BSc (Hons) Web & Mobile Development

A Comparative Performance Analysis of PHP MVC Frameworks: Laravel, Symfony, CodeIgniter, and Yii

Computing Honours Project (COMP10034)

Interim Report

Corey Black
B01651145

19/12/2025

Supervisor: Dr. Pablo Salva Garcia
Moderator: Mr. Sajjad Bagheri

1 Contents

2 Introduction

PHP is still one of the most widely used server-side programming languages in modern web development, supporting a large percentage of websites and web-based applications worldwide. The PHP ecosystem has progressed significantly, moving from simple procedural scripting toward robust, framework-driven development practices, that support better scalability, maintainability, and productivity. Central to this evolution is the Model-View-Controller (MVC) architectural pattern, which has become the predominant design pattern, enabling a clean separation between application logic, user interfaces, and data handling processes.

As PHP MVC frameworks have matured and improved, developers have many options to choose form, each with its own strengths and trade-offs. Popular frameworks like Laravel, Symfony, CodeIgniter, and Yii are commonly used, but the choice of framework is often based on what developers already know or what is most popular, rather than on actual performance data. This can lead to poor technical decisions, particularly in projects where performance, scalability, and efficient use of resources really matter.

The primary aim of this honours project is to asess and compare the performance, scalability and developer experience of four widely used PHP MVC frameworks. These frameworks were selected due to their prevalence in real-world PHP development and their differing architectural philosophies. In response to the limited availability of standardised performance comparisons, the study involves the development of functionally equivalent CRUD-based applications in each framework and testing them under the same conditions.

The objectives of this project include the development of functionality equivelent applications across the selected frameworks, the execution of performance benchmarking using industry-standard load testing tools, and the analysis of both quantitative metrics and qualitative development factors. By producing a comparative report supported by empirical data, this project aims to provide practical guidance to developers and organisations when selecting an appropriate PHP framework for real-world applications.

This interim report presents an overview of the project background, a review of relevant literature, the chosen methodology, progress achieved to date, and a detailed plan for completing the remaining stages of the project.

3 Background and Literature Review

3.1 PHP and the MVC Architectural Pattern

The Model-View-Controller (MVC) architectural pattern is a common architectural framework that organises application logic into three interconnected components. The model is responsible for managing data and application logic, the view manages the presentation layer, and the controller interprets user actions and facilitates communication between the model and view. This separation makes applications easier to maintain, test, and scale, especially in larger web based applications.

PHP MVC frameworks implement the MVC architectural pattern in a structured manner, equipping developers with essential tools including routing, database abstraction, templating and security features. Compared to ad-hoc procedural develoment approaches, the use of MVC frameworks has been shown to enhance code quality and reduce development time, especially in collaborative and long-term projects.

The architectural principles underpinning the Model-View-Controller (MVC) pattern are rooted in established software design theory. Gamma et al. (1994) describe design patterns as reusable solutions to recurring software design problems, emphasising separation of concerns as a key strategy for managing complexity in large systems. By isolating presentation logic from business rules and data management, MVC reduces the cognitive complexity of individual components and enables parallel development across teams. As a result, MVC has become a foundational pattern in modern web frameworks, including those within the PHP ecosystem.

3.2 Overview of Selected PHP Frameworks

While PHP frameworks offer significant productivity and maintainability benefits, their architectural design choices can directly influence runtime performance. Frameworks such as Laravel and Symfony prioritise abstraction, extensibility, and developer ergonomics, often introducing additional layers such as service containers, middleware pipelines, and dependency injection systems. These features improve long-term maintainability but may incur computational overhead, particularly under high concurrency (Pautasso, et al., 2017).

In contrast, lightweight frameworks such as CodeIgniter reduced abstraction in favour of direct execution paths, which can result in faster response times and lower memory consumption (Suraski, 2019). Yii adopts a hybrid approach, incorporating performance optimisations such as lazy loading and caching while still offering modern framework features. Understanding how these architectural trade offs manifest in real world performance scenarios is essential for informed framework selection, particularly for resource constrained or high traffic environments.

