JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 22 (2022) 18–25
Received: 15 October 2021
Accepted: 21 October 2021

Comparative analysis of PHP frameworks on the example of Laravel and Symfony

Analiza porównawcza szkieletów PHP na przykładzie Laravel i Symfony

Paulina Garbarz*, Małgorzata Plechawska-Wójcik

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract

This study aims to determine which of the analyzed PHP-based design patterns – Laravel or Symfony – is a more sufficient solution and which one of them is more complex from the code point of view. For this purpose, a comparative analysis was carried out based on the available documentation, as well as a comparison of the static and dynamic metrics obtained in the research environment of both tested patterns. As a result of a series of experiments and studies, it was established that both design patterns are an optimal and efficient solution, but their best application depends on the developer's individual needs and project requirements.
Keywords: PHP frameworks; Laravel; Symfony; comparative analysis

Streszczenie

Niniejsza praca ma na celu ustalenie, który z analizowanych wzorców projektowych opartych na języku PHP – Laravel czy Symfony – jest wydajniejszym rozwiązaniem, a także który z nich jest bardziej złożony z punktu widzenia kodu. W tym celu przeprowadzono analizę porównawczą opartą na dostępnej dokumentacj, a także porównaniu uzyskanych w środowisku badawczym metryk statycznych i dynamicznych obu badanych wzorców. Wynikiem serii eksperymentów i badań ustalono, że oba wzorce projektowe stanowią optymalne i wydajne rozwiązanie, jednak ich najkorzystniejsze zastosowanie jest zależne od indywidualnych potrzeb dewelopera oraz wymagań projektu.

Słowa kluczowe: szkielety PHP; Laravel; Symfony; analiza porównawcza

*Corresponding author

Email address: **garbarz.paulina@gmail.com** (P. Garbarz)

1. Introduction

Nowadays, working without a design pattern is associated with a long code development process and low efficiency, which is why more and more new solutions appear on the market, and the previously created ones are constantly developed. The most popular are the analyzed Laravel and Symfony - GitHub contains approximately 62,000 and approximately 24,000, respectively, projects based on these patterns (Figure 1), which are marked with stars to provide insight into them in the future.

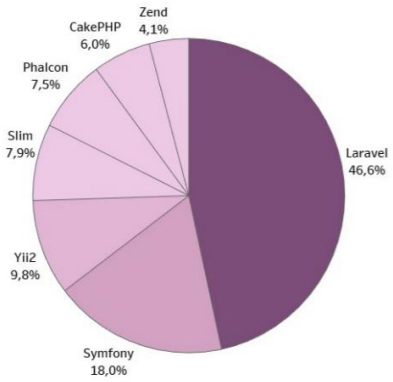

Figure 1: Popularity of particular PHP frameworks – own study based on projects popularity on GitHub.

Frameworks have a similar set of functionalities, but each of them individually has special features that distinguish them from the others. When choosing the right template, it should be taken into account not only the technical aspects and its possibilities, but also the subject of the project and the programmer's skills. Choosing a template can have a significant impact on the future and development of the entire project. That's why choosing the right one is so important.

The main idea behind Symfony is to create and develop long-term projects by default. It is a kind of skeleton for e-commerce platforms such as Magento, Drupal or PrestaShop, but also creates a base for numerous design patterns such as Laravel, Yii or CakePHP. The structure of Symfony is based on the MVC architecture, which provides modularity and complexity to the created projects due to the reusability of the same code in many different places. Its structure consists of a system of bundles, each of which fulfills a specific task.

Symfony is also independent of the selected database environment – it allows the use of a PDO or ORM module to establish communication with the database, due to which it can get connection with many types of databases, such as MySQL, Oracle or SQLServer. The ORM technique dedicated to Symfony is Doctrine ORM, which allows to improve the performance of queries and improve the security of transmitted data. This framework uses its own template engine, which is

called Twig. It allows the creation of communication between the view and the controler throught the model, making it possible to modify or display relevant data for the user.

The second analyzed framework, Laravel, is based on Symfony components and uses this framework as a kind of skeleton, but it is distinguished by its original logic and the way the code is implemented in PHP. Laravel is characterized by fast application development, a high level of abstraction and a high intuitiveness when writing code. Due to the simplicity of creating a project, it is often assigned to small, quickly implemented projects, but the range of possibilities offered also allows for the creation of extensive solutions. The structure of Laravel, as in the case of Symfony, is based on the MVC architecture. Already at the time of creating a new project, the user receives a fully functional environment with the necessary dependencies and functionalities, which allows to reduce the time spent on appropriate adaptation and configuration of this environment.

Laravel enables to use Eloquent ORM by default and PDO module by optional. Eloquent ORM allows to increase the level of application security, and also streamlines the process of communication with different database environments. The way of processing queries is not only more effective, but also more accessible for the programmer creating the application, which affects the efficiency of the application development process.

