DOCUMENT TRACKING SYSTEM

DTS

DEVELOPMENT PROCEDURES

Claude Code Setup • Git Workflow • Implementation Standards

Version 1.0 — 2026

SECTION 1 — ENVIRONMENT SETUP

1. Environment Setup

1.1 Prerequisites

PHP	8.2 or higher
Composer	2.x
Node.js	20.x or higher
npm	10.x or higher
PostgreSQL	15 or higher
Git	2.x
Claude Code CLI	Latest — @anthropic-ai/claude-code

1.2 Initial Project Setup

Run these commands once to set up the full development environment:

Backend + Frontend Setup

```
# Clone the repository
git clone <repo-url> dts
cd dts

# Backend setup
cd server-laravel
composer install
cp .env.example .env
php artisan key:generate

# Configure .env — set these values:
DB_CONNECTION=pgsql
DB_HOST=127.0.0.1
DB_PORT=5432
DB_DATABASE=dts
DB_USERNAME=your_username
DB_PASSWORD=your_password
APP_URL=http://localhost:8000

# Run migrations and seeders
```

```
php artisan migrate
php artisan db:seed

# Install Passport
php artisan passport:install

# Frontend setup
cd ../client
npm install
```

1.3 Claude Code Installation

Claude Code CLI Setup

```
# Install Claude Code globally
npm install -g @anthropic-ai/claude-code

# Verify installation
claude --version

# Navigate to project root (where CLAUDE.md lives)
cd /path/to/dts

# Start Claude Code
claude

# First run will prompt Anthropic authentication
# Follow the browser prompt to authenticate
```

CLAUDE.md Location

- **Place CLAUDE.md in the project ROOT directory** (same level as server-laravel/ and client/)
- Claude Code reads CLAUDE.md automatically on every session
- Never delete or rename CLAUDE.md — it is the primary context source
- Update CLAUDE.md whenever major architecture decisions are made

1.4 Running the Development Server

Development Server Commands

```
# Terminal 1 — Laravel API server
cd server-laravel
php artisan serve
# Runs on http://localhost:8000

# Terminal 2 — Laravel Queue worker
```

```
cd server-laravel
php artisan queue:listen --tries=1

# Terminal 3 — Vue dev server
cd client
npm run dev
# Runs on http://localhost:5173

# OR use the combined dev command from server-laravel:
cd server-laravel
composer run dev
# Starts all four: server, queue, logs, vite
```

SECTION 2 — CLAUDE CODE WORKFLOW

2. Claude Code Workflow

2.1 Starting a Session

1	**Navigate to project root** cd to the directory containing CLAUDE.md, server-laravel/, and client/
2	**Start Claude Code** Run: claude — Claude reads CLAUDE.md automatically
3	**State the task clearly** Tell Claude exactly what you are implementing. Reference the spec.
4	**Review before applying** Always review Claude's proposed changes before accepting
5	**Test after each change** Run tests, check API, verify frontend before moving to next task

2.2 How to Give Claude Tasks

Be specific. Reference the spec. Tell Claude what phase it is in.

Good Task Examples

- "Implement the Return to Sender action per the Chapter 1 spec. The route is POST /api/transactions/{trxNo}/return. Guards: status=Processing, isActive=true, has Received log, default recipients only. Required body fields: reason (dropdown) and remarks (free text). Effect: Returned To Sender log, returning office isActive→false, all pending isActive→false with Routing Halted log, Transaction→Returned, Document→Returned, notify origin and all halted offices with reason+remarks."
- "Create the migration to add parent_transaction_no, urgency_level (enum: Urgent|High|Normal|Routine default High), and due_date (nullable date) to document_transactions table. Do NOT modify existing migrations."
- "Update useActionVisibility.ts to add canMarkAsDone computed: FA action type only, isActive=true, has Received log, default recipient only, not CC or BCC."

