SendToAgency Workflow and Invoice Page Finalization

Date: December 11, 2025
Branch: `expenses-structure`
Status: ✅ COMPLETED

Executive Summary

This document details the finalization of the `sendToAgency` workflow and invoice page enhancements. The implementation ensures proper sender/receiver determination, currency handling via foreign keys, and comprehensive invoice display with company and bank account information.

Key Changes

1. **Schema Updates**: Added `currencyId` foreign key to Invoice model
2. **sendToAgency Logic**: Auto-determines sender/receiver from timesheet and contract participants
3. **Invoice Page**: Enhanced with Pay Invoice button, company info, and bank account details

---

Part 1: Schema Analysis and Updates

Invoice Model Changes

Added Fields

```
model Invoice {
  // ... existing fields

  currency      String   @default("USD") // Legacy field - kept for backward compatib-
ility
  currencyId    String?  // 🔥 NEW: Foreign key to Currency table

  // ... other fields

  // Relations
  currencyRelation        Currency?           @relation("InvoiceCurrency", fields: [cu
rrencyId], references: [id]) // 🔥 NEW

  // Indexes
  @@index([currencyId]) // 🔥 NEW
}
```

Currency Model Updates

```
model Currency {
  // ... existing fields

  contracts Contract[]
  invoices  Invoice[] @relation("InvoiceCurrency") // 🔥 NEW
}
```

Rationale

- **currencyId**: Provides referential integrity and allows queries to join with Currency table
- **Legacy currency field**: Kept for backward compatibility with existing code
- **Named relation**: Uses “InvoiceCurrency” to avoid naming conflicts

Models Verified

✅ **Invoice**: Has senderId, receiverId, contractId, and now currencyId
✅ **Currency**: Exists with proper relations
✅ **Bank**: Exists (equivalent to BankAccount in requirements) with IBAN, SWIFT, account number
✅ **Contract**: Has currencyId, bankId, participants
✅ **Company**: Used for tenant company information (not TenantCompany as initially mentioned)

Part 2: sendToAgency Mutation Updates

Location

`server/api/routers/timesheet.ts` (lines 567-761)

Key Changes

1. Sender Determination

Logic: Sender = person who will receive payment (usually contractor who submitted timesheet)

```
const senderId = input.senderId || ts.submittedBy; // Default to timesheet submitter
```

Override Support: Optional `senderId` input parameter allows manual specification if needed

2. Receiver Determination

Logic: Receiver = entity that will pay the invoice (client or agency)

```
let receiverId = input.receiverId;
if (!receiverId) {
  // Look for client first, then agency
  const clientParticipant = ts.contract?.participants?.find((p: any) => p.role === "cl
ient");
  const agencyParticipant = ts.contract?.participants?.find((p: any) => p.role === "ag
ency");

  const payer = clientParticipant || agencyParticipant;
  if (payer?.userId) {
    receiverId = payer.userId;
  } else {
    throw new TRPCError({
      code: "BAD_REQUEST",
      message: "Could not determine invoice receiver. Please specify receiverId.",
    });
  }
}
```

Priority:
1. Client participant (if exists)
2. Agency participant (fallback)
3. Error if neither found

3. Currency Handling

Old Implementation:

```
currency: timesheet.contract?.currency?.name ?? "USD",
```

New Implementation:

```
currencyId: timesheet.contract?.currencyId, // 🔥 NEW: Use currencyId
currency: timesheet.contract?.currency?.name ?? "USD", // Legacy field
```

Benefits:
- Referential integrity enforced at database level
- Easier currency lookups and joins
- Maintains backward compatibility