Due to the usage of the Blade engine template, it is possible to encrease the performance of communication between the view and the controller. Blade templates also allow direct injection of PHP code into the view, due to which additional modifications are not required unlike Symfony.

The article presents the similarities and differences, strengths and weaknesses, as well as the specific features of the Symfony and Laravel frameworks. Its purpose is to determine which of the analyzed frameworks is a better and more efficient solution among the patterns based on the PHP language. In order to carry out the research, a suitable environment has been created, consisting of two applications that perform basic CRUD functions (Create, Read, Update, Delete) using both frameworks. In addition, the paper presents a comparative analysis between the tested patterns, the results of which includes static metrics (such as the number of lines of written code, the amount of used classes and interfaces, the number of used methods and libraries, the overall size of both developed applications, the ease of introducing changes to them and the overall assessment performance), as well as dynamic metrics (average service response time and server load) [1].

2. Literature Review

Each design pattern offers an extensive set of tools, such as APIs and libraries, the meaning of which is described in article [2]. For the study of overall performance, the authors created an environment implementing the same task, based on three different frameworks and three MySQL databases with an identical structure. Ultimately, it was found that each excelled in specific research areas, with Laravel being considered the most advantageous for file read and write operations and the least optimal for complex data.

The article [3] raises an important issue, which is the selection of an appropriate framework, based on the similarities and differences between them. The model created by the authors compares the performance of Laravel and Symfony on specific levels, comparing their available functionalities and evaluating their performance. Both were rated as effective solutions for PHP applications, with Symfony being a more stable pattern and Laravel as a pioneer in creating dynamic solutions.

Currently, Laravel is the most used PHP-based framework. In the article [4] entirely devoted to this pattern, the authors presented its possibilities by analyzing the functionality based on the creation of an E-Commerce website, as well as comparing it to other patterns. The study showed that Laravel is a pioneer in reading and writing data in files as well as in database migration between different areas. The author of the article [5] compares Laravel with the Slim Framework by measuring the load test performance for three different scenarios. The study shows that Slim Framework is a faster and better solution, but Laravel, due to its size and the availability of numerous libraries and solutions, is better for large projects than the competitor. It is also worth mentioning that Laravel also owes its size to the fact that it contains Symfony components, which constitute its specific skeleton [6].

Symfony is one of the oldest PHP frameworks. Like most PHP frameworks, is based on MVC and uses ORM [7]. Despite the long experience, it is still a willingly chosen model, which was noted in the article [8], which proved that Symfony quickly keeps up with the dynamic changes in standards. It is also popular due to the large community and extensive documentation. During its lifetime, some versions were supported for a longer time than others. In the study [9], the three most popular versions were analyzed, using a research environment based on three applications with different versions of Symfony, which fulfill the same task, as well as comparing the performance of individual versions. Finally, the authors determined that Symfony 4.2 deserved a special distinction due to its advantage, for example, when returning a large amount of data from API.

3. Research Method

For the purpose of the study, it was created a research environment that contains two applications with identical functionalities, but implemented in different design patterns – Laravel and Symfony in the latest stable versions (8.5.7 and 5.1.3 respectively). The applications fulfill the tasks of a simple blog that performs basic CRUD operations and also allows for user authentication. Both versions have been tested for performance in communication with database. For this purpose, two

schemas were created for each project – one in MySQL, one in PostgreSQL. Both applications run on a local server. Table 1 shows the parameters of the equipment on which the research environment was realized.

Table 1: The parameters of the equipment

Parameter	Value
Processor	AMD Ryzen 5 4500U
RAM	16GB
Disc	SSD M.2PCIe 512GB
Graphic Card	AMD Radeon Graphics 2.38GHz
Operating System	Windows 10 Home 20H2

The created environment allows for a comparative analysis of Laravel and Symfony in order to determine their similarities, as well as to highlight significant differences. To complete this task, the author used appropriate methods and metrics, divided into static and dynamic metrics.

To perform the benchmarking using static metrics, a PHP tool called PHPLOC was used. It allows to calculate the exact number of lines of source code, the classes and methods contained in the program, and the overall size of the application (including the number of files and folders).

Two extensions for selected frameworks were used to determine dynamic metrics – Laravel Dusk for Laravel and WebTestCase for Symfony, respectively. They allow to simulate user actions, and thus collect data on SRT, QET and TPT metrics [10]. To measure the execution time of a database query, a series of tests should be carried out that perform a specific number of queries to the database. For each operation supported by the application (create, read, update, delete), tests containing 10, 100 and 1000 queries were performed. Before starting each test, it is necessary to clear the cache to maintain the reliability of the results.