Bad Task Examples — Too Vague

- "Add the return feature"
- "Fix the action buttons"
- "Update the database"

2.3 Claude Code Commands

claude	Start interactive session (reads CLAUDE.md)
claude "task description"	One-shot command — executes and exits
claude --continue	Continue the previous session
claude --print	Print response without interactive mode
/help	Show available Claude Code commands (inside session)
/clear	Clear session context (inside session)
/exit	Exit Claude Code (inside session)

2.4 Session Tips

- Start every session with: "Read CLAUDE.md and summarize the current implementation status" to orient Claude
- If Claude loses context mid-session, say: "Re-read CLAUDE.md and the current state of [filename]"
- For complex multi-file changes, break into smaller tasks: "Do only the migration first, then we will do the controller"
- Always tell Claude the phase: "We are in Phase 3 — Return to Sender controller update"
- After each completed task, say: "Update CLAUDE.md Implementation Status section to mark Return to Sender as done"

SECTION 3 — GIT WORKFLOW

3. Git Workflow

3.1 Branch Strategy

Branch	Purpose	Rules
main	Production ready code	Protected. No direct pushes. PR only. Must pass all tests.
develop	Integration branch	All features merge here first. Base for all feature branches.
feature/{name}	New feature development	Branch from develop. One feature per branch. PR to develop.
fix/{name}	Bug fixes	Branch from develop (or main for hotfixes). PR to develop.
migration/{name}	Database migrations only	Branch from develop. Reviewed separately before merge.

3.2 Branch Naming

Branch Naming Examples

```
# Feature branches
feature/return-to-sender
feature/mark-as-done
feature/manage-recipients
feature/official-notes
feature/close-document

# Migration branches
migration/document-status-enum
migration/transaction-urgency-fields
migration/document-versions-table

# Fix branches
fix/sequential-turn-guard
fix/fa-completion-logic
```

3.3 Commit Message Format

Follow Conventional Commits. Every commit must be clear and traceable.

Commit Message Examples

```
# Format
<type>(<scope>): <short description>

# Types
feat     → new feature
fix      → bug fix
migrate  → database migration
refactor → code improvement (no behavior change)
test     → adding/updating tests
docs     → documentation only
chore    → build, config, tooling

# Examples
feat(transactions): add Return to Sender action with reason + remarks
feat(transactions): add Mark as Done with proof attachment requirement
migrate(transactions): add urgency_level and due_date to document_transactions
fix(sequential): enforce active step guard on Receive action
refactor(status-service): split FA and FI completion logic
feat(frontend): add canMarkAsDone computed to useActionVisibility
feat(frontend): add MarkAsDoneModal with proof upload support
```

3.4 Feature Development Flow

1. **Pull latest develop**
   git checkout develop && git pull origin develop
2. **Create feature branch**
   git checkout -b feature/your-feature-name
3. **Implement with Claude Code**
   Use Claude Code to implement. Small focused commits.
4. **Run tests**
   php artisan test (backend) — verify no regressions
5. **Commit**
   git add . && git commit -m "feat(scope): description"
6. **Push branch**
   git push origin feature/your-feature-name

7 **Create PR to develop**
PR title matches commit format. Reference spec section in description.

8 **Review + merge**
Review code. Test on develop. Merge when approved.

3.5 Migration-Specific Rules

Database Migration Rules — Critical

- **NEVER modify an existing migration file** — always create a new one
- **Migrations must be reversible** — always implement the down() method
- **Test rollback before merging:** php artisan migrate:rollback
- Migration branches should be reviewed separately — DB changes affect the whole team
- After migration: run php artisan migrate:status to confirm clean state
- Coordinate with team before running migrations — they affect shared databases

SECTION 4 — IMPLEMENTATION STANDARDS

4. Implementation Standards

4.1 Backend — Controller Pattern

Every action controller method follows the same pattern. Do not deviate.

