Comprehensive Workflow System Implementation

Overview

This document describes the comprehensive workflow system implementation for the payroll SaaS platform. The system provides formal state management, state transitions, RBAC integration, audit trails, and margin calculations for all key entities.

Architecture

Components

1. **State Machines** (`lib/workflows/`)
   - Type-safe state definitions for each entity
   - Transition rules with permissions
   - Validation logic for state changes

2. **Workflow Services** (`lib/services/`)
   - `WorkflowExecutionService` : Executes state transitions with audit logging
   - `StateTransitionService` : High-level API with RBAC checks
   - `MarginCalculationService` : Handles margin calculations based on contract settings

3. **Database Schema** (`prisma/schema.prisma`)
   - Workflow state fields added to entities
   - `EntityStateHistory` model for audit trails
   - `WorkflowTemplate` and `TenantWorkflowSettings` for configurability

4. **tRPC Routers** (`server/api/routers/`)
   - Enhanced routers with workflow methods
   - RBAC-protected endpoints
   - Margin calculation endpoints

Supported Workflows

1. Timesheet Workflow

States:
- `draft` → `submitted` → `under_review` → `approved` | `rejected` | `changes_requested`

Actions:
- `submit` : Contractor submits timesheet
- `review` : Admin marks as under review
- `approve` : Admin approves timesheet
- `reject` : Admin rejects with reason
- `request_changes` : Admin requests modifications

Permissions:
- `timesheet.create.own`

- `timesheet.submit.own`
- `timesheet.review.global`
- `timesheet.approve.global`
- `timesheet.reject.global`
- `timesheet.modify.global`

2. Invoice Workflow

States:
- `draft` → `submitted` → `under_review` → `approved` | `rejected` → `sent` → `paid`

Actions:
- `submit` : Submit invoice for review
- `review` : Mark as under review
- `approve` : Approve invoice
- `reject` : Reject with reason
- `send` : Send approved invoice to client
- `mark_paid` : Mark invoice as paid

Permissions:
- `invoice.create.own`
- `invoice.submit.own`
- `invoice.review.global`
- `invoice.approve.global`
- `invoice.reject.global`
- `invoice.send.global`
- `invoice.modify.global`

Margin Calculations:
- Supports multiple payment modes: `gross` , `payroll` , `payroll_we_pay` , `split`
- Margin can be paid by: `client` , `agency` , `contractor`
- Automatic margin calculation based on contract settings

3. Payment Workflow

States:
- `pending` → `received` | `partially_received` → `confirmed`

Actions:
- `mark_received` : Mark payment as received
- `mark_partially_received` : Mark partial payment
- `confirm` : Confirm payment completion

Permissions:
- `payment.view.own`
- `payment.list.global`
- `payment.mark_received.global`
- `payment.confirm.global`

4. Payslip Workflow

States:
- `generated` → `validated` → `sent` → `paid`

Actions:
- `validate`: Validate payslip
- `send`: Send payslip to employee
- `mark_paid`: Mark as paid

Permissions:
- `payslip.view.own`
- `payslip.generate.global`
- `payslip.validate.global`
- `payslip.send.global`

5. Remittance Workflow

States:
- `generated` → `validated` → `sent` → `processing` → `completed`

Actions:
- `validate`: Validate remittance
- `send`: Send to payroll provider

Permissions:
- `remittance.view.own`
- `remittance.generate.global`
- `remittance.validate.global`
- `remittance.send.global`

Database Schema Changes

Modified Models

All entity models now include:
- `workflowState`: Formal workflow state
- Workflow-specific fields (approvedBy, rejectedBy, reviewedBy, etc.)
- Admin modification tracking (adminModifiedAmount, modifiedBy, etc.)

