iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 12 **Issue:** VI **Month of publication:** June 2024

DOI: https://doi.org/10.22214/ijraset.2024.63417

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 12 Issue VI June 2024- Available at www.ijraset.com

PHP Laravel - A Focus on Customization and Schedule Job Management

Akhil Duggirala
Sails Software Solutions, RGUKT - IIIT Srikakulam, India

Abstract: ***The PHP-based web framework Laravel is a prominent option for developers who want to create safe and effective online apps. Its rich feature set reduces the need for substantial planning from the ground up and saves developers much time. Using Laravel, developers may easily combine different customized packages for their projects, which expedites the coding process and improves the app's logical structure. Laravel plays a crucial role in the need for secure web development. Because of its robust security features, Laravel stands out as the most prominent framework. The schedule jobs library is updated by the author highlighting the framework's adaptability and flexibility for the project requirements.***
Keywords: Laravel, Spatie, Laravel Schedule Monitor, Artisan, Github

I. INTRODUCTION

In today's fast-paced digital environment, the efficient management and tracking of planned tasks within web applications is critical for guaranteeing peak performance and dependability. Laravel a renowned PHP-based web framework has been developed continuously to satisfy developers who want to build safe and robust effective web applications. The vast package ecosystem plays a significant role in Laravel's adaptability over the world which improves many facets of application development. One of the packages among them is "laravel-schedule-monitor" which is especially useful for detailed information about the jobs and commands. The integration of Laravel's schedule monitor enables the developers to track the time taken for the execution of the job, spotting the issues and performance reviews when the command runs. Laravel Schedule Monitor library permits synchronization with other services such as Oh Dear. The connection between them promptly notifies developers when the event of a task fails or is delayed by any means.

II. BACKGROUND OF STUDY

Regularly reviewing and evaluating the tasks to ensure they are completed as planned and to promptly address any deviations or issues is the responsibility of monitoring scheduled tasks. Its ongoing process helps maintain the promptness, reliability, and effectiveness of a project or organization. GitHub has the code that was utilized for the analysis. Most developers can save a great deal of time when looking into scheduled command failures by using the study's prediction of error logs when a command fails.

III. METHODOLOGY

A. Workflow of Schedule Commands

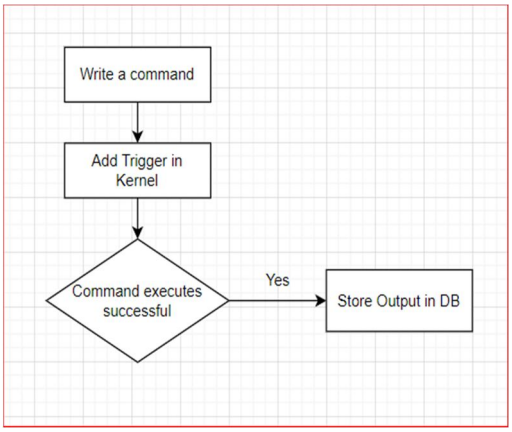


Figure 1: Workflow of schedule commands

©IJRASET: All Rights are Reserved

IV. PROPOSED SYSTEM

The package makes it simple to monitor, identify issues, and analyze performance by offering a dependable way to track and document the completion of planned tasks in Laravel applications. We present a feature in our suggested system that allows the failure causes of scheduled operations inside Laravel apps to be stored in a database. The improvement helps the developers effectively monitor the task and assists in investigating the failure reasons for task failures leading to easy problem-solving and proactive application performance monitoring. By systematically registering the failure causes, developers can obtain maximum insights and knowledge about the issues occurrences and make the necessary adjustments or modifications on the jobs which will eventually improve the application efficiency and reliability. The failure data that has been stored can also be used for additional analysis and further optimization technical improvements.