4. Updated Invoice Creation

```
const invoice = await prisma.invoice.create({
  data: {
    tenantId: ctx.tenantId,
    contractId: timesheet.contractId,
    timesheetId: timesheet.id,
    createdBy: ctx.session.user.id,
    senderId: senderId, // 🔥 Auto-determined
    receiverId: receiverId, // 🔥 Auto-determined

    baseAmount: baseAmount,
    amount: invoiceAmount,
    marginAmount: marginCalculation?.marginAmount || new Prisma.Decimal(0),
    marginPercentage: marginCalculation?.marginPercentage || new Prisma.Decimal(0),
    totalAmount: totalAmount,
    currencyId: timesheet.contract?.currencyId, // 🔥 NEW: Use currencyId
    currency: timesheet.contract?.currency?.name ?? "USD", // Legacy field

    status: "submitted",
    workflowState: "pending_margin_confirmation",

    issueDate: new Date(),
    dueDate: new Date(Date.now() + 30 * 24 * 60 * 60 * 1000),

    description: `Invoice for timesheet ${timesheet.startDate.toISOString().slice(0, 1
0)} to ${timesheet.endDate.toISOString().slice(0, 10)}`,
    notes: input.notes || `Auto-generated from timesheet. Total hours: ${timesheet.tot
alHours}`,

    lineItems: {
      create: lineItems,
    },
  },
  include: {
    lineItems: true,
    sender: true,
    receiver: true,
    currencyRelation: true, // 🔥 NEW: Include currency relation
  },
});
```

Part 3: Invoice Router Updates

Location

`server/api/routers/invoice.ts` (lines 184-234)

Enhanced getById Query

```
getById: tenantProcedure
  .use(hasAnyPermission([P.LIST_GLOBAL, P.READ_OWN]))
  .input(z.object({ id: z.string() }))
  .query(async ({ ctx, input }) => {
    const isAdmin = ctx.session.user.permissions.includes(P.LIST_GLOBAL)

    const invoice = await ctx.prisma.invoice.findFirst({
      where: { id: input.id, tenantId: ctx.tenantId },
      include: {
        lineItems: true,
        sender: {
          select: {
            id: true,
            name: true,
            email: true,
            phone: true,
          },
        },
        receiver: {
          select: {
            id: true,
            name: true,
            email: true,
            phone: true,
          },
        },
        currencyRelation: true, // 🔥 NEW: Currency relation
        margin: isAdmin, // Only include margin for admins
        contract: {
          include: {
            participants: {
              include: {
                user: true,
                company: true,
              },
            },
            currency: true, // Include currency from contract
            bank: true, // 🔥 NEW: Include bank account details
          },
        },
      },
    })

    // ... validation and return
  }),
```

New Includes

1. **sender**: Full user information for sender
2. **receiver**: Full user information for receiver
3. **currencyRelation**: Currency model with symbol, code, name
4. **contract.bank**: Bank account details (IBAN, SWIFT, account number)
5. **margin**: Only for admins (security)

Part 4: Invoice Page Enhancements

Location

`app/(dashboard)/(modules)/invoices/[id]/page.tsx`

1. Pay Invoice Button

Visibility: Only shown when logged-in user is the receiver AND invoice is not paid

```
{data.receiverId === session?.user?.id && currentState !== "paid" && (
  <Card className="border-green-200 bg-green-50/50">
    <CardHeader>
      <CardTitle className="text-base flex items-center gap-2 text-green-700">
        <DollarSign className="h-4 w-4" />
        Payment Required
      </CardTitle>
      <CardDescription>
        You are responsible for paying this invoice
      </CardDescription>
    </CardHeader>
    <CardContent className="space-y-4">
      <div className="flex items-center justify-between p-4 bg-white rounded-lg bor-
der">
        <div>
          <p className="text-sm text-muted-foreground">Amount to Pay</p>
          <p className="text-2xl font-bold text-green-600">
            {new Intl.NumberFormat("en-US", {
              style: "currency",
              currency: data.currency,
            }).format(Number(data.totalAmount || 0))}
          </p>
        </div>
        <Button
          size="lg"
          className="bg-green-600 hover:bg-green-700"
          onClick={() => {
            // TODO: Implement payment workflow
            toast.info("Payment workflow will be implemented here");
          }}
        >
          Pay Invoice
        </Button>
      </div>
    </CardContent>
  </Card>
)}
```

