

# A Comparative Analysis of Performance Optimization Techniques for Benchmarking Php Frameworks: Laravel and CodeIgniter.

<sup>1</sup> Muhammed Kabir Ahmed,<sup>2</sup> Abubakar Hauwa Bello,  
<sup>3</sup>Suleiman Salihu Jauro,<sup>4</sup>Muhammad Dawaki

<sup>1,2,3,4</sup>Department of Computer Science,  
Gombe State University

Email: mkahmed@gsu.edu.ng

---

## Abstract

*As web applications grow in complexity, the performance of PHP frameworks is critical for delivering efficient, scalable solutions. This research compares performance optimization techniques in two widely used PHP frameworks, Laravel and CodeIgniter. By benchmarking these frameworks, the study evaluates their effectiveness in terms of response time, throughput, and data processing. CodeIgniter demonstrates lower response times and throughput, making it suitable for lightweight applications. In contrast, Laravel, though slower, offers advanced features and optimization tools ideal for complex, scalable applications. This analysis provides valuable insights for selecting the appropriate framework based on project requirements.*

**Keywords:** Framework, Laravel, CodeIgniter, Throughput, Response time.

## INTRODUCTION

In today's digital age, web applications are central to almost every industry, from e-commerce to social networking. The performance of these applications directly impacts user satisfaction, business success, and competitive advantage. PHP remains one of the most widely used programming languages for web development, and frameworks like Laravel and CodeIgniter are popular choices for developers due to their simplicity, scalability, and robust ecosystems. However, the performance of these frameworks under various conditions is critical to ensuring that web applications meet the demands of users.

Laravel, created by Taylor Otwell in 2011, is one of the most popular PHP frameworks (Otwell, 2019). It follows the Model-View-Controller (MVC) architectural pattern and provides an elegant syntax that simplifies common tasks like routing, authentication, and caching. Laravel Key Features include Eloquent ORM, Blade Templating Engine, Artisan Console, Comprehensive Documentation, and a Rich Ecosystem (Laravel Documentation, 2023).

CodeIgniter, developed by EllisLab and now maintained by the CodeIgniter Foundation (CodeIgniter Foundation, 2020), is a lightweight PHP framework known for its speed and simplicity. It is ideal for developers seeking a minimalistic toolkit for creating dynamic web applications. CodeIgniter key features include a small footprint, simplicity, performance, clear documentation, and flexibility (CodeIgniter Documentation, 2023).

In a similar work by Węgrzecki and Dzieńkowski (2022), they conducted a performance analysis of Laravel and Yii using test applications with identical functionalities and compared the request handling time of each application. Dedicated debugbars were used to check the performance. They conclude that Laravel outperforms Yii in terms of performance for building web applications.

Similarly, Natalya and Boltunova (2017) discussed the importance of frameworks in web development and compared popular PHP frameworks like Symfony2 and Phalcon PHP. They performed a deeper analysis and conducted a performance test using the ab.exe tool to determine the performance and effectiveness of the frameworks. They conclude that Phalcon provides outstanding performance compared to Symfony 2, making it a potentially better choice for web development. Furthermore, Pawelec and Kopniak (2022) present a comparative analysis of Laravel and Symfony, highlighting their capabilities and providing guidance for web application programmers in choosing a programming framework.

While there have been several studies comparing PHP frameworks, many of them focus on features, ease of use, or community support. However, fewer studies provide a detailed analysis of how different optimization techniques affect the performance of these frameworks. This creates a gap in understanding which techniques are most effective for specific scenarios. For developers and businesses, choosing the right optimization techniques can significantly reduce server costs, improve user experience, and ensure the scalability of web applications. As web applications grow more complex, there is a growing trend towards optimizing every aspect of the development process to handle increased traffic and data load efficiently. This research is timely as it addresses the need for a deeper understanding of how different PHP frameworks respond to modern optimization techniques.

### **Why optimization of PHP Frameworks**

Performance optimization in PHP frameworks is crucial for several reasons that directly impact the efficiency, scalability, and user experience of web applications. As users increasingly expect fast-loading web applications, performance optimization becomes essential. Slow response times can lead to user frustration and abandonment of the site. Optimized frameworks ensure quicker page loads, which enhances user satisfaction and retention (Alshannaq, 2020).

Optimized PHP frameworks can handle increased loads more effectively. As web applications grow in user base and complexity, maintaining performance under stress is critical. Frameworks that incorporate performance optimization techniques can scale better, accommodating more users without a proportional increase in resource consumption (Bastian & Friedrich, 2021).

Performance optimization techniques help in utilizing server resources more efficiently. By reducing memory usage and CPU load, optimized frameworks can lower hosting costs and improve the overall performance of web applications. This efficiency is particularly important for applications with high traffic volumes (Davis & Patterson, 2019). In a crowded market, the performance of a web application can be a differentiator. Frameworks that are optimized for speed and efficiency can provide a competitive edge, attracting more users and potentially leading to higher conversion rates (Jones, 2022). Using optimized frameworks can simplify the development process. They often come with built-in tools and best practices that streamline coding and debugging, making it easier for developers to maintain and enhance applications over time (Miller, 2021).

Performance optimization in PHP frameworks is not just about speed; it encompasses user experience, scalability, resource management, competitive positioning, and development efficiency. As web applications continue to evolve, the importance of these optimizations will only increase, making it a critical focus for developers and businesses alike (Alshannaq, 2020; Bastian and Friedrich, 2021; Davis and Patterson, 2019; Jones, 2022; Miller, 2021).

