Payroll SaaS Project Analysis

Branch: expenses-structure
Date: December 21, 2025
Purpose: Analysis for implementing invoice expenses integration and payment workflow enhancements

---

Executive Summary

This document provides a comprehensive analysis of the payroll SaaS application codebase to identify the key files and components that need to be modified for the following requirements:

1. **Fix invoice total amount to include expenses**
2. **Fix payment timeline to show users in “By:” and “Confirmed By:” fields**
3. **Update calculation and margin section to include expenses**
4. **Add post-payment workflow actions based on salary type (GROSS, PAYROLL, PAYROLL_WE_PAY, SPLIT)**

Technology Stack

Frontend

- **Framework:** Next.js 14.2.28 (App Router)
- **UI Library:** React 18.2.0
- **Styling:** Tailwind CSS 3.3.3
- **Component Library:** Radix UI (shadcn/ui pattern)
- **State Management:** React Query (via tRPC), Zustand 5.0.3, Jotai 2.6.0
- **Forms:** React Hook Form 7.53.0 + Zod 3.23.8
- **Date Handling:** date-fns 3.6.0
- **Notifications:** Sonner 1.5.0

Backend

- **API:** tRPC 11.7.1 (type-safe API)
- **Database:** PostgreSQL (via Prisma)
- **ORM:** Prisma 6.7.0
- **Authentication:** NextAuth.js 4.24.11
- **File Storage:** AWS S3
- **Email:** SendGrid
- **SMS:** Twilio
- **Queue:** BullMQ 5.36.1 + Redis (Upstash)
- **Logging:** Winston 3.18.3

Database Schema Analysis

Key Models

Invoice Model

Location: `prisma/schema.prisma`

```
model Invoice {
  id            String  @id @default(cuid())
  tenantId      String
  contractId    String?
  timesheetId   String? @unique
  invoiceNumber String? @unique

  // Financial
  amount           Decimal
  baseAmount       Decimal?
  marginAmount     Decimal?
  marginPercentage Decimal?
  marginPaidBy     String?
  totalAmount      Decimal @default(0)

  // Payment tracking
  agencyMarkedPaidAt     DateTime?
  agencyMarkedPaidBy     String?
  amountPaidByAgency     Decimal?
  paymentReceivedAt      DateTime?
  paymentReceivedBy      String?
  amountReceived         Decimal?

  // Relations
  timesheet              Timesheet?
  margin                 Margin?
  lineItems              InvoiceLineItem[]
  sender                 User?
  receiver               User?
  agencyMarkedPaidByUser User?
  paymentReceivedByUser  User?
}
```

Key Finding: The Invoice model has payment tracking fields but doesn't have a direct `totalExpenses` field. Expenses are currently stored as line items with descriptions containing "expense".

Timesheet Model

Location: `prisma/schema.prisma`

```
model Timesheet {
  id       String @id @default(cuid())
  tenantId String

  // TOTALS & AMOUNT BREAKDOWN
  totalHours    Decimal
  baseAmount    Decimal?  // Work amount (hours × rate)
  marginAmount  Decimal?  // Calculated margin (HIDDEN from contractors)
  totalExpenses Decimal? @default(0)  // Sum of all expenses
  totalAmount   Decimal?  // Final total (baseAmount + marginAmount + totalExpenses)

  // Relations
  expenses     Expense[]  // Link to Expense table
}
```

Key Finding: Timesheets already have a proper `totalExpenses` field and relationship with Expense model. When invoices are created from timesheets, this expense data should be transferred.

Expense Model

Location: `prisma/schema.prisma`

```
model Expense {
  id          String @id @default(cuid())
  tenantId    String
  submittedBy String
  contractId  String?
  timesheetId String?

  title       String
  description String?
  amount      Decimal
  currency    String
  category    String

  status      String @default("draft")

  // Relations
  timesheet   Timesheet?
  contract    Contract?
}
```

Key Finding: Expenses can be linked to both timesheets and contracts. They have their own approval workflow.

