

Crowd2Found

AUDIT

Symfony Application

[DRAFT]

Introduction

The Crowd2Fund web application audit (C2F) includes an evaluation of various aspects of the system. Its purpose is to provide information and recommendations on:

- security,
- performance,
- ease of maintenance,
- compliance with best practices.

The audit is divided into several main assessment steps:

1. Technology stack.
2. Code quality and structure.
3. Security.
4. Performance.
5. Technical documentation + code comments.
6. Testing.

Technology Stack

Production Environment

[No information available for now. I assumed that it is similar to the containers available in the repository. See Local Environment section]

Local Environment

The local environment uses the following technology stack:

HTTP Server:	Nginx 1.10.3
Database:	MariaDB 10.3.39
Backend:	PHP 7.4.3 Symfony Framework 4.4.33

HTTP Server

Release date: 31.01.2017.

Outdated server version. Contains many security vulnerabilities that have been successively fixed in newer versions.

For NGINX version 1.10.3, several security vulnerabilities have been identified. Here are some key issues:

1. CVE-2017-7529: This integer overflow vulnerability in the range filter module affects NGINX versions from 0.5.6 up to 1.13.2. It allows attackers to trigger a leak of potentially sensitive information by sending a specially crafted request. This issue has been patched in versions 1.13.3 and later ([Server Fault](#)) ([GitHub](#)).
2. CVE-2016-0742, CVE-2016-0746, and CVE-2016-0747: These vulnerabilities involve issues with the resolver, including a NULL pointer dereference and use-after-free during CNAME response processing. These were fixed in later versions, starting from 1.9.10 ([NGINX](#)).

For a description of potential threats, see: https://nginx.org/en/security_advisories.html

Recommendations

Update to the latest possible version. Perhaps it needs to be done in stages, in sync with PHP and database version updates.

Database

Release date: 10.05.2023.

Outdated database version. Due to its relatively recent update, it contains a few potential security issues. One of them is:

1. **CVE-2022-47015**: This is a Denial of Service (DoS) vulnerability that affects MariaDB Server versions before 10.3.34 up to 10.9.3. The vulnerability is due to a possible null pointer dereference in the `spider_db_mbase::print_warnings` function. This issue has been addressed in MariaDB 10.3.39 ([MariaDB](#)) ([Tenable®](#)).

Recommendations

Update to the latest possible version. Perhaps it needs to be done in stages, in sync with PHP updates. Recommended upgrade to the current LTS version.

For a complete list of changes for each version, visit <https://mariadb.com/kb/en/release-notes/>.

PHP

Release date: 09.06.2022.

Outdated PHP version. Active and security support ended several years ago. High number of potential threats. Low efficiency.

Recommendations

Update to the latest possible version. The priority is to upgrade to either PHP 8.2 or 8.3.

Symfony

Released on: November 2019.

End of support: November 2023.

Outdated Symfony version. Active and security support ended several years ago. High number of potential threats. Lower efficiency.

Recommendations

The priority is to upgrade to Symfony 5, which is supported for several more months in terms of security patches. Up next to version 6.4 which is the LTS version

Code quality and structure

Symfony Best Practices (SBP)

Symfony Best Practices is a collection of tips and recommendations for working with Symfony-based applications. While these recommendations are a valuable guide for developers, they do not always have to be applied directly. It is important to consider the specific assumptions and purpose of the project. Adapting these practices to the specific needs and context of the application is key to achieving optimal results.

Directory structure

SBP suggests the following directory structure in the main branch of the project:

```
project_root/  
├─ assets/  
├─ bin/  
├─ config/  
├─ migrations/  
├─ public/  
├─ src/  
├─ templates/  
├─ tests/  
├─ translations/  
├─ var/  
└─ vendor/
```

Where the main place for the PHP code of the application is the *src/* folder.

SBP also recommends maintaining certain structures inside these folders, but in reality, the structure depends on how the application itself was designed. The main factors that determine the structure of the folders are what architecture and patterns were used to write the application.

Project

In the C2F application, the recommended structure has been partially preserved. The code placed in the ``src`` folder is shuffled. On one level, we can find standard PHP classes and whole modules (bundles). Such inconsistency leads to clutter and difficult code identification.

