Fund Relocation Feature - Complete Documentation

Overview

This document provides comprehensive documentation for implementing the **Fund Relocation Feature** that allows unspent funds from completed projects to be relocated to new projects of the same type and from the same center. This feature ensures complete traceability, follows the existing approval workflow, and integrates seamlessly with the reporting structure.

Table of Contents

Business Requirements

Core Requirements

1. **Unspent Funds Identification**
    - Identify projects with unspent funds (where `amount_sanctioned - total_expenses > 0`)
    - Only consider projects with status `approved_by_coordinator`
    - Calculate remaining balance from approved reports only
2. **Eligibility Criteria**
    - **Same Project Type**: The source and destination projects must have the same `project_type`
    - **Same Center**: The applicant (user) of both projects must be from the same `center`
    - **Source Project Status**: Source project must be `approved_by_coordinator` with unspent funds
    - **Destination Project Status**: Destination project can be in any status (draft, submitted, forwarded, approved)
3. **Complete Workflow Integration**
    - Applicant creates new project and can select available unspent funds
    - Provincial can view and edit fund relocation details during review
    - Coordinator can view and edit fund relocation details during review
    - Once approved, funds are added to the new project's budget
    - Full traceability maintained for auditing

4. **Reporting Integration**
    - Relocated funds should appear in project budgets
    - Reports should track expenses from relocated funds
    - Budget statements should show source of funds (original project reference)
5. **Legacy/Old Projects Fund Management**
    - Allow manual entry of unspent funds from previous/legacy projects
    - These funds can be from projects in `oldDevelopmentProjects` table or external projects
    - Legacy funds should be available for relocation to new projects
    - Track source type (current system project vs legacy project vs external)
    - Full documentation and traceability for legacy funds

System Architecture

High-Level Flow

```
1. Applicant creates new project
   ↓
2. System checks for eligible unspent funds (same type, same center)
   ↓
3. Applicant selects source project(s) and amount(s) to relocate
   ↓
4. Project submitted to Provincial (with fund relocation details)
   ↓
5. Provincial reviews and can edit fund relocation
   ↓
6. Project forwarded to Coordinator (with fund relocation details)
   ↓
7. Coordinator reviews and can edit fund relocation
   ↓
8. Coordinator approves project
   ↓
9. System adds relocated funds to project budget
   ↓
10. Funds available for reporting and expense tracking
```

Component Structure

```
app/Http/Controllers/Projects/FundRelocation/
├── FundRelocationController.php          # Main controller
├── FundRelocationValidationController.php # Validation logic
├── FundRelocationApprovalController.php  # Approval processing
└── LegacyFundController.php              # Legacy funds management

resources/views/projects/partials/FundRelocation/
├── select_source_projects.blade.php      # Source project selection
├── fund_relocation_summary.blade.php     # Summary display
├── fund_relocation_edit.blade.php        # Edit form for approvers
└── fund_relocation_traceability.blade.php # Audit trail view

resources/views/fund-relocation/legacy/
├── create.blade.php                      # Create legacy fund entry
├── edit.blade.php                        # Edit legacy fund entry
├── list.blade.php                        # List all legacy funds
└── show.blade.php                        # View legacy fund details
```

Database Schema

New Table: `fund_relocations`

```
CREATE TABLE `fund_relocations` (
    `id` BIGINT UNSIGNED NOT NULL AUTO_INCREMENT,
    `fund_relocation_id` VARCHAR(255) NOT NULL UNIQUE,
    `destination_project_id` VARCHAR(255) NOT NULL,
    `source_type` ENUM('current_project', 'legacy_fund') DEFAULT
'current_project',
    `source_project_id` VARCHAR(255) NULL,
    `source_legacy_fund_id` VARCHAR(255) NULL,
    `amount_relocated` DECIMAL(10, 2) NOT NULL,
    `status` ENUM('pending', 'approved', 'rejected', 'cancelled') DEFAULT
'pending',
    `requested_by` BIGINT UNSIGNED NOT NULL,
    `requested_at` TIMESTAMP NOT NULL,
    `approved_by` BIGINT UNSIGNED NULL,
    `approved_at` TIMESTAMP NULL,
    `rejection_reason` TEXT NULL,
    `notes` TEXT NULL,
    `created_at` TIMESTAMP NULL,
    `updated_at` TIMESTAMP NULL,
    PRIMARY KEY (`id`),
    INDEX `idx_destination_project` (`destination_project_id`),
    INDEX `idx_source_project` (`source_project_id`),
    INDEX `idx_source_legacy_fund` (`source_legacy_fund_id`),
    INDEX `idx_source_type` (`source_type`),
    INDEX `idx_status` (`status`),
```

```
    FOREIGN KEY (`destination_project_id`) REFERENCES
`projects`(`project_id`) ON DELETE CASCADE,
    FOREIGN KEY (`source_project_id`) REFERENCES `projects`(`project_id`) ON
DELETE RESTRICT,
    FOREIGN KEY (`source_legacy_fund_id`) REFERENCES
`legacy_funds`(`legacy_fund_id`) ON DELETE RESTRICT,
    FOREIGN KEY (`requested_by`) REFERENCES `users`(`id`) ON DELETE
RESTRICT,
    FOREIGN KEY (`approved_by`) REFERENCES `users`(`id`) ON DELETE SET NULL,
    CHECK (
        (source_type = 'current_project' AND source_project_id IS NOT NULL
AND source_legacy_fund_id IS NULL) OR
        (source_type = 'legacy_fund' AND source_legacy_fund_id IS NOT NULL
AND source_project_id IS NULL)
    )
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

New Table: `fund_relocation_history`