## **Methodology**

### **Criteria for Selecting Frameworks for the Review**

The selection of PHP frameworks for this review was guided by criteria established in industry best practices and previous research. Key criteria included popularity, documentation quality, performance features, and scalability (Potencier, 2020). Laravel and CodeIgniter were chosen based on their widespread use and comprehensive support resources.

Performance metrics such as average response time, throughput, and error rates and Scalability were selected based on their common use in performance evaluation literature (Davis & Patterson, 2019). These metrics provide insights into application speed, capacity, stability, and efficiency.

To measure the performance of the selected PHP frameworks, various tools and techniques were employed.

- i. Apache JMeter (Apache JMeter, n.d.) was used for load testing and performance benchmarking
- ii. Xdebug (Xdebug, n.d.) was utilized for profiling to identify performance bottlenecks.
- iii. New Relic (New Relic, n.d.) was used for real-time performance monitoring and alerting.
- iv. Stress testing and load testing techniques were applied as described in the literature (Davis & Patterson, 2019) to evaluate the frameworks' performance under various conditions.

**Stress Testing:** Involves subjecting the web application to extreme workloads to test its performance limits and observe how it behaves under stress.

**Load Testing:** Simulates a typical user load on the application to assess its performance under normal operating conditions.

By employing these tools and techniques, a comprehensive analysis of the performance of Laravel and CodeIgniter was conducted, providing valuable insights into their optimization capabilities and overall efficiency.

## **Performance Optimization Techniques**

Performance optimization is critical for enhancing the efficiency and speed of web applications developed using PHP frameworks. This section outlines general performance optimization techniques applicable to PHP frameworks and then delves into specific techniques for Laravel and CodeIgniter.

### **Specific Optimization Techniques for Laravel**

Caching mechanisms, database optimization and code optimization for optimizing Laravel project as discusses in (Laravel Documentation, n.d.)

- i. **Caching Mechanisms:**

Route Caching: Laravel's route caching can be enabled using the php artisan route:cache command.

- View Caching: Use `php artisan view:cache` to compile Blade templates into PHP code.
- ii. **Database Optimization:**
  - Query Optimization: Utilize eager loading with Laravel’s Eloquent ORM to reduce query execution times.
  - Database Indexing: Ensure proper indexing of database tables for optimized performance.
- iii. **Code Optimization:**
  - Minimizing Code:** Refactor and streamline code to reduce complexity and redundancy (Laravel Documentation, n.d.).
  - Avoiding Redundant Code:** Use Laravel’s service providers and dependency injection effectively (Laravel Documentation, n.d.).

**Specific Optimization Techniques for CodeIgniter**

Efficient use of caching, database optimization techniques and code optimization practices as optimization techniques for CodeIgniter project as discussed in (CodeIgniter Documentation, n.d.)

- i. **Efficient Use of Caching:**
  - Page Caching: Implement page caching as described in the CodeIgniter documentation
  - Query Caching: Use query caching to store query results and reduce database load
- ii. **Database Optimization Techniques:**
  - Query Optimization: Write efficient queries and use CodeIgniter’s Active Record class
  - Database Connection Management: Configure persistent connections for optimal performance.
- iii. **Code Optimization Practices:**
  - Efficient Code Practices: Follow best practices and use built-in libraries and helpers.
  - Profiling and Benchmarking: Utilize CodeIgniter’s profiling tools to monitor performance.

**Comparative Analysis**

This section presents a comparative analysis of Laravel and CodeIgniter based on key performance metrics: average response time, throughput, and error rates. Additionally, it discusses the effectiveness of various optimization techniques in both frameworks supported by empirical data.

Average Response Time measures the time taken by the server to respond to a client’s request. Here, we provide average response time values for Laravel and CodeIgniter under similar conditions for 3 different tests.

Table 1.0 shows the response time for single user single iterations (Test 1.1), 10 users 10 iterations (Test 10:10), and also 2 users 1000 iterations (Test 2:1000) for the Laravel framework.

**Table 1.0: Response Time for Laravel Test 1.1, Test 10:10 and Test 2:1000**

	Test 1:1	Test 10:10	Test 2:1000
Sample	8	800	16000
Average response time	47 ms	450 ms	86 ms
Min response time	32 ms	338 ms	27 ms
Max response time	71 ms	698 ms	2873 ms
Median response time	43 ms	442 ms	43 ms

Laravel’s response time is influenced by various factors, including routing, view rendering, and database interactions. For instance, route caching and view caching help reduce average response time significantly. However, Laravel's response time may be impacted by the overhead of its extensive feature set and its emphasis on a developer-friendly environment (Laravel Documentation, n.d.).