This project addresses this gap by empirically measuring the performance implications of these design philosophies using standardised benchmarks and equivalent application logic.

3.3 Summary of Framework Characteristics

Laravel is a feature-rich PHP framework designed to enhancee developer productivity and expressive syntax. It provides built-in support for routing, authentication, database migrations, and object-relational mapping through Eloquent. While Laravel is widely praised for its ease of use and expressive syntax, its extensive abstraction layers can introduce measurable performance overhead in certain scenarios, particularly under high concurrency.

Symfony is a highly modular and configurable framework commonly used in enterprise-level applications. Its component-based architecture allows developers to utilise individual components independently or as part of the full framework. Symfony prioritises flexibility and long-term maintainability, though this can result in a steeper learning curve and more complex configuration.

CodeIgniter is a lightweight PHP framework that focuses on simplicity and minimal overhead. It offers core MVC functionality without enforcing strict conventions, making it suitable for smaller applications or performance-sensitive environments. However, its reduced feature set may require additional manual implementation when developing complex systems.

Yii is a performance-oriented PHP framework that emphasises efficiency and rapid development. It includes features such as lazy loading, caching mechanisms, and code generation tools. Yii aims to strike a balance between performance and developer convenience, positioning itself as a high-performance alternative to more heavyweight frameworks.

3.4 Performance Metrics in Web Applications

Performance evaluation in web applications commonly focuses on measurable metrics that reflect both user experience and server efficiency. Response time represents the duration taken to process and return a request, directly influencing perceived application speed. Throughput measures the number of requests that can be handled within a given timeframe, providing insight into scalability under load. Memory usage reflects resource effciency and is particularly important in constrained hosting environments.

Benchmarking these metrics under controlled conditions allows for meaningful comparisons between frameworks, provided that testing environments, workloads, and configurations remain consistent.

3.5 Existing Comparative Studies

Previous studies and industry benchmarks have attempted to compare PHP frameworks; however, results often vary due to inconsistent methodologies, differing configurations, and non-equivalent application logic. Some studies prioritise micro-benchmarks, while others focus on synthetic workloads that may not reflect real-world usage. These limitations highlight the need for a standardised benchmarking approach using functionally equivalent applications and reproducible testing conditions, which this project seeks to implement.

Several academic and industry studies have examined the performance characteristics of PHP frameworks, though findings often differ due to methodological inconsistencies. Many comparative benchmarks focus on isolated request handling or synthetic workloads that do not reflect real-world application behaviour. In addition, variations in server configuration, framework versions, and database interaction patterns frequently limit the generalisability of reported results.

Some studies prioritise raw response time without considering scalability or resource usage, while others focus on developer productivity rather than runtime efficiency. As a result, conclusions drawn from such benchmarks may be misleading when applied to production environments. Furthermore, comparisons that rely on non-equivalent application logic risk attributing performance differences to frameworks rather than implementation choices.

These limitations highlight the need for a standardised and reproducible benchmarking approach that combines functionally equivalent applications with consistent testing conditions. By implementing identical CRUD-based aplications and controlling environmental variables, this project seeks to address gaps in existing comparative research and provide more reliable performance insights.

4 Methodology and Progress to Date

4.1 Research Methodology

This project adopts an experimental benchmarking research methodology, as it allows performance differences between frameworks to be examined under tightly controlled conditions. Thios approach was considered the most appropriate for ensuring fair and reproducible comparisons. Experimental approaches allow the independent variable (PHP framework choice) to be manipulated while holding dependent variables such as database schema, request type, and server configuration constant, ensuring a fair and reliable comparison between frameworks (Kitchenham et al., 2009).

Functionally equivalent CRUD-based web applications will be developed using Laravel, Symfony, CodeIgniter and Yii. CRUD applications were selected as they represent a common pattern in real-world web systems, encompassing database interaction, routing, controller logic, and view rendering. This ensures that benchmarking results reflect realistic application workloads rather than artificial micro-tests.