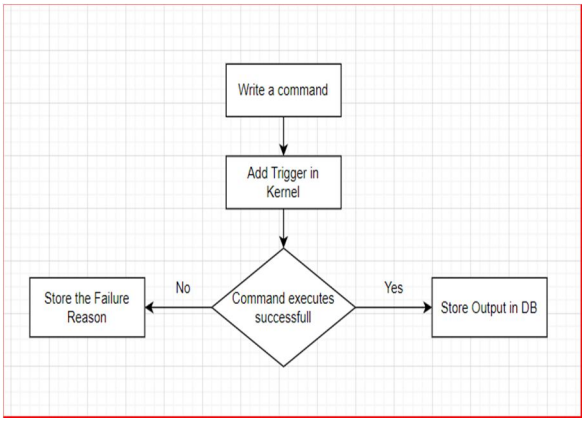


Figure 2: Proposed model for workflow of schedule commands

V. IMPLEMENTATION

A. *Library Installation*

You can install the package via composer [2]:

Installation

```
composer require spatie/laravel-schedule-monitor
```

Figure 3: Install the package

The command "composer require spatie/laravel-schedule-monitor" adds the "spatie/laravel-schedule-monitor" package as a dependency in a Laravel project. The Spatie helps in organizing and monitoring the scheduled tasks and enhances Laravel's capabilities also helps in obtaining insightful information on how the commands are being carried out such as storing the status of the job and monitoring the task.

B. *Database Preparation*

You must publish and run migrations:

Publishing

```
php artisan vendor:publish --
provider="Spatie\ScheduleMonitor\ScheduleMonitorServiceProvider" --
tag="schedule-monitor-migrations"

php artisan migrate
```

Figure 4: Publish the package and migrate

The command, "php artisan vendor: publish --provider="spatie\ScheduleMonitor\ScheduleMonitorServiceProvider" --tag="schedule-monitor-migrations" is used to publish the migrations provided by the “spatie/laravel-schedule-monitor”. The migrations create the tables in the database schema for the package functionality to store the execution logs..
The command “php-artisan migrate” is used to perform the migrations after the publishing. This will create relevant database tables in the database schema in the application. This connection enables the controlling of holding the appropriate data about scheduled jobs and their monitoring by the ‘spate/laravel-schedule-monitor’.

C. Schedule Command Creation

1. *Create a New Command [3]*:
First, create a new artisan command using the following command:

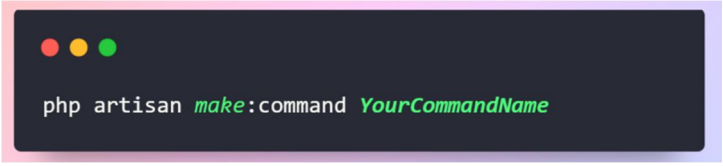


Figure 5: Command to create an artisan command.

2. Define the Schedule Logic:
Open the generated command file located at app/Console/Commands/. Define the logic of your command in the handle() function which will be executed when the scheduled command runs.

3. Usage
You need first run schedule-monitor: sync to keep an eye on your schedule. This command will examine your schedule and add a record to the monitored_scheduled_tasks table for every task.

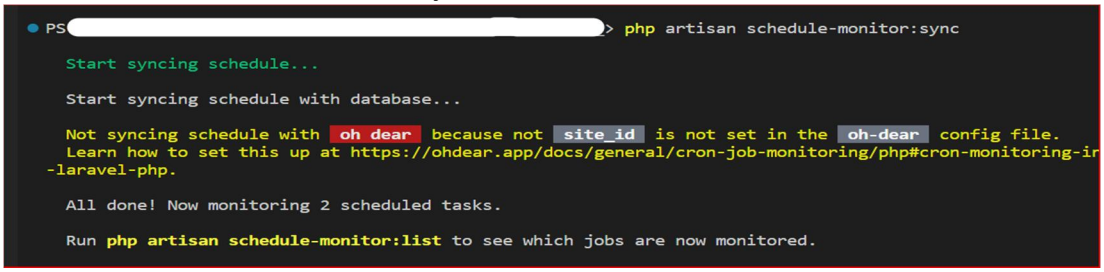


Figure 6: Sync the package.

