Upload Flow Diagram

Visual Flow of New Timesheet Upload System

Overview

The refactored timesheet upload system now follows the same pattern as the working contract upload system, with all S3 operations handled by the backend.

Complete Upload Flow

```
╔═══════════════════════════════════════════════════════════════════════════╗
║                    TIMESHEET FILE UPLOAD FLOW                             ║
║                  (FIXED - Matches Contract Pattern)                       ║
╚═══════════════════════════════════════════════════════════════════════════╝

┌──────────────────┐
│  1. USER ACTION  │
└──────────────────┘
         │
         ▼
┌─────────────────────────────────────────────────────────────────────────┐
│ TimesheetSubmissionForm Component                                       │
│ ┌─────────────────────────────────────────────────────────────────────┐ │
│ • User fills timesheet details                                          │
│ • User selects main timesheet file (optional)                           │
│ • User adds expenses with receipt files (optional)                      │
│ • User clicks "Submit"                                                  │
└─────────────────────────────────────────────────────────────────────────┘
         │
         ▼
┌─────────────────────────────────────────────────────────────────────────┐
│ STEP 1: Create Timesheet (createRange mutation)                         │
│ ┌─────────────────────────────────────────────────────────────────────┐ │
│ Input:                                                                  │
│   • contractId, startDate, endDate                                      │
│   • hoursPerDay, notes                                                  │
│   • expenses[] (without files)                                          │
│                                                                         │
│ Output:                                                                 │
│   • timesheetId (used for file uploads)                                 │
└─────────────────────────────────────────────────────────────────────────┘
         │
         ▼
┌─────────────────────────────────────────────────────────────────────────┐
│ STEP 2: Upload Files (Sequential)                                       │
│ ┌─────────────────────────────────────────────────────────────────────┐ │
│ For each file (main timesheet + expense receipts):                     │
│                                                                         │
│   2a. Convert File to Base64                                            │
│   ┌─────────────────────────────────────────────────────────────────┐   │
│   │ const base64 = await fileToBase64(file);                        │   │
│   │                                                                 │   │
│   │ function fileToBase64(file: File) {                             │   │
│   │   return new Promise((resolve, reject) => {                     │   │
│   │     const reader = new FileReader();                            │   │
│   │     reader.readAsDataURL(file);                                 │   │
│   │     reader.onload = () => {                                     │   │
│   │       const result = reader.result as string;                   │   │
│   │       const base64 = result.split(',')[1];                      │   │
│   │       resolve(base64);                                          │   │
│   │     };                                                          │   │
│   │   });                                                           │   │
│   │ }                                                               │   │
│   └─────────────────────────────────────────────────────────────────┘   │
│                                                                         │
│   2b. Call Upload Mutation                                              │
│   ┌─────────────────────────────────────────────────────────────────┐   │
│   │ await uploadTimesheetDocument.mutateAsync({                     │   │
│   │   timesheetId,                                                  │   │
│   │   fileName: file.name,                                          │   │
│   │   fileBuffer: base64,  // ✅ Send base64                        │   │
│   │   fileSize: file.size,                                          │   │
```


mimeType: file.type,
description: "...",
category: "timesheet" | "expense",
});
BACKEND PROCESSING
TRPC Router: timesheet.uploadExpenseDocument
Input Validation:
z.object({
timesheetId: z.string(),
fileName: z.string(),
fileBuffer: z.string(), // ✅ base64
fileSize: z.number(),
mimeType: z.string().optional(),
description: z.string().optional(),
category: z.string().default("expense"),
})
STEP 3a: Verify Ownership & Status
Check timesheet exists
Verify submittedBy = current user
Ensure status = "draft" (only draft timesheets can upload)
if (!ts) throw NOT_FOUND
if (ts.status !== "draft") throw BAD_REQUEST
STEP 3b: Upload to S3
const { uploadFile } = await import("@/lib/s3");
// Convert base64 to Buffer
const buffer = Buffer.from(input.fileBuffer, "base64");
// Build S3 path
const s3FileName =
`tenant_${tenantId}/timesheet/${timesheetId}/
${Date.now()}-${fileName}`;
// Upload to S3
try {
s3Key = await uploadFile(
buffer,
s3FileName,
mimeType || "application/octet-stream"
);
} catch (error) {
throw INTERNAL_SERVER_ERROR;

```
}
```

▼

S3 Upload Helper (lib/s3.ts)

```
export async function uploadFile(
  buffer: Buffer,
  fileName: string,
  contentType?: string
): Promise<string> {
  const key = buildKey(fileName);

  const command = new PutObjectCommand({
    Bucket: bucketName,
    Key: key,
    Body: buffer,
    ContentType: contentType,
  });

  await s3Client.send(command);
  return key;  // Return S3 key
}
```

