Code Quality Audit & Performance Evaluation of the Todolist Application

I. Introduction

A code quality and performance audit is a systematic and thorough evaluation of the source code of an application or software. Its main objective is to identify potential issues, improve code quality, and optimize performance. This meticulous review aims to ensure that the code adheres to development standards, is secure, maintainable, performant, and ready to meet the needs of the business.

II. Context

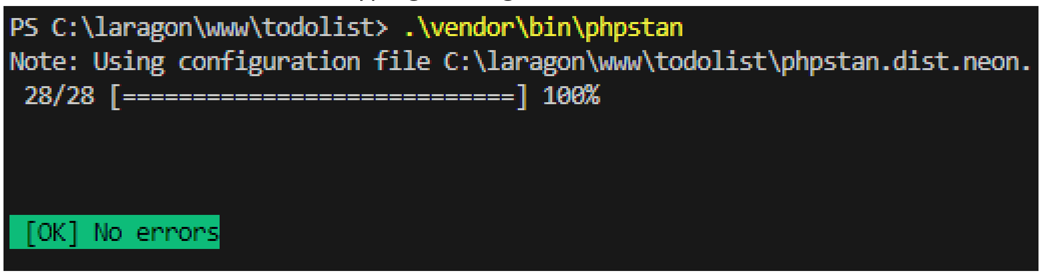
The company ToDo & Co developed an application in accelerated mode to demonstrate to potential investors that their concept was viable. The application was developed in PHP using the Symfony Framework.

Following a successful fundraising campaign, the company decided to continue developing the application and enlisted my services to further enhance it.

III. Code Quality

Code Validation Tool Usage

1. **PHP_Codesniffer** was used to validate that the code adheres to the standards of the PSR12 norm.
2. **PHPStan** was used to detect typing and logic issues in the code.

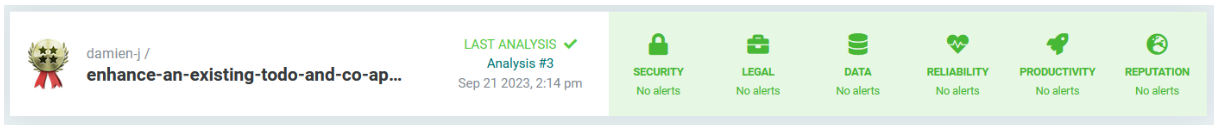


```
PS C:\laragon\www\todolist> .\vendor\bin\phpstan
Note: Using configuration file C:\laragon\www\todolist\phpstan.dist.neon.
 28/28 [============================] 100%



 [OK] No errors
```

3. **Symfony Insight** was used to detect potential security issues and ensure that Symfony's best practices were followed.



damien-j /
enhance-an-existing-todo-and-co-ap...

LAST ANALYSIS ✓
Analysis #3
Sep 21 2023, 2:14 pm

SECURITY
No alerts

LEGAL
No alerts

DATA
No alerts

RELIABILITY
No alerts

PRODUCTIVITY
No alerts

REPUTATION
No alerts

Symfony Coding Conventions

The following practices were followed to adhere to Symfony's recommendations:

1. **Class Naming :** Follows the "StudlyCaps" convention.
2. **File Naming :** Class files should be named using the same name as the class they contain.
3. **Indentation :** Uses 4 spaces for each level of indentation, without tabs.
4. **Annotations :** Symfony extensively uses annotations to configure routes, security, services, etc.
5. **Dependency Injection :** Symfony promotes dependency injection (DI) to manage dependencies between services.
6. **Configuration :** Configuration files in Symfony are typically in YAML, XML, or PHP formats.
7. **Functions :** Functions and methods should have descriptive names using the "camelCase" format.
8. **Unit Tests :** Unit tests are strongly encouraged in Symfony and are typically stored in the "tests" directory of your bundle.
9. **Route Naming :** Route names should be lowercase strings separated by underscores.

These coding conventions help maintain code consistency within the Symfony ecosystem and facilitate collaboration among developers.

Architecture and Project Structure

Symfony's architecture is based on the Model-View-Controller (MVC) pattern, which clearly separates the business logic from the presentation of the application.