D. Example

Let’s write a command GeneratePassword which updates the password every minute.



Figure 7: Basic Command

Lets register the command in Kernel.php

```
21      protected function schedule(Schedule $schedule)
22      {
23          $schedule->command('passport:purge')->hourly();
24
25          $schedule
26              ->command('encrypt:passwords')
27              ->everyMinute()
28              ->runInBackground()
29              ->withoutOverlapping()
30              ->storeOutputInDb();
31      }
32
```

Figure 8: Register the Command in Kernel

Run the command at the background
To run the command we use **schedule:work** command

```
php artisan schedule:work

 INFO  Running scheduled tasks every minute.


  2024-04-29 05:33:01 Running ["artisan" encrypt:passwords] in background ..... 324ms DONE
  ⇂ start /b cmd /v:on /c "("C:\php\php.exe" "artisan" encrypt:passwords & "C:\php\php.exe" "artisan" schedule:finish "framework\schedule-b978c1b
7f41fb028dc11e71ea9b8c9cd31a74e73" ^!ERRORLEVEL^!) > "C:\Users\SAILS-DM219\Desktop\Practice\research-paper\storage\logs/schedule-7625d7fbf514c43b
17e45fd977202f51482d42b0.log" 2>&1"
```

Figure 9: Schedule Work command execution

Let's see the users table for some users

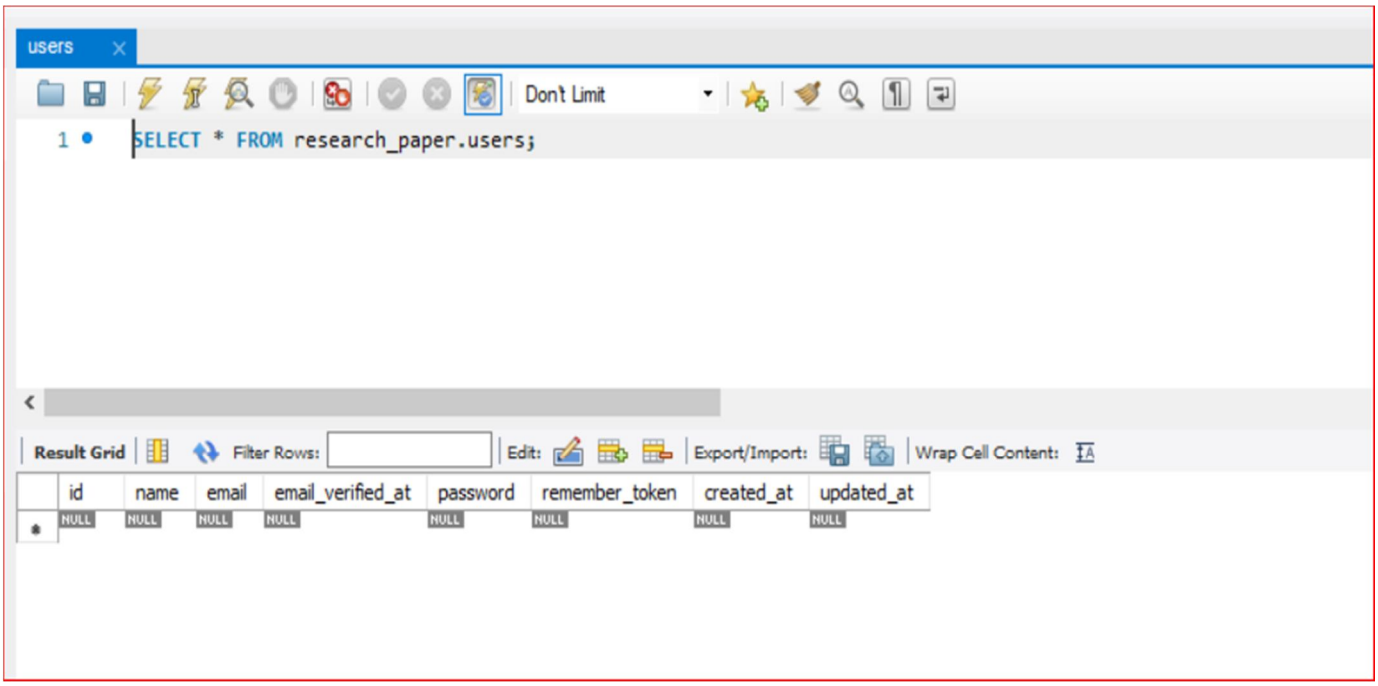