All frameworks will be implemented using PHP 8.x to ensure modern language features and performance improvements are utilised consistently. Each application will interact with the same database schema and perform identical operations to minimise variability arising from application logic differences.

Performance metrics, including response time, throughput, and memory usage, will be collected under identical hardware and software configurations to minimise environmental variance. Testing will be conducted in a non-production environment using PHP 8.x and MySQL, with server configurations kept consistent across all tests.

In addition to quantitative benchamarking, qualitative evaluation will be conducted to assess factors such as ease of setup, documentation quality, and overall developer experience. This mixed approach enables both objective performance measurement and contextual evaluation of framework usability.

4.2 Methodological Justification for Benchmarking Approach

The benchmarking methodology adopted in this project is grounded in established empirical software engineering research. Controlled experimentation enables reliable comparison by isolating a single independent variable – in this case, framework choice – while holding all other variables constant. Kitchenham et al. (2009) emphasise that reproducible experimental design is essential for producing valid and generalisable performance results. To further ensure measurement accuracy, identical workloads and execution environments are employed across all frameworks, reducing the influence of external variance. Jain (1991) recommends repeated test execution and result averaging as standard practice in performance evaluation to minimise the impact of transient system behaviour and measurement noise. Together, these principles support the validity and reliability of the benchmarking approach used in this study.

4.3 Benchmarking Tools and Load Testing Approaches

Load testing tools play a critical role in evaluating the scalability and robustness of web applications. Apache JMeter is widely adopted within industry and academia for its ability to simulate concurrent HTTP requests, configurable test plans, and detailed performance reporting (Apache Software Foundation, 2024). Its graphical interface allows precise control over thread groups, ramp-up periods, and request sampling. I selected Apache JMeter and Locust due to their complementary testing approaches and their widespread use in the industry.

Locust, by contrast, adopts a Python-based, event-driven approach that enables more flexible and programmatic load scenarios. Locust is particularly suited to modelling real-world user behaviour through scripted workflows rather than isolated request bursts (Locust.io, 2024). By utilising both tools, this project aims to balance repeatable, standardised benchmarks (JMeter) with behaviur-driven testing scenarios (Locust).

To ensure the validity and reliability of benchmarking results, all load tests will be executed multiple times, with average values recorded to reduce the impact of transient system fluctuations. Test scenarios will be identical across frameworks, including request types, concurrency levels, and execution order. This approach supports results repeatability and reduces the likelihood of anomalous outcomes influencing analysis.

Using both Apache JMeter and Locust further strengthens methodological robustness by allowing cross-verification of results generated through different testing paradigms. Where discrepancies arise, results will be examined in context rather than treated in isolation. This multi-tool approach enhances confidence in observed performance trends and supports more defensible conclusions.

4.4 Progress to Date

At the interim stage, the project has progressed in line with the approved specification and planned timeline. The research scope and objectives have been finalised, and a comprehensive background review has been conducted covering PHP MVC architecture, framework design philosophies, and web application performance metrics.

Technical evaluation of Laravel, Symfony, CodeIgniter, and Yii has been undertaken to assess installation complexity, configuration requirements, and arcitectural conventions. This has informed the decision to standardise application structure and database schema prior to implementation to ensure fairness across benchmarks.

Benchmarking tools, including Apache JMeter and Locust, have been evaluated and selected based on industry relevance and complementary testing capabilities. Initial test plans have been drafted to simulate concurrent user loads and capture response time, throughput, and memory usage metrics.

A preliminary database schema and CRUD endpoint specification have been designed to ensure consistency across framework implementations. The testing environment has been planned using a local XAMPP or Docker-based setup to ensure reproducibility and configuration parity.

Overall, the project is progressing in line with the original plan. Foundational research and design decisions have now been completed, allowing development and benchmarking to begin immediately after the interim milestone.

4.5 Preliminary Application Design

To ensure a fair and reproducible comparison between frameworks, a standardised application design has been defined prior to implementation. Each framework will implement a functionally equivalent CRUD-based web application, exposing identical endpoints and interacting with the same underlying database schema.

