivo: `/ProdutoController`	O Laravel centraliza a lógica de negócio no controlador; no PHP cada página executa ações isoladas.
Views (criar, editar, listar)	5 arquivos (`adiciona-produto`, etc...).	1 arquivo: `create.blade.php`.	O Laravel concentra a exibição em templates reutilizáveis; o PHP requer páginas separadas para cada ação.
Rotas	1 rota manual em cada arquivo.	1 arquivo com `Route::resource ('produtos', ...)`.	No Laravel a rota `resource` automatiza todas as ações CRUD.
Migration	Não possui.	5 migrations automáticas.	O Laravel gera e versiona o banco automaticamente; no PHP isso é feito manualmente pelo phpMyAdmin.
Request / Form	Não possui validação dedicada.	Validação simples no controller (sem classe `Request`).	Em versões mais completas do Laravel pode-se usar uma classe `Request` dedicada para validação.
Total de Arquivos Tocados	7 arquivos.	5 arquivos.	O Laravel exige menos arquivos e menos esforço manual, refletindo maior produtividade.

Fonte: Do Autor.

O Laravel monstrou maior prontidão para ambientes distribuídos e escaláveis, com suporte nativo para cache compartilhado, filas de tarefas assíncronas (Jobs/Listeners) e variáveis de ambiente (.env).

Essas funcionalidades permitem a execução de aplicações em múltiplos servidores, mantendo as sessões e o cache de forma centralizada. Já o PHP puro limita-se à gestão local de sessões, sem mecanismos de cache distribuído ou controle de variáveis de ambiente, conforme apresentado na Tabela 4. Tais limitações restringem sua utilização a aplicações de pequeno porte ou destinadas à execução local.

Esses resultados estão em conformidade com os estudos de (Paula; Oliveira, 2024) e (Costa, 2025), que destacam que os *frameworks* modernos oferecem suporte nativo à escalabilidade e à implementação em infraestruturas distribuídas. Assim, o Laravel mostra-se mais adequado para contextos empresariais e aplicações com alta demanda de processamento e acesso simultâneo.

Tabela 4 - Comparativo de Escalabilidade entre PHP Puro e Laravel

Critério Avaliado	Laravel	PHP Puro	Conclusão
Sessões	Armazenadas no banco de dados (driver `database`), permitindo compartilhamento entre servidores.	Armazenadas localmente com `session_start()` padrão.	O Laravel é mais escalável e preparado para ambientes distribuídos.
Cache	Configurado para usar banco de dados (`database driver`), possibilitando cache centralizado.	Não possui cache centralizado (sem uso de Redis, APCu ou Memcache).	O Laravel é mais eficiente na gestão e persistência de cache.
Filas / Jobs	Possui filas configuradas via `queue.php` (driver database), com suporte a tarefas assíncronas.	Não possui filas nem automações (sem cron jobs, workers ou scripts de background).	O Laravel oferece melhor suporte para processamento paralelo e alta demanda.
Variáveis de Ambiente (.env)	Utiliza arquivo `.env` para armazenar credenciais e configurações externas.	Credenciais fixas no arquivo `conecta.php` (sem `.env`).	O Laravel é mais seguro e flexível para implantação em diferentes ambientes.
Prontidão para Escalar	Estrutura pronta para load balancing, cache distribuído e execução paralela de tarefas.	Estrutura simples, adequada apenas para execução em um único servidor.	O Laravel apresenta melhor desempenho em sistemas corporativos e distribuídos.

Fonte: Do Autor.

Em termos de segurança, o Laravel apresentou clara superioridade. O *framework* oferece proteção automática contra ataques CSRF, autenticação por meio de *middleware* e consultas protegidas contra injeção de SQL através do Eloquent ORM. Embora a limitação de taxas não tenha sido implementada no projeto, o Laravel disponibiliza essa funcionalidade de forma nativa. Já o PHP puro depende de implementações manuais, como o uso de mysqli_prepare e variáveis de sessão, sem o suporte a *tokens* de segurança ou controle avançado de permissões. Conforme demonstrado na Tabela 5, o Laravel obteve desempenho superior em três dos quatro critérios avaliados CSRF, autenticação e injeção de SQL, o que reforça as observações de (Souza, 2021) e (Nunes, 2025) sobre a importância dos *frameworks* na mitigação de vulnerabilidades. Assim, conclui-se que o Laravel proporciona uma abordagem mais segura, integrada e menos suscetível a erros humanos.

