LARA-FILES

We are working on a file-system to database event-sourced system. The user will keep using his file system organization and our app will create a database and a nice UI for the user to find the documents. The user will not use any app for creating, renaming, deleting or moving folders or files. The app needs to watch changes in the file system to map them into the database.

We will use Laravel and the Spatie's package laravel-event-sourcing (v7) and Spatie's file-system-watcher package for real time watching. The stored_events table is now already being populated, using the code at the end of this explanation. So we have the initial events to work with. Now it's time to work on the first projector that will represent the file system.

We are using this system to manage the parts required to manufacture valves (.par) and the corresponding plans (.pdf), but the system should be product agnostic to keep it flexible. The main goal of the app is to help the user find the parts required to manufacture a valve and all the plans fastly, easily and with a great user experience.

We will have just one root or main folder-to-watch (let's call it /PLANOS). Inside it, we will have many directories. Each of these folders represents a main valve type (main product type) and the name of the main type corresponds to the folder name. All the folders (at different depths) inside the main valve type folder will represent different subtypes of the main product. A main type has no parent, but every subtype will have one parent.

There are three special folder names (once again we have to be flexible and be able to add more special folders): ARCHIVO, EN REVISION and all the folders that begin with 00 will have special treatment and should be ignored for the normal workflow.

Then we have the files. Each file represents a part of a product. The specific part to which each file belongs to, is defined by the file name excluding the extension and the revision (the path where the file lives is also important but the same file may live in different folders).

We will have three file extensions (but the system must be flexible and have the chance to add or remove file extensions to handle, for example .doc/x, xls/x or images like .jpg and ,png will be added to the system soon but not yet): .par, .dft and .pdf. The .par files are SolidEdge 3D individual parts, .dft are SolidEdge 2D individual drawings and the .pdf are just the same drawings as the .dft but in a portable format.

Revisions are defined in the file-name. It's easy to extract the revision. All the revisions are in alphabetical and/or numerical order, so 'revA' is older than 'revB' and 'version24' is older than 'version25'.

We have to define what we call the file-name and the part-name. The file-name and the part-name are similar concepts. The part-name is the name of a physical part that will be used when assembling a product which usually includes a prefix (PREFIX_core-name.par), obviously a part must be unique, but can exist different files with the same part-name in their file-names (obviously with different extensions), which will mean that they are linked to that part. In a file-name, the part-name is the substring of that file-name excluding the file extension and the revision. The core-name, will be the same as the part-name excluding the PREFIX. The part-name must be unique and it may have a .par, a .dft and a .pdf file or any other file extensions with the corresponding revisions. The files with different extensions that share the same part-name will be considered linked together to that part. This way a part may have linked to it a 3D model, a 2D drawing, an image, an excel file, a text file…

We will be able to set rules in a readable and flexible way in our app that will help the user eventually to keep the consistency of the file system and address hidden issues. The idea is that these rules will not stop the app, they will just warn the user about inconsistencies (remember that the user will keep manipulating the filesystem freely and this freedom is prone to inconsistencies).

Rules:

The same .par file may exist in different folders, but all of them must be the same (same content, same hash) and have the same file name. In fact, this is one of the core functionalities of the app: to be able to know in which products of our product range a specific part is used and also to warn if the same file-name with different content exists in the system (probably the oldest is outdated and can be a potential pitfall). The app also should warn if a file with the same content (hash) but different file-name has been found, as this situation should also be fixed by the user to keep the consistency of the system. We will need a column in the 'files' table called 'used_in' where we will store the path where each file is used.

On the contrary, only one .dft and .pdf may exist per '.par' file (one unique plan per part) in the whole main folder-to-watch. The app will work even if this rule is not accomplished, but it will keep track of these errors and eventually send notifications to the user until these kinds of discrepancies are resolved.

File-system example:

folder-to-watch/

      -/MAIN-TYPE-1/ (#parent = null, depth=1)

          -/SUB-TYPE-1   (#parent = MAIN-TYPE-1, depth=2)

               -SUB-TYPE-1_PART-1.par

               (#parent = SUB-TYPE-1, depth = 3, part-name = SUB-TYPE-1_PART-1, core-name = PART-1)

               -SUB-TYPE-1_PART-1.dft

               -SUB-TYPE-1_PART-1.pdf

               -SUB-TYPE-1_PART-2.par

               -SUB-TYPE-1_PART-2_revA.dft

               -SUB-TYPE-1_PART-2_revA.pdf (#core-name = PART-2)

               -/SUB-TYPE-2   (#parent = SUB-TYPE-1, #depth = 3)

                    -SUB-TYPE-2_PART-1.par (#parent = SUB-TYPE-2, depth =4)

                    -SUB-TYPE-2_PART-1.dft

                    -SUB-TYPE-2_PART-1.pdf

        -/SUB-TYPE-3 #parent = MAIN-TYPE-1

               -SUB-TYPE-1_PART-1.par ()

               -SUB-TYPE-3_PART-1.par

               -SUB-TYPE-3_PART-2.dft

               -SUB-TYPE-3_PART-2.pdf

               -SUB-TYPE-3_PART-3.pdf

               -SUB-TYPE-3_PART-3.dft

```
                    -SUB-TYPE-3_PART-4.dft
          -MAIN-TYPE-1_PART-1.par
          -MAIN-TYPE-1_PART-1.dft
          -MAIN-TYPE-1_PART-1.pdf
-/MAIN-TYPE-2/
          -/MAIN-TYPE-2 SUB-TYPE-1
                    -SUB-TYPE-1_PART-1.par
                    -SUB-TYPE-1_PART-1.dft
                    -SUB-TYPE-1_PART-1.pdf
                    -SUB-TYPE-1_PART-2.par
                    -SUB-TYPE-1_PART-2.dft
                    -SUB-TYPE-1_PART-2.pdf
          -/SUB-TYPE-2
                    -SUB-TYPE-1_PART-1.par
                    -SUB-TYPE-1_PART-1.dft
                    -SUB-TYPE-1_PART-1.pdf
                    -SUB-TYPE-2_PART-1.par
                    -SUB-TYPE-2_PART-2.dft
                    -SUB-TYPE-2_PART-2.pdf
          -MAIN-TYPE-1_PART-1.par
          -MAIN-TYPE-1_PART-1.dft
          -MAIN-TYPE-1_PART-1.pdf
```

STEP 1:

Write a projector that will populate the 'files' table, which will be a replica or a Model of the file-system (compendium of all the folders and files).
At this step the rules don't matter, we just want to recreate the real state of the file-system in a database for fast search&find. This is the idea I have for the columns, but I am open to ideas. We will also need a migration for the 'files' table:

Column	Type	Nullable	Default
id	INTEGER	NO	NULL
path	varchar	NO	NULL
name	varchar	NO	NULL
file_type	varchar	NO	NULL
extension	varchar	YES	NULL
revision	varchar	YES	NULL
part_name	varchar	YES	NULL
core_name	varchar	YES	NULL
product_main_type	varchar	YES	NULL
product_sub_type	varchar	YES	NULL
parent	varchar	YES	NULL
parent_path	varchar	YES	NULL
depth	INTEGER	NO	NULL
origin	varchar	NO	NULL
content_hash	varchar	YES	NULL
size	INTEGER	YES	NULL
modified_at	datetime	YES	NULL
created_at	datetime	YES	NULL
updated_at	datetime	YES	NULL

STEP 2:

Write the MasterFilesProjector.php. This projector will populate the 'masters' table for each event that will be used by the app to quickly find and provide the required .pdf documents to the user for all the .par files in a folder. Using a similar 'config' approach used in STEP 1 (FileSystemProjector), we will need a place where to define the file extensions that are candidates to become a 'master'. By the moment they will be: .par, .asm, .doc, .docx, .xls and .xlsx. To be a 'master' we need a 'slave', by the moment they will be: .pdf files but in the future other extensions may be candidates to be slaves. Following the approach in STEP1, we also require 'Directory name prefixes to omit' and 'Directory names that must be ignored completely' in the config file. The files in these directories will be omitted by this projector.

As the same .par file may live in multiple folders, we need a way to define one as the "master" of them. The 'master' will be a .par file (or .doc, .xlsx...). A specific .par file will be considered the master, if a slave .pdf with the same file name as the .par file (excluding the extension and revision) lives in the same folder as that .par file. The main goal of this app is to easily provide the unique slave .pdf document of any .par file used across the file system to the user. All the .par with the same name that live across the file system should have the same content as the 'master' (to be sure it's the same file copied across), but this rule will be monitored by another projector, it has nothing to do with this STEP 2. The same way, there should only exist one master, but this rule among others will be projected by another projector.

This projector must run only after the FileSystemProjector has run, as it will use the 'files' table as read-only to check whether the conditions for a file in the event stream to be a master are met. This projector also needs to remove from the table any file that after an event does not meet the conditions. We will need a migration and a model. The ideas for the columns could be:

path (the same as in 'files')

part_name (the same as in 'files')

master_revision (the same as in 'files')

extension (the same as in 'files')

parent_path (the same as in 'files')

content_hash (the same as in 'files')

slave_path (here will be the complete path to the .pdf file)

slave_revision (the same as in 'files')

modified_at (the same as in 'files')

(timestamps_columns)

STEP 3:

Write the PartsProjector.php. This projector will populate the 'parts' table for each event. This 'parts' table will be used by the app to quickly find and provide the required .pdf documents to the user for all the .par (docx, xlsx…) files in the whole file system. Using a similar 'config' approach used in STEP 1 and STEP 2 (FileSystemProjector and MasterFilesProjector), we will need a place to define the file extensions that are considered a 'part'. By the moment they will be: .par, .asm, .doc, .docx, .xls and .xlsx. A part may or may not have a corresponding 'master' in the 'masters' table. Following the approach in STEP1 AND 2, we also require 'Directory name prefixes to omit' and 'Directory names that must be ignored completely' in the config file. The files in these directories will be omitted by this projector.