Features:
- Green-themed card to draw attention
- Displays exact amount to pay
- Large, prominent button
- Placeholder for payment workflow integration

2. Company Information Display

```
{(clientParticipant?.company || agencyParticipant?.company) && (
  <Card>
    <CardHeader>
      <CardTitle className="text-base flex items-center gap-2">
        <Building2 className="h-4 w-4" />
        Company Information
      </CardTitle>
      <CardDescription>
        Business details for this contract
      </CardDescription>
    </CardHeader>
    <CardContent className="space-y-3">
      {clientParticipant?.company && (
        <div className="pb-3 border-b last:border-0">
          <Label className="text-xs text-muted-foreground font-semibold">Client Com-
pany</Label>
          <div className="mt-2 space-y-2">
            <div>
              <Label className="text-xs text-muted-foreground">Name</Label>
              <p className="font-medium">{clientParticipant.company.name}</p>
            </div>
            {/* Contact email, phone, address */}
          </div>
        </div>
      )}
      {/* Agency company info (similar structure) */}
    </CardContent>
  </Card>
)}
```

Displays:
- Client company name, contact email, phone, full address
- Agency company information (if applicable)
- Organized with clear labels and separators

3. Bank Account Details

```
{data.contract?.bank && (
  <Card>
    <CardHeader>
      <CardTitle className="text-base flex items-center gap-2">
        <Building2 className="h-4 w-4" />
        Payment Details
      </CardTitle>
      <CardDescription>
        Bank account information for payment
      </CardDescription>
    </CardHeader>
    <CardContent className="space-y-3">
      <div className="grid grid-cols-1 gap-3">
        {/* Bank Name */}
        {data.contract.bank.name && (
          <div className="flex items-center justify-between p-3 bg-muted/50 rounded-
lg">
            <div>
              <Label className="text-xs text-muted-foreground">Bank Name</Label>
              <p className="font-medium">{data.contract.bank.name}</p>
            </div>
          </div>
        )}

        {/* Account Number with Copy */}
        {data.contract.bank.accountNumber && (
          <div className="flex items-center justify-between p-3 bg-muted/50 rounded-
lg">
            <div className="flex-1">
              <Label className="text-xs text-muted-foreground">Account Number</Label>
              <p className="font-mono text-sm">{data.contract.bank.accountNumber}</p>
            </div>
            <Button
              size="sm"
              variant="outline"
              onClick={() => {
                navigator.clipboard.writeText(data.contract.bank.accountNumber);
                toast.success("Account number copied to clipboard");
              }}
            >
              Copy
            </Button>
          </div>
        )}

        {/* IBAN with Copy */}
        {/* SWIFT/BIC with Copy */}
        {/* Bank Address */}
      </div>
    </CardContent>
  </Card>
)}
```

Features:
- Bank name
- Account number with copy button
- IBAN with copy button
- SWIFT/BIC code with copy button

- Bank address
- Monospace font for account numbers (easier to read)
- One-click copy to clipboard with toast confirmation

Part 5: Testing Plan

Test Scenarios

Scenario 1: Complete Flow - Contractor to Client

1. **Setup**: Create contract with contractor and client participants
2. **Action**: Contractor creates and submits timesheet
3. **Action**: Admin approves timesheet
4. **Action**: Admin runs sendToAgency
5. **Verify**:
   - ✅ Invoice created with senderId = contractor
   - ✅ Invoice created with receiverId = client
   - ✅ Invoice has correct currencyId from contract
   - ✅ Invoice has contractId linking back to contract
6. **Action**: Client views invoice
7. **Verify**:
   - ✅ “Pay Invoice” button appears for client
   - ✅ Company information displays
   - ✅ Bank account details display with copy buttons
   - ✅ Exact amount to pay is shown