```
CREATE TABLE `fund_relocation_history` (
    `id` BIGINT UNSIGNED NOT NULL AUTO_INCREMENT,
    `fund_relocation_id` VARCHAR(255) NOT NULL,
    `action` ENUM('created', 'edited', 'approved', 'rejected', 'cancelled')
NOT NULL,
    `performed_by` BIGINT UNSIGNED NOT NULL,
    `performed_at` TIMESTAMP NOT NULL,
    `old_amount` DECIMAL(10, 2) NULL,
    `new_amount` DECIMAL(10, 2) NULL,
    `notes` TEXT NULL,
    `created_at` TIMESTAMP NULL,
    `updated_at` TIMESTAMP NULL,
    PRIMARY KEY (`id`),
    INDEX `idx_fund_relocation` (`fund_relocation_id`),
    INDEX `idx_performed_by` (`performed_by`),
    FOREIGN KEY (`fund_relocation_id`) REFERENCES
`fund_relocations`(`fund_relocation_id`) ON DELETE CASCADE,
    FOREIGN KEY (`performed_by`) REFERENCES `users`(`id`) ON DELETE RESTRICT
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

Migration File

File:
`database/migrations/YYYY_MM_DD_HHMMSS_create_fund_relocations_tables.php`

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up(): void
    {
        Schema::create('fund_relocations', function (Blueprint $table) {
            $table->id();
            $table->string('fund_relocation_id')->unique();
            $table->string('destination_project_id');
            $table->enum('source_type', ['current_project', 'legacy_fund'])->default('current_project');
            $table->string('source_project_id')->nullable();
            $table->string('source_legacy_fund_id')->nullable();
            $table->decimal('amount_relocated', 10, 2);
            $table->enum('status', ['pending', 'approved', 'rejected', 'cancelled'])->default('pending');
            $table->unsignedBigInteger('requested_by');
            $table->timestamp('requested_at');
            $table->unsignedBigInteger('approved_by')->nullable();
            $table->timestamp('approved_at')->nullable();
            $table->text('rejection_reason')->nullable();
            $table->text('notes')->nullable();
            $table->timestamps();

            $table->foreign('destination_project_id')
                  ->references('project_id')
                  ->on('projects')
                  ->onDelete('cascade');

            $table->foreign('source_project_id')
                  ->references('project_id')
                  ->on('projects')
                  ->onDelete('restrict');

            $table->foreign('source_legacy_fund_id')
                  ->references('legacy_fund_id')
                  ->on('legacy_funds')
                  ->onDelete('restrict');

            $table->foreign('requested_by')
                  ->references('id')
                  ->on('users')
                  ->onDelete('restrict');

            $table->foreign('approved_by')
                  ->references('id')
                  ->on('users')
                  ->onDelete('set null');

            $table->index('destination_project_id');
            $table->index('source_project_id');
```

```
            $table->index('source_legacy_fund_id');
            $table->index('source_type');
            $table->index('status');
        });

        Schema::create('fund_relocation_history', function (Blueprint
$table) {
            $table->id();
            $table->string('fund_relocation_id');
            $table->enum('action', ['created', 'edited', 'approved',
'rejected', 'cancelled']);
            $table->unsignedBigInteger('performed_by');
            $table->timestamp('performed_at');
            $table->decimal('old_amount', 10, 2)->nullable();
            $table->decimal('new_amount', 10, 2)->nullable();
            $table->text('notes')->nullable();
            $table->timestamps();

            $table->foreign('fund_relocation_id')
                  ->references('fund_relocation_id')
                  ->on('fund_relocations')
                  ->onDelete('cascade');

            $table->foreign('performed_by')
                  ->references('id')
                  ->on('users')
                  ->onDelete('restrict');

            $table->index('fund_relocation_id');
            $table->index('performed_by');
        });
    }

    public function down(): void
    {
        Schema::dropIfExists('fund_relocation_history');
        Schema::dropIfExists('fund_relocations');
    }
};
```

Legacy Funds Management

Overview

The Legacy Funds Management feature allows administrators and coordinators to manually enter unspent funds from previous projects that were completed before the current system was implemented, or from external projects. These funds can then be utilized for relocation to new projects in the system.

Use Cases

1. **Old Development Projects**: Funds from projects in the `oldDevelopmentProjects` table
2. **External Projects**: Funds from projects managed outside this system
3. **Historical Projects**: Projects completed before system implementation
4. **Manual Entry**: When exact project details are not in the system but funds need to be tracked

Database Schema for Legacy Funds

New Table: `legacy_funds`