3.1. Static Metrics

The following static metriccs were used in the study:

- **Number of lines of code** – Known as LOC (Lines of Code) or SLOC (Source Lines of Code), this is a type of size metric that allows to identify the lines of source code used in a project. Shows the scale of the software, and indicates classes and methods that are beyond the recommended size.
- **Number of classes and interfaces** – this metric allows to specify the exact number of classes and interfaces used in the software source code, it excludes internal classes.
- **Number of methods** – the metric responsible for indicating the number of methods existing in the application. It applies not only to the entire program, but also to individual classes or interfaces.
- **Program size** – a metric responsible for measuring the number of files and folders present in the application, as well as their size on the disk. Its size may be affected by the number of libraries attached to the project, the number of lines of code or the number of files and folders. This value was expressed in KB.

3.2. Dynamic Metrics

In addition to static metrics, the following dynamic metrics were also included in the study:

- **Total Processing Time (TPT)** – this metric is used to measure the time elapsed from sending the request until the application responds. It provides information on the total processing time of the user's command, from its creation, through processing by the server and database, to obtaining a reply for each CRUD operation. For each operation, the average processing time in milliseconds is calculated.
- **Service Response Time (SRT)** – is used to measure the response time of the application server, starting from sending a request to the network service until obtaining the first byte of the response. Similar to TPT, each CRUD operation is processed by the specified number of query samples, from which the average value for each operation is determined. The metric is expressed in milliseconds.
- **Query Execution Time (QET)** – a metric that measures the time that was needed to process the query sent by the application (in this case, the CRUD operation in the amount of samples specified above).

4. Comparative Analysis

Both frameworks – Symfony and Laravel – were analyzed in terms of static and dynamic metrics. This chapter summarizes their possibilities and the performed analysis.

4.1. Comparative Analysis Based on Static Metrics

The comparison of static metrics of projects implemented in Laravel and Symfony enables the comparison of the quality of the source code of the application being developed. The research environment provides information on the number of lines of code, number of classes and interfaces, number of methods, program size, scalability and overall performance score.

Figure 2 summarizes information on the total number of lines of code (LOC) for both projects that are part of the research environment.

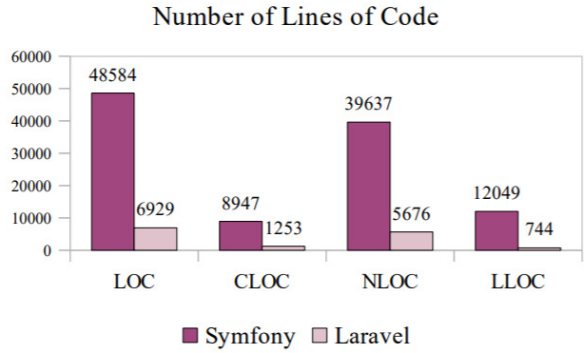

Figure 2: The number of lines of code in Symfony.

It also provides information about the number of comment and non-comment lines of code (CLOC and NLOC metrics respectively), and the number of logical lines of code (LLOC metric) consisting of classes and methods. The numerical advantage of LOC is clearly visible for the benefit of Symfony – the Symfony's LOC is almost 7 times higher than Laravel's. It is also worth noting that Symfony has a greater percentage of CLOC and LLOC (18.42% and 24.80% respectively) than Laravel (12.08% and 10.74% respectively). A higher number of comment lines of code may suggest the presence of more extensive documentation and guidance on the use of Symfony capabilities, while a higher number of lines of code placed in the application logic may result in slower performance as the project grows.

Figure 3 shows a graph showing the number of classes and interfaces available in both frameworks. Symfony, as with the number of lines of code, can boast a considerable number of classes in the project, which exceeds the number of classes in Laravel almost 14 times. Interestingly, none of the patterns offer any interface by default.

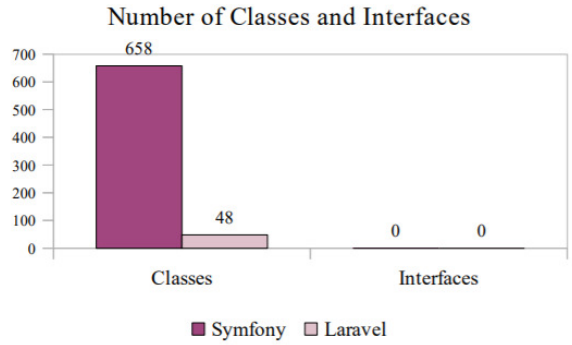

Figure 3: The number of classes and interfaces in Symfony and Laravel.

The above list of the number of classes shows the enormity and complexity of Symfony, but it is worth remembering that it is a fully proprietary framework, while Laravel complements the logic of Symfony with its logic, treating it as its base.

Figure 4 shows the quantification of the different types of method visibility in Symfony.