For each file in the event-stream, using the 'files' and 'masters' tables as read-only, the PartsProjector will check if the file of the event can be considered a part and if it has a part with the same 'part_name' or the same 'content_hash' in the 'masters' table. If so, it will update the 'slave_path' column in the 'parts' table with the same 'slave_path' that exists in the 'masters' table, if not it will write 'NULL'. If the part's 'hash_content' is the same as in the 'masters' table, the 'content_as_master' column will be set to 'TRUE' and to 'FALSE' if not.

This projector must run only after the FileSystemProjector and the MasterFilesProjector have run, as it will use the 'files' and 'masters' tables as read-only to decide how to manage each file in the event stream. This projector also needs to update the 'parts' table after each new event. We will need a migration and a model. The ideas for the columns could be:

path (the same as in 'files')

slave_path (as in 'masters' or 'NULL')

slave_revision (the same as in 'files')

part_name (the same as in 'files')

master_path (the same as in 'files')

master_revision (the same as in 'files')

core_name (the same as in 'files')

parent_path (the same as in 'files')

parent (the same as in 'files')

content_hash (the same as in 'files')

content_as_master ('TRUE' or 'FALSE')

modified_at (the same as in 'files')

(timestamps_columns)

Folder app\Events

app\Events\FileSystem\DirectoryCreated.php

```php
<?php

namespace App\Events\FileSystem;


class DirectoryCreated extends FileSystemEvent
{
    public function __construct(string $path, string $origin = 'real-time')
    {
        parent::__construct($path, $origin);
    }
}
```

app\Events\FileSystem\DirectoryDeleted.php

```php
<?php

namespace App\Events\FileSystem;

class DirectoryDeleted extends FileSystemEvent
{
    public function __construct(string $path, string $origin = 'real-time')
    {
        parent::__construct($path, $origin);
    }
}
```

app\Events\FileSystem\FileCreated.php

```php
<?php

namespace App\Events\FileSystem;

class FileCreated extends FileSystemEvent
{
    public function __construct(
        string $path,
        string $origin = 'real-time',
        ?string $hash = null,
        ?\DateTime $modifiedAt = null,
        ?int $size = null
    ) {
        parent::__construct($path, $origin, $hash, $modifiedAt, $size);
    }
}
```

app\Events\FileSystem\FileDeleted.php

```php
<?php

namespace App\Events\FileSystem;

class FileDeleted extends FileSystemEvent
{
    public function __construct(string $path, string $origin = 'real-time')
    {
        parent::__construct($path, $origin);
```

```php
    }
}

app\Events\FileSystem\FileModified.php
<?php

namespace App\Events\FileSystem;

class FileModified extends FileSystemEvent
{
    public ?string $previousHash;

    public function __construct(
        string $path,
        string $origin = 'real-time',
        ?string $hash = null,
        ?\DateTime $modifiedAt = null,
        ?int $size = null,
        ?string $previousHash = null
    ) {
        parent::__construct($path, $origin, $hash, $modifiedAt, $size);
        $this->previousHash = $previousHash;
    }

    public function toArray(): array
    {
        return array_merge(parent::toArray(), [
            'previous_hash' => $this->previousHash,
        ]);
```

```
        }
}

app\Events\FileSystem\FileSystemEvent.php
<?php

namespace App\Events\FileSystem;

use Spatie\EventSourcing\StoredEvents\ShouldBeStored;

abstract class FileSystemEvent extends ShouldBeStored
{
    public string $path;
    public string $origin; // 'initial', 'real-time', 'reconciled'
    public ?string $hash;
    public ?\DateTime $modifiedAt;
    public ?int $size;

    private array $metadata = [];

    public function __construct(
        string $path,
        string $origin = 'real-time',
        ?string $hash = null,
        ?\DateTime $modifiedAt = null,
        ?int $size = null
    ) {
        $this->path = $path;
        $this->origin = $origin;
```

```php
        $this->hash = $hash;
        $this->modifiedAt = $modifiedAt;
        $this->size = $size;
    }

    public function toArray(): array
    {
        $data = [
            'path' => $this->path,
            'origin' => $this->origin,
            'hash' => $this->hash,
            'modified_at' => $this->modifiedAt?->format('Y-m-d H:i:s'),
            'size' => $this->size,
            'type' => $this->getEventType(),
        ];

        return $data;
    }

    private function getEventType(): string
    {
        $className = get_class($this);

        if (str_contains($className, 'Directory')) {
            return 'directory';
        }

        if (str_contains($className, 'File')) {
            return 'file';
```

```php
        }

        return 'unknown';
    }

    public function addMetadata(array $data): void
    {
        $this->metadata = array_merge($this->metadata, $data);
    }

}
```