Here is a detailed explanation of each component of the Symfony architecture:

1. **The Model :** In Symfony, the model represents the application's business logic. This includes data management, database interactions, validation operations, and more. Models are typically stored in the *src/Entity* directory for Doctrine entities, but you can also create custom model classes if needed.
2. **The View :** The view is responsible for displaying data to users. In Symfony, views are usually created using the Twig template engine. View files are stored in the *templates/* directory.
3. **The Controller :** Controllers act as intermediaries between the model and the view. They receive HTTP requests, perform necessary processing using the appropriate model, and then return the response to the view. Controllers are typically stored in the *src/Controller* directory.
4. **Routing :** Symfony uses a routing system to map URLs to controllers. Routes are configured in the *config/routes.yaml* file or as attributes within controllers. They define which controller should be called based on the requested URL.
5. **Services :** Symfony encourages the use of services to group reusable application logic. Services are shared objects that can be injected into controllers, models, and other classes. They are configured in the *config/services.yaml* file.
6. **Configuration *:*** Symfony's configuration is primarily located in the *config/* directory. This includes database configuration, parameter management, security, and more.
7. **The public/ Directory :** This is where publicly accessible files, such as images, CSS files, and JavaScript files, are stored. These files are accessible via the application's base URL.

8. **Symfony Console :** Symfony also provides a powerful command-line console for various development tasks, such as code generation, database management, and more.

Code Security

Code security in a Symfony application is a crucial element to ensure the protection of sensitive data and the prevention of vulnerabilities. Symfony provides powerful tools and best practices to enhance the security of your application :

1. **Authentication and Authorization :** Symfony offers a robust authentication system, with components like Guard, to manage access to resources based on user roles and permissions.
2. **Data Validation :** Validating incoming data is crucial to prevent code injection attacks. Using Symfony validation constraints is necessary to filter and validate user data.
3. **CSRF Attack Protection :** Symfony includes built-in mechanisms to prevent Cross-Site Request Forgery (CSRF) attacks, including the automatic generation of CSRF tokens.
4. **Output Escaping :** Using Twig automatically escapes HTML output, reducing the risk of malicious code injection.
5. **Session Security :** Symfony provides configuration options to secure sessions, such as managing secure and HttpOnly cookies.
6. **Protection Against SQL Injection :** Using Doctrine, Symfony's database manager, allows for prepared statements to prevent SQL injections.
7. **Updates and Patches :** Keep Symfony and its dependencies up to date to benefit from security patches.
8. **Security Auditing :** Regularly conduct security audits to identify potential vulnerabilities in your code and dependencies.
9. **Use of Security Bundles :** Symfony offers security bundles like SecurityBundle to simplify security configuration and management.
10. **Training and Awareness :** Train your team in secure development practices and raise awareness about security best practices.

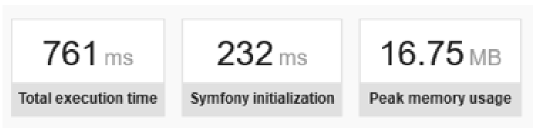
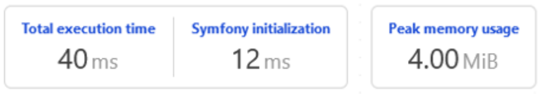
By following these best practices and utilizing the built-in security features of Symfony, it is possible to significantly strengthen the application's security and mitigate the risks associated with attacks and security vulnerabilities.

IV. Application Performance

The performance audit was conducted using Symfony's profiler, which is a powerful tool integrated into the Symfony framework for conducting performance audits. It collects detailed information about execution time, SQL queries, service calls, and more, enabling you to identify bottlenecks and optimize the application.