Figure 10: Checking Users table

You can observe that initially there are no users in the table and run the **schedule:work**

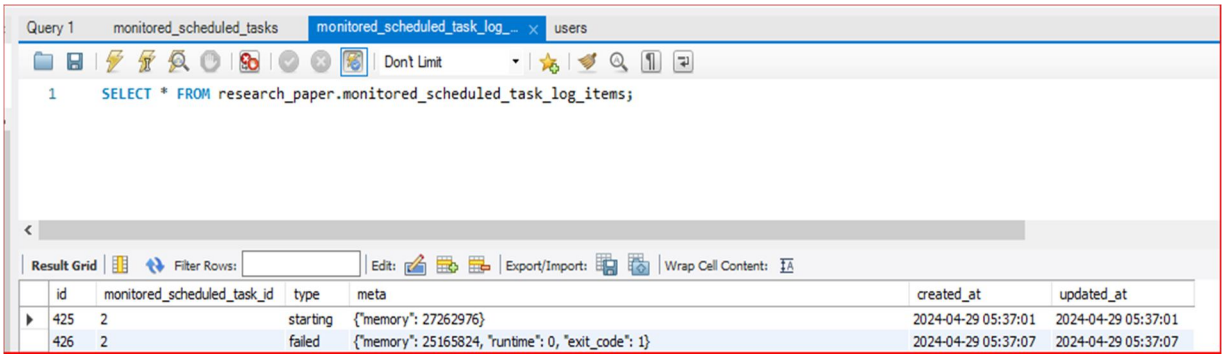


Figure 11: Checking Schedule Logs

E. Observation

We are facing a problem with minimal metadata being returned when a command fails. Examples of this metadata include "runtime", "exit_code", and "memory". It may not be enough to diagnose complex failures.

Based on these constrained information values, it's easier to identify the issue when working with fewer instructions. But when commands get more complicated that is, when you have to deal with a lot of lines, manipulate data, or perform intricate operations it gets harder to figure out the failure's primary cause with just this scant metadata.

The “MonitorScheduledTask” package is built in a way that, if any event or job fails it will store the runtime, exit code, and memory metadata fields in the database, but it doesn’t offer any further explanation and information on the context for the failure reasons. It is indeed necessary to improve the reliability of the monitoring system to record and collect more metadata of the error message which offers more insights into the underlying problems of the commands/jobs failure.