Recommendations

From the observation made, it appears that many programmers worked on the project. Each followed his own practice in dividing the code. It is also not insignificant that the project evolved all the way from Symfony 2, where the structure of the code differed significantly. Code refactoring is recommended. Depending on the complexity, it can be done modularly "all at once", or when fixing bugs or implementing new features

Architecture

Description

The right choice of architecture at the very beginning of a project is crucial to its success. The proper architecture provides a solid foundation that allows for easy extension, maintenance and scalability of the application. With a well-planned architecture, many problems can be avoided in later stages of development, such as:

1. Poor performance: A suboptimal architecture can lead to performance problems that can be difficult and costly to resolve after implementation.
2. Maintenance difficulties: Complex and chaotic code structures can significantly hamper maintenance work, leading to longer response times to errors and higher maintenance costs.
3. Lack of scalability: Applications with poorly chosen architectures can encounter difficulties in scaling, which can limit their ability to handle increasing numbers of users or data.
4. Integration difficulties: An ill-conceived architecture can lead to integration problems with other systems, limiting the flexibility and development possibilities of the application.

Regardless of the type of architecture chosen, a project such as C2F should have a clearly demarcated domain layer, i.e. one in which all the logic of the business assumptions is executed. Due to the complexity of the processes - i.e.: registration, acceptance, and investment validation processes - this is crucial, as it gives a broad picture of what is expected from the application.

In addition, an important factor for code quality is the Single Responsibility Principle. This principle implies that each class should only be responsible for one aspect of functionality. This makes the code clearly divided and understandable to the developer, reducing the need for additional technical documentation (the code describes itself).

Project

The project was created based on the standard Symfony schema. This framework uses several patterns such as MVC (Model-View-Controller) and SOA (Service Oriented Architecture). By design, these patterns are intended for smaller projects, but with proper care for the division of responsibility (Single Responsibility Principle), they can work well for larger projects.

The C2F project lacks separation of the domain layer (representing business assumptions). The Single Responsibility Principle is thoroughly ignored. Controllers contain a lot of application logic, repositories are overly bloated, and services execute many unrelated processes.

The database was configured by Doctrine ORM mostly. The configuration is inconsistent, e.g. table names are once written in came case once in snake case.

Recommendations

The threshold of entry into a project for new developers is very high. The problem is not the technology, but the way the application is written. The mix of dependencies, and the lack of a single responsibility, make the code unreadable and incomprehensible.

Missing the domain layer leads to a lack of understanding of the application's assumptions.

When working on a project, care should be taken to refactor the code in the area affected. Do not add additional logic, conditions, or workarounds to existing code, as this will lead to an even greater problem in understanding the code.

Refactoring should start to carve out the code for a single responsibility and, in the long run, to carve out a domain layer.

Code for new functionality should be written to the correct standards from the start.

Code standards

Description

The use of code standards in PHP is essential to maintain high quality, readability, and consistency of code. Code standards, such as PSR (PHP Standard Recommendations), serve several key functions:

1. **Readability:** code standards make code easier to read and understand. With uniform formatting, developers can quickly find their way around code written by other team members.
2. **Consistency:** Using uniform coding rules leads to consistency throughout the project. This facilitates teamwork as all developers follow the same rules and conventions.
3. **Ease of maintenance:** Code that is written according to standards is easier to maintain and develop. New team members can understand existing code more quickly, which accelerates the implementation process.
4. **Avoiding errors:** Code standards often include rules for writing safe and efficient code to help avoid common programming errors and security vulnerabilities.
5. **Automation:** The use of code standards makes it easier to implement automatic code analysis and formatting tools such as PHP_CodeSniffer or PHP-CS-Fixer. Automated tools can quickly detect and fix deviations from established standards.
6. **Professionalism:** The use of code standards demonstrates the professionalism of the development team. Clients and co-workers can have more confidence in the quality and reliability of the code.

Introducing and enforcing code standards in PHP is a key part of the process of developing and maintaining high-quality software, providing a long-term benefit for both the development team and the final product.

Project

Using the Symfony framework imposes a certain programming style up front. This coincides with the principles of PSR. However, there are many inconsistencies in the coding style of the project. A detailed analysis is provided by the code quality assessment tools (Quality Tools).

It should be noted that some quality tools have been configured in the project. However, it is difficult to know if they are used and how (possibly only for automatic code correction).

Recommendations

PHPStan

PHPStan is a static analysis tool for PHP code, which is used to detect errors and potential problems in code without having to run it. Its main goal is to improve code quality and reliability through early error detection.