Performance Metrics

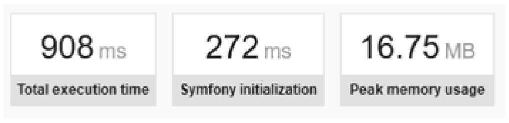
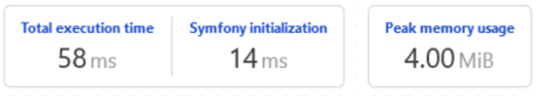
1. **Login (GET)**

Initial project	Final project
 761 ms Total execution time; 232 ms Symfony initialization; 16.75 MB Peak memory usage	 Total execution time 40 ms; Symfony initialization 12 ms; Peak memory usage 4.00 MiB

Improvement Rate

a. **Execution time : 94,74%**
b. **Memory : 76,12%**

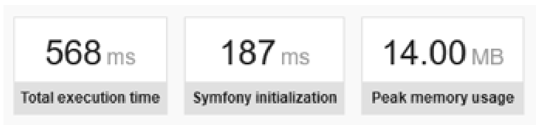
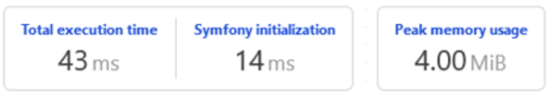
2. **Login (POST)**

Initial project	Final project
 908 ms Total execution time; 272 ms Symfony initialization; 16.75 MB Peak memory usage	 Total execution time 58 ms; Symfony initialization 14 ms; Peak memory usage 4.00 MiB

Improvement Rate

a. **Execution time : 93,61%**
b. **Memory : 76,12%**

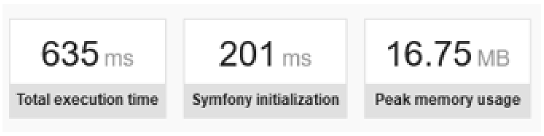
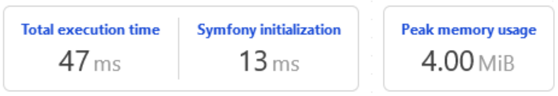
3. **Logout**

Initial project	Final project
 568 ms Total execution time; 187 ms Symfony initialization; 14.00 MB Peak memory usage	 Total execution time 43 ms; Symfony initialization 14 ms; Peak memory usage 4.00 MiB

Improvement Rate

a. **Execution time : 92,43%**
b. **Memory : 71,43%**

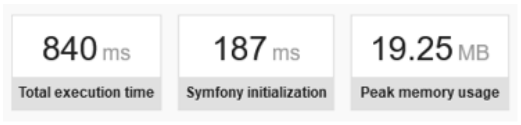
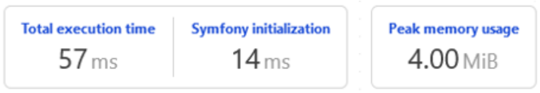
4. **Homepage**

Initial project	Final project
 635 ms Total execution time; 201 ms Symfony initialization; 16.75 MB Peak memory usage	 Total execution time 47 ms; Symfony initialization 13 ms; Peak memory usage 4.00 MiB

Improvement Rate

a. **Execution time : 92,60%**
b. **Memory : 76,12%**

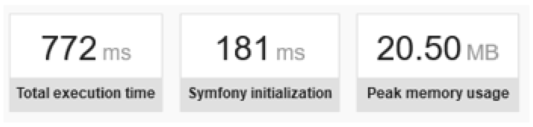
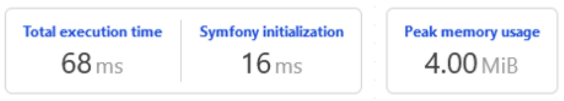
5. **Task list**

Initial project	Final project
 840 ms Total execution time; 187 ms Symfony initialization; 19.25 MB Peak memory usage	 Total execution time 57 ms; Symfony initialization 14 ms; Peak memory usage 4.00 MiB

Improvement Rate

a. **Execution time : 93,21%**
b. **Memory : 79,22%**

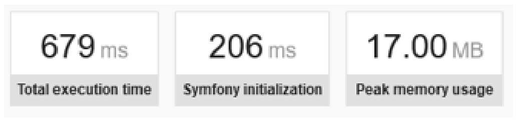
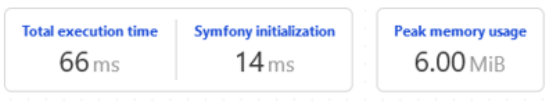
6. **Task create (GET)**