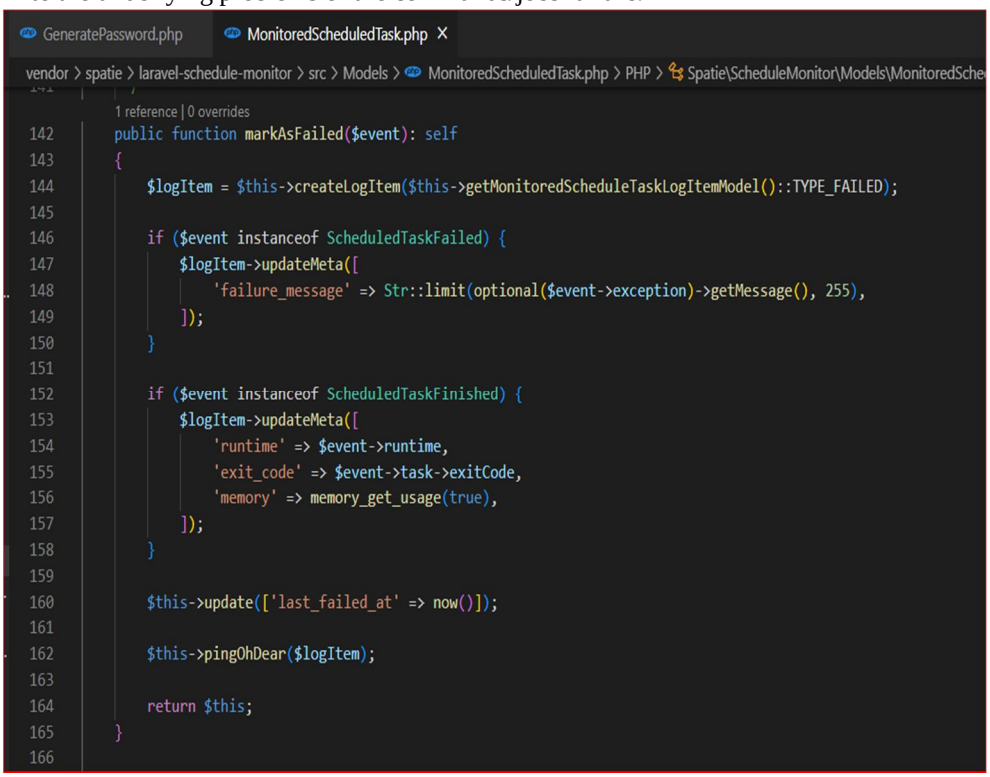


Figure 12: Monitored Schedule Task Model

The subject is accurate that a longer investigation may result from a lack of specific details regarding the reason a command failed which involves more manual tasks to debug the issues in the absence of sufficient context or error warning.

F. Monitored Scheduled Task - Package Update

Updating the package to record the cause of the failure reason helps in enhancing the troubleshooting and investigation time. This change in the package helps the monitoring system record the failure reason along with the metadata. This includes adding comprehensive error logging information to the MonitoredScheduledTask package. The package catches the error messages and helps in storing the new information in the database.

When looking into command failures, administrators and developers would have access to a larger collection of information by doing this. They could rapidly determine the failure's primary cause and take the necessary action to fix it by simply querying the database to obtain the error messages linked to unsuccessful commands..

```
'output' => $this->getEventTaskOutput($event),
```
[1]

Figure 13: Code to be added

At line number: 157, we have added the bit of code to capture the error logs.

Link: https://github.com/spatie/laravel-schedule-monitor/pull/97

```
142     public function markAsFailed($event): self
143     {
144         $logItem = $this->createLogItem($this->getMonitoredScheduleTaskLogItemModel()::TYPE_FAILED);
145
146         if ($event instanceof ScheduledTaskFailed) {
147             $logItem->updateMeta([
148                 'failure_message' => Str::limit(optional($event->exception)->getMessage(), 255),
149             ]);
150         }
151
152         if ($event instanceof ScheduledTaskFinished) {
153             $logItem->updateMeta([
154                 'runtime' => $event->runtime,
155                 'exit_code' => $event->task->exitCode,
156                 'memory' => memory_get_usage(true),
157                 'output' => $this->getEventTaskOutput($event),
158             ]);
159         }
160
161         $this->update(['last_failed_at' => now()]);
162
163         $this->pingOhDear($logItem);
164
165         return $this;
166     }
167
```

Figure 14: Updated Code

If we make the command run now, by schedule:work in terminal.

```
> php artisan schedule:work

  INFO  Running scheduled tasks every minute.


  2024-04-29 10:52:01 Running ["artisan" encrypt:passwords] in background ..... 380ms DONE
  ⇂ start /b cmd /v:on /c "("C:\php\php.exe" "artisan" encrypt:passwords & "C:\php\php.exe" "artisan" schedule:finish "framework\schedule-b978c1b
7f41fb028dc11e71ea9b8c9cd31a74e73" ^!ERRORLEVEL^!) > "C:\Users\SAILS-DM219\Desktop\Practice\research-paper\storage\logs/schedule-7625d7fbf514c43b
17e45fd977202f51482d42b0.log" 2>&1"
```