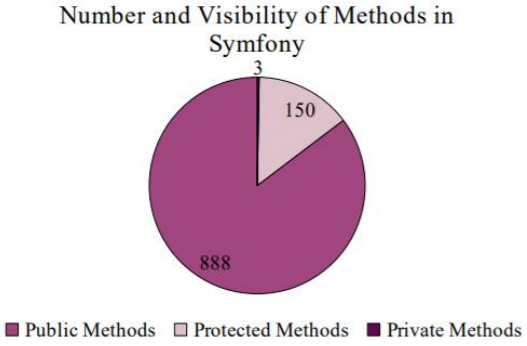

Figure 4: The number and the visibility of methods in Symfony.

It is clearly visible that this framework offers mostly public methods (888), but 15% of all methods (153 exactly) are protected or private. The sheer number of available methods in this framework is considerable, because the project implemented in Symfony in the research environment has 1041 of them. The audience of Symfony methods allows to conclude that the framework by default provides the standard high security of its components by restricting access to them.

The next graph (Figure 5) shows the number and visibility of the methods in the application implemented in Laravel. Their number, as in the case of LOC and the number of classes and interfaces, is several times lower than in the case of Symfony. It is worth noting that most of the methods available in the Laravel (77) application are public, while offering only 4 protected methods and no private methods.

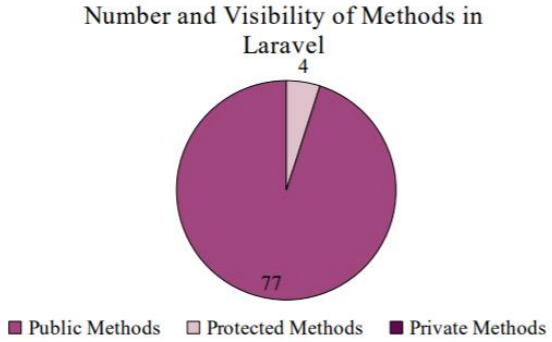

Figure 5: The number and the visibility of methods in Laravel.

Collected static metrics clearly show that Symfony creates larger-sized projects with a much higher number of components than Laravel. This is due to the construction of the entire framework, because Symfony takes full advantage of its proprietary solutions, while Larave relies on Symfony components, complementing them with its own logic.

The volume of the project can have a significant impact on the performance and smoothness of the application to the benefit of Laravel, but Symfony, due to the greater number of protected and private methods, can provide greater security of the designed solutions.

4.2. Comparative Analysis Based on Dynamic Metrics

Both Laravel and Symfony frameworks have been subjected tu dynamic metric analysis summarized in this chapter. It used data collected for 1000 samples, as it turned out to be the most reliable in the context of the entire study. The following therminology is used in the graphs: L for Laravel and S for Symfony.

At the beginning, the cooperation of both patterns with the PostgreSQL database was analyzed, which is visible at the Figure 6 describing the SRT for 1000 post.

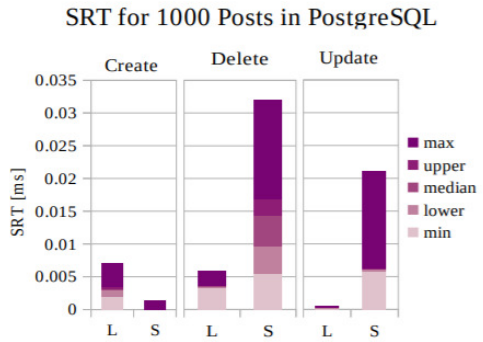

Figure 6: SRT metric for 1000 post sample in PostgreSQL.

It can be seen that when a new post was placed, Symfony obtained better results, but in other cases (delete and update) its result is much worse than Laravel in terms of efficiency – these operations take much longer

The situation is different for the SRT with a sample of 1000 comments presented at Figure 7.

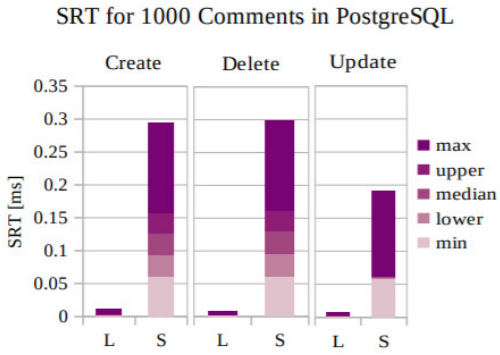

Figure 7: SRT metric for 1000 comments sample in PostgreSQL.

Laravel did better for each operation performed (the waiting time for a response is close to almost 0), while Symfony took definitely more time to deliver the first byte of the response, sometimes up to 0.3ms.

The time of querying the database at the time of creating a new post (Figure 8) was definitely in favor of Symfony.