The application will follow each framework's recommended MVC conventions while preserving identical business logic and request workflows. This design approach ensures that observed performance differences can be attributed primarily to framework architecture rather than implementation variance.

At the interim stage, the application structure, routing patterns, and endpoint specifications have been designed to support consistent benchmarking across all frameworks. These design artefacts form the foundation for the development phase that follows this milestone.

4.6 Standardised Database Schema

The CRUD-based applications will expose a consistent set of HTTP endpoints across all frameworks. These endpoints represent common real-world web application behaviour and allow meaningful performance comparison under load. The planned endpoint specification is outlined in Table 1.

Endpoint	**HTTP Method**	**Description**
/items	GET	Retrieve all records
/items/{id}	GET	Retrieve a single record
/items	POST	Create a new record
/items/{id}	PUT	Update an existing record
/items/{id}	DELETE	Delete a record

Each endpoint will execute equivalent database operations and response payloads across all frameworks. This ensures that benchmarking results reflect framework-level performance characteristics rather than differences in application logic or data access patterns.

4.7 Preliminary Database Schema

A preliminary relational database schema has been designed to support CRUD operations across all framework implementations. The schema is intentionally minimal to avoid introducing unnecessary complexity while remaining representative of real-world applications.

Field	Type	Description
Id	INT (PK)	Unique record identifier
Name	VARCHAR(255)	Item name
description	TEXT	Item description
created_at	TIMESTAMP	Creation timestamp
updated_at	TIMESTAMP	Last update timestamp

4.8 Limitations and Scope of Study

While this project aims to provide meaningful insights into the comparative performance of PHP MVC frameworks, certain limitations must be acknowledged. Performance testing will be conducted within a controlled, non-production environment, which may not fully replicate the complexity of live deployment scenarios. Factors such as network latency, hardware diversity, and real user behaviour are therefore outside the scope of this study.

Additionally, the use of CRUD-based applications, while representetive of common web development patterns, does not encompass all possible framework use cases. Applications with extensive background processing, real-time communication, or complex business logic may exhibit different performance characteristics.

The scope of the project is deliberately to ensure feasibility within the available timeframe. Only core framework features will be evaluated, and optimal extensions or third-party packages will be excluded unless required for basic operation. Despite these limitations, the study provides a controlled and reproducible comparison that offers practical value for developers selecting PHP frameworks based on performance and developer experience.

5 Planned Work and Project Timeline

5.1 Planned Work

The next phase of the project will focus on the implementation of functionally equivalent CRUD-based web applications across the selected PHP MVC frameworks. Development will begin with lighter frameworks, namely CodeIgniter and Yii, before progressing to the more feature-rich frameworks, Laravel and Symfony. This sequencing has been chosen to reduce cognitive overhead and allow lessons learned from simpler implementations to inform subsequent development stages.

Once development is complete, performance benchmarking will be conducted using predefined load testing scenarios. These tests will simulate increasing numbers of concurrent users and varying request loads in order to capture performance behaviour under realistic conditions. Quantitative metrics including response time, throughput, and memory usage will be collected for each framework.

Following data collection, results will be analysed using statistical summares and data visualisation techniques to identify trends, performance differences, and scalability characteristics between frameworks. In parallel, a qualitative evaluation will be conducted based on development effort, configuration complexity, documentation quality, and overall developer experience encountered during implementation.

The final stage of the project will involve synthesising findings into a comprehensive dissertation and preparing a formal presentation that communicates both the technical results and their practical implications.

5.2 Risks and Mitigation

Potential risk associated with this project include variations in framework configuration that could affect performance results, time constraints related to implementing multiple frameworks, and unexpected technical challenges during benchmarking.

To mitigate these risks, strict configuration consistancy will be maintained across all test environments, including identical server settings, database schemas, and application logic. The scope of each application will be deliberately limited to core CRUD functionality to prevent feature creep and ensure timely completion.

Time management risks will be mitigated through a structured development schedule and regular progress monitoring against the project timeline. Additionally, early identification testing of benchmarking tools will reduce the likelihood of delays during the evaluation phase.