Used configuration: `./quality_tools/phpstan.dist.neon`

Report: `./reports/phpstan_report.txt`

PHPStan indicated 7448 errors. The analysis detail was set to level 6 (not max).

Some of the errors identified by this tool are not very significant. They are due to the age of the project and the fact that it was probably initially written in PHP 5. Instead, it is a very useful indication of what needs to be changed to move to PHP 8.

Particularly noteworthy are places where variables are operated on that may not exist or be of a different type than assumed. This also applies to arrays where there are direct references to keys, without first checking whether such a key even exists. Such places are a potential source of application runtime errors, resulting in an error 500 status being returned.

PHP Coding Standards Fixer

PHP Coding Standards Fixer (PHP-CS-Fixer) is a tool that is used to automatically format and fix PHP code according to specific coding standards. It is an extremely useful tool for developers and development teams who want to maintain code consistency and readability in their projects.

Used configuration: `./quality_tools/.php-cs-fixer.dist.php`

Report: `./reports/php-cs-fixer_report.txt`

The tool was configured to check for PSR and Symfony standards. The report identified 853 files to be repaired.

PHP-CS-Fixer can fix the errors found on its own, but with such many files, manual change control is recommended.

PHPMD (PHP Mess Detector)

PHPMD (PHP Mess Detector) is a static analysis tool for PHP code that helps detect potential problems, bugs and suboptimal code fragments. Its main goal is to improve code quality and make it easier to maintain by identifying places that may need refactoring.

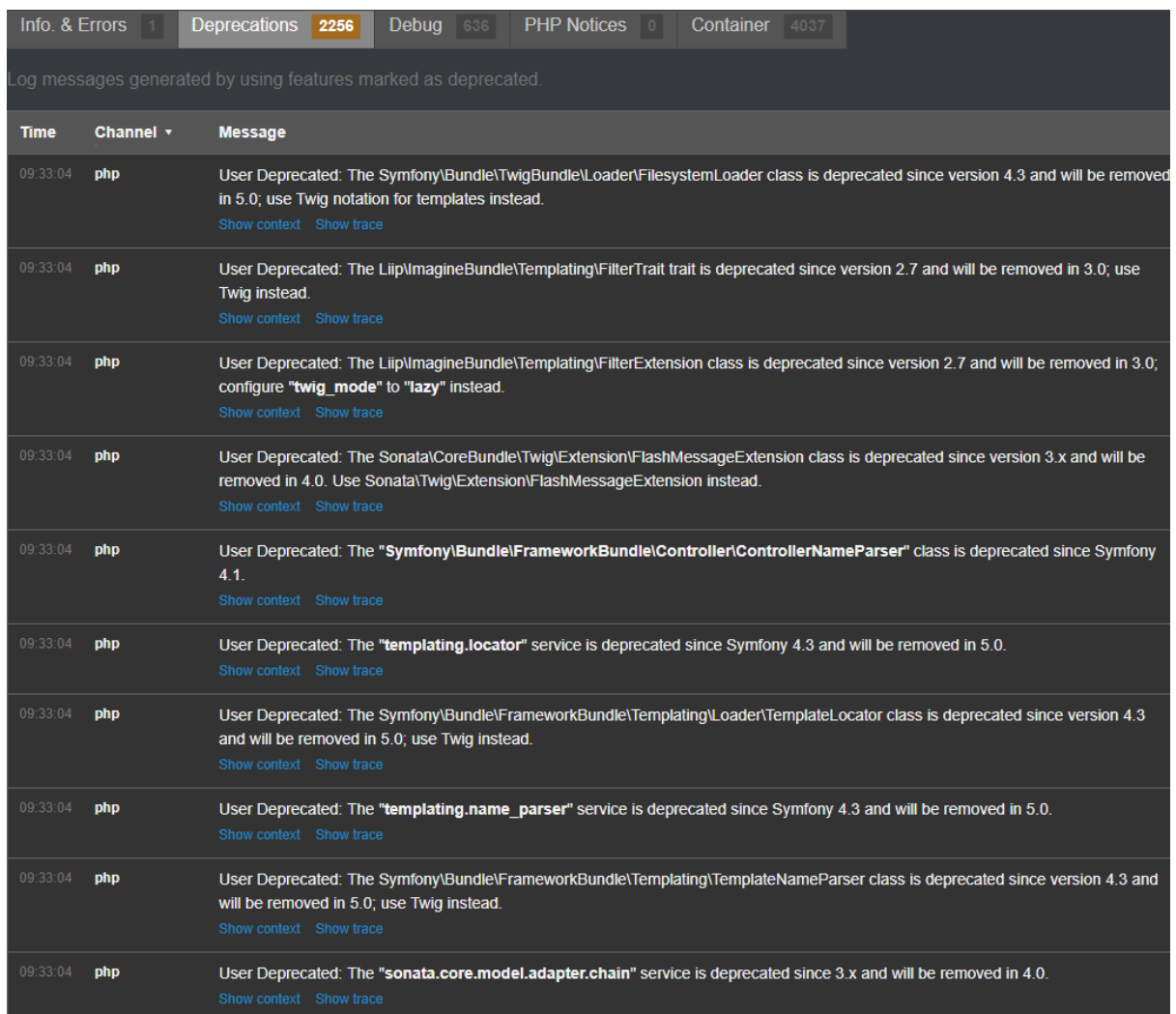
Used configuration: ./quality_tools/.php-cs-fixer.dist.php
Report: ./reports/phpmd_src-report.txt