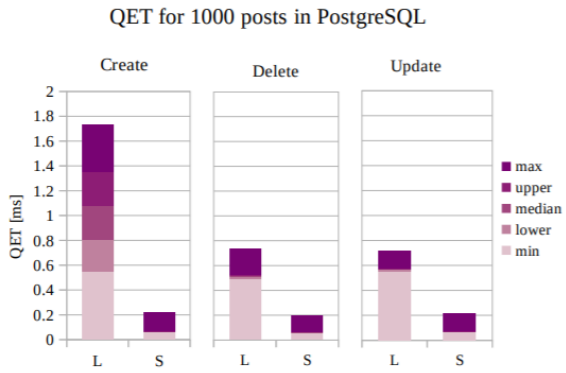

Figure 8: QET metric for 1000 posts sample in PostgreSQL.

With each operation, it is visibly lower than in the case of Laravel (almost 9 times for the create operation itself). The operation of creating a post is the most absorbing among those listed for Laravel, while the results for Symfony are very similar and remain at the level of 0.2 ms.

The situation is similar in the case of posting a comment on the blog (Figure 9). The query processing time by Symfony is much lower than in the case of Laravel, despite the fact that both patterns maintain the values or all the above-mentioned operations at a similar level.

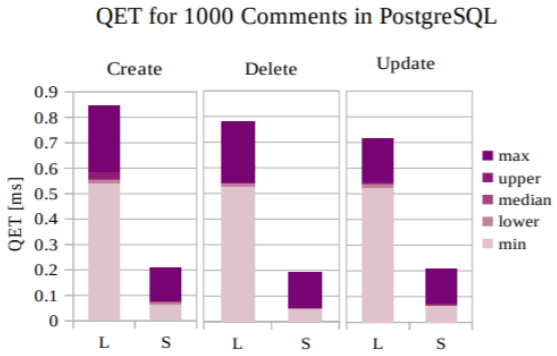

Figure 9: QET metric for 1000 comments sample in PostgreSQL.

It is worth noting, however, that the time for adding a comment by Laravel is significantly lower than the time for adding a post - this is due to the fact that less data lands in the database (a comment is much shorter than a blog entry). Symfony is in the area of 0.2ms in both cases.

The difference in query processing time may be due to the fact that both frameworks, Laravel and Symfony, use separate object-relational mapping – Eloquent ORM and Doctrine ORM respectively. The main difference between them is that Doctrine is entirely based on pure old PHP language, while Eloquent inherits all the ORM persistence logic.

The TPT metric (Figure 10) for the sample of 1000 posts in the case of Laravel was the worst for the Create operation - it clearly surpasses the other values on the chart.

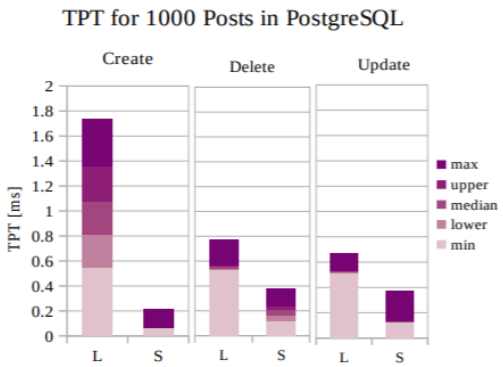

Figure 10: TPT metric for 1000 posts sample in PostgreSQL.

This is due to the fact that the query processing time was much higher than in other operations. A similar tendency was shown by Symfony - a significant part of TPT was spent on query processing, but the result is still more optimal (oscillates between 0.2 – 0.4ms).

In the case of TPT for a sample of 1000 comments (Figure 11), the situation for both patterns is relatively similar - both frameworks show similar values for each operation, but it is worth noting that in the case of Laravel this value is 2 times lower than in the case

of the sample of 1000 posts. There was an exemplary trend in Symfony - each operation requires almost 0.4ms of TPT. This is because Symfony has a higher SRT which automatically increases TPT.

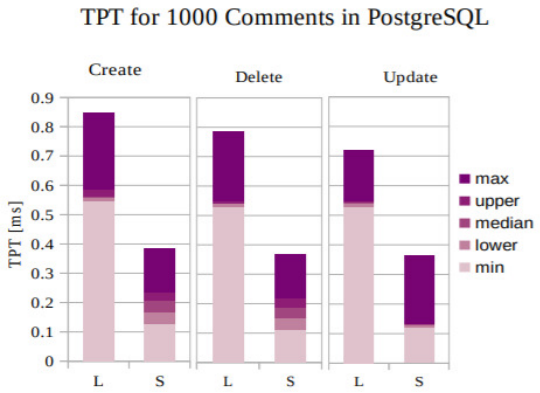

Figure 11: TPT metric for 1000 comments sample in PostgreSQL.