```
CREATE TABLE `legacy_funds` (
    `id` BIGINT UNSIGNED NOT NULL AUTO_INCREMENT,
    `legacy_fund_id` VARCHAR(255) NOT NULL UNIQUE,
    `source_type` ENUM('old_development_project', 'external_project',
'manual_entry') NOT NULL,
    `old_project_id` BIGINT UNSIGNED NULL,
    `project_title` VARCHAR(255) NOT NULL,
    `project_type` VARCHAR(255) NOT NULL,
    `user_id` BIGINT UNSIGNED NOT NULL,
    `center` VARCHAR(255) NOT NULL,
    `amount_available` DECIMAL(10, 2) NOT NULL,
    `amount_sanctioned` DECIMAL(10, 2) NULL,
    `amount_utilized` DECIMAL(10, 2) DEFAULT 0.00,
    `amount_relocated` DECIMAL(10, 2) DEFAULT 0.00,
    `commencement_date` DATE NULL,
    `completion_date` DATE NULL,
    `society_name` VARCHAR(255) NULL,
    `in_charge_name` VARCHAR(255) NULL,
    `description` TEXT NULL,
    `supporting_documents` JSON NULL,
    `status` ENUM('active', 'fully_utilized', 'archived') DEFAULT 'active',
    `entered_by` BIGINT UNSIGNED NOT NULL,
    `entered_at` TIMESTAMP NOT NULL,
    `verified_by` BIGINT UNSIGNED NULL,
    `verified_at` TIMESTAMP NULL,
    `verification_notes` TEXT NULL,
    `created_at` TIMESTAMP NULL,
    `updated_at` TIMESTAMP NULL,
    PRIMARY KEY (`id`),
    INDEX `idx_legacy_fund_id` (`legacy_fund_id`),
    INDEX `idx_source_type` (`source_type`),
    INDEX `idx_old_project_id` (`old_project_id`),
    INDEX `idx_user_id` (`user_id`),
    INDEX `idx_center` (`center`),
    INDEX `idx_project_type` (`project_type`),
```

```
    INDEX `idx_status` (`status`),
    FOREIGN KEY (`old_project_id`) REFERENCES `oldDevelopmentProjects`(`id`)
ON DELETE SET NULL,
    FOREIGN KEY (`user_id`) REFERENCES `users`(`id`) ON DELETE RESTRICT,
    FOREIGN KEY (`entered_by`) REFERENCES `users`(`id`) ON DELETE RESTRICT,
    FOREIGN KEY (`verified_by`) REFERENCES `users`(`id`) ON DELETE SET NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

Migration File

File:
`database/migrations/YYYY_MM_DD_HHMMSS_create_legacy_funds_table.php`

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up(): void
    {
        Schema::create('legacy_funds', function (Blueprint $table) {
            $table->id();
            $table->string('legacy_fund_id')->unique();
            $table->enum('source_type', ['old_development_project',
'external_project', 'manual_entry']);
            $table->unsignedBigInteger('old_project_id')->nullable();
            $table->string('project_title');
            $table->string('project_type');
            $table->unsignedBigInteger('user_id');
            $table->string('center');
            $table->decimal('amount_available', 10, 2);
            $table->decimal('amount_sanctioned', 10, 2)->nullable();
            $table->decimal('amount_utilized', 10, 2)->default(0.00);
            $table->decimal('amount_relocated', 10, 2)->default(0.00);
            $table->date('commencement_date')->nullable();
            $table->date('completion_date')->nullable();
            $table->string('society_name')->nullable();
            $table->string('in_charge_name')->nullable();
            $table->text('description')->nullable();
            $table->json('supporting_documents')->nullable();
            $table->enum('status', ['active', 'fully_utilized',
'archived'])->default('active');
            $table->unsignedBigInteger('entered_by');
            $table->timestamp('entered_at');
            $table->unsignedBigInteger('verified_by')->nullable();
            $table->timestamp('verified_at')->nullable();
            $table->text('verification_notes')->nullable();
            $table->timestamps();
```

```
            $table->foreign('old_project_id')
                  ->references('id')
                  ->on('oldDevelopmentProjects')
                  ->onDelete('set null');

            $table->foreign('user_id')
                  ->references('id')
                  ->on('users')
                  ->onDelete('restrict');

            $table->foreign('entered_by')
                  ->references('id')
                  ->on('users')
                  ->onDelete('restrict');

            $table->foreign('verified_by')
                  ->references('id')
                  ->on('users')
                  ->onDelete('set null');

            $table->index('legacy_fund_id');
            $table->index('source_type');
            $table->index('old_project_id');
            $table->index('user_id');
            $table->index('center');
            $table->index('project_type');
            $table->index('status');
        });
    }

    public function down(): void
    {
        Schema::dropIfExists('legacy_funds');
    }
};
```

Integration with Fund Relocation

Legacy funds should be included in the eligible source projects when:

- Project type matches
- Center matches
- Legacy fund status is 'active'
- Available amount > 0 (amount_available - amount_relocated > 0)

Legacy Fund Controller: `LegacyFundController.php`

Location:
`app/Http/Controllers/Projects/FundRelocation/LegacyFundController.php`

Key Methods:

1. **`index(Request $request)`**
   - Lists all legacy funds with filtering options
   - Filter by center, project type, status, source type
2. **`create()`**
   - Shows form to create new legacy fund entry
   - Option to link to old development project or create manual entry
3. **`store(Request $request)`**
   - Creates new legacy fund entry
   - Validates data
   - Auto-generates legacy_fund_id
   - Links to old project if source_type is 'old_development_project'
4. **`show($legacyFundId)`**
   - Displays legacy fund details
   - Shows relocation history
   - Shows available amount
5. **`edit($legacyFundId)`**
   - Shows edit form for legacy fund
   - Only editable if status is 'active' and no relocations exist
6. **`update(Request $request, $legacyFundId)`**
   - Updates legacy fund details
   - Validates changes
   - Creates history entry
7. **`verify(Request $request, $legacyFundId)`**
   - Verifies legacy fund entry (by coordinator/admin)

- Updates verified_by and verified_at
- Adds verification notes

8. `getAvailableFunds($projectType, $center)`
    - Returns available legacy funds for relocation
    - Used by FundRelocationController

Model: `LegacyFund.php`