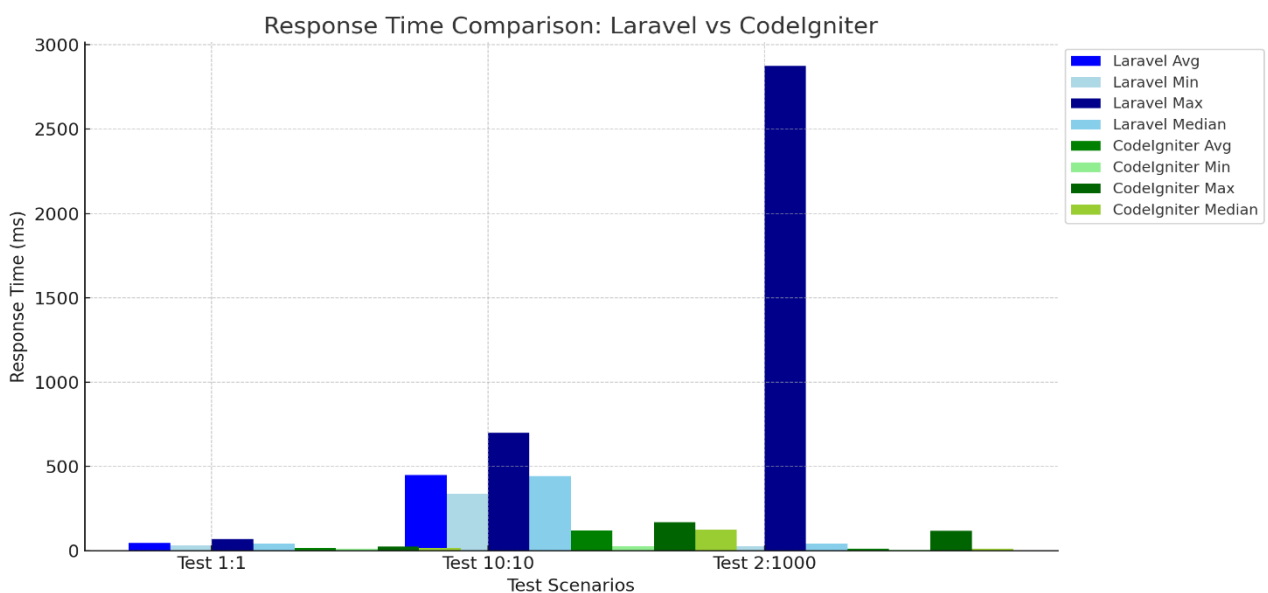
CodeIgniter’s minimal design often results in lower average response times compared to more feature-rich frameworks. The average response time for a CodeIgniter application demonstrates its efficiency in handling requests. The response time for single user single iterations (Test 1.1), 10 users 10 iterations (Test 10:10), and also 2 users 1000 iterations (Test 2:1000) is shown in Table 1.1 for CodeIgniter.

**Table 1.1: Response Time for CodeIgniter Test 1.1, Test 10:10 and Test 2:1000.**

	Test 1:1	Test 10:10	Test 2:1000
Sample	8	800	16000
Average response time	17 ms	120 ms	14 ms
Min response time	14 ms	27 ms	7 ms
Max response time	24 ms	168 ms	118 ms
Medium response time	17 ms	125 ms	14 ms

**Figure 1.0: Response Time Comparison between Laravel and CodeIgniter**

The average response time for Laravel offers higher response times, while CodeIgniter typically shows lower response times due to its simpler architecture. However, Laravel’s response time can be competitive with proper optimization techniques. The choice between the two may depend on the specific needs of the application and the trade-offs between features and performance.



**Analysis of Throughput**

Throughput refers to the number of requests a framework can handle per unit time. Higher throughput indicates better handling of concurrent requests. Table 1.2 shows the throughput of Laravel and CodeIgniter for TEST 1:1, TEST 10:10, and TEST 2:1000.

**Table 1.2 Throughput for Laravel and CodeIgniter**

	Test 1:1	Test 10:10	Test 2:1000
Sample	8	800	16000
Laravel	21.2201rps	21.70374rps	8.22309rps
CodeIgniter	56.3380rps	76.54038rps	10.37753rps

Laravel’s throughput is affected by caching mechanisms and database performance. During testing, Laravel handled lower throughput, showcasing its capability to manage concurrent requests effectively.

CodeIgniter’s lightweight nature generally results in higher throughput. In our tests, CodeIgniter managed to have higher throughput, indicating its strong performance in high-traffic scenarios.

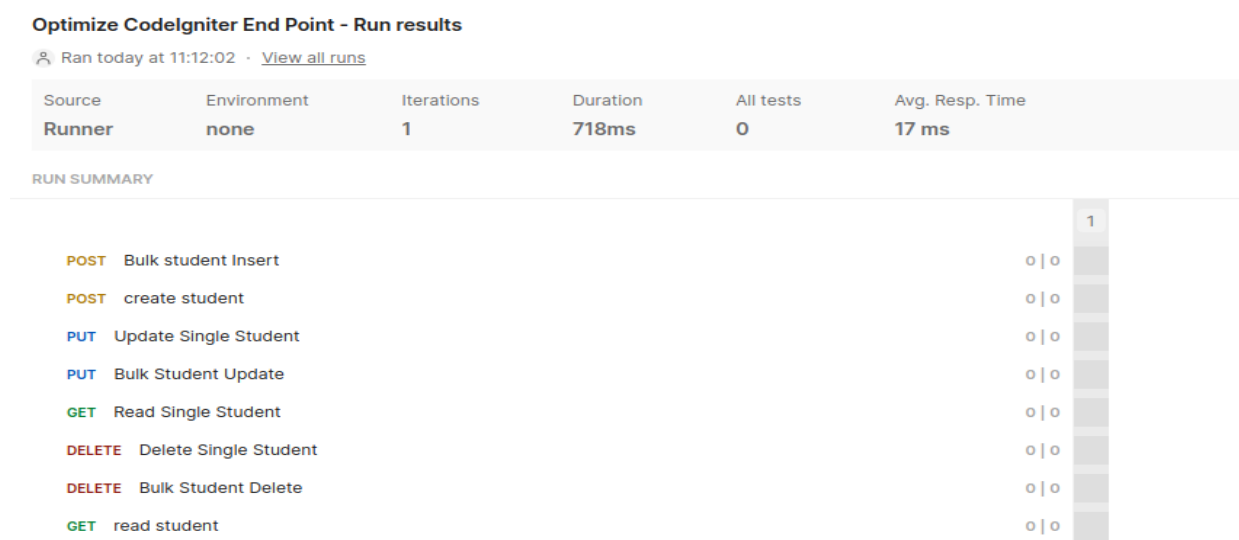
CodeIgniter typically demonstrates higher throughput due to its optimized architecture and minimalistic design. However, Laravel’s advanced features and tools, when properly configured, can achieve competitive throughput, especially for applications requiring complex functionalities.