In total, the report identified more than 6700 bugs. It is recommended to focus on errors of the CyclomaticComplexity type, meaning too much code complexity in a single functionality.

Symfony Profiler

Symfony Profiler is an advanced debugging and monitoring tool for Symfony applications that provides detailed information about the performance of the application in real time. Its main purpose is to help developers diagnose problems and optimise application performance.

The tool points out deficiencies in translation files and the use of code marked as deprecated (obsolete).



The screenshot shows the 'Deprecations' tab in the Symfony Profiler, displaying a list of 10 deprecated classes and services. The tab is highlighted in orange and shows a count of 2256 deprecations. Below the tab, there is a table with columns for Time, Channel, and Message. Each message includes a deprecation notice and links to 'Show context' and 'Show trace'.

Time	Channel	Message
09:33:04	php	User Deprecated: The <code>Symfony\Bundle\TwigBundle\Loader\FilesystemLoader</code> class is deprecated since version 4.3 and will be removed in 5.0; use Twig notation for templates instead. Show context Show trace
09:33:04	php	User Deprecated: The <code>Lip\ImagineBundle\Templating\FilterTrait</code> trait is deprecated since version 2.7 and will be removed in 3.0; use Twig instead. Show context Show trace
09:33:04	php	User Deprecated: The <code>Lip\ImagineBundle\Templating\FilterExtension</code> class is deprecated since version 2.7 and will be removed in 3.0; configure <code>"twig_mode"</code> to <code>"lazy"</code> instead. Show context Show trace
09:33:04	php	User Deprecated: The <code>Sonata\CoreBundle\Twig\Extension\FlashMessageExtension</code> class is deprecated since version 3.x and will be removed in 4.0. Use <code>Sonata\Twig\Extension\FlashMessageExtension</code> instead. Show context Show trace
09:33:04	php	User Deprecated: The <code>"Symfony\Bundle\FrameworkBundle\Controller\ControllerNameParser"</code> class is deprecated since Symfony 4.1. Show context Show trace
09:33:04	php	User Deprecated: The <code>"templating_locator"</code> service is deprecated since Symfony 4.3 and will be removed in 5.0. Show context Show trace
09:33:04	php	User Deprecated: The <code>Symfony\Bundle\FrameworkBundle\Templating\Loader\TemplateLocator</code> class is deprecated since version 4.3 and will be removed in 5.0; use Twig instead. Show context Show trace
09:33:04	php	User Deprecated: The <code>"templating.name_parser"</code> service is deprecated since Symfony 4.3 and will be removed in 5.0. Show context Show trace
09:33:04	php	User Deprecated: The <code>Symfony\Bundle\FrameworkBundle\Templating\TemplateNameParser</code> class is deprecated since version 4.3 and will be removed in 5.0; use Twig instead. Show context Show trace
09:33:04	php	User Deprecated: The <code>"sonata.core.model.adapter.chain"</code> service is deprecated since 3.x and will be removed in 4.0. Show context Show trace

Security audit

Package dependencies

Package dependencies in the context of PHP applications refer to the external libraries, frameworks and tools that are necessary for the application to run. Managing these dependencies is a key aspect of developing and maintaining modern PHP applications.