Folder app\Services

```php
app\Services\FileSystemReconciler.php
<?php

namespace App\Services;

use SplFileInfo;
use Carbon\Carbon;
use RecursiveIteratorIterator;
use RecursiveDirectoryIterator;
use Illuminate\Support\Facades\DB;
use Illuminate\Support\Facades\Log;
use App\Events\FileSystem\FileCreated;
use App\Events\FileSystem\FileModified;
use App\Events\FileSystem\FileDeleted;
use App\Events\FileSystem\DirectoryCreated;
use App\Events\FileSystem\DirectoryDeleted;
use Spatie\EventSourcing\StoredEvents\Models\EloquentStoredEvent;

class FileSystemReconciler
{
    private string $basePath;
    private array $discrepancies = [];

    public function __construct(string $basePath)
    {
        $this->basePath = rtrim($basePath, '/\\');
```

```php
    }

    public function execute(): array
    {
        Log::info("Starting file system reconciliation", ['path' => $this->basePath]);

        $currentState = $this->crawlFileSystem();
        $eventTimeline = $this->getEventTimeline();

        // Two-phase reconciliation
        $this->findMissingItems($currentState, $eventTimeline);  // Phase 1: Find deletions
        $this->findExistingItemDiscrepancies($currentState, $eventTimeline);  // Phase 2: Find
additions/modifications

        $generatedEvents = $this->generateReconciliationEvents();

        Log::info("Reconciliation completed", [
            'items_scanned' => count($currentState),
            'discrepancies_found' => count($this->discrepancies),
            'events_generated' => $generatedEvents
        ]);

        return [
            'scanned' => count($currentState),
            'discrepancies' => count($this->discrepancies),
            'events_created' => $generatedEvents
        ];
    }
```

```php
private function crawlFileSystem(): array
{
    $currentState = [];
    $iterator = new RecursiveIteratorIterator(
        new RecursiveDirectoryIterator($this->basePath, RecursiveDirectoryIterator::SKIP_DOTS),
        RecursiveIteratorIterator::SELF_FIRST
    );

    foreach ($iterator as $item) {
        $path = $item->getPathname();
        $relativePath = $this->getRelativePath($path);

        $currentState[$relativePath] = [
            'type' => $item->isDir() ? 'directory' : 'file',
            'path' => $relativePath,
            'modified_at' => Carbon::createFromTimestamp($item->getMTime()),
            'size' => $item->isFile() ? $item->getSize() : 0,
            'hash' => $item->isFile() ? $this->calculateFileHash($path) : null
        ];
    }

    return $currentState;
}

private function getEventTimeline(): array
{
    $events = EloquentStoredEvent::query()
        ->where('event_class', 'LIKE', '%FileSystem%')
        ->orderBy('created_at', 'desc')
```

```php
            ->get(['event_class', 'event_properties', 'created_at']);

        $timeline = [];

        foreach ($events as $event) {
            $props = $event->event_properties;
            $path = $props['path'] ?? null;

            if (!$path) continue;

            // Only keep the most recent event per path
            if (!isset($timeline[$path])) {
                $timeline[$path] = [
                    'type' => $this->getEventType($event->event_class),
                    'event_class' => $event->event_class,
                    'created_at' => $event->created_at,
                    'properties' => $props
                ];
            }
        }

        return $timeline;
    }

    /**
     * Phase 1: Find items that exist in events but are missing from filesystem
     * These should be marked as deleted
     */
    private function findMissingItems(array $currentState, array $eventTimeline): void
```

```php
{
    foreach ($eventTimeline as $path => $event) {
        // Skip if this path doesn't belong to our base path
        if (!$this->pathBelongsToBasePath($path)) {
            continue;
        }

        // Skip if the last event was already a deletion
        if ($this->isDeleteEvent($event['event_class'])) {
            continue;
        }

        // If item exists in events but not in current filesystem, it was deleted
        if (!isset($currentState[$path])) {
            $this->discrepancies[$path] = [
                'type' => $event['type'],
                'reason' => 'missing_from_filesystem',
                'current' => null,
                'event' => $event
            ];
        }
    }
}

/**
 * Phase 2: Find discrepancies for items that exist in filesystem
 * Original logic for additions and modifications
 */
private function findExistingItemDiscrepancies(array $currentState, array $eventTimeline): void
```

```php
{
    foreach ($currentState as $path => $current) {
        $event = $eventTimeline[$path] ?? null;

        if (!$event) {
            // Item exists but has no recorded event
            $this->discrepancies[$path] = [
                'type' => $current['type'],
                'reason' => 'missing_event',
                'current' => $current,
                'event' => null
            ];
            continue;
        }

        if ($this->isDeleteEvent($event['event_class'])) {
            // Item exists but was deleted in event history
            $this->discrepancies[$path] = [
                'type' => $current['type'],
                'reason' => 'deleted_but_exists',
                'current' => $current,
                'event' => $event
            ];
            continue;
        }

        if ($current['type'] === 'file') {
            $eventTime = Carbon::parse($event['created_at']);
            $currentTime = $current['modified_at'];
```

```php
            if ($currentTime->gt($eventTime)) {
                // File modified after last event
                $this->discrepancies[$path] = [
                    'type' => 'file',
                    'reason' => 'modified_after_event',
                    'current' => $current,
                    'event' => $event
                ];
            }
        }
    }
}

private function generateReconciliationEvents(): int
{
    $count = 0;

    // Sort discrepancies to handle deletions of parent folders before children
    $sortedDiscrepancies = $this->sortDiscrepanciesForDeletion($this->discrepancies);

    foreach ($sortedDiscrepancies as $path => $discrepancy) {
        try {
            switch ($discrepancy['reason']) {
                case 'missing_from_filesystem':
                    // Create deletion events for items that no longer exist
                    if ($discrepancy['type'] === 'directory') {
                        event(new DirectoryDeleted($path, 'reconciled'));
                        $count++;
```

```php
        } else {
            event(new FileDeleted($path, 'reconciled'));
            $count++;
        }
        break;

    case 'missing_event':
    case 'deleted_but_exists':
        if ($discrepancy['type'] === 'directory') {
            event(new DirectoryCreated($path, 'reconciled'));
            $count++;
        } else {
            event(new FileCreated(
                $path,
                'reconciled',
                $discrepancy['current']['hash'],
                $discrepancy['current']['modified_at'],
                $discrepancy['current']['size']
            ));
            $count++;
        }
        break;

    case 'modified_after_event':
        event(new FileModified(
            $path,
            'reconciled',
            $discrepancy['current']['hash'],
            $discrepancy['current']['modified_at'],
```

```php
                    $discrepancy['current']['size'],
                    $discrepancy['event']['properties']['hash'] ?? null
                ));
                $count++;
                break;
            }

            Log::debug("Generated reconciliation event", [
                'path' => $path,
                'reason' => $discrepancy['reason'],
                'type' => $discrepancy['type']
            ]);
        } catch (\Exception $e) {
            Log::error("Failed to generate reconciliation event", [
                'path' => $path,
                'error' => $e->getMessage()
            ]);
        }
    }

    return $count;
}

/**
 * Sort discrepancies to handle directory deletions properly
 * Directories should be deleted after their contents (deepest first)
 */
private function sortDiscrepanciesForDeletion(array $discrepancies): array
{
```

```php
    $deletions = [];
    $others = [];

    foreach ($discrepancies as $path => $discrepancy) {
        if ($discrepancy['reason'] === 'missing_from_filesystem') {
            $deletions[$path] = $discrepancy;
        } else {
            $others[$path] = $discrepancy;
        }
    }

    // Sort deletions by path depth (deepest first) to delete children before parents
    uksort($deletions, function ($a, $b) {
        $depthA = substr_count($a, DIRECTORY_SEPARATOR);
        $depthB = substr_count($b, DIRECTORY_SEPARATOR);

        if ($depthA === $depthB) {
            return strcmp($b, $a); // Reverse alphabetical for same depth
        }

        return $depthB - $depthA; // Deeper paths first
    });

    return array_merge($deletions, $others);
}

/**
 * Check if a path belongs to our base path
 */
```

```php
private function pathBelongsToBasePath(string $path): bool
{
    // For deleted items, we can't use realpath since the path doesn't exist
    // Instead, do a simple string comparison
    $normalizedPath = str_replace(['/', '\\'], DIRECTORY_SEPARATOR, $path);

    // Check if it's a relative path within our base path structure
    // Paths should not start with / or contain ..
    return !str_starts_with($normalizedPath, DIRECTORY_SEPARATOR) &&
           !str_contains($normalizedPath, '..');
}

private function getRelativePath(string $absolutePath): string
{
    return str_replace($this->basePath . DIRECTORY_SEPARATOR, '', $absolutePath);
}

private function calculateFileHash(string $path): ?string
{
    if (!is_file($path)) return null;

    try {
        return hash_file('sha256', $path);
    } catch (\Exception $e) {
        Log::warning("Could not calculate file hash", ['path' => $path]);
        return null;
    }
}
```

```php
    private function getEventType(string $className): string
    {
        if (str_contains($className, 'Directory')) return 'directory';
        if (str_contains($className, 'File')) return 'file';
        return 'unknown';
    }

    private function isDeleteEvent(string $className): bool
    {
        return str_contains($className, 'Deleted');
    }
}
```