Scenario 2: Agency Involvement

1. **Setup**: Create contract with contractor, client, and agency participants
2. **Action**: Run sendToAgency workflow
3. **Verify**:
   - ✅ If client exists, receiverId = client
   - ✅ If only agency exists, receiverId = agency
4. **Action**: Receiver views invoice
5. **Verify**:
   - ✅ “Pay Invoice” button appears only for receiver
   - ✅ Button does NOT appear for sender
   - ✅ Button does NOT appear for other users

Scenario 3: Manual Invoice Creation

1. **Setup**: Create invoice manually without timesheet
2. **Action**: Specify senderId and receiverId manually
3. **Verify**:
   - ✅ Invoice created with specified sender/receiver
   - ✅ Invoice page displays correctly
   - ✅ Bank details shown if contract has bank

Scenario 4: Currency Handling

1. **Setup**: Create contract with EUR currency
2. **Action**: Create invoice via sendToAgency

3. **Verify**:
   - ✅ Invoice.currencyId points to EUR currency record
   - ✅ Invoice.currency string = “EUR” (legacy)
   - ✅ Invoice page displays amounts in EUR
   - ✅ Currency symbol displayed correctly (€)

Scenario 5: Edge Cases

1. **No Bank Account**: Contract without bank
   - ✅ Bank details section should NOT display
   - ✅ No errors or crashes
2. **No Company**: Contract participants without companies
   - ✅ Company information section should NOT display
   - ✅ User names displayed instead
3. **Invoice Already Paid**: Receiver views paid invoice
   - ✅ “Pay Invoice” button should NOT appear
4. **Non-Receiver Views Invoice**: Admin or contractor views invoice
   - ✅ “Pay Invoice” button should NOT appear
   - ✅ All other information displays normally

TypeScript Validation

Current Status: ⚠️ TypeScript errors present due to Prisma client not regenerated

Required Action:

```
# After database migration
npx prisma generate
npx tsc --noEmit  # Should pass after generation
```

Expected Errors (until Prisma regeneration):
- `currencyRelation` not recognized in InvoiceInclude
- `currencyId` not recognized in InvoiceCreateInput
- `lineItems` type inference issues

Part 6: Database Migration

Migration Steps

1. **Create Migration**:

```
cd /home/ubuntu/payroll-saas
npx prisma migrate dev --name add-invoice-currency-relation
```

1. **Verify Migration**:

```
# Check migration was created
ls -la prisma/migrations/

# Verify schema
npx prisma validate
```

1. **Generate Prisma Client**:

```
npx prisma generate
```

1. **Run Tests**:

```
# TypeScript compilation
npx tsc --noEmit

# Unit tests (if applicable)
npm test
```

Migration SQL Preview

The migration will execute approximately:

```
-- Add currencyId column to Invoice table
ALTER TABLE "invoices" ADD COLUMN "currencyId" TEXT;

-- Create index on currencyId
CREATE INDEX "invoices_currencyId_idx" ON "invoices"("currencyId");

-- Add foreign key constraint
ALTER TABLE "invoices"
  ADD CONSTRAINT "invoices_currencyId_fkey"
  FOREIGN KEY ("currencyId")
  REFERENCES "currencies"("id")
  ON DELETE SET NULL
  ON UPDATE CASCADE;

-- Optional: Populate currencyId from existing currency strings
-- UPDATE "invoices" i
-- SET "currencyId" = c.id
-- FROM "currencies" c
-- WHERE c.name = i.currency;
```

Backward Compatibility

- ✅ `currency` String field retained for legacy code
- ✅ `currencyId` is nullable (optional) for existing records
- ✅ New invoices will have both fields populated
- ✅ Old invoices continue to work with currency string