The security report performed with the Composer Audit tool identified 14 security issues.

Some of these are closely related to an outdated version of Symfony. Other problems relate to the libraries responsible for communicating with AWS and for sending HTTP requests (Guzzle).

Report: `./reports/composer_audit-report.txt`

The same problems are indicated by the Symfony Security Checker tool.

Recommendations

The first step is to update the `aws/aws-sdk-php` and `guzzlehttp/guzzle` libraries to the latest possible version. However, these will not be the latest versions available, stripped of all fixes, as these packages are also dependent on the version of PHP used in the project.

Start upgrading Symfony to at least version 5 to get rid of security issues related to Symfony and Twig libraries.

Upgrade to PHP version 8.

Target stack: PHP 8.3, Symfony 6.4

SQL Injection

SQL Injection is one of the most common and dangerous forms of attacks on web applications, which involves injecting malicious SQL code into database queries. This attack exploits application security vulnerabilities that improperly process user input.

Project

Queries to the database are performed using Doctrine's ORM. There are places where queries are written manually, but always with the use of parameters. This means that all values are properly protected by the Doctrine library.

No sites were found that were potentially vulnerable to SQL Injection.

Forms protection (CSRF)

Cross-Site Request Forgery (CSRF) protection is an important aspect of web application security, designed to prevent attacks in which an attacker prompts a user to perform unwanted actions on a website to which they are logged in.

Project

This project contains many forms, some of which have CSRF protection enabled, some of which do not.

There are, for example, such places, with a commentary, but it is not clear from this why this protection has been excluded:

```
// todo - find out wtf is going on here  
'csrf_protection' => false,
```

Recommendations

The use of all forms supported by Symfony should be analysed. It is likely that CSRF protection has been disabled, due to the forms' handling of AJAX requests.

If possible, adapt the code to support forms with CSRF protection enabled.

Users Authentication and Authorization

The site has two access ports for users.

1. Front - a place intended for investors and business.
2. Back/Admin Panel - a place intended for system administrators.

Access to both ports is restricted, the user should be logged in to use the functionality. Access to both ports is managed through a separate firewall.

An analysis of the firewall code and settings shows that each requested URL is properly handled by the firewall and requires authentication to open it (except for the login page).

Each user and administrator is assigned appropriate roles that block or allow access to specific system functionality. The correctness of role assignment and handling cannot be verified due to insufficient knowledge of the design assumptions.

Business/Investor users are mixed with Admin users in the database. There is no separate context.

Recommendations

It is good practice to separate the context of regular users from admin users. For refactorings, create separate data sources and separate access firewalls.

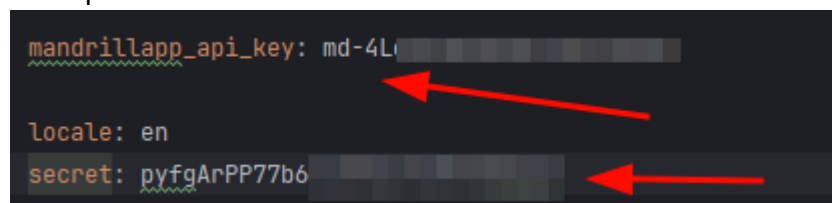
Sensitive data

There is a lot of information in the project that appears to be sensitive. This is particularly true of the various secret keys that are officially available for reading in the repository. This is particularly dangerous where these keys are responsible for the operation of the PROD application.

Any sensitive data used in the PROD version should not be made visible in the repository. This can lead to undesirable, dangerous situations and badly affect the final perception by the regular user.

Examples:

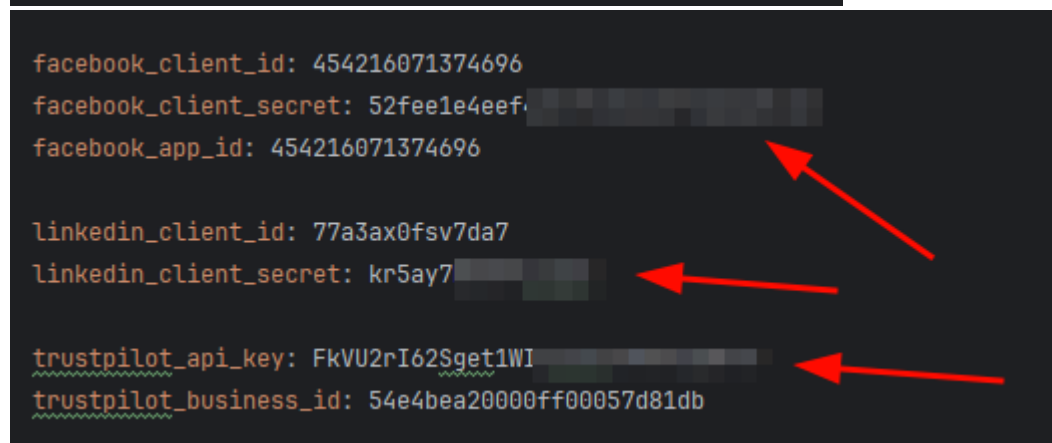
```
mandrillapp_api_key: md-4L[REDACTED]
locale: en
secret: pyfgArPP77b6[REDACTED]
```

A screenshot of a terminal window showing environment variables. The first line is 'mandrillapp_api_key: md-4L[REDACTED]' with a red arrow pointing to the key name. The second line is 'locale: en'. The third line is 'secret: pyfgArPP77b6[REDACTED]' with a red arrow pointing to the key name.

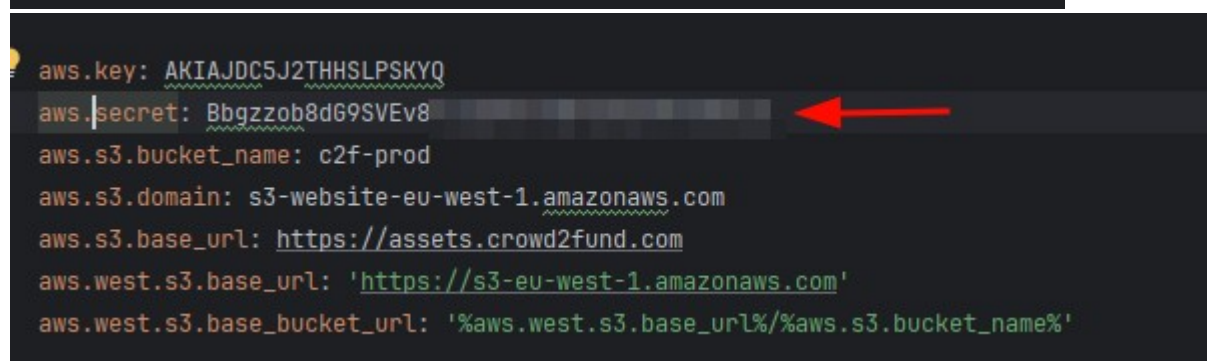
```
facebook_client_id: 454216071374696
facebook_client_secret: 52fee1e4eef[REDACTED]
facebook_app_id: 454216071374696

linkedin_client_id: 77a3ax0fsv7da7
linkedin_client_secret: kr5ay7[REDACTED]

trustpilot_api_key: FkVU2rI62Sget1WI[REDACTED]
trustpilot_business_id: 54e4bea20000ff00057d81db
```

A screenshot of a terminal window showing environment variables for social media and Trustpilot. The first three lines are for Facebook: 'facebook_client_id: 454216071374696', 'facebook_client_secret: 52fee1e4eef[REDACTED]', and 'facebook_app_id: 454216071374696'. The next three lines are for LinkedIn: 'linkedin_client_id: 77a3ax0fsv7da7', 'linkedin_client_secret: kr5ay7[REDACTED]'. The last two lines are for Trustpilot: 'trustpilot_api_key: FkVU2rI62Sget1WI[REDACTED]' and 'trustpilot_business_id: 54e4bea20000ff00057d81db'. Red arrows point to the key names for Facebook, LinkedIn, and Trustpilot.

```
aws.key: AKIAJDC5J2THHSLPSKYQ
aws.secret: Bbgzzob8d69SVEv8[REDACTED]
aws.s3.bucket_name: c2f-prod
aws.s3.domain: s3-website-eu-west-1.amazonaws.com
aws.s3.base_url: https://assets.crowd2fund.com
aws.west.s3.base_url: 'https://s3-eu-west-1.amazonaws.com'
aws.west.s3.base_bucket_url: '%aws.west.s3.base_url%/%aws.s3.bucket_name%'
```

A screenshot of a terminal window showing environment variables for AWS. The first line is 'aws.key: AKIAJDC5J2THHSLPSKYQ'. The second line is 'aws.secret: Bbgzzob8d69SVEv8[REDACTED]' with a red arrow pointing to the key name. The remaining lines are 'aws.s3.bucket_name: c2f-prod', 'aws.s3.domain: s3-website-eu-west-1.amazonaws.com', 'aws.s3.base_url: https://assets.crowd2fund.com', 'aws.west.s3.base_url: 'https://s3-eu-west-1.amazonaws.com'', and 'aws.west.s3.base_bucket_url: '%aws.west.s3.base_url%/%aws.s3.bucket_name%'.