app\Services\FileSystemScanner.php

```php
<?php

namespace App\Services;

use App\Events\FileSystem\DirectoryCreated;
use App\Events\FileSystem\FileCreated;
use Illuminate\Support\Facades\Log;
use RecursiveDirectoryIterator;
use RecursiveIteratorIterator;
use SplFileInfo;

class FileSystemScanner
{
    private string $basePath;
    private int $totalItems = 0;
```

```php
private int $processedItems = 0;
private array $stats = [
    'directories' => 0,
    'files' => 0,
    'total_size' => 0,
    'errors' => 0,
];

public function __construct(string $basePath)
{
    $this->basePath = rtrim($basePath, '/\\');
}

public function scan(callable $progressCallback = null): array
{
    Log::info("Starting initial file system scan", ['path' => $this->basePath]);

    if (!is_dir($this->basePath)) {
        throw new \InvalidArgumentException("Path does not exist or is not a directory: {$this->basePath}");
    }

    // First pass: count total items for progress tracking
    $this->countItems();

    // Second pass: process items and emit events
    $this->processItems($progressCallback);

    Log::info("File system scan completed", $this->stats);
```

```php
        return $this->stats;
    }

    private function countItems(): void
    {
        try {
            $iterator = new RecursiveIteratorIterator(
                new RecursiveDirectoryIterator($this->basePath, RecursiveDirectoryIterator::SKIP_DOTS),
                RecursiveIteratorIterator::SELF_FIRST
            );

            $this->totalItems = iterator_count($iterator);
        } catch (\Exception $e) {
            Log::error("Error counting items", ['error' => $e->getMessage()]);
            $this->totalItems = 0;
        }
    }

    private function processItems(callable $progressCallback = null): void
    {
        try {
            $iterator = new RecursiveIteratorIterator(
                new RecursiveDirectoryIterator($this->basePath, RecursiveDirectoryIterator::SKIP_DOTS),
                RecursiveIteratorIterator::SELF_FIRST
            );

            foreach ($iterator as $fileInfo) {
                try {
                    $this->processItem($fileInfo);
```

```php
                $this->processedItems++;

                if ($progressCallback && $this->totalItems > 0) {
                    $progress = ($this->processedItems / $this->totalItems) * 100;
                    $progressCallback($progress, $this->processedItems, $this->totalItems,
$fileInfo->getPathname());
                }
            } catch (\Exception $e) {
                $this->stats['errors']++;
                Log::error("Error processing item", [
                    'path' => $fileInfo->getPathname(),
                    'error' => $e->getMessage()
                ]);
            }
        }
    } catch (\Exception $e) {
        Log::error("Error during file system scan", ['error' => $e->getMessage()]);
        throw $e;
    }
}

private function processItem(SplFileInfo $fileInfo): void
{
    $relativePath = $this->getRelativePath($fileInfo->getPathname());

    if ($fileInfo->isDir()) {
        $this->processDirectory($relativePath);
    } else {
        $this->processFile($fileInfo, $relativePath);
```

```php
        }
    }

    private function processDirectory(string $relativePath): void
    {
        event(new DirectoryCreated($relativePath, 'initial'));
        $this->stats['directories']++;
    }

    private function processFile(SplFileInfo $fileInfo, string $relativePath): void
    {
        $hash = $this->calculateFileHash($fileInfo->getPathname());
        $modifiedAt = new \DateTime('@' . $fileInfo->getMTime());
        $size = $fileInfo->getSize();

        event(new FileCreated(
            $relativePath,
            'initial',
            $hash,
            $modifiedAt,
            $size
        ));

        $this->stats['files']++;
        $this->stats['total_size'] += $size;
    }

    private function calculateFileHash(string $filePath): ?string
    {
```

```php
        try {
            // For large files, we might want to use a more efficient method
            if (filesize($filePath) > 100 * 1024 * 1024) { // 100MB
                return hash_file('md5', $filePath);
            }
            return hash_file('sha256', $filePath);
        } catch (\Exception $e) {
            Log::warning("Could not calculate hash for file", [
                'path' => $filePath,
                'error' => $e->getMessage()
            ]);
            return null;
        }
    }

    private function getRelativePath(string $absolutePath): string
    {
        return str_replace($this->basePath . DIRECTORY_SEPARATOR, '', $absolutePath);
    }

    public function getStats(): array
    {
        return $this->stats;
    }

    public function getProgress(): float
    {
        return $this->totalItems > 0 ? ($this->processedItems / $this->totalItems) * 100 : 0;
    }
```

```php
}

app\Services\FileSystemWatcher.php
<?php

namespace App\Services;

use App\Events\FileSystem\DirectoryCreated;
use App\Events\FileSystem\DirectoryDeleted;
use App\Events\FileSystem\FileCreated;
use App\Events\FileSystem\FileDeleted;
use App\Events\FileSystem\FileModified;
use Illuminate\Support\Facades\Log;
use Spatie\Watcher\Watch;

class FileSystemWatcher
{
    private string $basePath;
    private array $fileHashes = [];

    public function __construct(string $basePath)
    {
        $this->basePath = rtrim($basePath, '/\\');
        $this->loadExistingHashes();
    }

    public function start(): void
    {
        Log::info("Starting file system watcher", ['path' => $this->basePath]);
```

```php
        Watch::path($this->basePath)
            ->onFileCreated(function (string $path) {
                $this->handleFileCreated($path);
            })
            ->onFileUpdated(function (string $path) {
                $this->handleFileUpdated($path);
            })
            ->onFileDeleted(function (string $path) {
                $this->handleFileDeleted($path);
            })
            ->onDirectoryCreated(function (string $path) {
                $this->handleDirectoryCreated($path);
            })
            ->onDirectoryDeleted(function (string $path) {
                $this->handleDirectoryDeleted($path);
            })
            ->start();
    }

    private function handleFileCreated(string $absolutePath): void
    {
        try {
            $relativePath = $this->getRelativePath($absolutePath);

            if (!file_exists($absolutePath)) {
                Log::warning("File creation event received but file doesn't exist", ['path' => $absolutePath]);
                return;
            }
```

```php
        $hash = $this->calculateFileHash($absolutePath);
        $modifiedAt = new \DateTime('@' . filemtime($absolutePath));
        $size = filesize($absolutePath);

        // Store hash for future comparison
        $this->fileHashes[$relativePath] = $hash;

        event(new FileCreated($relativePath, 'real-time', $hash, $modifiedAt, $size));

        Log::info("File created", [
            'path' => $relativePath,
            'size' => $size,
            'hash' => substr($hash, 0, 8) . '...'
        ]);

    } catch (\Exception $e) {
        Log::error("Error handling file creation", [
            'path' => $absolutePath,
            'error' => $e->getMessage()
        ]);
    }
}

private function handleFileUpdated(string $absolutePath): void
{
    try {
        $relativePath = $this->getRelativePath($absolutePath);
```

```php
if (!file_exists($absolutePath)) {
    Log::warning("File update event received but file doesn't exist", ['path' => $absolutePath]);
    return;
}

$newHash = $this->calculateFileHash($absolutePath);
$modifiedAt = new \DateTime('@' . filemtime($absolutePath));
$size = filesize($absolutePath);
$previousHash = $this->fileHashes[$relativePath] ?? null;

// Only emit event if content actually changed
if ($newHash !== $previousHash) {
    $this->fileHashes[$relativePath] = $newHash;

    event(new FileModified(
        $relativePath,
        'real-time',
        $newHash,
        $modifiedAt,
        $size,
        $previousHash
    ));

    Log::info("File modified", [
        'path' => $relativePath,
        'size' => $size,
        'old_hash' => $previousHash ? substr($previousHash, 0, 8) . '...' : 'unknown',
        'new_hash' => substr($newHash, 0, 8) . '...'
    ]);
```

```php
        }

    } catch (\Exception $e) {
        Log::error("Error handling file update", [
            'path' => $absolutePath,
            'error' => $e->getMessage()
        ]);
    }
}

private function handleFileDeleted(string $absolutePath): void
{
    try {
        $relativePath = $this->getRelativePath($absolutePath);

        // Remove from hash tracking
        unset($this->fileHashes[$relativePath]);

        event(new FileDeleted($relativePath, 'real-time'));

        Log::info("File deleted", ['path' => $relativePath]);

    } catch (\Exception $e) {
        Log::error("Error handling file deletion", [
            'path' => $absolutePath,
            'error' => $e->getMessage()
        ]);
    }
}
```

```php
private function handleDirectoryCreated(string $absolutePath): void
{
    try {
        $relativePath = $this->getRelativePath($absolutePath);

        event(new DirectoryCreated($relativePath, 'real-time'));

        Log::info("Directory created", ['path' => $relativePath]);

    } catch (\Exception $e) {
        Log::error("Error handling directory creation", [
            'path' => $absolutePath,
            'error' => $e->getMessage()
        ]);
    }
}

private function handleDirectoryDeleted(string $absolutePath): void
{
    try {
        $relativePath = $this->getRelativePath($absolutePath);

        // Remove all file hashes for files in this directory
        $this->fileHashes = array_filter(
            $this->fileHashes,
            fn($path) => !str_starts_with($path, $relativePath . '/'),
            ARRAY_FILTER_USE_KEY
        );
```

```php
            event(new DirectoryDeleted($relativePath, 'real-time'));

            Log::info("Directory deleted", ['path' => $relativePath]);

        } catch (\Exception $e) {
            Log::error("Error handling directory deletion", [
                'path' => $absolutePath,
                'error' => $e->getMessage()
            ]);
        }
    }

    private function calculateFileHash(string $filePath): ?string
    {
        try {
            // For large files, use MD5 for performance
            if (filesize($filePath) > 100 * 1024 * 1024) { // 100MB
                return hash_file('md5', $filePath);
            }
            return hash_file('sha256', $filePath);
        } catch (\Exception $e) {
            Log::warning("Could not calculate hash for file", [
                'path' => $filePath,
                'error' => $e->getMessage()
            ]);
            return null;
        }
    }
```

```php
    private function getRelativePath(string $absolutePath): string
    {
        return str_replace($this->basePath . DIRECTORY_SEPARATOR, '', $absolutePath);
    }

    private function loadExistingHashes(): void
    {
        // TODO: Load existing file hashes from database/cache
        // This would help detect modifications during watcher downtime
        Log::info("Loading existing file hashes for comparison");
    }
}
```

Folder app\Listeners

```php
app\Listeners\FileSystemEventListener.php
<?php

namespace App\Listeners;

use App\Events\FileSystem\FileSystemEvent;
use Illuminate\Contracts\Queue\ShouldQueue;
use Spatie\EventSourcing\StoredEvents\StoredEvent;

class FileSystemEventListener implements ShouldQueue
{
    public function handle($event)
    {
        if (!$event instanceof FileSystemEvent) {
            return;
        }

        // Enhance event properties with additional metadata
        $event->addMetadata([
            'file_type' => $this->determineFileType($event),
            'event_type' => class_basename($event),
            'origin' => $event->origin,
        ]);
    }

    private function determineFileType(FileSystemEvent $event): string
```

```php
    {
        $eventClass = class_basename($event);

        if (str_contains($eventClass, 'Directory')) {
            return 'directory';
        }

        if (str_contains($eventClass, 'File')) {
            return 'file';
        }

        return 'unknown';
    }
}
```

app\Listeners\LogFileSystemEvent.php

```php
<?php

namespace App\Listeners;

use App\Events\FileSystem\FileSystemEvent;
use Illuminate\Support\Facades\Log;

class LogFileSystemEvent
{
    public function handle(FileSystemEvent $event)
    {
        Log::info("FileSystemEvent received", [
            'event' => get_class($event),
```

```php
            'path' => $event->path,
            'origin' => $event->origin
        ]);
    }
}
```

Folder app\Console\Commands

```php
app\Console\Commands\InitialFileSystemScan.php
<?php

namespace App\Console\Commands;

use App\Services\FileSystemScanner;
use Illuminate\Console\Command;
use Illuminate\Support\Facades\DB;

class InitialFileSystemScan extends Command
{
    protected $signature = 'filesystem:scan
                            {path : The path to scan}
                            {--no-progress : Disable progress bar}';

    protected $description = 'Perform initial scan of file system and create events';

        // Constructor to set custom help message
    public function __construct()
    {
        parent::__construct();

        $this->setHelp(<<<'HELP'
          DESCRIPTION:
            Perform the initial scan of a directory and record filesystem events in the database.
            This command will:
```

- Recursively scan the given directory
  - Generate events for every file and directory found
  - Store these events in the `stored_events` table
  - Display statistics and sample events at completion
  💡 Use for initial setup - not recommended for active systems.
USAGE:
  php artisan filesystem:scan <path> [options]
ARGUMENTS:
  path   Absolute path to scan (required)
OPTIONS:
  --no-progress   Disable progress bar display (useful for CI environments)
OUTPUT:
  - Progress bar showing current file being processed (unless disabled)
  - Summary table with metrics:
      • Directories found
      • Files found
      • Total size
      • Errors encountered
      • Events created
      • Duration
      • Items per second
  - Sample of last 5 events created
SCANNING BEHAVIOR:
  - Processes both files and directories
  - Follows symbolic links
  - Skips unreadable paths (counted as errors)
  - Records creation events for all found items
SAMPLE OUTPUT:
  Starting initial file system scan...