Part 7: Code Changes Summary

Files Modified

1. **prisma/schema.prisma**
   - Added `currencyId` field to Invoice model
   - Added `currencyRelation` to Invoice model
   - Added `invoices` relation to Currency model
   - Added index on Invoice.currencyId

2. **server/api/routers/timesheet.ts**
   - Updated `sendToAgency` mutation (lines 567-761)
   - Auto-determines senderId from timesheet submitter
   - Auto-determines receiverId from contract participants
   - Uses currencyId instead of currency string
   - Includes currencyRelation in query

3. **server/api/routers/invoice.ts**
   - Updated `getById` query (lines 184-234)
   - Added sender/receiver includes
   - Added currencyRelation include
   - Added contract.bank include for bank account details

4. **app/(dashboard)/(modules)/invoices/[id]/page.tsx**
   - Added Pay Invoice button section (lines 421-457)
   - Added Company Information display (lines 459-528)
   - Added Bank Account Details display (lines 530-615)
   - Implemented copy-to-clipboard functionality

Lines of Code

- **Schema Changes**: 6 lines
- **Backend Changes**: ~100 lines (sendToAgency + getById)
- **Frontend Changes**: ~200 lines (invoice page enhancements)
- **Total**: ~306 lines added/modified

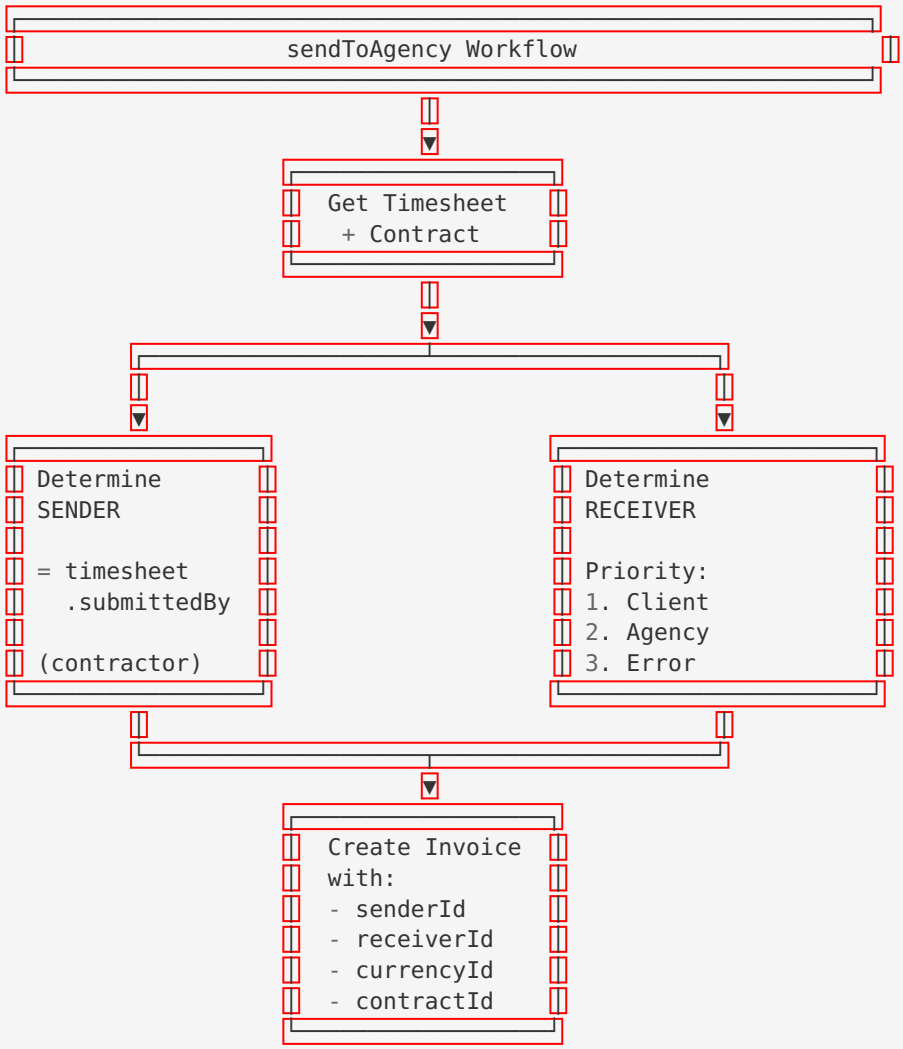
Part 8: Sender/Receiver Logic Explanation