5.3 Project Timeline

The project will follow the timeline outlined below:

- **Weeks 1-3:** Application development across all selected frameworks
- **Weeks 4-5**: Load testing and benchmarking using Apache JMeter and Locust
- **Weeks 6-7:** Data analysis, evaluation of preformance results, and initial write up of findings
- **Weeks 8-9:** Final report refinement, integration of results, and data visualisation
- **Week 10:** Final review, proofreading, and submission

This structured timeline is designed to ensure steady progress while allowing sufficient time for analysis and refinement prior to submission.

6 Conclusion

This interim report has outlined the background, objectives, methodology, and current progress of my honours project investigating the comparative performance of PHP MVC frameworks. Foundational research has been completed, and a clear experimental methodology has been established to support the development and benchmarking phases of the project.

Progress to date indicates that the project is advancing in line with the approved specification, with key technical decisions regarding framework selection, benchmarking tools, and testing environments already established. The proposed approach aims to produce meaningful and reproducible performance comparisons supported by both quantitative metrics and qualitative evaluation.

Upon completion, this project is expected to provide practical insights that assist developers and organisations in selecting PHP MVC frameworks based on empirical evidence rather than assumption. Continued adherence to the planned timeline and methodology will support the successful completion of the project within the academic year.

7 Bibliography

Apache Software Foundation, 2024. *Apache JMeter.* [Online]
Available at: https://jmeter.apache.org/
[Accessed 17 December 2025].

Gamma, E., Helm, R., Johnson, R. & Vlissides, J., 1994. *Design Patterns: Elements of Reusable Object-Oriented Software.* Boston: Addison-Wesley.

Jain, R., 1991. *The Art of Computer Systems Performance Analysis.* New York: Wiley.

Kitchenham, B. et al., 2009. Systematic Literature Reviews in Software Engineering – A Systematic Literature Review. *Information and Software Technology,* 51(1), pp. 7-15.

Locust.io, 2024. *Locust Documentation.* [Online]
Available at: https://locust.io/
[Accessed 17 December 2025].

Pautasso, C., Zimmermann, O. & Leymann, F., 2017. RESTful Web Services vs. Big Web Services: Making the Right Architectural Decision. *World Wide Web,* 20(1), pp. 87-133.

Suraski, J., 2019. *PHP Architect's Guide to PHP Design Patterns.* San Diego: PHP Architect.

8 Student Declaration

UWS Assessment Student Declaration

This Declaration must be completed and submitted along with your assessment. Please ensure sections 1 and 2 are completed before submission.

- For written assessments, insert the Declaration as the first page of your document.
- For assessments not in the written format (e.g. video, audio, presentation, or practical work), submit the Declaration as a separate file.
- For group assessments, unless otherwise directed by your lecturer, the group may submit a single shared declaration.

1. Declaration that this is your OWN work	
All UWS students are expected to uphold the values of academic integrity: UWS Student Academic Integrity Procedure. Academic Integrity means a commitment to, and upholding of, the values of honesty, trust, fairness, respect, responsibility and courage in learning, teaching, research and engagement with the University community. You are responsible for ensuring that the work you submit is your own, and that any use of Generative AI follows the guidance provided for the assessment.	
I confirm this assessment is my own work and complies with the guidance provided.	**YES**
2. Declaration of appropriate use of Generative AI	
You must check the assessment guidance to understand what uses, if any, of Generative AI are permitted for this assessment.	
I used Generative AI for this assessment.	**NO**

Extenuating Circumstances

The University recognises that, from time to time, you may encounter circumstances that affect your ability to complete or submit an assessment. If this happens you can submit an Extenuating Circumstances Submission (ECS). In submitting each piece of coursework or completing an examination or class test, you are confirming that you are 'fit to sit' the assessment and wish that any mark achieved for that assessment should stand. You can submit an ECS up to 48 hours after the assessment deadline, including where you have submitted the assessment but believe your academic performance has been affected by extenuating circumstances. The School Assessment Board will know that an ECS has been submitted when recording your module marks. See the UWS Extenuating Circumstances Submission Procedure for further guidance.