Path: /var/www
✅ Initial scan completed successfully!

Metric	Value	
Directories found	1,024	
Files found	12,345	
Total size	1.23 GB	
Errors	3	
Events created	13,369	
Duration	5.2 seconds	
Items per second	2,571	

📋 Sample events created (last 5):

Event	Path	Type	Created	
FileCreated	image.jpg	file	2023-01-01 12:34:56	
DirCreated	documents	directory	2023-01-01 12:34:55	

EXAMPLES:
  1. Scan with progress bar:
     php artisan filesystem:scan /home/user/documents
  2. Scan without progress bar:
     php artisan filesystem:scan /mnt/data --no-progress
NOTES:
  - Requires write permission to the database
  - Large directories may take significant time
  - Check error count for accessibility issues

```php
            HELP
                );
    }

    public function handle()
    {
        $path = $this->argument('path');
        $showProgress = !$this->option('no-progress');

        $this->info("Starting initial file system scan...");
        $this->info("Path: {$path}");
        $this->newLine();

        $scanner = new FileSystemScanner($path);

        $progressBar = null;
        if ($showProgress) {
            $progressBar = $this->output->createProgressBar();
            $progressBar->setFormat(' %current%/%max% [%bar%] %percent:3s%% - %message%');
        }

        $startTime = microtime(true);
        $eventCountBefore = DB::table('stored_events')->count();

        try {
            $stats = $scanner->scan(function ($progress, $current, $total, $currentPath) use ($progressBar,
$showProgress) {
                if ($showProgress && $progressBar) {
                    $progressBar->setMaxSteps($total);
```

```php
                    $progressBar->setProgress($current);
                    $progressBar->setMessage(basename($currentPath));
                }
            });

            if ($showProgress && $progressBar) {
                $progressBar->finish();
                $this->newLine(2);
            }

            $endTime = microtime(true);
            $duration = round($endTime - $startTime, 2);
            $eventCountAfter = DB::table('stored_events')->count();
            $eventsCreated = $eventCountAfter - $eventCountBefore;

            // Display results
            $this->displayResults($stats, $duration, $eventsCreated);

            // Show sample events
            $this->showSampleEvents();

        } catch (\Exception $e) {
            $this->error("Scan failed: " . $e->getMessage());
            return Command::FAILURE;
        }

        return Command::SUCCESS;
    }
```

```php
private function displayResults(array $stats, float $duration, int $eventsCreated): void
{
    $this->info("✅ Initial scan completed successfully!");
    $this->newLine();

    $this->table(
        ['Metric', 'Value'],
        [
            ['Directories found', number_format($stats['directories'])],
            ['Files found', number_format($stats['files'])],
            ['Total size', $this->formatBytes($stats['total_size'])],
            ['Errors', $stats['errors']],
            ['Events created', number_format($eventsCreated)],
            ['Duration', "{$duration} seconds"],
            ['Items per second', $stats['directories'] + $stats['files'] > 0 ?
                round(($stats['directories'] + $stats['files']) / $duration) : 0],
            ['Event types', 'Directory: '.$stats['directories'].', File: '.$stats['files']],
        ]
    );
}

private function showSampleEvents(): void
{
    $this->newLine();
    $this->info("📋 Sample events created (last 5):");

    $sampleEvents = DB::table('stored_events')
        ->where('event_class', 'like', '%FileSystem%') // Filter filesystem events
        ->orderBy('id', 'desc')
```

```php
            ->limit(5)
            ->get(['event_class', 'event_properties', 'created_at']);

        if ($sampleEvents->isEmpty()) {
            $this->warn("No events found.");
            return;
        }

        $tableData = $sampleEvents->map(function ($event) {
            $properties = json_decode($event->event_properties, true);

            return [
                'Event' => class_basename($event->event_class),
                'Path' => $properties['path'] ? basename($properties['path']) : 'N/A',
                'Type' => $properties['type'] ?? 'N/A',
                'Created' => $event->created_at,
            ];
        })->toArray();

        $this->table(
            ['Event', 'Path', 'Type', 'Created'],
            $tableData
        );
    }

    private function formatBytes(int $bytes): string
    {
        $units = ['B', 'KB', 'MB', 'GB', 'TB'];
```

```php
        for ($i = 0; $bytes > 1024 && $i < count($units) - 1; $i++) {
            $bytes /= 1024;
        }

        return round($bytes, 2) . ' ' . $units[$i];
    }
}
```

app\Console\Commands\MonitorStoredEvents.php

```php
<?php

namespace App\Console\Commands;

use Illuminate\Console\Command;
use Spatie\EventSourcing\StoredEvents\Models\EloquentStoredEvent;

class MonitorStoredEvents extends Command
{
    protected $signature = 'events:monitor-db
                            {--delay=1 : Seconds to wait after detecting an event}
                            {--last=5 : Display last N events before monitoring}';

    protected $description = 'Monitor the stored_events table for real-time filesystem event persistence
verification';

    private $shouldExit = false;

    public function __construct()
    {
```

```
parent::__construct();

$this->setHelp(<<<'HELP'
  DESCRIPTION:
    Monitor the stored_events table to verify real-time persistence of filesystem events.
    Displays new events as they occur in the database with color-coded event types.
    Features:
    - Shows historical events before starting live monitoring
    - Color-coded event types for quick identification:
      • Green: File/Directory creation
      • Blue: File modification
      • Red: File/Directory deletion
    - Normalizes Windows paths to Unix-style
    - Graceful exit with Ctrl+C
  USAGE:
    php artisan events:monitor-db [options]
  OPTIONS:
    --delay=<seconds>   Delay between processing events (minimum 0.5s) [default: 1s]
    --last=<number>     Show last N historical events before monitoring [default: 5]
  EXAMPLES:
    Start monitoring with default settings:
      php artisan events:monitor-db
    Monitor with custom settings (show last 3 events, 0.5s delay):
      php artisan events:monitor-db --last=3 --delay=0.5
  OUTPUT FORMAT:
    [LIVE] [HH:MM:SS] <ICON> <COLORED_EVENT_TYPE>: <PATH>
    [HIST] [HH:MM:SS] <ICON> <COLORED_EVENT_TYPE>: <PATH>
    Prefixes:
      [LIVE] - Real-time events detected during monitoring
```

```
             [HIST] - Historical events shown at startup
           Icons:
               📄  - File event
               📁  - Directory created
               🗑️  - Directory deleted
               🔄  - File modified
               ❌  - File deleted
               ❓  - Unknown event type
         COLOR SCHEME:
            \e[32mGreen\e[0m  - File/Directory creation
            \e[34mBlue\e[0m   - File modification
            \e[31mRed\e[0m    - File/Directory deletion
         HELP
      );
   }

   public function handle()
   {
       // Setup signal handler for graceful exit
       if (function_exists('pcntl_async_signals')) {
           pcntl_async_signals(true);
           pcntl_signal(SIGINT, fn() => $this->shouldExit = true);
       }

       $lastId = EloquentStoredEvent::max('id') ?? 0;
       $delay = max(0.5, (float)$this->option('delay'));
       $showLast = max(0, (int)$this->option('last'));

       $this->info("Monitoring stored_events table. Press Ctrl+C to exit.");
```

```php
$this->line("Initial last event ID: $lastId");
$this->line("Delay after event: {$delay}s");
$this->line("Displaying last {$showLast} events");
$this->line(str_repeat('-', 60));

// Display recent events if requested
if ($showLast > 0) {
    $this->displayRecentEvents($showLast);
    $this->line(str_repeat('-', 60));
}

while (!$this->shouldExit) {
    $events = EloquentStoredEvent::where('id', '>', $lastId)
        ->orderBy('id')
        ->get();

    if ($events->isNotEmpty()) {
        foreach ($events as $event) {
            if ($this->shouldExit) break 2;

            $this->displayEvent($event);
            $lastId = $event->id;

            // Add delay after each event
            usleep((int)($delay * 1000000));
        }
    } else {
        if ($this->shouldExit) break;
        usleep(500000);  // 0.5s sleep when no events
```

```php
        }
    }

    $this->newLine();
    $this->info('Monitoring stopped.');
}

protected function displayRecentEvents(int $count)
{
    $events = EloquentStoredEvent::orderBy('id', 'desc')
        ->take($count)
        ->get()
        ->reverse();

    if ($events->isEmpty()) {
        $this->line('No historical events found');
        return;
    }

    $this->line("=== LAST {$count} EVENTS ===");
    foreach ($events as $event) {
        $this->displayEvent($event, true);
    }
}

protected function displayEvent(EloquentStoredEvent $event, bool $isHistorical = false)
{
    $properties = $event->event_properties;
    $path = $properties['path'] ?? '';
```

```php
        // Normalize Windows paths
        $path = str_replace('\\', '/', $path);

        // Extract both date and time from the datetime string
        $date = substr($event->created_at, 0, 10); // yyyy-mm-dd
        $time = substr($event->created_at, 11, 8); // HH:ii:ss

        $prefix = $isHistorical ? '[HIST] ' : '[LIVE] ';

        $this->line(sprintf(
            "{$prefix}[%s %s] %s: %s",
            $date,
            $time,
            $this->getColoredEventType($event->event_class),
            $path
        ));
    }

    protected function getColoredEventType(string $className): string
    {
        $type = $this->getEventType($className);

        // Apply colors based on event type
        if (str_contains($type, 'CREATED')) {
            return "<fg=green>$type</>";
        } elseif (str_contains($type, 'MODIFIED')) {
            return "<fg=blue>$type</>";
        } elseif (str_contains($type, 'DELETED')) {
```

```php
            return "<fg=red>$type</>";
        }

        return $type;
    }

    protected function getEventType(string $className): string
    {
        return match (true) {
            str_contains($className, 'FileCreated') => '📄 FILE CREATED',
            str_contains($className, 'FileDeleted') => '❌ FILE DELETED',
            str_contains($className, 'FileModified') => '🔄 FILE MODIFIED',
            str_contains($className, 'DirectoryCreated') => '📁 DIRECTORY CREATED',
            str_contains($className, 'DirectoryDeleted') => '🗑️ DIRECTORY DELETED',
            default => '❓ ' . class_basename($className)
        };
    }
}
```