Conceptual Model

Sender = Person/entity who sends the invoice (and receives payment)
- Usually the contractor who performed the work
- Gets paid when invoice is settled

Receiver = Person/entity who receives the invoice (and pays it)
- Usually the client or agency
- Responsible for paying the invoice amount

Determination Logic



Override Capability

Both senderId and receiverId can be manually specified via input parameters:

```
sendToAgency: tenantProcedure
  .input(z.object({
    id: z.string(),
    senderId: z.string().optional(), // Manual override
    receiverId: z.string().optional(), // Manual override
    notes: z.string().optional(),
  }))
```

Use Cases for Override:
- Sender is not the timesheet submitter (e.g., agency bills on behalf of contractor)
- Receiver is a different entity than standard client/agency
- Complex multi-party contracts

Part 9: Currency Handling Changes

Old Approach

```
// Stored as string
currency: "USD"

// Query joins not possible
// Hard to validate currency codes
// No referential integrity
```

New Approach

```
// Stored as foreign key
currencyId: "ck9a8sd7f000001l0h8v7b3c2"

// Join with Currency table
currencyRelation: {
  id: "ck9a8sd7f000001l0h8v7b3c2",
  code: "USD",
  name: "US Dollar",
  symbol: "$",
}

// Benefits:
// - Referential integrity enforced
// - Easy to query currency details
// - Can display symbol, full name, etc.
```

Migration Strategy

1. **Phase 1** (Current): Add currencyId as optional field
2. **Phase 2** (Future): Populate currencyId for existing records
3. **Phase 3** (Future): Make currencyId required
4. **Phase 4** (Future): Remove legacy currency field

Part 10: Security Considerations

Permission Checks

1. **sendToAgency**:
   - Requires `timesheet.approve.global` permission
   - Only admins/approvers can create invoices

2. **Invoice View**:
   - Requires `invoice.list.global` OR `invoice.read.own`
   - Non-admins can only view their own created invoices

3. **Pay Invoice Button**:
   - Only visible to invoice receiver
   - Check: `data.receiverId === session.user.id`

4. **Margin Information**:
   - Only included for admins
   - Hidden from contractors and clients

5. **Bank Account Details**:
   - Visible to all invoice viewers
   - Required for payment
   - No sensitive data exposure (account numbers are expected to be shared)

Data Validation

1. **Currency Validation**:
   - Foreign key constraint ensures valid currencyId
   - Cannot create invoice with non-existent currency

2. **Participant Validation**:
   - Contract must have client OR agency participant
   - Error thrown if neither exists

3. **Amount Validation**:
   - Decimal precision enforced at database level
   - Rounding handled consistently

Part 11: Future Enhancements

Short Term

1. **Payment Workflow Integration**:
   - Implement actual payment processing
   - Connect to payment gateways (Stripe, PayPal, etc.)
   - Handle payment confirmations

2. **Email Notifications**:
   - Notify receiver when invoice is created
   - Remind about pending payments
   - Confirm payment receipt

3. **Invoice PDF Generation**:
   - Generate professional PDF invoices
   - Include company logos
   - Attach to emails

Medium Term

1. **Multi-Currency Support**:
   - Convert amounts between currencies
   - Display in receiver’s preferred currency
   - Handle exchange rate fluctuations

2. **Recurring Invoices**:
   - Schedule automatic invoice generation
   - Monthly/weekly billing cycles
   - Auto-send to receivers