PaymentModel Enum

Location: `prisma/schema.prisma`

```
enum PaymentModel {
  GROSS              // Client pays gross amount
  PAYROLL            // Payroll partner handles payment
  PAYROLL_WE_PAY     // We pay contractor, payroll reimburses
  SPLIT              // Split payment across multiple methods
}
```

Key Files and Components Analysis

1. Invoice Calculation Issues

Current Implementation

File: `app/(dashboard)/(modules)/invoices/[id]/page.tsx` (Lines 160-185)

```
// Calculate totals from line items
const lineItemsTotals = useMemo(() => {
  if (!data?.lineItems) return { subtotal: 0, expenses: 0, workTotal: 0 };

  let workTotal = 0;
  let expenses = 0;

  data.lineItems.forEach((item: any) => {
    const amount = Number(item.amount || 0);
    if (item.description?.toLowerCase().includes('expense')) {
      expenses += amount;
    } else {
      workTotal += amount;
    }
  });

  return {
    subtotal: workTotal + expenses,
    expenses,
    workTotal,
  };
}, [data?.lineItems]);
```

Issues Identified:
1. Expenses are identified by checking if description includes “expense” - this is fragile
2. Invoice total calculation doesn’t properly include expenses from related timesheets
3. No separate expense tracking beyond line items

Files to Modify:

1. **Invoice Router** - `server/api/routers/invoice.ts`
   - Add logic to fetch expenses from related timesheet
   - Calculate totalExpenses separately
   - Include expenses in totalAmount calculation

2. **Invoice Detail Page** - `app/(dashboard)/(modules)/invoices/[id]/page.tsx`
   - Update calculation logic to use timesheet expenses if available
   - Display expenses from Expense table instead of line items

3. **Database Schema** - `prisma/schema.prisma`
   - Consider adding `totalExpenses` field to Invoice model
   - Add relation from Invoice to Expense

2. Payment Timeline User Display Issues

Current Implementation

File: `components/invoices/PaymentTrackingCard.tsx` (Lines 168-191)

```
{/* Agency Marked Paid */}
{paymentStatus.agencyMarkedPaidAt && (
  <div className="flex items-start gap-3 p-3 rounded-lg bg-white border">
    <div className="flex-1">
      <p className="text-sm font-medium">Marked as Paid by Agency</p>
      <p className="text-xs text-muted-foreground">
        {format(new Date(paymentStatus.agencyMarkedPaidAt), "PPpp")}
      </p>
      {paymentStatus.agencyMarkedPaidBy && (
        <p className="text-xs text-muted-foreground">
          By: {paymentStatus.agencyMarkedPaidBy.name}
        </p>
      )}
    </div>
  </div>
)}
```

Issues Identified:

1. The component expects `paymentStatus.agencyMarkedPaidBy` to be a user object with a `name` property
2. However, the Invoice model stores `agencyMarkedPaidBy` as a string (userId)
3. The same issue exists for `paymentReceivedBy` field

Files to Modify:

1. **Invoice Router** - `server/api/routers/invoice.ts`
   - Include user relations in invoice queries:
   ```
   typescript
       include: {
         agencyMarkedPaidByUser: {
           select: { id: true, name: true, email: true }
         },
         paymentReceivedByUser: {
           select: { id: true, name: true, email: true }
         }
       }
   ```

2. **PaymentTrackingCard** - `components/invoices/PaymentTrackingCard.tsx`
   - Update interface to expect user objects
   - Handle cases where user data might be missing

3. **Invoice Detail Page** - `app/(dashboard)/(modules)/invoices/[id]/page.tsx`
   - Pass correct user objects to PaymentTrackingCard component

3. Calculation and Margin Section Updates

Current Implementation

File: `app/(dashboard)/(modules)/invoices/[id]/page.tsx` (Lines 870-920)

```
{/* MARGINS & TOTALS */}
<div className="space-y-3">
  <div className="flex justify-end">
    <div className="w-96 space-y-3">
      {/* Base Amount (Subtotal) */}
      <div className="flex justify-between items-center px-4 py-2">
        <span className="text-sm">Subtotal (Base Amount):</span>
        <span className="font-medium">
          {formatCurrency(Number(data.baseAmount || data.amount || 0))}
        </span>
      </div>

      {/* Margin Calculation */}
      {marginBreakdown && marginBreakdown.marginAmount > 0 && (
        // ... margin display
      )}

      {/* Total Amount */}
      <div className="flex justify-between items-center px-4 py-4 bg-green-600 text-
white rounded-lg">
        <span className="text-lg font-bold">TOTAL AMOUNT DUE:</span>
        <span className="text-2xl font-bold">
          {formatCurrency(Number(data.totalAmount || 0))}
        </span>
      </div>
    </div>
  </div>
</div>
```

Issues Identified:

1. Expenses are displayed separately from the main calculation section
2. The total calculation flow doesn't clearly show: Base + Expenses + Margin = Total
3. MarginCalculationDisplay component doesn't account for expenses

Files to Modify:

1. **MarginCalculationDisplay Component** - `components/workflow/MarginCalculationDisplay.tsx`
   - Add `totalExpenses` to the breakdown interface
   - Display expenses as a separate line item in the calculation
   - Update calculation flow: Base + Expenses = Subtotal, Subtotal + Margin = Total