```php
<?php

namespace App\Models;

use App\Models\OldProjects\OldDevelopmentProject;
use App\Models\User;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsTo;

class LegacyFund extends Model
{
    protected $table = 'legacy_funds';

    protected $fillable = [
        'legacy_fund_id',
        'source_type',
        'old_project_id',
        'project_title',
        'project_type',
        'user_id',
        'center',
        'amount_available',
        'amount_sanctioned',
        'amount_utilized',
        'amount_relocated',
        'commencement_date',
        'completion_date',
        'society_name',
        'in_charge_name',
        'description',
        'supporting_documents',
        'status',
        'entered_by',
        'entered_at',
        'verified_by',
        'verified_at',
        'verification_notes',
    ];

    protected $casts = [
        'amount_available' => 'decimal:2',
```

```
        'amount_sanctioned' => 'decimal:2',
        'amount_utilized' => 'decimal:2',
        'amount_relocated' => 'decimal:2',
        'commencement_date' => 'date',
        'completion_date' => 'date',
        'supporting_documents' => 'array',
        'entered_at' => 'datetime',
        'verified_at' => 'datetime',
    ];

    public function oldProject(): BelongsTo
    {
        return $this->belongsTo(OldDevelopmentProject::class,
'old_project_id');
    }

    public function user(): BelongsTo
    {
        return $this->belongsTo(User::class, 'user_id');
    }

    public function enteredBy(): BelongsTo
    {
        return $this->belongsTo(User::class, 'entered_by');
    }

    public function verifiedBy(): BelongsTo
    {
        return $this->belongsTo(User::class, 'verified_by');
    }

    public function fundRelocations()
    {
        return $this->hasMany(FundRelocation::class,
'source_legacy_fund_id', 'legacy_fund_id');
    }

    /**
     * Calculate available amount for relocation
     */
    public function getAvailableAmountAttribute()
    {
        return max(0, $this->amount_available - $this->amount_relocated);
    }
}
```

Updated Fund Relocation Schema

The `fund_relocations` table needs to support legacy funds:

```
// Add to fund_relocations migration
$table->string('source_legacy_fund_id')->nullable()-
```

```
>after('source_project_id');
$table->enum('source_type', ['current_project', 'legacy_fund'])-
>default('current_project');

// Add foreign key
$table->foreign('source_legacy_fund_id')
      ->references('legacy_fund_id')
      ->on('legacy_funds')
      ->onDelete('restrict');
```

Updated FundRelocationController

The `getEligibleSourceProjects()` method should include legacy funds:

```
public function getEligibleSourceProjects($projectType, $center)
{
    $eligibleProjects = [];

    // Get current system projects
    $projects = Project::where('project_type', $projectType)
        ->where('status', 'approved_by_coordinator')
        ->whereHas('user', function($query) use ($center) {
            $query->where('center', $center);
        })
        ->with(['user', 'reports' => function($query) {
            $query->where('status', 'approved_by_coordinator');
        }])
        ->get();

    foreach ($projects as $project) {
        $availableFunds = $this->calculateAvailableFunds($project-
>project_id);

        if ($availableFunds > 0) {
            $eligibleProjects[] = [
                'source_type' => 'current_project',
                'source_id' => $project->project_id,
                'project_title' => $project->project_title,
                'amount_sanctioned' => $project->amount_sanctioned ?? 0,
                'available_funds' => $availableFunds,
            ];
        }
    }

    // Get legacy funds
    $legacyFunds = LegacyFund::where('project_type', $projectType)
        ->where('center', $center)
        ->where('status', 'active')
        ->whereRaw('amount_available - amount_relocated > 0')
        ->get();

    foreach ($legacyFunds as $legacyFund) {
```

```
            $availableFunds = $legacyFund->available_amount;

            if ($availableFunds > 0) {
                $eligibleProjects[] = [
                    'source_type' => 'legacy_fund',
                    'source_id' => $legacyFund->legacy_fund_id,
                    'project_title' => $legacyFund->project_title,
                    'amount_sanctioned' => $legacyFund->amount_sanctioned ?? 0,
                    'available_funds' => $availableFunds,
                    'source_type_label' => $legacyFund->source_type,
                ];
            }
        }