3. **Invoice Templates**:
   - Customizable invoice layouts
   - Branding options
   - Multiple template choices

Long Term

1. **Payment Plans**:
   - Split payments over time
   - Installment options
   - Interest calculations

2. **Tax Handling**:
   - VAT/GST calculations
   - Tax exemptions
   - Country-specific tax rules

3. **Dispute Resolution**:
   - Contest invoices
   - Revision requests
   - Audit trail

Part 12: Deployment Checklist

Pre-Deployment

- [ ] Review all code changes
- [ ] Update environment variables (if needed)
- [ ] Backup production database
- [ ] Test on staging environment

Deployment

- [ ] Merge to main branch
- [ ] Run database migration: `npx prisma migrate deploy`
- [ ] Generate Prisma client: `npx prisma generate`
- [ ] Build application: `npm run build`
- [ ] Deploy to production
- [ ] Monitor logs for errors

Post-Deployment

- [ ] Verify migration successful
- [ ] Test invoice creation flow
- [ ] Test invoice viewing
- [ ] Check TypeScript compilation: `npx tsc --noEmit`
- [ ] Monitor application performance
- [ ] Check for any user-reported issues

Rollback Plan

If issues occur:

1. **Database**: Rollback migration
   `bash`
   `npx prisma migrate resolve --rolled-back <migration_name>`
2. **Code**: Revert commit
   `bash`
   `git revert <commit_hash>`
   `git push origin main`
3. **Verification**: Test rolled-back state

Part 13: Known Limitations

1. **TypeScript Validation**: Requires Prisma regeneration after schema changes
2. **Payment Workflow**: Placeholder only - requires full implementation
3. **Currency Conversion**: Not yet implemented
4. **Invoice Editing**: Limited editing capabilities after creation
5. **Multi-Receiver Invoices**: Only supports single receiver per invoice

---

Part 14: Support and Troubleshooting

Common Issues

Issue 1: TypeScript Errors After Schema Changes

Symptom: `currencyRelation does not exist in type InvoiceInclude`

Solution:

```
npx prisma generate
```

Issue 2: “Could not determine invoice receiver”

Symptom: Error when running sendToAgency

Solution:
- Ensure contract has client OR agency participant with userId
- Manually specify receiverId in input

Issue 3: Bank Details Not Showing

Symptom: Bank account details card not displayed

Solution:
- Verify contract has bankId set
- Check Bank record exists and has data
- Ensure invoice query includes `contract: { include: { bank: true } }`

Issue 4: Pay Invoice Button Not Visible

Symptom: Receiver doesn't see payment button

Solution:
- Verify invoice.receiverId matches logged-in user.id
- Check invoice status is not "paid"
- Ensure session is properly loaded

Part 15: Documentation References

Related Documentation

- IMPLEMENTATION_SUMMARY.md (./IMPLEMENTATION_SUMMARY.md) - Margin system implementation
- PHASE3_UI_CHANGES_SUMMARY.md (./PHASE3_UI_CHANGES_SUMMARY.md) - UI enhancements
- MIGRATION_SUMMARY.md (./MIGRATION_SUMMARY.md) - Database migrations
- TIMESHEET_REFACTOR_SUMMARY.md (./TIMESHEET_REFACTOR_SUMMARY.md) - Timesheet improvements

External References

- Prisma Documentation (https://www.prisma.io/docs/) - Schema and migrations
- tRPC Documentation (https://trpc.io/docs/) - API routing
- Next.js Documentation (https://nextjs.org/docs) - Framework reference