▼

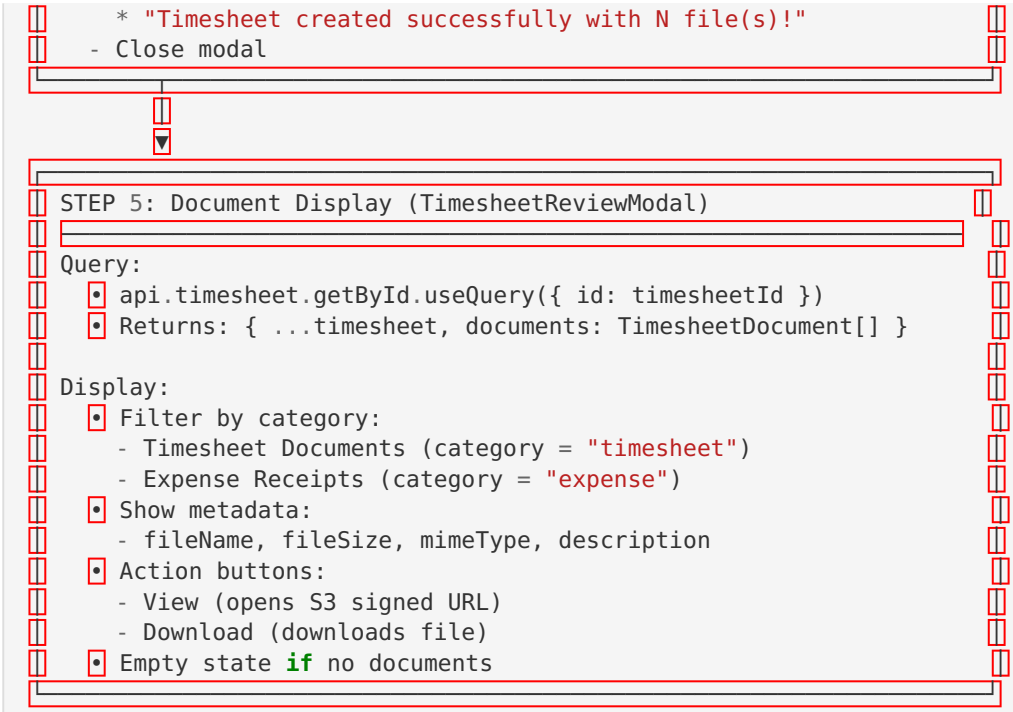
STEP 3c: Create Database Record

```
const document = await prisma.timesheetDocument.create({
  data: {
    timesheetId,
    fileName,
    fileUrl: s3Key,  // ✅ Store S3 key
    fileSize,
    mimeType,
    description,
    category,  // "timesheet" or "expense"
  },
});

console.log("Document uploaded:", {
  documentId: document.id,
  timesheetId,
  s3Key,
  fileName,
});

return document;
```

▼

STEP 4: Frontend Success Handling

- Increment uploadedCount
- Continue with next file (if any)
- After all files uploaded:
  - Invalidate queries:
    * utils.timesheet.getMyTimesheets.invalidate()
    * utils.timesheet.getById.invalidate({ id: timesheetId })
  - Show success toast:


* "Timesheet created successfully with N file(s)!"
- Close modal
STEP 5: Document Display (TimesheetReviewModal)
Query:
api.timesheet.getById.useQuery({ id: timesheetId })
Returns: { ...timesheet, documents: TimesheetDocument[] }
Display:
Filter by category:
- Timesheet Documents (category = "timesheet")
- Expense Receipts (category = "expense")
Show metadata:
- fileName, fileSize, mimeType, description
Action buttons:
- View (opens S3 signed URL)
- Download (downloads file)
Empty state if no documents