Standard Controller Method Pattern

```
public function actionName(Request $request, string $trxNo): JsonResponse
{
    // 1. Validate request body
    $validated = $request->validate([
        "remarks" => "required|string|max:500",
        "reason"  => "required|string",
    ]);

    $user = $request->user();

    // 2. Find transaction with relations
    $transaction = DocumentTransaction::with(["document","recipients","logs"])
        ->where("transaction_no", $trxNo)->first();

    if (!$transaction) {
        return response()->json(["success" => false, "message" => "Transaction not found."],
404);
    }

    // 3. Status guard
    if ($transaction->status !== "Processing") {
        return response()->json(["success" => false, "message" => "Transaction is not
active."], 422);
    }

    // 4. Actor guard — verify recipient row + isActive
    $recipient = DocumentRecipient::where("transaction_no", $trxNo)
        ->where("office_id", $user->office_id)
        ->where("isActive", true)->first();

    if (!$recipient) {
        return response()->json(["success" => false, "message" => "Not authorized."], 403);
    }

    // 5. Duplicate action guard
    $alreadyDone = $transaction->logs
        ->where("status", "ActionStatus")
        ->where("office_id", $user->office_id)->isNotEmpty();

    if ($alreadyDone) {
        return response()->json(["success" => false, "message" => "Already actioned."], 409);
    }
```

```
    // 6. Execute inside DB::transaction()
    DB::transaction(function () use ($trxNo, $transaction, $validated, $user, $recipient) {
        // Write log
        DocumentTransactionLog::create([...]);

        // Update recipient/transaction state
        $recipient->update(["isActive" => false]);

        // Evaluate completion
        TransactionStatusService::evaluate($trxNo);
    });

    // 7. Return refreshed transaction
    return response()->json([
        "success" => true,
        "message" => "Action completed successfully.",
        "data"    => $transaction->refresh()->load([
            "document", "recipients", "signatories", "attachments", "logs"
        ])
    ], 200);
}
```

4.2 Backend — Model Conventions

Table naming	Plural snake_case: document_transactions, document_recipients
Primary key	Use string for transaction_no/document_no, bigIncrements for others
Soft deletes	NOT used — use isActive boolean instead
Relationships	Always define in model: hasMany, belongsTo, hasManyThrough
Fillable	Always define $fillable — never use $guarded
Casts	Cast boolean fields: "isActive" => "boolean"
Scopes	Use query scopes for common filters: scopeActive(), scopeForOffice()

4.3 Backend — Migration Pattern

Migration Examples

```
// Adding columns to existing table
Schema::table("document_transactions", function (Blueprint $table) {
    $table->string("parent_transaction_no")->nullable()->after("transaction_no");
    $table->foreign("parent_transaction_no")
        ->references("transaction_no")
```

```
        ->on("document_transactions")
        ->nullOnDelete();

    $table->enum("urgency_level", ["Urgent","High","Normal","Routine"])
        ->default("High")->after("status");

    $table->date("due_date")->nullable()->after("urgency_level");
});

// Creating new table
Schema::create("document_notes", function (Blueprint $table) {
    $table->id();
    $table->string("document_no", 50);
    $table->string("transaction_no", 50);
    $table->foreign("document_no")->references("document_no")
        ->on("documents")->cascadeOnDelete();
    $table->text("note");
    $table->string("office_id", 50);
    $table->string("office_name", 150);
    $table->foreignUuid("created_by_id")->references("id")
        ->on("users")->onDelete("cascade");
    $table->string("created_by_name", 150);
    $table->timestamps();
});
```

4.4 Frontend — Composable Pattern

Composable Pattern

```
// composables/useTransaction.ts
import { ref } from "vue"
import API from "@/api"

export function useTransaction() {
    const transaction = ref<any>(null)
    const isLoading = ref(false)
    const error = ref<string | null>(null)

    function setTransaction(data: any) {
        transaction.value = data
    }

    async function someAction(trxNo: string, payload: { remarks: string }) {
        isLoading.value = true
        error.value = null
        try {
            const { data } = await API.post(`/transactions/${trxNo}/action`, payload)
            if (data.data) setTransaction(data.data)
            return data
        } catch (e: any) {
            const msg = e.response?.data?.message || e.message || "Action failed"
            error.value = msg
            throw new Error(msg)
        } finally {
            isLoading.value = false
        }
```

```
    }

    return { transaction, isLoading, error, setTransaction, someAction }
}
```