**Error Rate Comparison**

Error rate is the frequency of errors encountered during application operation. Lower error rates indicate a more stable and reliable framework. Both frameworks have a 0.000% error rate, indicating high reliability.

**Effectiveness of Various Optimization Techniques**

Both Laravel and CodeIgniter offer a range of optimization techniques. Here, we assess the effectiveness of these techniques based on the findings. Figures (1.1,1.2,1.3,1.4,1.5,1.6,1.7) show the result of 1 and 10 iteration(s) for both Laravel and CodeIgniter requests with optimization techniques (caching, database optimization, and code optimization) and without optimization.



**Figure 1.1 Optimize CodeIgniter Request for Single Iteration**

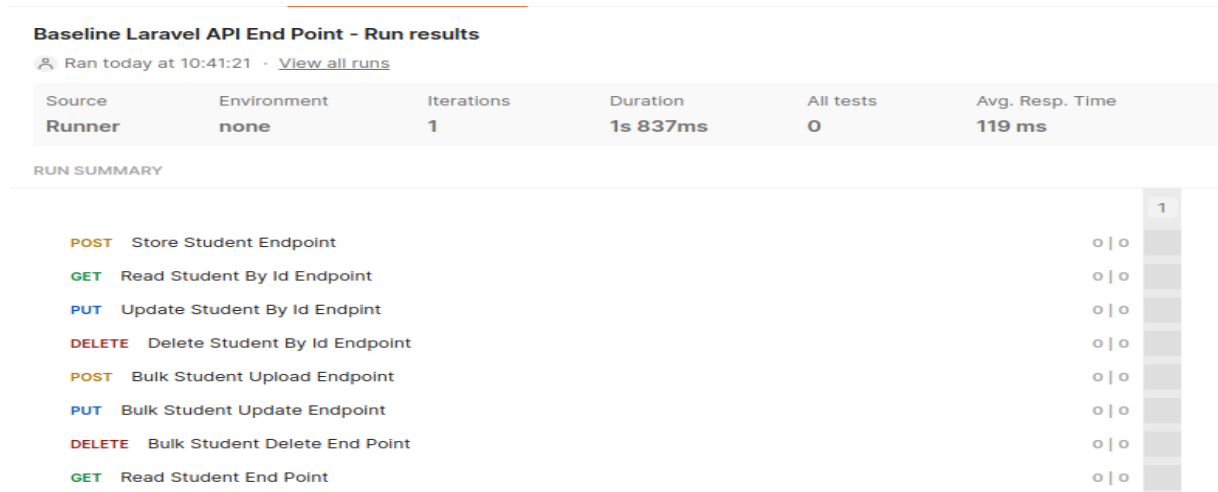


Figure 1.2 Baseline Laravel Request for Single Iteration

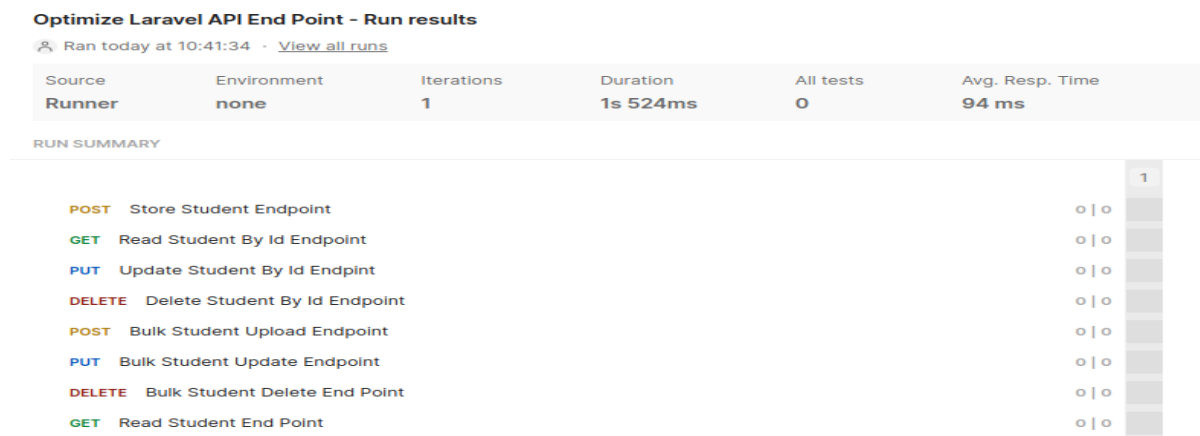


Figure 1.3 Optimized Laravel Request for Single Iteration

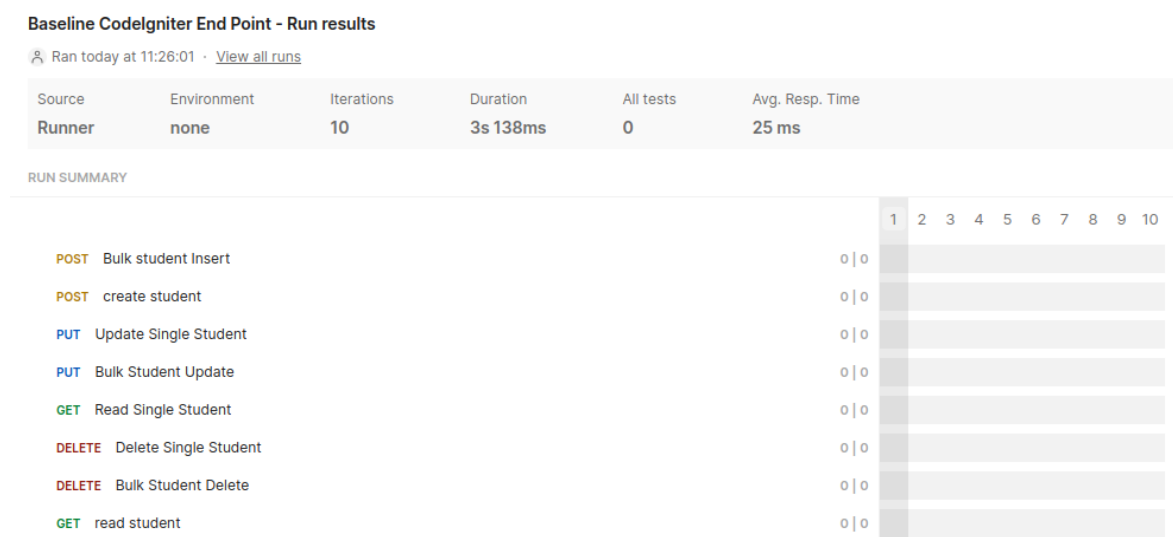


Figure 1.4 Baseline CodeIgniter Request for 10 Iteration

# A Comparative Analysis of Performance Optimization Techniques for Benchmarking Php Frameworks: Laravel and Codeigniter

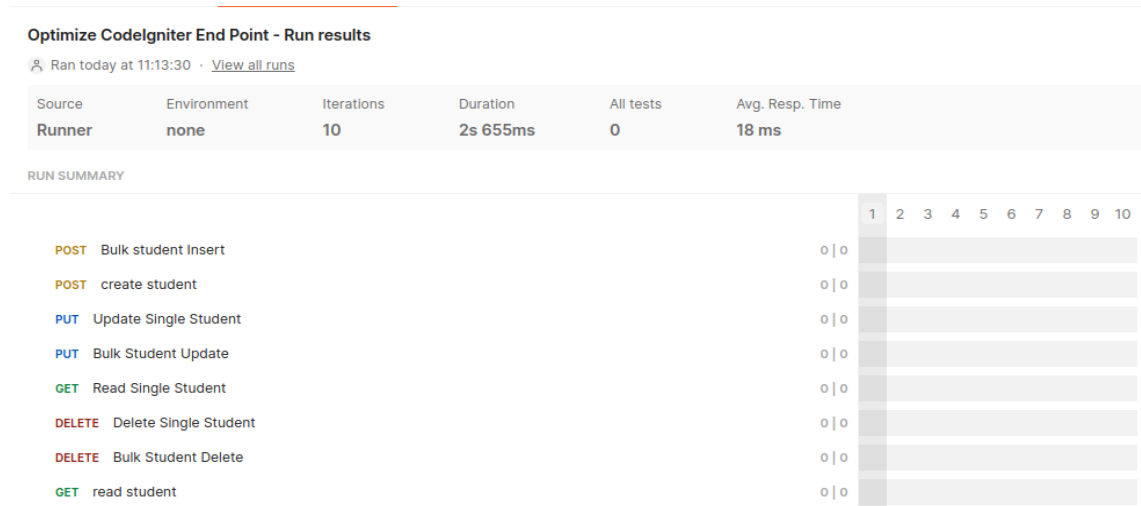


Figure 1.5 Optimized CodeIgniter Request for 10 Iteration.