Tabela 5 - Comparativo de Segurança entre PHP Puro e Laravel

CRITÉRIO AVALIADO	LARAVEL	PHP PURO	CONCLUSÃO
Proteção CSRF	Proteção automática com diretiva `@csrf` em formulários e middleware de verificação.	Não possui tokens CSRF nos formulários.	O Laravel protege contra requisições maliciosas; PHP puro é vulnerável a ataques CSRF.
Autenticação e Autorização	Autenticação via `auth middleware`; não utiliza `policies` para permissões avançadas.	Verificação manual de sessão sem níveis de permissão.	O Laravel fornece autenticação padronizada; PHP puro depende de controle manual.
Injeção SQL	Totalmente protegido via Eloquent ORM e `query bindings`.	Protegido com `mysqli_prepare` e `bind_param`.	Ambos seguros quanto à injeção de SQL, porém Laravel oferece proteção automatizada.
Rate Limiting	Não implementado, mas disponível via middleware `throttle`.	Não implementado.	Nenhum sistema possui limitação de tentativas; porém Laravel possui suporte nativo à configuração.

Fonte: Do Autor.

O Laravel é compatível com diversos sistemas de gerenciamento de bancos

de dados (SGBD) e oferece suporte nativo para múltiplas conexões, transações automáticas e controle de versões por meio de *migrations*. Em contrapartida, o PHP puro depende de configurações manuais e não dispõe de recursos de versionamento ou recuperação automatizada. Conforme apresentado na Tabela 6, o Laravel proporciona rastreabilidade completa das alterações realizadas na base de dados, permitindo o retorno a versões anteriores e a criação de diferentes estados estruturais, o que o torna mais adequado a ambientes colaborativos e de evolução contínua. Esses resultados corroboram as observações de (Vicente; Sirqueira, 2020), que destacam a importância dos *frameworks* PHP modernos para assegurar a integridade e a manutenção das bases de dados.

Tabela 6 - Comparativo de Banco de Dados entre PHP Puro e Laravel

CRITÉRIO AVALIADO	LARAVEL	PHP PURO	CONCLUSÃO
Migrações	Possui 5 migrations (4 executadas), com versionamento via *artisan migrate* e controle de histórico.	Não possui migrações; o banco é configurado manualmente em SQL.	Laravel controla a estrutura do banco e permite rollback automatizado.
Transações	Suporte nativo via `DB::transaction()` mesmo que não utilizado no projeto.	Não utiliza `beginTransaction`, `commit` ou `rollback`.	Laravel possui suporte pronto; PHP depende do controle manual do desenvolvedor.
Múltiplas conexões (multi-DB)	Configurável no projeto, permitindo múltiplos bancos (MySQL, PostgreSQL, SQLite, etc.).	Apenas uma conexão definida em `conecta.php`.	Laravel é mais flexível para aplicações com múltiplos SGBDs.
Versionamento automático	As migrations controlam a evolução e a reversão do banco, garantindo histórico de alterações.	Não há versionamento; estrutura criada e alterada manualmente.	Laravel oferece rastreabilidade total das modificações no banco.
Consistência transacional	Suporte nativo a transações, mesmo que não implementado neste projeto.	Não há rollback automático, o que gera risco de inconsistência em falhas.	Laravel garante consistência e recuperação em caso de erros.
Suporte multi-banco	Sim, facilmente configurável pelo arquivo `.env` (ex.: `mysql2`, `pgsql`).	Não possui suporte; seria necessário configurar múltiplas variáveis manuais.	Laravel é mais escalável e compatível com diferentes SGBDs.

Fonte: Do Autor, 2025.

Com base nas análises realizadas, o Laravel apresentou desempenho superior em praticamente todos os critérios avaliados estrutura, produtividade, escalabilidade, segurança e compatibilidade com bancos de dados, onde o PHP puro demonstrou melhor desempenho bruto em tarefas simples, porém careceu de recursos automáticos, segurança integrada e padronização estrutural, estes resultados estão em conformidade com as conclusões de (Vicente; Sirqueira, 2020), (Almeida; outros, 2025), (Saroni; Mulyanti, 2020) e (Paula; Oliveira, 2024), que apontam o Laravel como uma evolução significativa no ecossistema PHP, por oferecer maior eficiência, segurança e confiabilidade no desenvolvimento de sistemas web modernos.

4 CONCLUSÃO

O presente estudo teve como objetivo analisar comparativamente as tecnologias PHP e o *framework* Laravel, considerando aspectos como estrutura, desempenho, escalabilidade, segurança e compatibilidade com bancos de dados. Para alcançar esse propósito, foram desenvolvidas duas aplicações equivalentes uma em PHP puro e outra com o *framework* Laravel, além da realização de uma revisão bibliográfica sobre o uso

e a evolução dessas tecnologias no contexto do desenvolvimento web moderno. Os resultados das análises demonstraram que o Laravel apresentou desempenho superior em praticamente todos os critérios avaliados, especialmente no que se refere à organização estrutural, à produtividade, à segurança e à escalabilidade. O *framework* aplicou de forma consistente o padrão MVC (*Model-View-Controller*), com uma organização modular e o uso de recursos como Eloquent ORM, Query Builder, *migrations* e Artisan CLI, que proporcionaram maior padronização, rastreabilidade e agilidade no desenvolvimento. Esses elementos reduziram o esforço manual, minimizaram erros e tornaram o processo de manutenção mais eficiente.