Initial project	Final project
 772 ms Total execution time 181 ms Symfony initialization 20.50 MB Peak memory usage	 Total execution time 68 ms Symfony initialization 16 ms Peak memory usage 4.00 MiB

Improvement Rate

a. **Execution time : 91,19%**
b. **Memory : 70,73%**

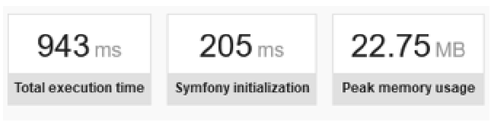
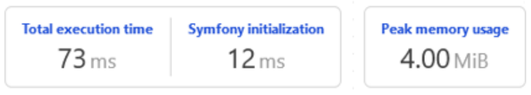
7. **Task create (POST)**

Initial project	Final project
 679 ms Total execution time 206 ms Symfony initialization 17.00 MB Peak memory usage	 Total execution time 66 ms Symfony initialization 14 ms Peak memory usage 6.00 MiB

Improvement Rate

a. **Execution time : 90,28%**
b. **Memory : 64,70%**

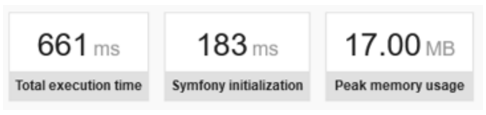
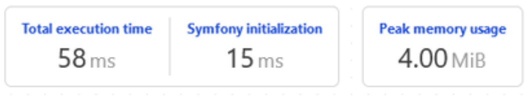
8. **Task edit (GET)**

Initial project	Final project
 943 ms Total execution time 205 ms Symfony initialization 22.75 MB Peak memory usage	 Total execution time 73 ms Symfony initialization 12 ms Peak memory usage 4.00 MiB

Improvement Rate

a. **Execution time : 92,26%**
b. **Memory : 82,42%**

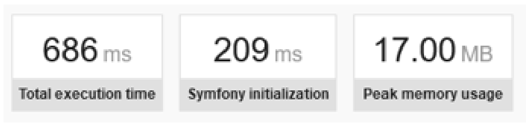
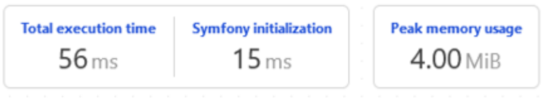
9. **Task edit (POST)**

Initial project	Final project
 661 ms Total execution time · 183 ms Symfony initialization · 17.00 MB Peak memory usage	 Total execution time 58 ms · Symfony initialization 15 ms · Peak memory usage 4.00 MiB

Improvement Rate

a. **Execution time : 91,22%**
b. **Memory : 76,47%**

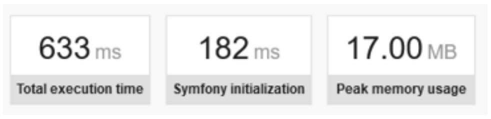
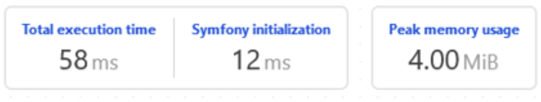
10. **Task delete**

Initial project	Final project
 686 ms Total execution time · 209 ms Symfony initialization · 17.00 MB Peak memory usage	 Total execution time 56 ms · Symfony initialization 15 ms · Peak memory usage 4.00 MiB

Improvement Rate

a. **Execution time : 91,84%**
b. **Memory : 76,47%**

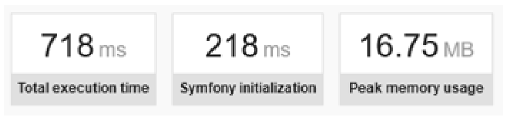
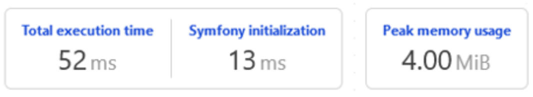
11. **Task toggle**

Initial project	Final project
 633 ms Total execution time · 182 ms Symfony initialization · 17.00 MB Peak memory usage	 Total execution time 58 ms · Symfony initialization 12 ms · Peak memory usage 4.00 MiB