app\Console\Commands\ReconcileFileSystem.php
```php
<?php

namespace App\Console\Commands;

use App\Services\FileSystemReconciler;
use Illuminate\Console\Command;
use Illuminate\Support\Facades\Log;

class ReconcileFileSystem extends Command
```

```php
{
    protected $signature = 'filesystem:reconcile
                            {path : The path to reconcile}
                            {--skip-scan : Skip filesystem scan, use last state}';

    protected $description = 'Reconcile file system state with event history';

    // Constructor to set custom help message
    public function __construct()
    {
        parent::__construct();

        $this->setHelp(<<<'HELP'
DESCRIPTION:
  Reconcile the current file system state with stored event history.
  This command will:
  1. Scan the target directory (unless skipped)
  2. Compare current state against stored events
  3. Identify discrepancies (missing/modified files)
  4. Generate reconciliation events to fix inconsistencies
  5. Output summary statistics
  💡 Recommended for periodic maintenance and data integrity checks.
WORKFLOW:
  [Scan Phase]      : Scan file system (optional, unless --skip-scan used)
  [Comparison Phase]: Compare against last known state from events
  [Reconciliation]  : Generate events to fix discrepancies
  [Reporting]       : Show results table
USAGE:
  php artisan filesystem:reconcile <path> [options]
```

```
ARGUMENTS:
  path            Absolute path to reconcile (required)
OPTIONS:
  --skip-scan     Use last scan state instead of re-scanning (faster)
OUTPUT:
  - Summary table with reconciliation metrics:
      • Items scanned
      • Discrepancies found
      • Events created
      • Duration
      • Items per second
EXAMPLES:
  1. Full reconciliation with new scan:
     php artisan filesystem:reconcile /var/www
  2. Fast reconciliation using last scan state:
     php artisan filesystem:reconcile /mnt/data --skip-scan
SAMPLE OUTPUT:
  Starting file system reconciliation...
  Path: /var/www
  ✅ Reconciliation completed successfully!

  | Metric                        | Value    |     |
  |-------------------------------|----------|-----|
  | Items scanned                 | 15,342   |     |
  | Discrepancies found           | 27       |     |
  | Reconciliation events created | 27       |     |
  | Duration                      | 8.3 sec  |     |
  | Items per second              | 1,848    |     |
```

```php
NOTES:
  - Without --skip-scan: Does fresh scan (slower but more accurate)
  - With --skip-scan: Uses last scan state (faster but requires recent scan)
  - Discrepancies include: missing files, modified files, orphan events
  - Created events will fix inconsistencies in the event history
  - Run during low-traffic periods for large directories
HELP
        );
    }

    public function handle()
    {
        $path = $this->argument('path');

        $this->info("Starting file system reconciliation...");
        $this->line("Path: {$path}");
        $this->newLine();

        $startTime = microtime(true);

        try {
            $reconciler = new FileSystemReconciler($path);
            $result = $reconciler->execute();

            $endTime = microtime(true);
            $duration = round($endTime - $startTime, 2);

            $this->displayResults($result, $duration);
```

```php
        } catch (\Exception $e) {
            $this->error("Reconciliation failed: " . $e->getMessage());
            Log::error("Reconciliation failed", ['error' => $e->getMessage()]);
            return Command::FAILURE;
        }

        return Command::SUCCESS;
    }

    private function displayResults(array $result, float $duration): void
    {
        $this->info("✅ Reconciliation completed successfully!");
        $this->newLine();

        $this->table(
            ['Metric', 'Value'],
            [
                ['Items scanned', number_format($result['scanned'])],
                ['Discrepancies found', number_format($result['discrepancies'])],
                ['Reconciliation events created', number_format($result['events_created'])],
                ['Duration', "{$duration} seconds"],
                ['Items per second', $result['scanned'] > 0 ? round($result['scanned'] / $duration) : 0],
            ]
        );
    }
}
```

app\Console\Commands\TestEventDispatch.php
```php
<?php
```

```php
namespace App\Console\Commands;

use App\Events\FileSystem\FileCreated;
use Illuminate\Console\Command;
use Illuminate\Support\Facades\Log;

class TestEventDispatch extends Command
{
    protected $signature = 'test:event';
    protected $description = 'Test event dispatching';

    public function handle()
    {
        Log::info("Dispatching test event");

        event(new FileCreated(
            '/test/path.txt',
            'test',
            'hash123',
            now(),
            1024
        ));

        Log::info("Test event dispatched");

        $this->info("✅ Test event dispatched. Check logs.");
        return 0;
    }
}
```

```
}
```

app\Console\Commands\WatchFileSystem.php

```php
<?php

namespace App\Console\Commands;

use App\Services\FileSystemWatcher;
use Illuminate\Console\Command;
use Illuminate\Support\Facades\Log;

class WatchFileSystem extends Command
{
    protected $signature = 'filesystem:watch
                            {path : The path to watch}
                            {--timeout=0 : Stop watching after N seconds (0 = infinite)}';

    protected $description = 'Watch file system for real-time changes';

    public function handle()
    {
        $path = $this->argument('path');
        $timeout = (int) $this->option('timeout');

        if (!is_dir($path)) {
            $this->error("Path does not exist or is not a directory: {$path}");
            return Command::FAILURE;
        }
```

```php
$this->info("🔍 Starting file system watcher...");
$this->info("📁 Watching: {$path}");

if ($timeout > 0) {
    $this->info("⏱️  Timeout: {$timeout} seconds");
} else {
    $this->info("⏱️  Running indefinitely (Ctrl+C to stop)");
}

$this->newLine();

// Set up signal handling for graceful shutdown
if (extension_loaded('pcntl')) {
    pcntl_signal(SIGTERM, [$this, 'gracefulShutdown']);
    pcntl_signal(SIGINT, [$this, 'gracefulShutdown']);
}

try {
    $watcher = new FileSystemWatcher($path);

    // Set timeout if specified
    if ($timeout > 0) {
        $this->setTimeoutAlarm($timeout);
    }

    $this->info("✅ Watcher started. Monitoring for changes...");
    $this->displayInstructions();

    // Start watching (this will block)
```

```php
            $watcher->start();

        } catch (\Exception $e) {
            $this->error("❌ Watcher failed: " . $e->getMessage());
            Log::error("File system watcher failed", [
                'path' => $path,
                'error' => $e->getMessage(),
                'trace' => $e->getTraceAsString()
            ]);
            return Command::FAILURE;
        }

        return Command::SUCCESS;
    }

    private function displayInstructions(): void
    {
        $this->newLine();
        $this->line("📋 <fg=yellow>Instructions:</fg=yellow>");
        $this->line("   • Create, modify, or delete files/folders in the watched directory");
        $this->line("   • Check the logs for real-time event tracking");
        $this->line("   • Use Ctrl+C to stop watching gracefully");
        $this->line("   • Check stored_events table to see captured events");
        $this->newLine();
    }

    private function setTimeoutAlarm(int $seconds): void
    {
        if (extension_loaded('pcntl')) {
```

```php
        pcntl_alarm($seconds);
        pcntl_signal(SIGALRM, function () {
            $this->info("⏰ Timeout reached. Stopping watcher...");
            exit(0);
        });
    }
}

public function gracefulShutdown(): void
{
    $this->newLine();
    $this->info("🛑 Shutting down file system watcher gracefully...");
    $this->info("✅ Watcher stopped.");
    exit(0);
}
}
```