Recommendations

Check that the publicly available keys are indeed from the production environment. If so, generate new keys and remove them from the repository.

This data should be placed on the production server directly (preferably using some Vault, encoding sensitive data)

Performance audit

Profiling

Profiling was performed while using the system manually, combined with observation of the reports provided by the Symfony Profiler.

Pages do not show excessive performance drops during operation. Database queries are performing optimally.

The repositories contain SQL queries that perform searches on columns without indexes.

Recommendations

Configure and perform stress test and performance tests on the pages most prioritised by the regular user.

If the amount of data is increased, care should be taken to ensure that there are appropriate indexes on the tables in the database.

Cache System

The application uses standard caching solutions provided by the Symfony Framework, Doctrine and external bundles.

Recommendations

None.

Technical Documentation

The project was not accompanied by technical documentation.

The code has no comments at the points of complex procedures, except for single vague notations.

Recommendations

During the work on the project, complete comments in places that perform difficult and complex operations.

New code should be properly documented.

Tests

Writing tests for code is a key part of the professional software development process. Tests ensure that the application works as intended, facilitate the early detection of bugs, and support the maintenance and development of the code.

Properly written tests are very helpful for developers starting work with a project, as they clarify how the application works.

Project

Initially, tests were written using PHPSpec and PHPUnit. Unit, integration and functional/application tests can be found. The PHPSpec and PHPUnit libraries were used to write the tests.

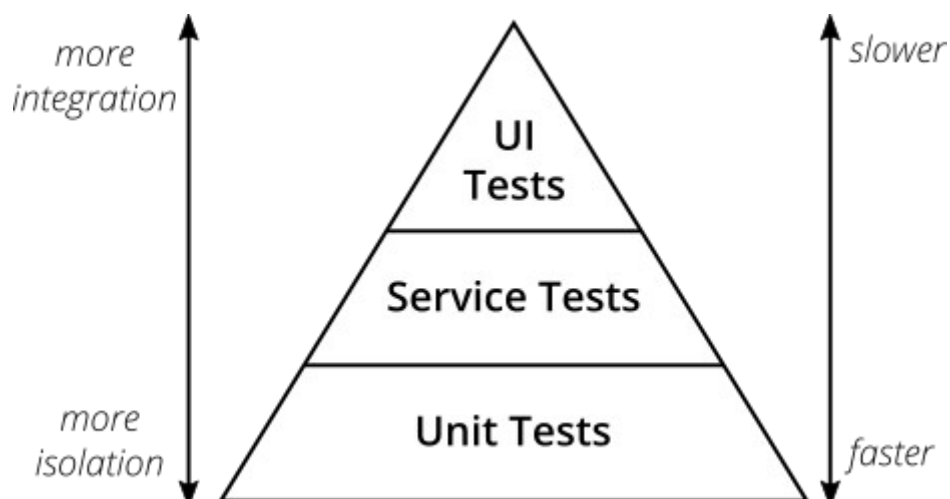
However, according to the commit history, no new tests were written, or existing tests were modified for several years. Furthermore, there are no libraries in the project's package dependencies to run these tests.

It can be assumed that the current tests are not up-to-date and do not cover the current project assumptions.

Recommendations

When working on an application, care should be taken to test the code, both new and modified, as part of the task.

Ultimately, the Test Pyramid should be used:



Vendor overview

Identification of unused libraries

No use found for the library:

- `incenteev/composer-parameter-handler`

Identification of library versions

All libraries used are not updated to the latest possible version for the project.

Recommendations

The first step should be to update the libraries to the latest possible version for the current state of the project. This can be read from the `composer.lock` file, which has older versions blocked.

The next steps should be to update Symfony to version 5.4 and then 6.4.

Deploys

TODO

Symfony Framework

TODO