The Read operation for each framework and sample is shown in a separate graph (Figure 12). It clearly shows that the situation is very similar in both cases - both Laravel and Symfony show low SRT, and most TPT is sending and processing a query to the database and getting a response from it. However, it is worth paying attention to the time in which the above-mentioned operation is performed - for Laravel this time is 1.8ms for posts and 1.6ms for comments, while for Symfony - 0.2ms.

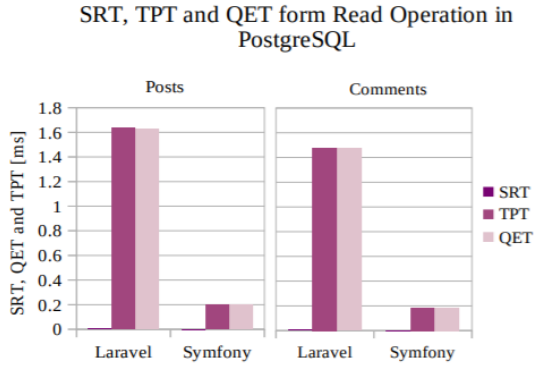

Figure 12: Dynamic metrics for Read operation in PostgreSQL.

The next stage of the analysis was the compilation of dynamic engines for the MySQL database. Figure 13 shows the SRT metric related to a sample of 1000 posts. The most overwhelming operation in this case was the Delete operation - for both Laravel and Symfony.

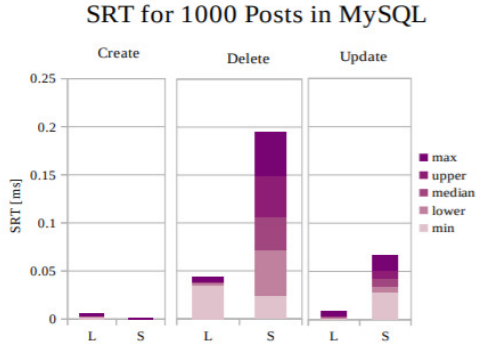

Figure 13: SRT metric for 1000 posts sample in MySQL.

It is worth mentioning that Symfony has a much higher SRT for Update and Delete than Laravel. This may be due to the fact that the size of the post, due to the number of characters, is an aggravating query for the database.

The situation is different for the SRT metric for a sample of 1000 comments (Figure 14).

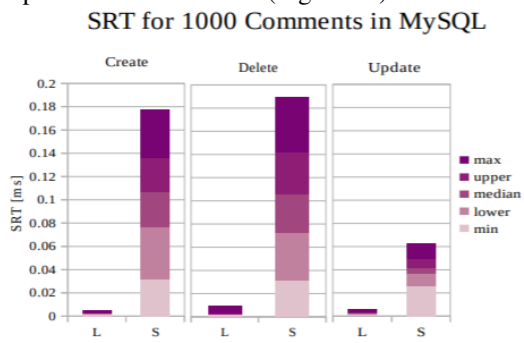

Figure 14: SRT metric for 1000 comments sample in MySQL.

While in the case of creating a post, SRT for both patterns was relatively low and high for the remaining operations, in this case the SRT is clearly (even 10 times) higher for Symfony than Laravel for each performed operation. The graph also clearly shows that the SRT for Laravel comments remains at a similar level of around 0.01ms.

In the case of the query processing time in the MySQL database for a sample of 1000 posts (Figure 15), Laravel did worse in the create and delete operation than Symfony - its execution time was significantly longer. Symfony, however, required much more time to update an existing post (almost 2ms), which is not only a few times higher result in the stocunt to Laravel, but also in relation to the create and delete operation in Symfony itself.

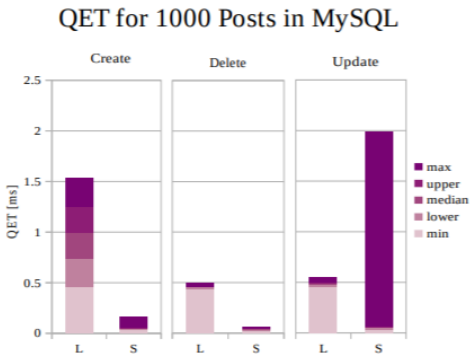

Figure 15: QET metric for 1000 posts sample in MySQL.

The situation looks different for QET metric for comments (Figure 16). In this case, all operations temporarily disadvantage Laravel - the query processing time, from sending it to obtaining a response from the database, is much higher than in the case of Symfony. Contrary to the sample of 10,000 posts, the delete operation was the most absorbing - probably because the comment is additionally burdened with a foreign key.

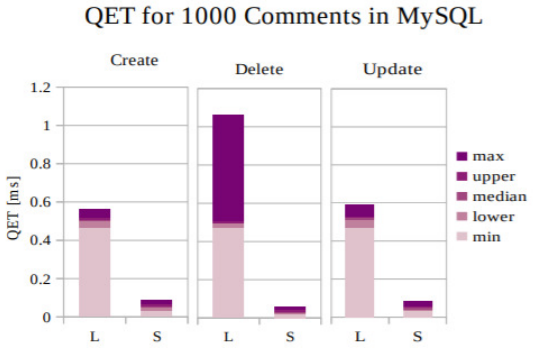

Figure 16: QET metric for 1000 comments sample in MySQL.