---

Appendix A: Code Snippets

Complete sendToAgency Mutation

```
sendToAgency: tenantProcedure
  .use(hasPermission(P.APPROVE))
  .input(z.object({
    id: z.string(),
    senderId: z.string().optional(), // Optional override
    receiverId: z.string().optional(), // Optional override
    notes: z.string().optional(),
  }))
  .mutation(async ({ ctx, input }) => {
    // 1️⃣ Verify timesheet is approved
    const ts = await ctx.prisma.timesheet.findFirst({
      where: { id: input.id, tenantId: ctx.tenantId },
      include: {
        contract: {
          include: {
            participants: true,
            currency: true,
          },
        },
      },
    });

    if (!ts) throw new TRPCError({ code: "NOT_FOUND" });

    if (ts.workflowState !== "approved") {
      throw new TRPCError({
        code: "BAD_REQUEST",
        message: "Timesheet must be approved before sending to agency"
      });
    }

    // 2️⃣ Check if invoice already exists
    if (ts.invoiceId) {
      throw new TRPCError({
        code: "BAD_REQUEST",
        message: "Invoice already created for this timesheet"
      });
    }

    // 3️⃣ Determine Sender and Receiver
    const senderId = input.senderId || ts.submittedBy;

    let receiverId = input.receiverId;
    if (!receiverId) {
      const clientParticipant = ts.contract?.participants?.find((p: any) => p.role
=== "client");
      const agencyParticipant = ts.contract?.participants?.find((p: any) => p.role
=== "agency");

      const payer = clientParticipant || agencyParticipant;
      if (payer?.userId) {
        receiverId = payer.userId;
      } else {
        throw new TRPCError({
          code: "BAD_REQUEST",
          message: "Could not determine invoice receiver. Please specify receiverId.",
        });
      }
    }

    // 4️⃣ Create invoice in transaction
    const invoice = await ctx.prisma.$transaction(async (prisma) => {
```

```
      // ... (full implementation in timesheet.ts)
    });

    return { success: true, invoiceId: invoice.id };
  }),
```

Appendix B: Visual Flow Diagram

```
┌─────────────────────────────────────────────────────────────┐
│                    Invoice Creation Flow                    │
└─────────────────────────────────────────────────────────────┘

1. Contractor Creates Timesheet
   │
   ├─> Entries (hours worked)
   ├─> Expenses (if any)
   └─> Documents (receipts)

2. Contractor Submits Timesheet
   │
   └─> Status: draft → submitted

3. Admin Reviews Timesheet
   │
   ├─> Approve → Status: approved
   └─> Reject → Status: rejected

4. Admin Runs sendToAgency
   │
   ├─> Determine Sender (contractor)
   ├─> Determine Receiver (client/agency)
   ├─> Get Currency ID
   ├─> Calculate Margin
   │
   └─> Create Invoice
       │
       ├─> senderId: contractor.id
       ├─> receiverId: client.id
       ├─> currencyId: contract.currencyId
       ├─> contractId: contract.id
       ├─> Line Items (work + expenses)
       └─> Margin Entry

5. Invoice Created
   │
   └─> Status: pending_margin_confirmation

6. Receiver Views Invoice
   │
   ├─> See "Pay Invoice" button
   ├─> See Company Info
   ├─> See Bank Details
   └─> Click "Pay Invoice"

7. Payment Workflow (Future)
   │
   └─> Process payment → Status: paid
```

Conclusion

The sendToAgency workflow and invoice page finalization successfully implements:

1. ✅ **Automatic sender/receiver determination** based on contract participants
2. ✅ **Currency foreign key relationship** for referential integrity
3. ✅ **Enhanced invoice page** with payment button, company info, and bank details
4. ✅ **Copy-to-clipboard** functionality for easy payment processing
5. ✅ **Permission-based UI** showing relevant information to each user role
6. ✅ **Backward compatibility** maintained with legacy currency field

Next Steps

1. Run database migration
2. Generate Prisma client
3. Test complete flow in development
4. Deploy to staging for QA
5. Implement payment workflow
6. Add email notifications

Document Version: 1.0
Last Updated: December 11, 2025
Author: AI Development Team
Review Status: Ready for Implementation