Figure 15: Schedule Work

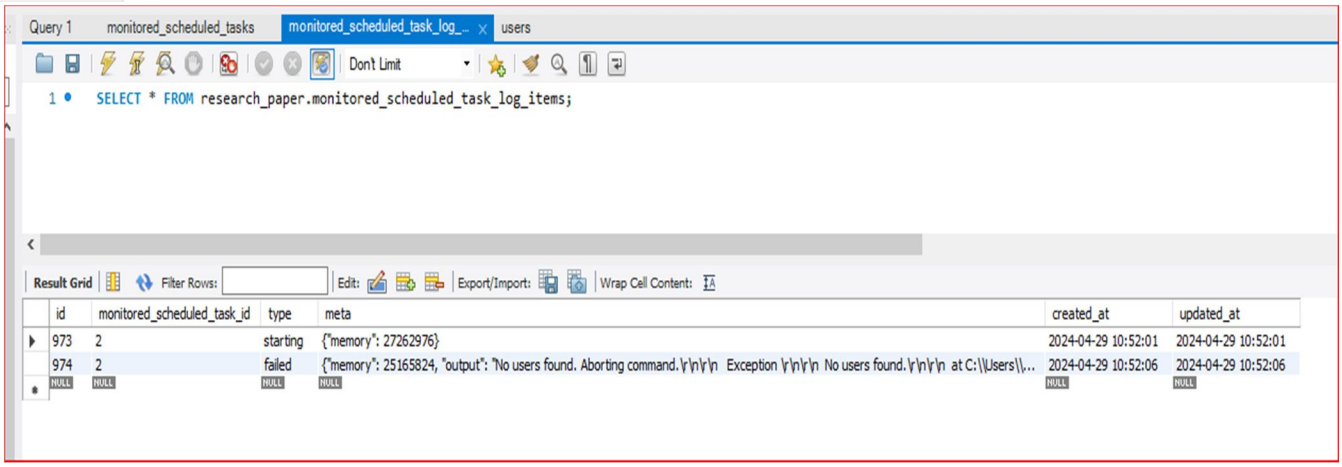


Figure 16: Error Log in Schedule Logs Table

VI. CONCLUSION

The ability to record and keep comprehensive error data in the database has made it easier for developers to locate and fix the core reasons for command failures. This change gives them the capacity to identify and fix problems quickly, reducing downtime and increasing system dependability. Furthermore, centralizing error data in the database makes it easier to do continuous analysis and follow trends. Development teams can prioritize improvements and improve system performance over time by identifying reoccurring problems or areas for improvement. The system will adapt to meet changing demands and requirements thanks to this iterative approach, which promotes ongoing progress. In conclusion up, adding thorough error logging to the package is a proactive step that improves the system's maintainability and dependability. This improvement enhances operational excellence and yields measurable time and resource savings by providing developers with the knowledge they need to resolve problems effectively and enable systematic analysis for long-term optimization.