TPT for MySQL posts shown at Figure 17 - unlike PostgreSQL - does not unequivocally speak in favor of any of the analyzed patterns. It is true that the operation of adding a new post and removing it takes Laravel more time, but its update is several times lower than in the case of Symfony. This is due to the high QET for a Symfony post update - all this metric is the vast majority of the total processing time in each case.

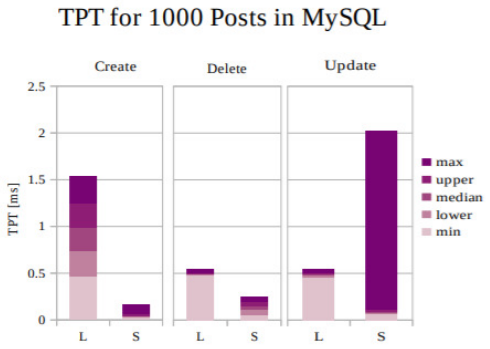

Figure 17: TPT metric for 1000 posts sample in MySQL.

The TPT for the sample of 1000 comments shown in Figure 18 clearly shows that in this case Symfony is again keeping the result at 0.2ms. The thing that stands out is the processing time for the delete operation in Laravel - it is significantly longer than for the other operations and the second pattern. This fact is due to the low SRT and high QET.

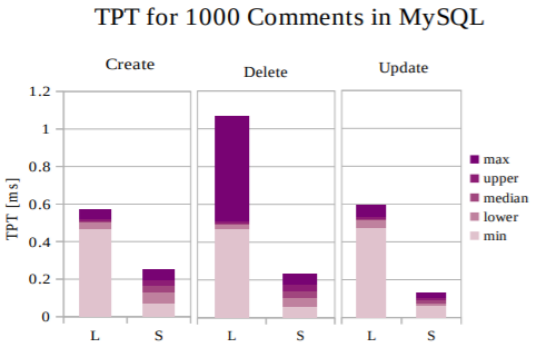

Figure 18: TPT metric for 1000 comments sample in MySQL.

Figure 19 shows the mean values of SRT, QET, and TPT for 1000 comments and 1000 posts read operations. As in the case of other operations, the vast majority of TPT is occupied by QET (approx. 3 ms for Laravel and approx. 0.5 ms for Symfony, respectively). SRT is almost 0 in both cases.

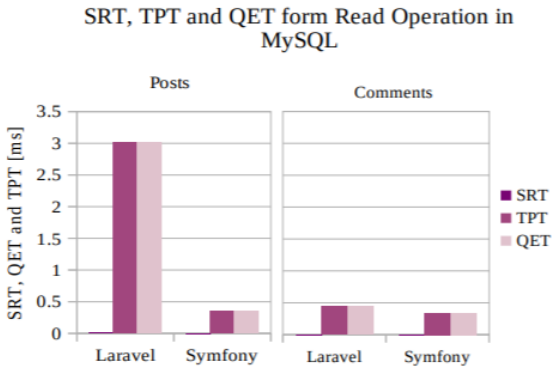

Figure 19: Dynamic metrics for Read operation in MySQL.

The collected values of static and dynamic metrics made it possible to compare the possibilities and quality of the code and the functioning of both analyzed design patterns itself. Dynamic metrics have been compiled in tables to compile the final results.

Table 2 presents dynamic metrics for a sample of 1000 PostgreSQL posts. It is clearly visible the performance of Symfony showing better communication with the database as well as better SRT for half of the CRUD operations.

Table 2: Dynamic metrics for 1000 posts sample in PostgreSQL.

	SRT	QET	TPT
Create	S	S	S
Read	S	S	S
Update	L	S	S
Delete	L	S	S

Table 3 shows a similar trend with an advantage for Symfony - Laravel is a clear favorite for the SRT metric, but Symfony performs better for the rest of the cases.

Table 3: Dynamic metrics for 1000 comments sample in PostgreSQL.

	SRT	QET	TPT
Create	L	S	S
Read	S	S	S
Update	L	S	S
Delete	L	S	S

Table 4 summarizes the dynamic capabilities of both frameworks for a sample of 1000 MySQL posts. In this case, Laravel did better not only for the SRT, but also for each metric for the Delete operation.

Table 4: Dynamic metrics for 1000 posts sample in MySQL.

	SRT	QET	TPT
Create	S	S	S
Read	S	S	S
Update	L	S	S
Delete	L	L	L

The last table (Table 5) shows the dynamic metrics for a sample of 1000 MySQL comments. The situation is the same as for the sample of 1000 PostgreSQL

comments - Laravel has better performance for the SRT metric, except for the Read operation.