Error Handling Flow

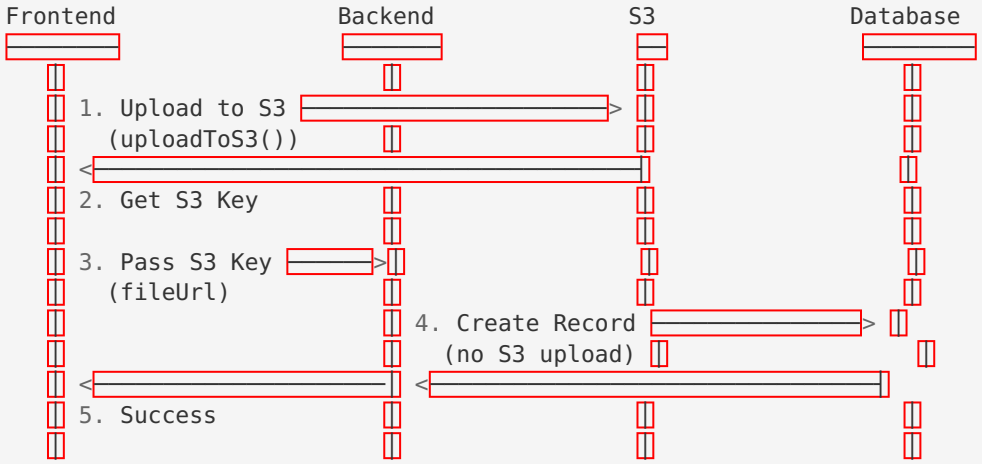
Data Flow Diagram


User Browser
(React Client)
1. File selected (File object)
fileToBase64(file)
Input: File object
Output: Base64 string
2. Base64 string
uploadTimesheetDocument.mutateAsync()
Payload:
{
timesheetId: "xyz",
fileName: "receipt.pdf",
fileBuffer: "JVBERi0xLjQK...",
fileSize: 102400,
mimeType: "application/pdf",
category: "expense"
}
3. TRPC Request (HTTP)
Backend TRPC Router
(timesheet.uploadExpenseDocument)
4. Validate & Extract
Buffer.from(fileBuffer, "base64")
Input: Base64 string
Output: Buffer
5. Buffer
lib/s3.ts
uploadFile()
AWS SDK: PutObjectCommand
Bucket: payroll-saas-documents
Key: tenant_123/timesheet/xyz/...
Body: Buffer


6. S3 Upload Success
Amazon S3
File stored at:
s3://bucket/tenant_123/timesheet/...
Returns: S3 Key
7. S3 Key
Prisma (PostgreSQL)
timesheetDocument.create()
INSERT INTO timesheet_documents
(id, timesheetId, fileName,
fileUrl, fileSize, mimeType,
category, uploadedAt)
VALUES
('doc-123', 'xyz', 'receipt.pdf',
's3://...', 102400, 'app/pdf',
'expense', NOW())
8. DB Record Created
Return TimesheetDocument
{
id: "doc-123",
timesheetId: "xyz",
fileName: "receipt.pdf",
fileUrl: "s3://...",
fileSize: 102400,
category: "expense",
uploadedAt: "2024-01-15T10:30:00Z"
}
9. TRPC Response
Frontend Success Handler
• Increment uploadedCount
• Invalidate queries
• Show success toast

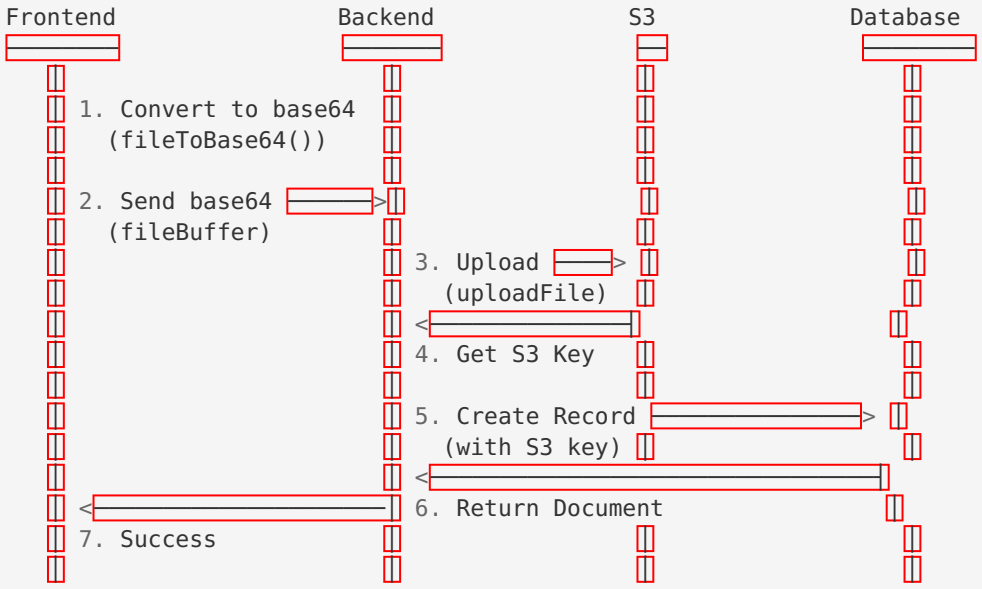
Comparison: Before vs After

Before Fix (Broken)



❌ Problems:
- Frontend has S3 credentials (security risk)
- Split operations (not atomic)
- Inconsistent error handling
- Orphaned S3 files if DB fails

After Fix (Working)



✅ Benefits:
- S3 credentials only in backend (secure)
- Atomic operations (reliable)
- Centralized error handling
- No orphaned files
- Matches contract pattern (consistent)

Summary

The refactored timesheet upload system now follows a clean, secure, and reliable pattern:

1. **Frontend**: Converts files to base64 and sends to backend
2. **Backend**: Handles S3 upload and database record creation
3. **S3**: Stores files with organized paths
4. **Database**: Stores metadata and S3 keys
5. **Display**: Queries TimesheetDocument records and displays

All operations are atomic, secure, and consistent with the working contract upload system.