4.5 Frontend — Modal Pattern

Modal Component Pattern

```
<!-- ReceiveModal.vue -->
<script setup lang="ts">
import { ref } from "vue"
import { useTransaction } from "@/composables/useTransaction"
import { useToast } from "@/composables/useToast"

const props = defineProps<{ trxNo: string }>()
const emit = defineEmits<{ received: [] }>()

const { receiveDocument, isLoading } = useTransaction()
const toast = useToast()
const remarks = ref("")

async function submit() {
    try {
        const result = await receiveDocument(props.trxNo, { remarks: remarks.value })
        toast.success(result.message)
        emit("received")
    } catch (e: any) {
        toast.error(e.message)
    }
}
</script>
```

4.6 Route Registration Pattern

Route Registration

```
// server-laravel/routes/api.php
// Always inside: Route::middleware(["auth:api"])->group(...)

Route::prefix("/transactions")->group(function () {
    // Existing
    Route::post("/{trxNo}/release",  [TransactionController::class, "releaseDocument"]);
    Route::post("/{trxNo}/receive",  [TransactionController::class, "receiveDocument"]);
    Route::post("/{trxNo}/forward",  [TransactionController::class, "forwardDocument"]);

    // New routes to add
    Route::post("/{trxNo}/return",             [TransactionController::class,
"returnToSender"]);
    Route::post("/{trxNo}/subsequent-release", [TransactionController::class,
"subsequentRelease"]);
```

```
    Route::post("/{trxNo}/done",               [TransactionController::class, "markAsDone"]);
    Route::post("/{trxNo}/reply",              [TransactionController::class,
"replyDocument"]);
    Route::patch("/{trxNo}/recipients",        [TransactionController::class,
"manageRecipients"]);
});

Route::prefix("documents")->group(function () {
    Route::post("/{docNo}/close",      [DocumentController::class, "closeDocument"]);
    Route::post("/close-bulk",         [DocumentController::class, "closeBulk"]);
    Route::put("/{docNo}/re-release",  [DocumentController::class, "reRelease"]);
    Route::post("/{docNo}/copy",       [DocumentController::class, "copyDocument"]);
    Route::get("/{docNo}/notes",       [DocumentNotesController::class, "index"]);
    Route::post("/{docNo}/notes",      [DocumentNotesController::class, "store"]);
});
```

SECTION 5 — TESTING

5. Testing

5.1 Backend Testing

Running Tests

```
# Run all tests
cd server-laravel
php artisan test

# Run specific test file
php artisan test tests/Feature/TransactionTest.php

# Run with coverage
php artisan test --coverage

# Run specific test method
php artisan test --filter test_return_to_sender_halts_all_routing
```

5.2 Manual API Testing — Required Checks per Action

After implementing each action, verify these manually using Postman or curl:

Action	Manual Checks Required
Return to Sender	1) reason required — 422 without it 2) remarks required — 422 without it 3) ALL pending recipients isActive→false after 4) Transaction status = Returned 5) Document status = Returned 6) Origin notified 7) All halted offices have Routing Halted log
Mark as Done	1) Only works for FA action type 2) CC/BCC cannot Mark as Done — 403 3) requires_proof=true → 422 without attachment 4) Remarks required — 422 without it 5) isActive→false after 6) StatusService evaluates — check Transaction status
Manage Recipients	1) Cannot remove office that has Received — 422 2) Remove logs written for each removed office 3) Removed office notified 4) Sequential: removed active step advances to next 5) All operations atomic — partial failure rolls back 6) StatusService evaluates after all changes
Close Document	1) Only origin can close — 403 for others 2) Remarks required — 422 without 3) Ongoing transactions: all pending isActive→false 4) Ongoing: Routing Halted log on each 5) Document status = Closed 6) Official Notes locked (cannot add after close)

5.3 Frontend Verification Checklist

- Action buttons appear/disappear correctly per useActionVisibility rules
- Modals show correct fields (proof upload only when requires_proof=true)
- Error messages from backend displayed in toast
- Transaction state refreshes after every action — no stale data
- Sequential: only active step shows action buttons
- CC/BCC: no Forward/Return/Done/Release buttons
- Origin only: Manage Recipients and Close buttons
- Status labels update correctly (Awaiting → In Progress → Done)

SECTION 6 — PHASE IMPLEMENTATION PLAN

6. Phase Implementation Plan

DB migrations run in parallel — create the migration first, then implement the feature that needs it.