        return $eligibleProjects;
    }
```

Controller Structure

Main Controller: `FundRelocationController.php`

Location: `app/Http/Controllers/Projects/FundRelocation/FundRelocationController.php`

Key Methods:

1. **`getEligibleSourceProjects($projectType, $center)`**
   - Returns projects with unspent funds matching criteria
   - Calculates remaining balance from approved reports
2. **`store(Request $request, $projectId)`**
   - Creates fund relocation requests
   - Validates eligibility
   - Creates history entry
3. **`update(Request $request, $fundRelocationId)`**
   - Updates fund relocation amount (by approvers)

- Creates history entry
- Validates new amount doesn't exceed available

4. `show($fundRelocationId)`
    - Displays fund relocation details with traceability
5. `getByProject($projectId)`
    - Returns all fund relocations for a project

Validation Controller: `FundRelocationValidationController.php`

Location: `app/Http/Controllers/Projects/FundRelocation/FundRelocationValidationController.php`

Key Methods:

1. `validateEligibility($sourceProjectId, $destinationProjectId)`
    - Checks project type match
    - Checks center match
    - Checks source project has unspent funds
2. `calculateAvailableFunds($projectId)`
    - Calculates unspent funds from approved reports
    - Returns available amount
3. `validateAmount($sourceProjectId, $amount)`
    - Ensures amount doesn't exceed available funds
    - Checks for existing relocations from same source

Approval Controller: `FundRelocationApprovalController.php`

Location:
`app/Http/Controllers/Projects/FundRelocation/FundRelocationApprovalController.php`

Key Methods:

1. `approveRelocations($projectId)`
    - Called when project is approved by coordinator
    - Updates fund relocation status to 'approved'
    - Adds funds to destination project budget
    - Updates source project to mark funds as relocated
2. `rejectRelocation($fundRelocationId, $reason)`
    - Rejects a fund relocation request
    - Updates status and creates history

View/Partial Structure

Partial: `select_source_projects.blade.php`

Location:
`resources/views/projects/partials/FundRelocation/select_source_projects.blade.php`

Purpose: Allows applicant to select source projects and amounts during project creation/edit

Features:

- Dropdown/list of eligible source projects
- Display available unspent amount for each
- Input field for amount to relocate
- Validation (max available amount)
- Add multiple source projects
- Summary of total funds to be relocated

Partial: `fund_relocation_summary.blade.php`

Location: `resources/views/projects/partials/FundRelocation/fund_relocation_summary.blade.php`

Purpose: Displays summary of fund relocations for a project

Features:

- List of all fund relocations
- Source project details
- Amounts
- Status
- Total relocated funds

Partial: `fund_relocation_edit.blade.php`

Location: `resources/views/projects/partials/FundRelocation/fund_relocation_edit.blade.php`

Purpose: Allows Provincial and Coordinator to edit fund relocation amounts

Features:

- Editable amount fields
- Validation
- Notes/comments field
- Save changes button

Partial: `fund_relocation_traceability.blade.php`

Location:
`resources/views/projects/partials/FundRelocation/fund_relocation_traceability.blade.php`

Purpose: Displays complete audit trail

Features:

- History of all changes
- Who made changes
- When changes were made
- Old and new values
- Notes/reasons

Workflow Implementation

Step 1: Project Creation (Applicant)

1. Applicant creates new project
2. System checks for eligible source projects:

```
$eligibleProjects =
FundRelocationController::getEligibleSourceProjects(
    $projectType,
    $user->center
);
```

3. If eligible projects exist, show fund relocation partial
4. Applicant selects source projects and amounts
5. Fund relocation records created with status 'pending'

Step 2: Project Submission to Provincial

1. Project status changes to 'submitted_to_provincial'
2. Provincial can view fund relocations in project details
3. Provincial can edit amounts (with validation)
4. Changes logged in history table

Step 3: Project Forwarding to Coordinator

1. Project status changes to 'forwarded_to_coordinator'
2. Coordinator can view fund relocations in project details
3. Coordinator can edit amounts (with validation)
4. Changes logged in history table

Step 4: Project Approval

1. Coordinator approves project
2. `FundRelocationApprovalController::approveRelocations()` is called
3. For each approved fund relocation:
    - Status updated to 'approved'
    - Funds added to destination project budget
    - Source project marked (optional: add field `funds_relocated` to projects table)
    - History entry created

Step 5: Budget Integration

1. Relocated funds added to project's `amount_sanctioned` or `overall_project_budget`
2. Funds available in budget tables for reporting
3. Reports can track expenses from relocated funds

Implementation Steps

Phase 1: Database Setup

1. ✅ Create migration file for `fund_relocations` table
2. ✅ Create migration file for `fund_relocation_history` table
3. ✅ Create migration file for `legacy_funds` table
4. ✅ Run migrations
5. ✅ Create Eloquent models:

- `app/Models/FundRelocation.php`
- `app/Models/FundRelocationHistory.php`
- `app/Models/LegacyFund.php`

Phase 2: Controller Implementation

1. ✅ Create `FundRelocationController`
2. ✅ Create `FundRelocationValidationController`
3. ✅ Create `FundRelocationApprovalController`
4. ✅ Create `LegacyFundController`
5. ✅ Add routes in `routes/web.php`

Phase 3: View/Partial Implementation

1. ✅ Create `select_source_projects.blade.php`
2. ✅ Create `fund_relocation_summary.blade.php`
3. ✅ Create `fund_relocation_edit.blade.php`
4. ✅ Create `fund_relocation_traceability.blade.php`
5. ✅ Create legacy fund views (create, edit, list, show)
6. ✅ Integrate into project create/edit forms
7. ✅ Integrate into project show views

Phase 4: Integration with Project Workflow

1. ✅ Add fund relocation section to project creation form
2. ✅ Add fund relocation display to project show page
3. ✅ Add fund relocation edit capability for Provincial
4. ✅ Add fund relocation edit capability for Coordinator
5. ✅ Integrate approval logic into project approval process
6. ✅ Update `getEligibleSourceProjects()` to include legacy funds
7. ✅ Update fund relocation approval to handle legacy funds

Phase 5: Budget Integration

1. ✅ Update budget calculation to include relocated funds
2. ✅ Update reporting to show source of funds

3. ✅ Update budget summaries to include relocated funds

Phase 6: Testing

1. ✅ Test project creation with fund relocation
2. ✅ Test Provincial review and editing
3. ✅ Test Coordinator review and editing
4. ✅ Test approval process
5. ✅ Test budget integration
6. ✅ Test reporting with relocated funds
7. ✅ Test traceability/audit trail

Detailed Implementation Guide

1. Model: `FundRelocation.php`

