LARA-FILES

We are working on a file-system to database 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.

Folder app\Events

```
app\Events\FileSystem\DirectoryCreated.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

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);
    }
}
```

app\Events\FileSystem\FileModified.php

```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;
        $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';
        }

        return 'unknown';
    }

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

}
```

Folder app\Services

app\Services\FileSystemReconciler.php

```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\DirectoryCreated;
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();
    $this->compareStates($currentState, $eventTimeline);

    $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
    ];
}

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')
      ->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;
}

private function compareStates(array $currentState, array $eventTimeline): void
{
    foreach ($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'];

    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;

    foreach ($this->discrepancies as $path => $discrepancy) {
        try {
            switch ($discrepancy['reason']) {
                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'],
                        $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;
}
```

```php
    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;
        }
    }

    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;
    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, '/\\');
    }
```

```php
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);

    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);
                    $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);
    }
}

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());
```

```php
    $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 $filePath): ?string
{
    try {
        // For large files, we might want to use a more efficient method
        if (filesize($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;
```

```php
        }
    }

    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;
    }
}
```

app\Services\FileSystemWatcher.php

```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;
```

```php
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]);

        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;
        }

        $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);

        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) . '...'
            ]);
        }

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

private function handleFileDeleted(string $absolutePath): void
{
```

```php
    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()
        ]);
    }
}

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
        );

        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()
        ]);
    }
}
```

```php
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;
    }
}

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

```
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,
        ]);
    }
```

```php
    private function determineFileType(FileSystemEvent $event): string
    {
        $eventClass = class_basename($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),
            'path' => $event->path,
            'origin' => $event->origin
        ]);
    }
}
```

Folder app\Console\Commands

```
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
      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);
                    $progressBar->setProgress($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;
}

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')
        ->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'];

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

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

app\Console\Commands\MonitorStoredEvents.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.");
        $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));
```

```php
// 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
    }
}

$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'] ?? '';

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

```php
    // Extract time portion from the datetime string
    $time = substr($event->created_at, 11, 8);

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

    $this->line(sprintf(
        "{$prefix}[%s] %s: %s",
        $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($type, 'DELETED')) {
        return "<fg=red>$type</>";
    }

    return $type;
}
```

```php
    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($class
```

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
{
    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        │
  └──────────────────────────────────┴──────────────┘
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);

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

```php
        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

namespace App\Console\Commands;

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

```php
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;
```

```php
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;
        }

        $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)
    $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 $seconds): void
{
    if (extension_loaded('pcntl')) {
        pcntl_alarm($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);
    }
}
```

sd