2. **Invoice Detail Page** - `app/(dashboard)/(modules)/invoices/[id]/page.tsx`
   - Update `marginBreakdown` useMemo to include expenses
   - Reorganize calculation section to show clearer flow
   - Consider using a dedicated calculation card component

4. Post-Payment Workflow Actions

Current Implementation

File: `lib/services/PaymentWorkflowService.ts`

The service already has comprehensive logic for handling different payment models:
- `executeGrossPaymentWorkflow()` - Lines 94-132
- `executePayrollWorkflow()` - Lines 138-195
- `executePayrollWePayWorkflow()` - Lines 201-310
- `executeSplitPaymentWorkflow()` - Lines 316-452

Issues Identified:
1. The workflows are defined but may not be triggered automatically after payment confirmation
2. Need to ensure these workflows are called in the invoice payment received flow

Files to Modify:

1. **Invoice Router** - `server/api/routers/invoice.ts`
   - In `markPaymentReceived` mutation, add call to PaymentWorkflowService
   - Trigger appropriate workflow based on paymentModel
   - Example:
   ```typescript
   // After marking payment as received
   if (invoice.paymentModel) {
     await PaymentWorkflowService.executePaymentWorkflow({
       invoiceId: invoice.id,
       paymentModel: invoice.paymentModel,
       userId: ctx.session.user.id,
       tenantId: ctx.tenantId,
     });
   }
   ```

2. **Invoice Detail Page** - `app/(dashboard)/(modules)/invoices/[id]/page.tsx`
   - Display post-payment workflow tasks/status
   - Show next steps based on payment model

3. **PaymentWorkflowService** - `lib/services/PaymentWorkflowService.ts`
   - Ensure all workflow methods properly handle invoice totals with expenses
   - Add proper error handling and rollback logic

RBAC Permissions System

Relevant Permissions

File: `server/rbac/permissions.ts`

```
export enum Resource {
  INVOICE = "invoice",
  PAYMENT = "payment",
  EXPENSE = "expense",
}

export enum Action {
  CREATE = "create",
  READ = "read",
  UPDATE = "update",
  DELETE = "delete",
  LIST = "list",
  APPROVE = "approve",
  PAY = "pay",
  CONFIRM = "confirm",
}

export enum PermissionScope {
  GLOBAL = "global",
  OWN = "own",
  TENANT = "tenant",
}
```

Invoice Permissions

File: `lib/workflows/invoice-state-machine.ts`

```
export const InvoicePermissions = {
  SUBMIT_OWN: 'invoice.submit.own',
  CONFIRM_MARGIN_OWN: 'invoice.confirmMargin.own',
  PAY_OWN: 'invoice.pay.own',

  LIST_ALL: 'invoice.list.global',
  REVIEW_ALL: 'invoice.review.global',
  APPROVE_ALL: 'invoice.approve.global',
  SEND_ALL: 'invoice.send.global',
  MARK_PAID_ALL: 'invoice.pay.global',
  CONFIRM_PAYMENT_ALL: 'invoice.confirm.global',
}
```

Key Findings:
- Proper RBAC is in place for all invoice operations
- Agency users can mark invoices as paid (PAY_OWN)
- Admin users can confirm payment received (CONFIRM_PAYMENT_ALL)
- No changes needed to RBAC for the required modifications

Invoice Workflow State Machine

States

File: `lib/workflows/invoice-state-machine.ts`

```
export enum InvoiceState {
  DRAFT = 'draft',
  SUBMITTED = 'submitted',
  PENDING_MARGIN_CONFIRMATION = 'pending_margin_confirmation',
  UNDER_REVIEW = 'under_review',
  APPROVED = 'approved',
  SENT = 'sent',
  MARKED_PAID_BY_AGENCY = 'marked_paid_by_agency',
  PAYMENT_RECEIVED = 'payment_received',  // <-- Trigger payment workflows here
  REJECTED = 'rejected',
  PAID = 'paid',
}
```

Payment Workflow Trigger Point

The `PAYMENT_RECEIVED` state is where post-payment workflows should be triggered based on the paymentModel.