```php
<?php

namespace App\Models;

use App\Models\OldProjects\Project;
use App\Models\User;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsTo;

class FundRelocation extends Model
{
    protected $table = 'fund_relocations';

    protected $fillable = [
        'fund_relocation_id',
        'destination_project_id',
        'source_type',
        'source_project_id',
        'source_legacy_fund_id',
        'amount_relocated',
        'status',
        'requested_by',
        'requested_at',
        'approved_by',
        'approved_at',
        'rejection_reason',
        'notes',
    ];
```

```
    protected $casts = [
        'amount_relocated' => 'decimal:2',
        'requested_at' => 'datetime',
        'approved_at' => 'datetime',
    ];

    public function destinationProject(): BelongsTo
    {
        return $this->belongsTo(Project::class, 'destination_project_id',
'project_id');
    }

    public function sourceProject(): BelongsTo
    {
        return $this->belongsTo(Project::class, 'source_project_id',
'project_id');
    }

    public function sourceLegacyFund(): BelongsTo
    {
        return $this->belongsTo(LegacyFund::class, 'source_legacy_fund_id',
'legacy_fund_id');
    }

    public function requestedBy(): BelongsTo
    {
        return $this->belongsTo(User::class, 'requested_by');
    }

    public function approvedBy(): BelongsTo
    {
        return $this->belongsTo(User::class, 'approved_by');
    }

    public function history()
    {
        return $this->hasMany(FundRelocationHistory::class,
'fund_relocation_id', 'fund_relocation_id');
    }
}
```

2. Controller: `FundRelocationController.php` (Key Methods)

```
<?php

namespace App\Http\Controllers\Projects\FundRelocation;
```

```php
use App\Http\Controllers\Controller;
use App\Models\FundRelocation;
use App\Models\OldProjects\Project;
use App\Models\Reports\Monthly\DPReport;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\DB;
use Illuminate\Support\Facades\Log;

class FundRelocationController extends Controller
{
    /**
     * Get eligible source projects for fund relocation
     */
    public function getEligibleSourceProjects($projectType, $center)
    {
        // Get approved projects of same type and center with unspent funds
        $projects = Project::where('project_type', $projectType)
            ->where('status', 'approved_by_coordinator')
            ->whereHas('user', function($query) use ($center) {
                $query->where('center', $center);
            })
            ->with(['user', 'reports' => function($query) {
                $query->where('status', 'approved_by_coordinator');
            }])
            ->get();

        $eligibleProjects = [];

        foreach ($projects as $project) {
            $availableFunds = $this->calculateAvailableFunds($project->project_id);

            if ($availableFunds > 0) {
                $eligibleProjects[] = [
                    'project_id' => $project->project_id,
                    'project_title' => $project->project_title,
                    'amount_sanctioned' => $project->amount_sanctioned ?? 0,
                    'available_funds' => $availableFunds,
                ];
            }
        }

        return $eligibleProjects;
    }

    /**
     * Calculate available unspent funds for a project
     */
    private function calculateAvailableFunds($projectId)
    {
        $project = Project::where('project_id', $projectId)->first();

        if (!$project || $project->status !== 'approved_by_coordinator') {
            return 0;
        }
```

```
        $amountSanctioned = $project->amount_sanctioned ?? 0;

        // Calculate total expenses from approved reports
        $totalExpenses = DPReport::where('project_id', $projectId)
            ->where('status', 'approved_by_coordinator')
            ->with('accountDetails')
            ->get()
            ->sum(function($report) {
                return $report->accountDetails->sum('total_expenses');
            });

        // Calculate already relocated funds
        $alreadyRelocated = FundRelocation::where('source_project_id',
$projectId)
            ->where('status', 'approved')
            ->sum('amount_relocated');

        $available = $amountSanctioned - $totalExpenses - $alreadyRelocated;

        return max(0, $available);
    }

    /**
     * Store fund relocation request
     */
    public function store(Request $request, $projectId)
    {
        $request->validate([
            'source_type' => 'required|in:current_project,legacy_fund',
            'source_project_id' =>
'required_if:source_type,current_project|nullable|exists:projects,project_id
',
            'source_legacy_fund_id' =>
'required_if:source_type,legacy_fund|nullable|exists:legacy_funds,legacy_fun
d_id',
            'amount_relocated' => 'required|numeric|min:0.01',
        ]);

        $destinationProject = Project::where('project_id', $projectId)-
>firstOrFail();
        $availableFunds = 0;
        $sourceType = $request->source_type;

        if ($sourceType === 'current_project') {
            $sourceProject = Project::where('project_id', $request-
>source_project_id)->firstOrFail();

            // Validate eligibility
            if ($destinationProject->project_type !== $sourceProject-
>project_type) {
                return back()->withErrors(['error' => 'Project types must
match']);
            }

            if ($destinationProject->user->center !== $sourceProject->user-
>center) {
                return back()->withErrors(['error' => 'Projects must be from
```

```
the same center']);
            }

            $availableFunds = $this->calculateAvailableFunds($sourceProject-
>project_id);
        } else {
            // Legacy fund
            $legacyFund = LegacyFund::where('legacy_fund_id', $request-
>source_legacy_fund_id)->firstOrFail();

            // Validate eligibility
            if ($destinationProject->project_type !== $legacyFund-
>project_type) {
                return back()->withErrors(['error' => 'Project types must
match']);
            }

            if ($destinationProject->user->center !== $legacyFund->center) {
                return back()->withErrors(['error' => 'Projects must be from
the same center']);
            }

            if ($legacyFund->status !== 'active') {
                return back()->withErrors(['error' => 'Legacy fund is not
active']);
            }