REFERENCES

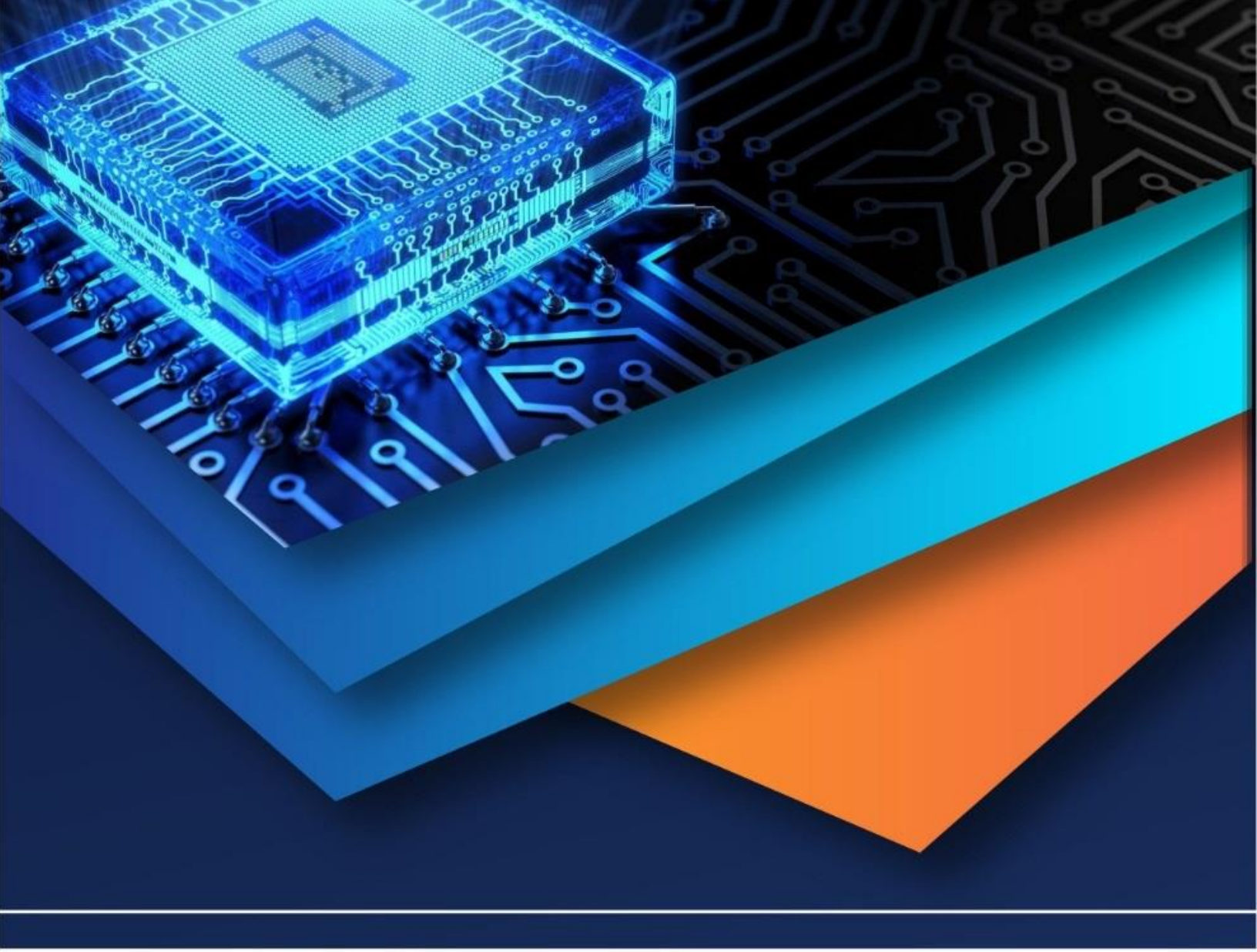
[1] https://github.com/spatie/laravel-schedule-monitor/pull/97
[2] https://spatie.be/docs/laravel-query-builder/v5/installation-setup
[3] https://www.cloudways.com/blog/custom-artisan-commands-laravel/
[4] https://laravel.com/docs/11.x

ABOUT THE AUTHOR



Akhil Duggirala is a Software Developer at Sails Software Solutions having 2+ years of experience in Full Stack Development and graduated from RGUKT IIIT SRIKAKULAM. He is very fascinated with technology and has a publication in Machine Learning in IJSRD titled Bank Loan Personal Modelling using Classification Algorithms of Machine Learning. His interest lies in various Machine Learning Techniques, Full Stack Development and open-source contributions.

©IJRASET: All Rights are Reserved



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)