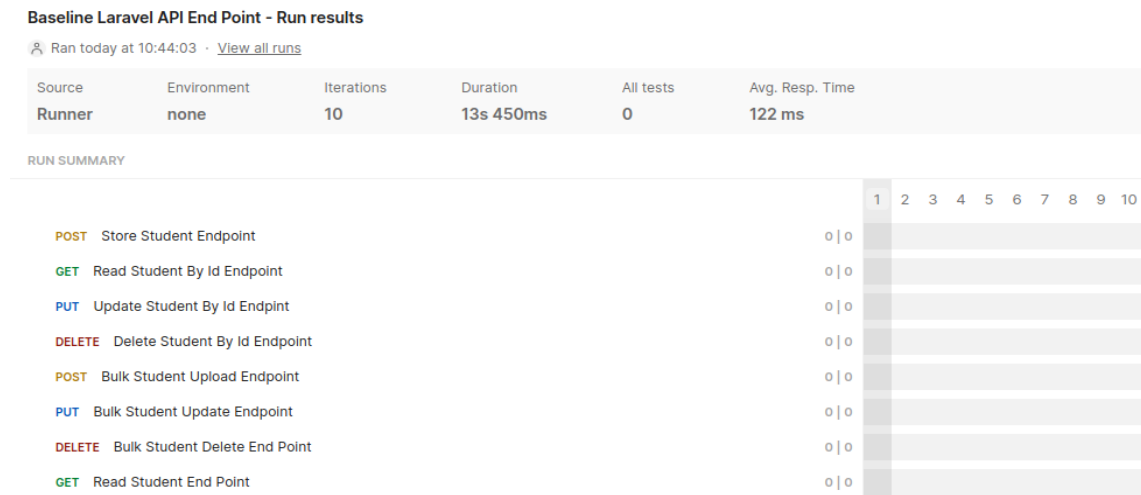


Figure 1.6 Baseline Laravel Request for 10 Iterations.

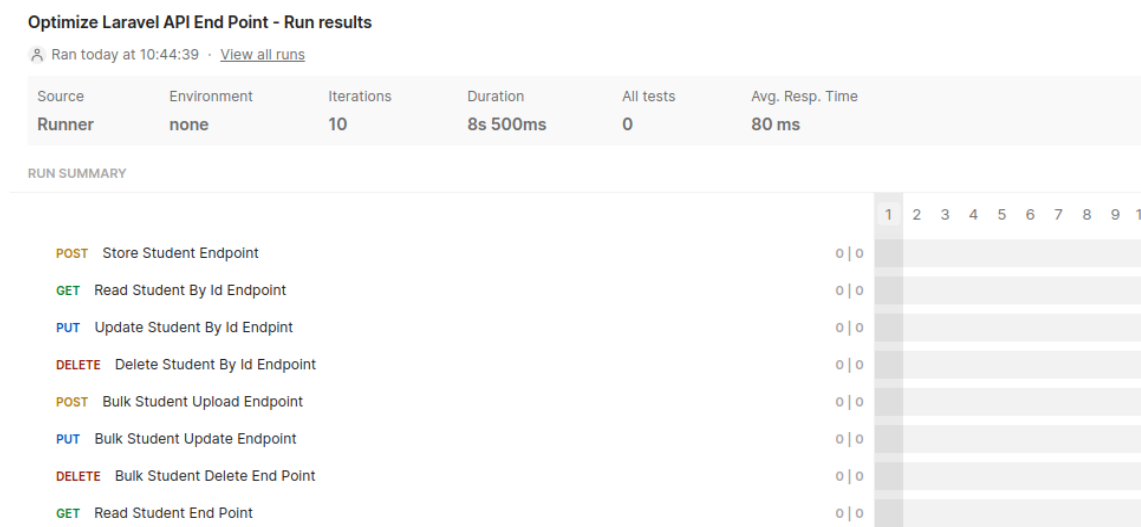


Figure 1.7 Optimize Laravel Request for 10 Iterations.



For Laravel, The implementation of route and view caching, query optimization and eager loading, code refactoring, and minimization efforts in Laravel significantly reduced response times. Data indicates a change from 119 ms to 94 ms given the difference of 25 ms reduction in average response time for single iteration and from 122 ms to 80 ms difference of 42 ms in 10 iterations. And also, CodeIgniter's efficient caching techniques, database indexing, and query caching and efficient code practices also contributed to enhanced performance, with a reduction in response time. Data indicates a change from 29 ms to 17 ms given the difference of 12 ms reduction in average response time for single iteration and from 25 ms to 18 ms difference of 7 ms in 10 iterations.

Both frameworks benefit from their respective optimization techniques. Laravel's advanced features and tools provide significant performance improvements when properly utilized. CodeIgniter's simpler, more straightforward optimization techniques also prove effective, particularly in high-performance scenarios.

### **Analysis**

The comparative analysis of performance optimization techniques in Laravel and CodeIgniter reveals significant improvements in response times for both frameworks, showcasing the effectiveness of various strategies employed. However, CodeIgniter framework is more efficient compared to Laravel framework in terms of efficiency.

For Laravel, the implementation of the below optimization techniques demonstrated a marked improvement in performance:

- i. Utilizing Laravel's built-in caching mechanisms, such as route caching and view caching, significantly reduced the response times by minimizing the overhead associated with route resolution and view rendering.
- ii. Optimizing database queries and implementing eager loading to reduce the number of queries executed resulted in notable performance gains.
- iii. Code Refactoring and Minimization: Refactoring code to eliminate redundancy and streamline processes

CodeIgniter also exhibited substantial performance enhancements through the application of various optimization techniques:

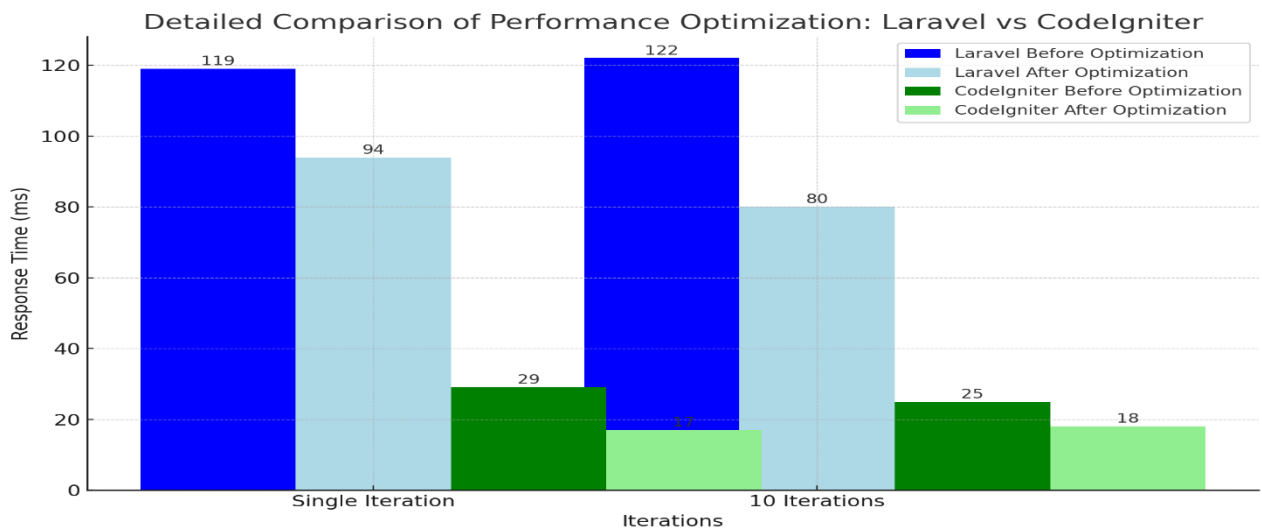
- i. Efficient Caching Techniques: CodeIgniter's caching mechanisms effectively reduced response times by storing frequently accessed data and minimizing the need for repeated computations.
- ii. Database Indexing and Query Caching: Implementing database indexing and query caching improved query execution times and overall database performance.
- iii. Efficient Code Practices: Adopting best practices for writing clean, efficient code minimized processing overhead and optimized resource utilization.

### **Data Analysis**

These results underscore the impact of Laravel's comprehensive optimization features, which collectively contribute to improved application performance and user experience. And also, findings from tables 1.0, 1.1, and 1.2 show that CodeIgniter has a slow response time and higher throughput compared to Laravel. CodeIgniter's inherent efficiency and its ability to achieve high performance with minimal overhead make it a suitable choice for applications where speed and simplicity are needed.

For Laravel Single Iteration, when optimization was considered, the average response time decreased from 119ms to 94ms, reflecting a reduction of 25 ms. while for 10 iterations, the average response time decreased from 122 ms to 80 ms, indicating a significant reduction of

42 ms. For CodeIgniter Single Iteration, when optimization was considered, the average response time decreased from 29 ms to 17 ms, reflecting a reduction of 12 ms. Also, for 10 iterations, the average response time decreased from 25 ms to 18 ms, indicating a reduction of



7 ms. As shown in figure 1.8

**Figure 1.8 Detailed comparison of performance optimization of Laravel and CodeIgniter.**

### Comparative Insight

The data indicates that both Laravel and CodeIgniter benefit significantly from performance optimization techniques, though the degree of improvement and the specific areas of impact vary between the two frameworks:

The more feature-rich framework, Laravel, showed substantial gains particularly in scenarios with multiple iterations, demonstrating its ability to handle complex, resource-intensive applications efficiently. The reductions of 25ms and 42ms in average response times for single and multiple iterations respectively, highlight the effectiveness of Laravel’s caching, query optimization, and code refactoring techniques.

Known for its simplicity and minimalistic design, CodeIgniter exhibited lower initial response times and continued to show improvements with optimizations. The reductions of 12ms and 7ms for single and multiple iterations respectively, indicate the high efficiency achieved through caching, database optimization, and code efficiency practices.

### Key Findings

In this paper, a comprehensive review and comparative analysis of two popular PHP frameworks was conducted, Laravel and CodeIgniter, focusing on their performance optimization techniques. The key findings from the study are summarized below:

#### Performance Optimization Techniques

**Laravel:** The implementation of route and view caching, query optimization, eager loading, and code refactoring significantly reduced response times. These techniques collectively contributed to substantial performance improvements, making Laravel well-suited for complex, feature-rich applications.

**CodeIgniter:** Efficient caching techniques, database indexing, query caching, and streamlined code practices also led to notable reductions in response times. CodeIgniter’s minimalist architecture and simplicity enabled high performance, particularly in lightweight applications.

### **Comparative Analysis of Performance Metrics**

Average response time, throughput and error rate is considered as the performance metrics

Average Response Time:

Laravel showed a reduction from 119ms to 94ms for single iterations and from 122ms to 80ms for ten iterations. CodeIgniter showed a reduction from 29ms to 17ms for single iterations and from 25ms to 18ms for ten iterations. While Laravel benefited more significantly from optimization in multi-iteration scenarios, CodeIgniter maintained lower initial response times.

Both frameworks demonstrated improved throughput with optimization techniques, with CodeIgniter typically achieving higher throughput due to its simpler design.

Both frameworks maintained low error rates, with Laravel's advanced error handling features providing robust support for complex applications, and CodeIgniter's straightforward setup minimizing potential errors.

### **Effectiveness of Optimization Techniques**

Laravel: Advanced features and tools like Eloquent ORM, Blade templating engine, and Artisan Console, when optimized effectively, can achieve high performance and scalability.