O PHP puro demonstrou melhor desempenho em tarefas simples e isoladas, devido à execução direta de comandos SQL e à ausência de camadas intermediárias. No entanto, careceu de funcionalidades automáticas, validações integradas e mecanismos de segurança nativos, o que limita sua aplicação em sistemas mais complexos e ambientes distribuídos. O Laravel destacou-se ainda por sua capacidade de integração com múltiplos bancos de dados, suporte a cache compartilhado, filas de tarefas assíncronas e variáveis de ambiente, recursos que o tornam mais adequado a projetos de médio e grande porte. Além disso, o *framework* apresentou vantagens significativas em segurança, oferecendo proteção nativa contra ataques CSRF, injeções de SQL e falhas de autenticação.

Os resultados obtidos estão em conformidade com os estudos de Saroni e Mulyanti (2020), Vicente e Sirqueira (2020), Paula e Oliveira (2024), Almeida et al. (2025) e Laaziri et al. (2019), os quais confirmam que *frameworks* modernos, como o Laravel, representam uma evolução significativa no ecossistema PHP, ao oferecer maior eficiência, segurança e confiabilidade no desenvolvimento de sistemas web. Desta forma, conclui-se que o Laravel se mostra a opção mais eficiente e robusta para o desenvolvimento de aplicações web que exigem escalabilidade, segurança e manutenção facilitada, enquanto o PHP puro continua sendo uma alternativa viável para projetos menores e de implementação simples.

REFERÊNCIAS

ABUTALEB, H.; TAMIMI, A.; ALRAWASHDEH, T. **Empirical Study of Most Popular PHP Framework**. 2021. Acesso em: 10 jun. 2023. Disponível em: <https://doi.org/10.1109/ICIT52682.2021.9491694>.

ADAM, S. I.; ANDOLO, S. **A new PHP web application development framework based on MVC architectural pattern and ajax technology**. 2019. Acesso em: 26 maio 2025. Disponível em: <https://doi.org/10.1109/ICORIS.2019.8874871>.

ALMEIDA, M. C.; OUTROS. **Sistema de gestão e inteligência de negócios para organizações sociais: LOGDEV**. São João da Boa Vista: UNIFEOB, 2025. Acesso em: 26 maio 2025. Disponível em: <http://ibict.unifeob.edu.br:8080/jspui/bitstream/prefix/7047/1/PI.COMP-NUVEM.B.G8.pdf>.

CAMILO, T. S. **Desenvolvimento de sistema web para gerenciamento de salas de integração e recursos**. 2019.

COSTA, B. A. **Além da cobertura de código: avaliando a qualidade de testes em componentes Symfony**. Dissertação (Mestrado) — Universidade Federal do Ceará, Fortaleza, 2025. Acesso em: 26 maio 2025. Disponível em: <https://repositorio.ufc.br/handle/riufc/80115>.

FERREIRA, B. S. **Framework Laravel: um estudo de caso full stack development**. 2021. Trabalho de Conclusão de Curso (Graduação em Sistemas de Informação) – Universidade Federal do Ceará, Fortaleza.

KHAN, S.; KHANAM, A. T. **Study on mvc framework for web development in php**. International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 2023, p. 414–419.

LAAZIRI, M.; OUTROS. **A comparative study of php frameworks performance**. Procedia Manufacturing, 2019, v. 32, p. 864–871.

NUNES, G. M. **Uma solução de monitoramento no fornecimento de energia solar residencial**. Dissertação (Mestrado) — Universidade Federal do Ceará, Fortaleza, 2025. Acesso em: 26 maio 2025. Disponível em: <https://repositorio.ufc.br/handle/riufc/80962>.

OLIVEIRA, D. **PHP em ambientes corporativos: escalabilidade e integração**. Brasília: EdUNB, 2023.

PAULA, M.; OLIVEIRA, S. **Integração e segurança em aplicações modernas com frameworks php**. Revista Brasileira de Sistemas Web, 2024, Belo Horizonte, v. 10, n. 1, p. 22–39.

SARONI, M. I. N.; MULYANTI, B. **Hypertext preprocessor framework in the development of web applications**. 2020. Acesso em: 10 jun. 2023. Disponível em: <https://doi.org/10.1088/1757-899X/821/1/012096>.

SOUZA, M. A. **Banco de dados e linguagens dinâmicas: integração com php**. Revista Brasileira de Tecnologia da Informação, 2021, v. 9, n. 1, p. 51–67.

SUNARDI, A. et al. **Mvc architecture: A comparative study between laravel framework and slim framework in freelancer project monitoring system web based**. Procedia Computer Science, 2019, Elsevier, v. 157, p. 134–141.

VICENTE, J.; SIRQUEIRA, T. **Uma ferramenta de análise estática de código fonte para aplicações web php**. 2020, v. 7.