Phase 1 — Database Schema Updates

Do these migrations FIRST before any Phase 2-4 work

1. documents: status enum → add Active|Returned|Completed|Closed, remove Processing|Archived
2. documents: add allow_copy (boolean), qr_code (varchar nullable)
3. document_transactions: status enum → add Returned
4. document_transactions: add parent_transaction_no (FK self), urgency_level (enum), due_date (date nullable)
5. document_transaction_logs: status enum → add Done|Closed|Routing Halted|Document Revised|Recipient Added|Recipient Removed|Recipients Reordered
6. document_transaction_logs: add reason (varchar nullable)
7. action_library: add type (FA|FI), default_urgency_level, reply_is_terminal, requires_proof, proof_description
8. document_type_library: add default_urgency_level
9. CREATE document_versions table
10. CREATE document_notes table

Phase 2 — Core Services

TransactionStatusService	Update to FA/FI split logic. FA: check terminal action logs. FI: check Received logs. Separate from document-level evaluation.
DocumentStatusService	New service. Evaluate document.status after every transaction status change. Active if any Processing. Completed if all Completed.
OverdueService	New service. Calculate overdue status per recipient. Uses received_at + urgency threshold vs due_date.
NotificationService	New service. Unified dispatch for all notification triggers. Abstracted from controllers.

Phase 3 — Backend Actions

Action	Status	Key Work
Return to Sender (update)	**Update**	Add reason field, halt ALL pending, Routing Halted log per halted office, notify halted offices
Subsequent Release (new)	**New**	Releasing isActive→false, target reactivated, same TRX, existing recipients only
Mark as Done (new)	**New**	FA+default only, proof guard, Done log, StatusService evaluate
Reply (new)	**New**	New document_no + TRX, Single locked, origin auto-recipient, reply_is_terminal check
Close Single (new)	**New**	Origin only, remarks required, halt ongoing routing, Document→Closed
Close Bulk (new)	**New**	Completed docs only, shared remark, atomic close all
Edit & Re-release (new)	**New**	Snapshot to document_versions, new TRX, Start Fresh vs Continue
Copy to New Document (new)	**New**	New document_no + TRX, all fields editable, origin only
Manage Recipients (new)	**New**	Atomic PATCH, remove guard (no Received), reorder (locked immovable), StatusService
Official Notes (new)	**New**	document_notes CRUD, scoped to document_no, locked on Close
QR Code on Create (update)	**New**	Generate on store(), save to documents.qr_code

Phase 4 — Frontend

useActionVisibility.ts	Add all new computeds: canMarkAsDone, canSubsequentRelease, canClose, canManageRecipients, canReply (update)
useTransaction.ts	Add all new action methods: returnToSender (update), subsequentRelease, markAsDone, replyDocument, closeDocument, closeBulk, manageRecipients

useDocumentNotes.ts	New composable replacing useComments — fetchNotes, addNote, scoped to document_no
New Modals	ReturnModal (update), MarkAsDoneModal, ReplyModal, CloseModal, ManageRecipientsModal, SubsequentReleaseModal
My Documents tabs	Update from Processing|Archived → Draft|Active|Returned|Completed|Closed
Incoming Documents tabs	Add Overdue tab, In Progress tab. Update For Action logic.
ViewDocument side panel	Two tabs: Transaction Logs + Official Notes. Replace Comments tab.
Status labels	Update all label logic: Awaiting Your Action|In Progress|Done|Forwarded|Document Recalled|Removed from Recipients|Completed|Closed

SECTION 7 — COMMON GOTCHAS

7. Common Gotchas — Read Before Implementing

7.1 Database

Never Do These

- **Hard delete a DocumentRecipient row.** Always set isActive=false.
- **Manually set Transaction.status = Completed.** Always call TransactionStatusService::evaluate()
- **Modify an existing migration file.** Always create a new migration.
- **Set document.status = Active manually.** Derive from transaction statuses via DocumentStatusService.
- **Use document_comments or document_logs tables for new data.** Use document_notes and document_transaction_logs.