Improvement Rate

a. **Execution time : 90,84%**
b. **Memory : 76,47%**

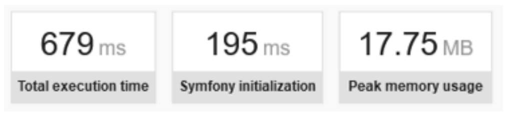
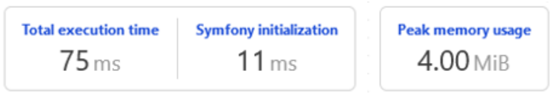
12. **User list**

Initial project	Final project
 718 ms Total execution time; 218 ms Symfony initialization; 16.75 MB Peak memory usage	 Total execution time 52 ms; Symfony initialization 13 ms; Peak memory usage 4.00 MiB

Improvement Rate

a. **Execution time : 92,76%**
b. **Memory : 76,12%**

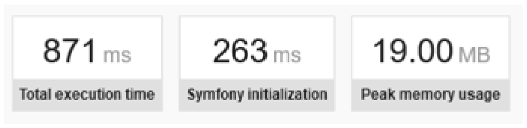
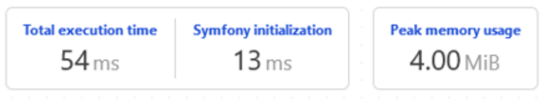
13. **User create (GET)**

Initial project	Final project
 679 ms Total execution time; 195 ms Symfony initialization; 17.75 MB Peak memory usage	 Total execution time 75 ms; Symfony initialization 11 ms; Peak memory usage 4.00 MiB

Improvement Rate

a. **Execution time : 88,95%**
b. **Memory : 77,46%**

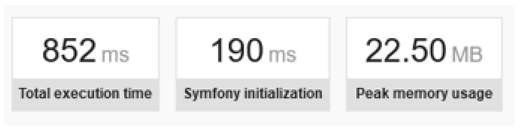
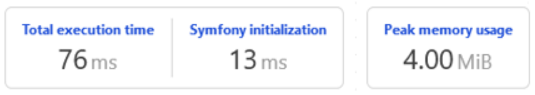
14. **User create (POST)**

Initial project	Final project
 871 ms Total execution time; 263 ms Symfony initialization; 19.00 MB Peak memory usage	 Total execution time 54 ms; Symfony initialization 13 ms; Peak memory usage 4.00 MiB

Improvement Rate

a. **Execution time : 93,80%**
b. **Memory : 78,95%**

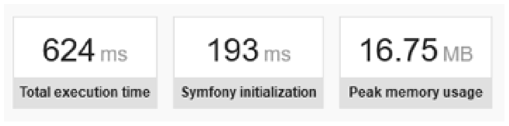
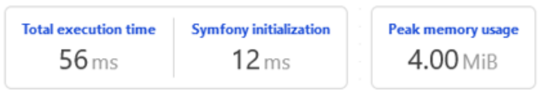
15. User edit (GET)

Initial project	Final project
 852 ms Total execution time; 190 ms Symfony initialization; 22.50 MB Peak memory usage	 Total execution time 76 ms; Symfony initialization 13 ms; Peak memory usage 4.00 MiB

Improvement Rate

a. **Execution time : 91,08%**
b. **Memory : 82,22%**

16. User edit (POST)

Initial project	Final project
 624 ms Total execution time; 193 ms Symfony initialization; 16.75 MB Peak memory usage	 Total execution time 56 ms; Symfony initialization 12 ms; Peak memory usage 4.00 MiB