Table 5: Dynamic metrics for 1000 comments sample in MySQL.

	SRT	QET	TPT
Create	L	S	S
Read	S	S	S
Update	L	S	S
Delete	L	S	S

While analyzing the above results, it is worth bearing in mind that the main part of the TPT metric is the database query processing time (QET).

5. Conclusion

The aim of the study was to determine which of the analyzed frameworks - Laravel or Symfony - is a better solution for the developer. Based on the collected results, it can be concluded that Laravel as a framework is definitely a less complex solution than Symfony. Its construction based on static metrics shows a lower complexity and volume in relation to the second analyzed model (the values of the metrics sometimes show a several times higher result in relation to the Symfony - Laravel ratio). However, it is worth remembering that Symfony uses only its own proprietary solutions, while Laravel uses Symfony components as a specific base and its backbone, which has a decisive impact on the volume of the code.

Performance was tested based on dynamic metrics. Due to this operation it was possible to test the performance of both frameworks. The analysis revealed that Laravel as a framework shows better results for the SRT metric - this means that compared to Symfony it delivers the first byte of response faster, regardless of the operation performed, but Symfony seems to be a pattern better connected with databases. The query processing time for most operations with samples of 1000 posts and 1000 comments in Symfony most often oscillated around 0.2ms, while in Laravel it was able to achieve the result of less than 2ms.

All CRUD operations showed a similar relationship - in each case, the greater part of TPT took up the processing time of the database query. For both frameworks, this operation was the most absorbing and took a long time to carry out from start to finish.

The above results allow to conclude that both analyzed frameworks are an efficient and optimal solution for a developer, but each of them has its own advantages and disadvantages. Laravel is a better solution for the processing of application content and due to the volume of the code, it allows for easier organization and implementation of changes, but Symfony is better connected with PostgreSQL and MySQL databases. The choice of the best framework should therefore depend on the individual needs of the developer, project assumptions, as well as the requirements set by the client. More extensive research is planned to explore this topic.

References

[1] A. Gdula, M. Plechawska-Wójcik, Porównanie wydajności wybranych technologii tworzenia usług sieciowych w aspekcie zastosowań w aplikacjach internetowych, Journal of Computer Sciences Institute 1 (2016) 14-19, https://doi.org/10.35784/jcsi.61.

[2] M. Jailia, A. Kumar, M. Agarwal, I. Sinha, Behavior of MVC (Model View Controller) based Web Application developed in PHP and .NET framework, International Conference on ICT in Business Industry & Government (ICTBIG) (2016) 1-5, https://doi.org/10.1109/ictbig.2016.7892651.

[3] U. Sa'adah, J. Akhmad, M. Hisyam, Implementing Singleton method in design of MVC-based PHP framework, International Electronics Symposium (IES) (2015) 212-217, https://doi.org/10.1109/elecsym.2015.7380843.

[4] J. Adamu, R. Hamzah, M. M. Rosli, Security issues and framework of electronic medical record: A review, Bulletin of Electrical Engineering and Informatics 9(2) (2020) 565-572, https://doi.org/10.11591/eei.v9i2.2064.

[5] R. F. Olanrewaju, T. Islam, N. A. Ali, An empirical study of the evolution of PHP MVC framework, Advanced Computer and Communication Engineering Technology (2015) 399-410, https://doi.org/10.1007/978-3-319-07674-4_40.

[6] X. Li, S. Karnan, J. A. Chishti, An empirical study of three PHP frameworks, 4th International Conference on Systems and Informatics (ICSAI) (2017) 1636-1640, https://doi.org/10.1109/icsai.2017.8248546.

[7] M. Laaziri, K. Benmoussa, S. Khoulji, K. M. Larbi, A. El Yamami, A comparative study of Laravel and Symfony PHP frameworks, International Journal of Electrical and Computer Engineering 9(1) (2019) 704-712, https://doi.org/10.11591/ijece.v9i1.pp704-712.

[8] S. Singh, J. Iyer, Comparative Study of MVC (Model View Controller) Architecture with respect to Struts Framework and PHP, International Journal of Computer Science Engineering 5(3) (2016) 142-150.

[9] X. K. Liu, G. G. Cheng, Analysis and Implementation of ASP. Net and PHP Frameworks Based on MVC Architecture, In Advanced Materials Research 798 (2013) 749-752, https://doi.org/10.4028/www.scientific.net/amr.798-799.749.

[10] M. R. Daraż, P. Kopniak, Analiza możliwości współpracy aplikacji mobilnych z usługami sieciowymi typu REST i Web Service, Journal of Computer Sciences Institute 11 (2019) 155-162, https://doi.org/10.35784/jcsi.181.