            $availableFunds = $legacyFund->available_amount;
        }

        if ($request->amount_relocated > $availableFunds) {
            return back()->withErrors(['error' => 'Amount exceeds available
funds']);
        }

        DB::beginTransaction();
        try {
            $fundRelocationId = 'FR-' . str_pad(FundRelocation::count() + 1,
6, '0', STR_PAD_LEFT);

            $fundRelocationData = [
                'fund_relocation_id' => $fundRelocationId,
                'destination_project_id' => $projectId,
                'source_type' => $sourceType,
                'amount_relocated' => $request->amount_relocated,
                'status' => 'pending',
                'requested_by' => Auth::id(),
                'requested_at' => now(),
                'notes' => $request->notes,
            ];

            if ($sourceType === 'current_project') {
                $fundRelocationData['source_project_id'] = $request-
>source_project_id;
            } else {
                $fundRelocationData['source_legacy_fund_id'] = $request-
>source_legacy_fund_id;
```

```
            }

            $fundRelocation = FundRelocation::create($fundRelocationData);

            // Create history entry
            $this->createHistoryEntry($fundRelocation, 'created',
Auth::id(), null, $request->amount_relocated, $request->notes);

            DB::commit();

            return back()->with('success', 'Fund relocation request created
successfully');
        } catch (\Exception $e) {
            DB::rollBack();
            Log::error('Error creating fund relocation', ['error' => $e-
>getMessage()]);
            return back()->withErrors(['error' => 'Failed to create fund
relocation request']);
        }
    }

    /**
     * Update fund relocation (by approvers)
     */
    public function update(Request $request, $fundRelocationId)
    {
        $request->validate([
            'amount_relocated' => 'required|numeric|min:0.01',
            'notes' => 'nullable|string',
        ]);

        $fundRelocation = FundRelocation::where('fund_relocation_id',
$fundRelocationId)->firstOrFail();

        if ($fundRelocation->status !== 'pending') {
            return back()->withErrors(['error' => 'Only pending relocations
can be edited']);
        }

        $oldAmount = $fundRelocation->amount_relocated;
        $availableFunds = $this->calculateAvailableFunds($fundRelocation-
>source_project_id);

        // Add back the old amount to available funds
        $totalAvailable = $availableFunds + $oldAmount;

        if ($request->amount_relocated > $totalAvailable) {
            return back()->withErrors(['error' => 'Amount exceeds available
funds']);
        }

        DB::beginTransaction();
        try {
            $fundRelocation->update([
                'amount_relocated' => $request->amount_relocated,
                'notes' => $request->notes ?? $fundRelocation->notes,
            ]);
```

```
            // Create history entry
            $this->createHistoryEntry(
                $fundRelocation,
                'edited',
                Auth::id(),
                $oldAmount,
                $request->amount_relocated,
                $request->notes
            );

            DB::commit();

            return back()->with('success', 'Fund relocation updated
successfully');
        } catch (\Exception $e) {
            DB::rollBack();
            Log::error('Error updating fund relocation', ['error' => $e-
>getMessage()]);
            return back()->withErrors(['error' => 'Failed to update fund
relocation']);
        }
    }

    /**
     * Create history entry
     */
    private function createHistoryEntry($fundRelocation, $action,
$performedBy, $oldAmount, $newAmount, $notes = null)
    {
        return \App\Models\FundRelocationHistory::create([
            'fund_relocation_id' => $fundRelocation->fund_relocation_id,
            'action' => $action,
            'performed_by' => $performedBy,
            'performed_at' => now(),
            'old_amount' => $oldAmount,
            'new_amount' => $newAmount,
            'notes' => $notes,
        ]);
    }
}
```

3. Approval Controller: FundRelocationApprovalController.php

```
<?php

namespace App\Http\Controllers\Projects\FundRelocation;

use App\Http\Controllers\Controller;
use App\Models\FundRelocation;
```

```php
use App\Models\OldProjects\Project;
use Illuminate\Support\Facades\DB;
use Illuminate\Support\Facades\Log;

class FundRelocationApprovalController extends Controller
{
    /**
     * Approve all fund relocations for a project
     * Called when project is approved by coordinator
     */
    public function approveRelocations($projectId)
    {
        $fundRelocations = FundRelocation::where('destination_project_id',
$projectId)
            ->where('status', 'pending')
            ->get();

        if ($fundRelocations->isEmpty()) {
            return;
        }

        DB::beginTransaction();
        try {
            $totalRelocated = 0;

            foreach ($fundRelocations as $fundRelocation) {
                // Update status
                $fundRelocation->update([
                    'status' => 'approved',
                    'approved_by' => auth()->id(),
                    'approved_at' => now(),
                ]);

                // Update source (project or legacy fund)
                if ($fundRelocation->source_type === 'current_project') {
                    // Source project funds are already tracked via reports
                    // No additional update needed
                } else {
                    // Update legacy fund
                    $legacyFund = LegacyFund::where('legacy_fund_id',
$fundRelocation->source_legacy_fund_id)->first();
                    if ($legacyFund) {
                        $legacyFund->increment('amount_relocated',
$fundRelocation->amount_relocated);

                        // Update status if fully utilized
                        if ($legacyFund->amount_relocated >= $legacyFund-
>amount_available) {
                            $legacyFund->update(['status' =>
'fully_utilized']);
                        }
                    }
                }

                // Create history entry
                $this->createHistoryEntry(
                    $fundRelocation,
```

```
                    'approved',
                    auth()->id(),
                    null,
                    $fundRelocation->amount_relocated,
                    'Approved with project approval'
                );