Folder app\Projectors

3 printable files

(file list disabled)

app\Projectors\FileSystemProjector.php

```php
<?php

namespace App\Projectors;

use App\Events\FileSystem\{
    DirectoryCreated, DirectoryDeleted,
    FileCreated, FileDeleted, FileModified,
    FileSystemEvent
};
use App\Models\File;
use Spatie\EventSourcing\EventHandlers\Projectors\Projector;
```

```php
class FileSystemProjector extends Projector
{
    /* ========= Event hooks ========= */

    public function onFileCreated(FileCreated $event): void
    {
        $this->update($event);
    }

    public function onDirectoryCreated(DirectoryCreated $event): void
    {
        $this->update($event, true);
    }

    public function onFileModified(FileModified $event): void
    {
        $this->update($event); // just overwrite hash / size / modified_at
    }
}
```

```php
public function onFileDeleted(FileDeleted $event): void

{

File::where('path', $event->path)->delete();

}


public function onDirectoryDeleted(DirectoryDeleted $event): void

{

// delete directory itself *and* everything below it

File::where('path', 'like', $event->path.'%')->delete();

}


/* ========= Helpers ========= */


private function update(FileSystemEvent $event, bool $isDir = false): void

{

/* —— Skip entire path if it lives inside an omitted directory —— */

if ($this->shouldSkipPath($event->path)) {
```

```php
    return; // ignore completely
}


[$name, $ext] = $this->splitNameExt($event->path);


/* ——— skip unwanted file types ------------------------------------- */
if (!$isDir && $this->shouldSkipExtension($ext)) {
    return;
}



File::updateOrCreate(
    ['path' => $event->path],
    [
        'name' => $name,
        'file_type' => $isDir ? 'directory' : 'file',
        'extension' => $isDir ? null : ltrim($ext, '.'),
        'revision' => $isDir ? null : $this->extractRevision($name),
```

```php
            'part_name' => $isDir ? null : $this->extractPartName($name),

            'core_name' => $isDir ? null : $this->extractCoreName($name),

            'product_main_type' => $this->segment($event->path, 0),

            'product_sub_type' => $this->segment($event->path, 1, true),

            'parent' => $this->parentFolder($event->path),

            'parent_path' => $this->parentPath($event->path),

            'depth' => substr_count($event->path, '/'),

            'origin' => $event->origin,

            'content_hash' => $event->hash ?? null,

            'size' => $event->size ?? null,

            'modified_at' => $event->modifiedAt ?? now(),

        ]

    );

}


/* ========= Files & Folders Omitting utilities ========= */


/**
```

* Return TRUE when the file extension is in the omit list.

*/

```php
private function shouldSkipExtension(string $ext): bool

{

$clean = strtolower(ltrim($ext, '.'));


return in_array(

$clean,

array_map('strtolower', config('projectors.filesystem.omit_extensions', [])),

true

);

}
```

/**

* Return TRUE when any segment of the given path

* • equals an "omit_directories" entry, OR

* • starts with one of the "omit_directory_prefixes".

*/

```php
private function shouldSkipPath(string $path): bool
{
$segments = explode('/', $path);

$omit = array_map('strtolower', config('projectors.filesystem.omit_directories', []));

$prefix = array_map('strtolower', config('projectors.filesystem.omit_directory_prefixes', []));

foreach ($segments as $seg) {
$seg = strtolower($seg);

// exact directory names
if (in_array($seg, $omit, true)) {
return true;
}

// prefixes (e.g. everything that starts with '00')
foreach ($prefix as $p) {
if ($p !== '' && str_starts_with($seg, $p)) {
```

```php
            return true;

        }

    }

}

return false;

}


/* ========= Parsing utilities ========= */


private function splitNameExt(string $path): array

{

$basename = basename($path);

$pos = strrpos($basename, '.');

return $pos === false

? [$basename, '']

: [substr($basename, 0, $pos), substr($basename, $pos)];

}
```

```php
/**
 * Return the revision (e.g. "revA", "v1", "20250617") or NULL when the tail
 * does not match one of the allowed patterns[1].
 *
 * ── How the unified regular expression works ───────────────────────────────
 * 1. rev[0-9a-z]+ → matches strings that start with "rev" followed by
 * one or more digits or lowercase letters, e.g. "revA", "rev01"[1].
 * 2. v[0-9]+[a-z]* → matches strings that start with "v" (upper- or
 * lower-case) followed by one or more digits and *optionally* a
 * trailing letter sequence, e.g. "v1", "V1", "v2b", "v10a"[2].
 * 3. [0-9]{8} → matches exactly eight consecutive digits, which we
 * use for dates like "20250617" (YYYYMMDD)[3].
 * 4. The /i modifier makes the whole pattern case-insensitive, so
 * "_V1" and "_v1" are treated the same[4].
 */
private function extractRevision(string $name): ?string
{
$pos = strrpos($name, '_'); // last underscore[1]
```

```php
if ($pos === false) { // none => no revision[1]

return null; // early exit[1]

}


$token = substr($name, $pos + 1); // candidate after "_"[2]


// Allowed patterns combined into one regular expression[2]

$isRevision = preg_match(

'/^(?:rev[0-9a-z]+|v[0-9]+[a-z]*|[0-9]{8})$/i', // unified regex[2]

$token

);


return $isRevision ? $token : null; // keep or ignore[2]

}


/**

* Strip "_revision" only when one was actually detected[1].

*/
```

```php
private function extractPartName(string $name): string

{

$rev = $this->extractRevision($name); // reuse logic[2]

return $rev !== null

? substr($name, 0, -strlen('_' . $rev)) // cut off tail[1]

: $name; // leave intact[1]

}
/**

* Turn something like "SUB-TYPE-1_ PART TWO _revA.pdf" into "PART TWO".

*/

private function extractCoreName(string $fileName): ?string

{

// Extract the part before revision or file extension

$part = $this->extractPartName($fileName); // e.g. "SUB-TYPE-1_ PART TWO"


if ($part === null) {

return null;

}
```

```php
// Find the first underscore and take everything after it

$pos = strpos($part, '_');

if ($pos === false) {

return trim($part); // Return the whole part if no underscore is found

}


// Trim leading/trailing spaces and return the core name

return trim(substr($part, $pos + 1)); // Keep the internal spaces intact

}


/**

* segment(…, 0) == MAIN-TYPE-N

* segment(…, 1) == first subtype, segment(…, 2) the next …

* If $concat = true, concatenate all subtypes with "/".

*/

private function segment(string $path, int $index, bool $concat = false): ?string

{
```

```php
$segments = explode('/', $path); // already relative

if ($index === 0) {

return $segments[0] ?? null;

}


$subs = array_slice($segments, 1, $concat ? null : 1);

return $subs ? ($concat ? implode('/', $subs) : $subs[0]) : null;

}


/**

* Return the immediate parent directory name or NULL when the file/folder

* is stored at the repository root.

*

* examples
——————————————————————————————————————

* "A/B/C.txt" → "B"

* "A/B" (directory) → "A"

* "rootFile.txt" → null
```

```php
    */
    private function parentFolder(string $path): ?string
    {
        if (!str_contains($path, '/')) {
            return null; // file/dir lives at repo root
        }


        return basename(dirname($path)); // "B"
    }


    /** Full path to the parent folder ("A/B" in "A/B/C.txt") or NULL at root. */
    private function parentPath(string $path): ?string
    {
        if (!str_contains($path, '/')) {
            return null;
        }


        $parent = dirname($path); // "A/B"
```

```php
        return $parent === '.' ? null : $parent;

    }

}
```

app\Projectors\MasterFilesProjector.php

```php
<?php

namespace App\Projectors;

use App\Events\FileSystem\{

DirectoryDeleted, // we only need delete events for directories

FileCreated, FileModified,

FileDeleted,

FileSystemEvent

};

use App\Models\File; // read-only source
```

```php
use App\Models\Master; // write model

use Illuminate\Support\Facades\Config;

use Spatie\EventSourcing\EventHandlers\Projectors\Projector;


class MasterFilesProjector extends Projector
{
/* ========= Projector Order ========= */

public int $weight = 2; //Projectors with a lower weight run first


/* ========= Event hooks ========= */


public function onFileCreated(FileCreated $event): void
{
// early-out when path lives in an omitted directory
if ($this->shouldSkipPath($event->path)) {
return;
}
```

```php
        $this->refreshForPath($event->path);
    }

    public function onFileModified(FileModified $event): void
    {
        // early-out when path lives in an omitted directory
        if ($this->shouldSkipPath($event->path)) {
            return;
        }

        $this->refreshForPath($event->path);
    }

    public function onFileDeleted(FileDeleted $event): void
    {
        // remove a vanished master **or** a vanished slave
        Master::where('path', $event->path)
            ->orWhere('slave_path', $event->path)
```

```php
    ->delete();

    // a slave deletion may invalidate other masters
    $this->refreshNeighbouringMasters($event->path);
}


public function onDirectoryDeleted(DirectoryDeleted $event): void
{
    // drop everything inside the deleted tree
    Master::where('path', 'like', $event->path.'%')
        ->orWhere('slave_path','like', $event->path.'%')
        ->delete();
}


/* ========= Core logic ========= */

private function refreshForPath(string $path): void
{
```

```php
$file = File::where('path', $path)->first();

if (!$file) {

return; // race-condition guard – FileSystemProjector not committed yet

}


if ($this->isMasterExt($file->extension)) {

$this->evaluateMaster($file); // (re-)create or delete

}


if ($this->isSlaveExt($file->extension)) {

// a new / changed slave may create new masters in the folder

$candidates = File::query()

->where('parent_path', $file->parent_path)

->whereIn('extension', Config::get('projectors.masterfiles.master_extensions', []))

->where('part_name', $file->part_name)

->get();


foreach ($candidates as $candidate) {
```

```php
        $this->evaluateMaster($candidate);

    }

}

}


/**

 * Check whether the given File *qualifies* as a master

 * (i.e. has an accompanying slave in the same folder) and

 * update the `masters` table accordingly.

 */

private function evaluateMaster(File $master): void

{

// Never touch files that live in an omitted path

if ($this->shouldSkipPath($master->path)) {

Master::where('path', $master->path)->delete(); // clean up if it was inserted earlier

return;

}
```

```php
$slave = $this->locateSlave($master);


if ($slave) {

Master::updateOrCreate(

['path' => $master->path],

[

'master_revision' => $master->revision,

'parent_path' => $master->parent_path,

'content_hash' => $master->content_hash,

'slave_path' => $slave->path,

'slave_revision' => $slave->revision,

'modified_at' => $master->modified_at,

'part_name' => $master->part_name,

'extension' => $master->extension,

],

);

} else {

// No slave any more → not a master
```

```php
Master::where('path', $master->path)->delete();

}

}


/** Search for a slave (.pdf, …) that lives in the same folder and

* shares the same *part_name*.

*/

private function locateSlave(File $master): ?File

{

return File::query()

->where('parent_path', $master->parent_path)

->where('part_name', $master->part_name)

->whereIn(

'extension',

Config::get('projectors.masterfiles.slave_extensions', [])

)

->orderByDesc('revision')

->get() // fetch all candidates in that order
```

```php
    ->first(fn ($f) => // keep the first one that
        !$this->shouldSkipPath($f->path) // isn't in an omitted dir
    );
}


private function refreshNeighbouringMasters(string $deletedPath): void
{
    $dir = str_contains($deletedPath, '/') ? dirname($deletedPath) : null;
    if (!$dir) {
        return;
    }


    // grab every candidate master in that folder and re-evaluate
    $candidates = File::query()
        ->where('parent_path', $dir)
        ->whereIn('extension', Config::get('projectors.masterfiles.master_extensions', []))
        ->get();
```

```php
foreach ($candidates as $candidate) {

$this->evaluateMaster($candidate);

}

}


/* ========= Helpers ========= */


private function isMasterExt(?string $ext): bool

{

return in_array(strtolower($ext ?? "),
Config::get('projectors.masterfiles.master_extensions', []), true);

}


private function isSlaveExt(?string $ext): bool

{

return in_array(strtolower($ext ?? "), Config::get('projectors.masterfiles.slave_extensions',
[]), true);

}
```

```php
/* ========= Path-omitting helper ========= */

private function shouldSkipPath(string $path): bool

{

$segments = explode('/', $path);


$omit = array_map('strtolower',

Config::get('projectors.masterfiles.omit_directories', [])

);

$prefix = array_map('strtolower',

Config::get('projectors.masterfiles.omit_directory_prefixes', [])

);


foreach ($segments as $seg) {

$seg = strtolower($seg);


if (in_array($seg, $omit, true)) {

return true; // full match
```

```php
    }


    foreach ($prefix as $p) { // prefix match

    if ($p !== '' && str_starts_with($seg, $p)) {

    return true;

    }

    }

    }

    return false;

    }

    }
```