Summary of Required Changes

Priority 1: Fix Invoice Total Calculation with Expenses

Files to Modify:

1. `server/api/routers/invoice.ts`
- Add expense fetching from timesheet
- Calculate totalExpenses properly
- Include in totalAmount calculation

1. `app/(dashboard)/(modules)/invoices/[id]/page.tsx`
   - Update calculation display logic
   - Show expenses from Expense table

2. `prisma/schema.prisma` (Optional but Recommended)
   - Add `totalExpenses` field to Invoice model
   - Add InvoiceExpense relation table

Priority 2: Fix Payment Timeline User Display

Files to Modify:

1. `server/api/routers/invoice.ts`
- Include user relations in queries:
- `agencyMarkedPaidByUser`
- `paymentReceivedByUser`

1. `components/invoices/PaymentTrackingCard.tsx`
   - Update interface expectations
   - Handle user objects properly

2. `app/(dashboard)/(modules)/invoices/[id]/page.tsx`
   - Pass correct user objects to component

Priority 3: Update Calculation and Margin Display

Files to Modify:

1. `components/workflow/MarginCalculationDisplay.tsx`
- Add expenses to breakdown
- Update calculation flow display

1. `app/(dashboard)/(modules)/invoices/[id]/page.tsx`
   - Reorganize calculation section
   - Show clearer flow: Base + Expenses + Margin = Total

Priority 4: Add Post-Payment Workflow Actions

Files to Modify:

1. `server/api/routers/invoice.ts`
- In `markPaymentReceived` mutation:
- Call PaymentWorkflowService.executePaymentWorkflow()
- Pass invoice paymentModel

1. `app/(dashboard)/(modules)/invoices/[id]/page.tsx`
   - Display workflow status after payment
   - Show next steps for each payment model

2. `lib/services/PaymentWorkflowService.ts`
   - Ensure workflows handle expenses properly
   - Add better error handling

Recommended Implementation Order

1. **Phase 1: Database and Backend Logic**
   - Update Invoice model (add totalExpenses field if needed)
   - Update invoice router to fetch and calculate expenses
   - Include user relations in payment tracking

2. **Phase 2: Component Updates**
   - Fix PaymentTrackingCard user display
   - Update MarginCalculationDisplay to include expenses
   - Update invoice detail page calculations

3. **Phase 3: Workflow Integration**
   - Integrate PaymentWorkflowService with payment received flow
   - Test all payment models (GROSS, PAYROLL, PAYROLL_WE_PAY, SPLIT)
   - Display workflow status in UI

4. **Phase 4: Testing and Validation**
   - Test expense calculations with various scenarios
   - Verify user display in payment timeline
   - Test payment workflows for all models
   - Verify RBAC permissions work correctly

Technical Notes

Expense Handling Strategy

Currently, expenses are stored in two ways:
1. **As InvoiceLineItems** - with description containing “expense”
2. **As Expense records** - linked to timesheets

Recommendation: Transition to using the Expense model exclusively:
- Create proper Invoice-Expense relations
- Migrate existing expense line items to Expense records
- Update all calculations to use Expense model

Payment Model Workflow Triggers

The PaymentWorkflowService should be triggered at the `PAYMENT_RECEIVED` state:

```
// In invoice.markPaymentReceived mutation
await StateTransitionService.executeTransition({
  entityType: WorkflowEntityType.INVOICE,
  entityId: invoiceId,
  action: WorkflowAction.MARK_PAYMENT_RECEIVED,
  userId: ctx.session.user.id,
  tenantId: ctx.tenantId,
});

// After state transition succeeds
if (invoice.paymentModel) {
  const workflowResult = await PaymentWorkflowService.executePaymentWorkflow({
    invoiceId: invoice.id,
    paymentModel: invoice.paymentModel,
    userId: ctx.session.user.id,
    tenantId: ctx.tenantId,
  });

  // Store workflow result in invoice metadata or create tasks
}
```

Files Reference

Core Files to Modify

1. **Database Schema**
   - `prisma/schema.prisma`
2. **Backend API**
   - `server/api/routers/invoice.ts`
   - `server/api/routers/expense.ts`
3. **Services**
   - `lib/services/PaymentWorkflowService.ts`
   - `lib/services/MarginCalculationService.ts`
4. **UI Components**
   - `app/(dashboard)/(modules)/invoices/[id]/page.tsx`

- `components/invoices/PaymentTrackingCard.tsx`
- `components/workflow/MarginCalculationDisplay.tsx`

5. **Workflow**
   - `lib/workflows/invoice-state-machine.ts`

Supporting Files

- `server/rbac/permissions.ts` - Permission definitions
- `lib/services/StateTransitionService.ts` - State machine execution
- `lib/audit.ts` - Audit logging for changes

Next Steps

1. Review this analysis with the development team
2. Create detailed technical specifications for each change
3. Set up development environment and branch
4. Implement changes in the recommended order
5. Write tests for each modification
6. Deploy to staging for testing
7. Conduct UAT (User Acceptance Testing)
8. Deploy to production

End of Analysis