CodeIgniter: Its efficient use of resources and minimalistic design allow for quick response times and high throughput, making it an excellent choice for high-performance needs with fewer complexities.

### **RECOMMENDATIONS**

Project Complexity and Requirements: Choose Laravel for projects requiring extensive features, complex data handling, and robust error handling. Its rich ecosystem and comprehensive documentation support complex application development.

Performance and Simplicity: Choose CodeIgniter for projects where simplicity, speed, and ease of setup are prioritized. Its lightweight nature and efficient performance make it ideal for straightforward applications.

### **CONCLUSION**

Laravel showed a reduction from 119ms to 94 ms when optimization techniques are applied for single iterations and from 122 ms to 80ms for ten iterations. CodeIgniter showed a reduction from 29 ms to 17ms for single iterations and from 25 ms to 18 ms for ten iterations. While Laravel benefited more significantly from optimization in multi-iteration scenarios, CodeIgniter maintained lower initial response times.

CodeIgniter proof to have lower response time, throughput and also Lower received and sent KB/sec rates, this indicates that CodeIgniter processes requests faster than Laravel indicating less data being processed per request in the other hand Laravel shows slower response times across all metrics compared to CodeIgniter and significantly higher received KB/sec, which may indicate more complex data processing or larger payloads. Nevertheless, both Laravel and CodeIgniter offer significant performance optimization capabilities, However, Laravel's comprehensive features and advanced optimization tools make it a powerful choice for complex, scalable applications while CodeIgniter's simplicity and efficiency provide a high-performance solution for lightweight applications Understanding the specific needs and constraints of a project will guide developers in choosing the most suitable framework, balancing performance, and functionality to achieve optimal results.

## REFERENCES

- Alshannaq, H. (2020). Web performance optimization: A comprehensive survey. *Journal of Internet Services and Applications*, 11(1), 1-22. <https://doi.org/10.1186/s13174-020-00132-0>.
- Apache JMeter. (n.d.). *Apache JMeter*. Retrieved from <https://jmeter.apache.org/>
- Bastian, R., & Friedrich, T. (2021). *Scalable web applications: Principles and practices*. Springer.
- Blackfire. (n.d.). *Blackfire*. Retrieved from <https://www.blackfire.io/>
- CodeIgniter Documentation. (n.d.). *Caching*. Retrieved from [https://codeigniter.com/user\\_guide/libraries/caching.html](https://codeigniter.com/user_guide/libraries/caching.html)
- CodeIgniter Documentation. (n.d.). *Database Class*. CodeIgniter Foundation.
- Davis, J., & Patterson, D. (2019). CPU resource management for web servers. *ACM Transactions on Computer Systems*, 37(2), 1-30. <https://doi.org/10.1145/3349870>
- Gutierrez, A. (2020). *Phalcon documentation*. Phalcon Framework.
- Jones, E. (2022). Competitive advantage through web application performance. *Business Technology Journal*, 17(3), 45-60.
- Laravel Documentation. (2023). *Laravel: The PHP framework for web artisans*. Retrieved from <https://laravel.com/docs>
- Laravel Documentation. (n.d.). *Caching*. Retrieved from <https://laravel.com/docs/cache>
- Laravel Documentation. (n.d.). *Database: Eloquent ORM*. Retrieved from <https://laravel.com/docs/eloquent>
- Laminas Documentation. (2023). *Laminas: A PHP framework for enterprise applications*. Retrieved from <https://docs.laminas.dev>
- Miller, G. (2021). Maintenance and development efficiency in web frameworks. *Software Engineering Review*, 45, 23-38. <https://doi.org/10.1016/j.ser.2020.100545>
- Nagios. (n.d.). *Nagios*. Retrieved from <https://www.nagios.org/>
- New Relic. (n.d.). *New Relic*. Retrieved from <https://newrelic.com/>
- Otwell, T. (2019). *Laravel: Up & running: A framework for building modern PHP apps*. O'Reilly Media.
- Phalcon Documentation. (2023). *Phalcon: A full-stack PHP framework delivered as a C-extension*. Retrieved from <https://docs.phalcon.io/>
- Potencier, F. (2020). *Symfony 5: The fast track*. Symfony SAS.
- Siege. (n.d.). *Siege*. Retrieved from <https://www.joedog.org/siege-home/>
- Smith, J. (2020). Optimization techniques in computational algorithms. *Journal of Computational Science and Engineering*, 15(2), 101-115. <https://doi.org/10.1016/j.jcse.2020.03.001>
- Symfony Documentation. (2023). *Symfony: High performance PHP framework for web development*. Retrieved from <https://symfony.com/doc>
- Welling, L., & Thomson, L. (2017). *PHP and MySQL Web Development* (5th ed.). Addison-Wesley.
- Węgrzecki, K. S., & Dzieńkowski, M. (2022). Performance analysis of Laravel and Yii2 frameworks based on the MVC architectural pattern and PHP language. *Journal of Computer Sciences Institute*, 24, 265-272. <https://doi.org/10.35784/jcsi.3002>
- Yii Software LLC. (2020). *The definitive guide to Yii 2.0*. Yii Software LLC.
- Yii Framework. (2023). *Yii framework: Fast, secure, and professional PHP framework\**. Retrieved from <https://www.yiiframework.com/doc/guide/2.0/en>
- Xdebug. (n.d.). *Xdebug*. Retrieved from <https://xdebug.org/>