Improvement Rate

a. **Execution time : 91,03%**
b. **Memory : 76,12%**

Results

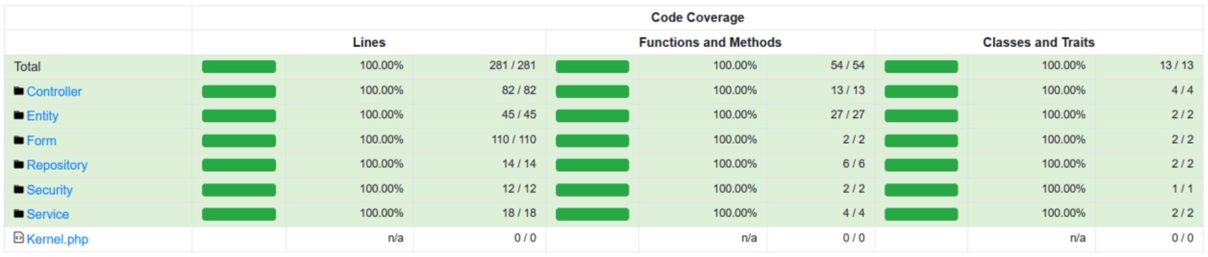
The various analyses show significant performance improvements, with an average reduction in execution time of over 90%, and even up to 94.74% for the login page. Additionally, there is an average reduction in memory usage of over 70%, reaching as high as 82.42% for task editing.

V. Unit and Functional Tests :

Test Coverage

The implementation of unit and functional tests in a Symfony application is essential to ensure its quality and robustness. Unit tests focus on verifying the proper functioning of individual components, such as classes and methods, by isolating each unit for independent testing. Functional tests, on the other hand, assess the behavior of the application as a whole by simulating real user interactions.

Thanks to PHPUnit and Symfony's WebTestCase component, I was able to create effective unit and functional tests with a test coverage rate of 100%.



	Code Coverage								
	Lines			Functions and Methods			Classes and Traits		
Total	🟩	100.00%	281 / 281	🟩	100.00%	54 / 54	🟩	100.00%	13 / 13
Controller	🟩	100.00%	82 / 82	🟩	100.00%	13 / 13	🟩	100.00%	4 / 4
Entity	🟩	100.00%	45 / 45	🟩	100.00%	27 / 27	🟩	100.00%	2 / 2
Form	🟩	100.00%	110 / 110	🟩	100.00%	2 / 2	🟩	100.00%	2 / 2
Repository	🟩	100.00%	14 / 14	🟩	100.00%	6 / 6	🟩	100.00%	2 / 2
Security	🟩	100.00%	12 / 12	🟩	100.00%	2 / 2	🟩	100.00%	1 / 1
Service	🟩	100.00%	18 / 18	🟩	100.00%	4 / 4	🟩	100.00%	2 / 2
Kernel.php		n/a	0 / 0		n/a	0 / 0		n/a	0 / 0

VI. Conclusion

This code and performance audit has been a crucial step in assessing the current state of the Symfony application. The results obtained are extremely positive, with notable strengths, including a significant improvement in response times and memory usage, as well as compliance with the standards set by the Symfony framework and PHP standards.

The remarkable improvement in response times and the reduction in memory usage testify to the effectiveness of the optimizations implemented. These improvements contribute to providing a better user experience while minimizing the operational costs of the application.

Adhering to coding standards imposed by PHP and Symfony demonstrates a commitment to best development practices, which enhances the stability and maintainability of the application in the long run. It also facilitates collaboration within the development team and ensures a better understanding of the code.

To sustain this positive momentum, we recommend the following actions :

1. **Continuous Updates of Symfony Framework and PHP :** It's essential to keep up with the current versions of the Symfony framework and PHP to benefit from the latest features, security fixes, and performance enhancements. Ensure that updates are planned and executed regularly.
2. **Performance Monitoring :** Continue to proactively monitor the application's performance using tools like the Symfony profiler. Identify potential bottlenecks and optimize them accordingly.
3. **Ongoing Team Training :** Ensure that members of the development team are continually trained in the latest Symfony and PHP development practices. This will ensure that the code remains in line with best practices.
4. **Automated Testing :** Keep investing in high-quality unit and functional tests to maintain a 100% test coverage rate. This will help quickly detect potential regressions during updates and code changes.
5. **Dependency Management :** Keep a close eye on the application's dependencies and ensure they are also kept up to date to avoid security and compatibility issues.
6. **Regular Code Review :** Encourage regular code reviews within the team to maintain high standards of code quality.

By following these recommendations, you can not only maintain the current quality and performance of the application but also ensure its continuous growth and evolution to meet the changing needs of the business and users.