                $totalRelocated += $fundRelocation->amount_relocated;
            }

            // Add relocated funds to destination project budget
            $project = Project::where('project_id', $projectId)->first();

            if ($project) {
                $currentBudget = $project->amount_sanctioned ?? 0;
                $project->update([
                    'amount_sanctioned' => $currentBudget + $totalRelocated,
                ]);
            }

            DB::commit();
            Log::info('Fund relocations approved', [
                'project_id' => $projectId,
                'total_relocated' => $totalRelocated,
                'count' => $fundRelocations->count(),
            ]);
        } catch (\Exception $e) {
            DB::rollBack();
            Log::error('Error approving fund relocations', [
                'project_id' => $projectId,
                'error' => $e->getMessage(),
            ]);
            throw $e;
        }
    }

    private function createHistoryEntry($fundRelocation, $action,
$performedBy, $oldAmount, $newAmount, $notes = null)
    {
        return \App\Models\FundRelocationHistory::create([
            'fund_relocation_id' => $fundRelocation->fund_relocation_id,
            'action' => $action,
            'performed_by' => $performedBy,
            'performed_at' => now(),
            'old_amount' => $oldAmount,
            'new_amount' => $newAmount,
            'notes' => $notes,
        ]);
    }
}
```

4. Integration with Project Approval

Update `CoordinatorController::approveProject()`:

```php
public function approveProject($project_id)
{
    // ... existing approval code ...

    // Approve fund relocations
    $fundRelocationController = new FundRelocationApprovalController();
    $fundRelocationController->approveRelocations($project_id);

    // ... rest of approval code ...
}
```

Testing Checklist

Functional Testing

- ☐ **Legacy Funds Management**
  - ☐ Coordinator/Admin can create legacy fund entries
  - ☐ Legacy funds can be linked to old development projects
  - ☐ Legacy funds can be created as manual entries
  - ☐ Legacy funds can be verified
  - ☐ Legacy funds appear in eligible source projects list
  - ☐ Legacy funds show correct available amount
- ☐ **Project Creation with Fund Relocation**
  - ☐ Applicant can see eligible source projects (current + legacy)
  - ☐ Applicant can select source projects and amounts
  - ☐ Applicant can select legacy funds
  - ☐ Validation prevents selecting more than available
  - ☐ Validation ensures same project type
  - ☐ Validation ensures same center
  - ☐ Fund relocation records created with status 'pending'
- ☐ **Provincial Review**
  - ☐ Provincial can view fund relocations

    - ☐ Provincial can edit amounts
    - ☐ Validation prevents invalid amounts
    - ☐ History entries created for edits

- ☐ **Coordinator Review**
    - ☐ Coordinator can view fund relocations
    - ☐ Coordinator can edit amounts
    - ☐ Validation prevents invalid amounts
    - ☐ History entries created for edits

- ☐ **Project Approval**
    - ☐ Fund relocations approved when project approved
    - ☐ Funds added to destination project budget
    - ☐ Source project funds marked as relocated
    - ☐ History entries created

- ☐ **Budget Integration**
    - ☐ Relocated funds appear in project budget
    - ☐ Budget calculations include relocated funds
    - ☐ Reports can track expenses from relocated funds

- ☐ **Traceability**
    - ☐ Complete history visible
    - ☐ All changes logged with user and timestamp
    - ☐ Audit trail complete

Edge Cases

- ☐ Multiple relocations from same source project
- ☐ Multiple relocations from same legacy fund
- ☐ Relocation amount equals available funds
- ☐ Relocation amount less than available funds
- ☐ No eligible source projects available
- ☐ Source project has zero unspent funds
- ☐ Legacy fund has zero available funds
- ☐ Project rejection (fund relocations should be cancelled)

- ☐ Project deletion (fund relocations should be handled)
- ☐ Legacy fund fully utilized (status change)

Notes and Considerations

1. **Performance**: Consider caching eligible projects list for better performance
2. **Notifications**: Consider adding email notifications for fund relocation approvals
3. **Reporting**: Update budget reports to show source of funds (original project reference or legacy fund)
4. **UI/UX**: Make fund relocation section collapsible/expandable in project forms
5. **Validation**: Add real-time validation in frontend for better UX
6. **Permissions**: Ensure proper role-based access control for editing fund relocations
7. **Legacy Funds**: Only coordinators/admins should be able to create/verify legacy funds
8. **Legacy Fund Verification**: Consider requiring verification before legacy funds can be used
9. **Documentation**: Legacy funds should have supporting documents uploaded when possible
10. **Audit Trail**: All legacy fund entries and changes should be logged for audit purposes

---

Future Enhancements

1. **Bulk Relocation**: Allow relocating funds to multiple projects at once
2. **Partial Relocation**: Allow relocating partial amounts with remainder staying in source
3. **Relocation Reversal**: Allow reversing approved relocations (with proper approval)
4. **Advanced Filtering**: Add filters for year, project type, center in eligible projects list
5. **Export/Import**: Export fund relocation data for external analysis
6. **Dashboard Widget**: Show fund relocation statistics on dashboards
7. **Legacy Fund Import**: Bulk import legacy funds from CSV/Excel
8. **Legacy Fund Linking**: Auto-link legacy funds to old development projects based on matching criteria

9. **Legacy Fund Reports**: Generate reports showing utilization of legacy funds
10. **Legacy Fund Archival**: Auto-archive legacy funds when fully utilized

---

Last Updated: [Current Date] **Version**: 1.0 **Status**: Documentation Complete - Ready for Implementation