7.2 FA vs FI — The Most Common Source of Bugs

FA vs FI Rules — Memorize These

- **FI: Receive IS terminal.** Transaction completes when all FI recipients Receive. No further action needed or expected.
- **FA: Receive is NOT terminal.** Do NOT complete an FA transaction on Receive. Completion only on Done/Forward/Return/Reply/Release.
- **Mark as Done: FA default recipients only.** CC/BCC cannot Mark as Done even if they Received. Check action_library.type=FA AND recipient_type=default.
- **Action type comes from action_library.** Do not hardcode FA/FI — always look up from action_library.type field.

7.3 Sequential Routing

- Only the currently active step can Receive. Guard: lowest sequence number with no terminal action AND isActive=true.
- When a step completes: set their isActive→false, find next pending recipient, set their isActive→true, notify them.

- Reordering via Manage Recipients: locked recipients (have Received) are immovable. Only pending recipients can be reordered.
- Removing the active step: advance to next pending. If none, StatusService evaluates.

7.4 Return to Sender — Halt ALL Pending

Return halts ALL routing — not just the returning office

- Get ALL DocumentRecipient rows where isActive=true on this transaction
- Exclude the returning office (already handled)
- For each remaining active recipient: isActive→false + write Routing Halted log
- Notify each halted office with the reason and remarks from the returning office
- Then: Transaction→Returned, Document→Returned, notify origin

7.5 BCC Visibility

- BCC recipients must be filtered from API responses for non-origin callers
- When returning document recipients, check if caller is origin. If not, exclude BCC rows.
- This applies to: show(), history(), all document list endpoints
- BCC notes in Official Notes ARE visible to all — only routing visibility is restricted

7.6 Manage Recipients — Atomic

- All three operations (add/remove/reorder) must happen in a single DB::transaction()
- If any operation fails, ALL roll back — no partial state
- StatusService MUST run after the full atomic operation completes
- Re-adding a removed office: find existing DocumentRecipient row (no Received log) → reactivate. Do NOT create duplicate row.

SECTION 8 — QUICK REFERENCE

8. Quick Reference

8.1 Useful Artisan Commands

Artisan Commands

```
# Migrations
php artisan migrate                          # Run pending migrations
php artisan migrate:status                  # Check migration status
php artisan migrate:rollback                 # Rollback last batch
php artisan make:migration add_urgency_to_transactions

# Models & Controllers
php artisan make:model DocumentNote -m       # Model + migration
php artisan make:controller DocumentNotesController --api
php artisan make:request StoreDocumentNoteRequest

# Testing
php artisan test
php artisan test --filter test_method_name

# Cache
php artisan config:clear
php artisan cache:clear
php artisan route:clear

# Routes
php artisan route:list --path=api/transactions
```

8.2 Key File Locations

CLAUDE.md	Project root — master context for Claude Code
routes/api.php	All API route definitions
app/Http/Controllers/	All controllers
app/Models/	All Eloquent models
app/Services/TransactionStatusService.php	Completion logic — update here
database/migrations/	All DB migrations — never edit existing

client/src/composables/	useTransaction, useActionVisibility, useToast
client/src/stores/	Pinia stores: auth, document, libraries
client/src/router/index.ts	Vue Router routes
client/src/api/index.ts	Axios instance + auth interceptor

8.3 Chapter Implementation Status

Chapter / Module	Status	Notes
Chapter 1 — Transaction Flow	**Partial**	Core actions implemented. Many actions missing or incomplete.
Chapter 2 — Dashboard Module	**Not Started**	Design complete. Implementation pending.
Chapter 2 — Reports Module	**Not Started**	Design complete. Implementation pending.
Chapter 2 — Advanced Search	**Not Started**	Design complete. AI deferred to Chapter 4.
Chapter 2 — Templating	**Not Started**	Design complete. Implementation pending.
Chapter 2 — Notifications Center	**Not Started**	Design pending. Chapter 2 design in progress.
Chapter 2 — User & Office Settings	**Not Started**	Design pending.
Chapter 2 — Signatory Management	**Not Started**	Design pending.
Chapter 3 — System Administration	**Not Started**	Design pending.
Chapter 4 — AI + Future Features	**Not Started**	Deferred. AI integration planned.

End of Development Procedures

Document Tracking System · Version 1.0 · 2026