app\Projectors\PartsProjector.php

```php
    <?php


    namespace App\Projectors;
```

```php
use App\Events\FileSystem\{
FileCreated, FileModified, FileDeleted,
DirectoryDeleted
};
use App\Models\File; // read-only
use App\Models\Master; // read-only
use App\Models\Part; // write
use Illuminate\Support\Facades\Config;
use Spatie\EventSourcing\EventHandlers\Projectors\Projector;


class PartsProjector extends Projector
{
/* run after MasterFilesProjector (weight = 2) */
public int $weight = 3;


/* ===== Event hooks ===================================== */
```

```php
public function onFileCreated(FileCreated $event): void
{
$this->dispatchRefreshes($event->path);
}


public function onFileModified(FileModified $event): void
{
$this->dispatchRefreshes($event->path);
}


public function onFileDeleted(FileDeleted $event): void
{
/* 1. remove the row when the deleted file *was* a part */
Part::where('path', $event->path)->delete();


/* 2. if a slave disappeared, the related parts lose their link */
$file = File::where('path', $event->path)->first();
if ($file && $this->isSlaveExt($file->extension)) {
```

```php
        $this->refreshByPartName($file->part_name);

    }

}


public function onDirectoryDeleted(DirectoryDeleted $event): void

{

Part::where('path', 'like', $event->path.'%')->delete();

}


/* ——— Dispatcher
————————————————————————————————————————————— */


/**

* Decide whether the incoming path is

* • a "part" → refresh itself

* • a "slave" → refresh every sibling part that shares the part_name

*/

private function dispatchRefreshes(string $path): void
```

```php
{
    if ($this->shouldSkipPath($path)) {
        return;
    }

    $file = File::where('path', $path)->first();
    if (!$file) {
        return; // FileSystemProjector not committed yet
    }

    if ($this->isPartExt($file->extension)) {
        $this->refreshForPath($file->path);
    }

    if ($this->isSlaveExt($file->extension)) {
        $this->refreshByPartName($file->part_name);
    }
}
```

```php
private function refreshByPartName(string $partName): void

{

Part::where('part_name', $partName)

->pluck('path')

->each(fn (string $p) => $this->refreshForPath($p));

}



/* ===== Core ============================================= */



private function refreshForPath(string $path): void

{

$file = File::where('path', $path)->first();

if (!$file || !$this->isPartExt($file->extension)) {

return;

}



/* Find a candidate master by name OR identical hash */
```

```php
$master = Master::query()
->where(function ($q) use ($file) {
$q->where('part_name', $file->part_name)
->orWhere('content_hash', $file->content_hash);
})
->orderByDesc('slave_revision') // newest slave first
->first();

$contentMatches = $master
&& $file->content_hash
&& $file->content_hash === $master->content_hash;

Part::updateOrCreate(
['path' => $file->path],
[
'part_name' => $file->part_name,
'core_name' => $file->core_name,
'parent_path' => $file->parent_path,
```

```php
            'parent' => $file->parent,

            'extension' => $file->extension,

            'master_path' => $master?->path,

            'master_revision' => $master?->master_revision,

            'slave_path' => $master?->slave_path,

            'slave_revision' => $master?->slave_revision,

            'content_hash' => $file->content_hash,

            'content_as_master' => $contentMatches,

            'modified_at' => $file->modified_at,

        ]
    );
}


/* ===== Helpers ========================================= */


private function isPartExt(?string $ext): bool
{
    return in_array(
```

```php
        strtolower($ext ?? ''),

        Config::get('projectors.parts.part_extensions', []),

        true

    ); // "par, asm, doc/x, xls/x" by default[1]

}


private function isSlaveExt(?string $ext): bool

{

    return in_array(

        strtolower($ext ?? ''),

        Config::get('projectors.masterfiles.slave_extensions', []),

        true

    ); // currently only "pdf"[1]

}


/**
 * TRUE when any path segment
 * • matches an entry in omit_directories, OR
```

```php
 * • starts with a prefix omit_directory_prefixes
 */
private function shouldSkipPath(string $path): bool
{
$segments = explode('/', $path);


$omit = array_map('strtolower', Config::get('projectors.parts.omit_directories', []));
$prefix = array_map('strtolower', Config::get('projectors.parts.omit_directory_prefixes', []));


foreach ($segments as $seg) {
$seg = strtolower($seg);


if (in_array($seg, $omit, true)) {
return true;
}


foreach ($prefix as $p) {
```

```
if ($p !== '' && str_starts_with($seg, $p)) {

return true;

}

}

}

return false;

}

}
```

Folder config\projectors

2 printable files

(file list disabled)

config\projectors\filesystem.php

```php
<?php

return [

    /*
    |----------------------------------------------------------------------
    | Extensions that must be ignored
    |----------------------------------------------------------------------
    | Write them without the leading dot. Comparison is case-insensitive.
    */
    'omit_extensions' => [
        'cfg',
```

```php
    'db',

    // …

],


/*
|----------------------------------------------------------------------
| Directory names that must be ignored completely
|----------------------------------------------------------------------
| Example: build, _trash, debug … Case-insensitive.
*/
'omit_directories' => [

'build',

'debug',

// …

],


/*
|----------------------------------------------------------------------
```

| Directory name prefixes to omit

|--

| Each entry is compared with "startsWith". The most common request is

| to drop *every* directory that starts with "00".

*/

'omit_directory_prefixes' => [

'00', // ignore 00, 001_tmp, 00-old, 00Whatever …

// add more prefixes here …

],

];

config\projectors\masterfiles.php

```php
<?php


return [
```

```
/*

|------------------------------------------------------------

| Which extensions *may become* a "master" file?

|------------------------------------------------------------

| Order does not matter; use lowercase without the leading dot.

*/

'master_extensions' => [

'par', 'asm',

'doc', 'docx',

'xls', 'xlsx',

],


/*

|------------------------------------------------------------

| Which extensions are considered "slaves" (documents

| that must live in the *same folder* and share the same

| part-name to make the master valid)?

|------------------------------------------------------------
```

```php
    */
    'slave_extensions' => [
        'pdf',
    ],


    /* ─────────────────────────────────────────────────────────────

    | Omit logic
    | - Any folder whose name matches an entry in
    | "omit_directories" must be omitted
    | - Any folder whose name _starts with_ one of the prefixes
    | in "omit_directory_prefixes" must be omitted
    ───────────────────────────────────────────────────────────── */

    'omit_directories' => [ // exact names
        'ARCHIVO', 'MODIFICAR', '.git', '.svn',
    ],
```

```
'omit_directory_prefixes' => [ // anything that _starts with_

'00',

'_',

'.', // 00*, _*, .* …

],


];
```