New Models

1. **EntityStateHistory**
   ```
   prisma
     model EntityStateHistory {
       id           String   @id @default(cuid())
       tenantId     String
       entityType   String   // timesheet, invoice, payment, etc.
       entityId     String
       fromState    String?
       toState      String
       action       String
       actorId      String
       actorName    String
       actorRole    String?
       reason       String?
       metadata     Json?
   ```

```
    transitionedAt DateTime @default(now())
  }
```

2. **WorkflowTemplate**

```
prisma
  model WorkflowTemplate {
    id          String  @id @default(cuid())
    tenantId    String
    name        String
    entityType  String
    states      Json    // State definitions
    transitions Json    // Transition rules
    permissions Json    // Permission mappings
  }
```

3. **TenantWorkflowSettings**

```
prisma
  model TenantWorkflowSettings {
    id       String @id @default(cuid())
    tenantId String @unique
    // Per-entity workflow template IDs
    timesheetWorkflowTemplateId  String?
    invoiceWorkflowTemplateId    String?
    // Global settings
    requireApprovalForTimesheets Boolean
    autoSendInvoiceOnApproval    Boolean
    defaultMarginPaidBy          String?
  }
```

API Usage

Timesheet Workflow

```
// Submit timesheet
await trpc.timesheet.submitTimesheet.mutate({ id: timesheetId })

// Review timesheet
await trpc.timesheet.reviewTimesheet.mutate({
  id: timesheetId,
  notes: "Reviewing hours"
})

// Approve timesheet
await trpc.timesheet.approve.mutate({ id: timesheetId })

// Request changes
await trpc.timesheet.requestChanges.mutate({
  id: timesheetId,
  changesRequested: "Please correct entry for Monday"
})

// Modify amounts (admin only)
await trpc.timesheet.modifyAmounts.mutate({
  id: timesheetId,
  totalAmount: 1500,
  adminModificationNote: "Adjusted for overtime"
})

// Get available actions
const actions = await trpc.timesheet.getAvailableActions.query({ id: timesheetId })

// Get state history
const history = await trpc.timesheet.getStateHistory.query({ id: timesheetId })
```

Invoice Workflow

```
// Review invoice
await trpc.invoice.reviewInvoice.mutate({
  id: invoiceId,
  notes: "Checking amounts"
})

// Approve invoice
await trpc.invoice.approveInvoiceWorkflow.mutate({
  id: invoiceId,
  notes: "Approved for payment"
})

// Calculate margin
const margin = await trpc.invoice.calculateMargin.mutate({
  id: invoiceId,
  contractId: contractId,
  baseAmount: 10000
})

// Modify invoice amounts
await trpc.invoice.modifyInvoiceAmounts.mutate({
  id: invoiceId,
  amount: 10500,
  marginAmount: 500,
  marginPercentage: 5,
  adminModificationNote: "Adjusted margin"
})

// Send invoice
await trpc.invoice.sendInvoiceWorkflow.mutate({ id: invoiceId })
```

Payment Workflow

```
// Mark payment as received
await trpc.payment.markPaymentReceived.mutate({
  id: paymentId,
  amountReceived: 5000, // Optional for partial payment
  notes: "First installment received"
})

// Confirm payment
await trpc.payment.confirmPayment.mutate({
  id: paymentId,
  notes: "Payment verified and confirmed"
})
```

Margin Calculation

The `MarginCalculationService` supports flexible margin calculations:

Payment Modes

- `gross` : Standard payment
- `payroll` : Through payroll system
- `payroll_we_pay` : Payroll paid by agency

- `split`: Split payment method

Margin Paid By

- `client`: Margin added to client invoice
- `contractor`: Margin deducted from contractor payment
- `agency`: Margin absorbed by agency

Example Usage

```
import { MarginCalculationService, MarginPaidBy } from '@/lib/services'

const calculation = MarginCalculationService.calculateMargin({
  baseAmount: 10000,
  marginPercentage: 10,
  marginPaidBy: MarginPaidBy.CLIENT,
  paymentMode: PaymentMode.GROSS
})

// Result:
// {
//   baseAmount: 10000,
//   marginAmount: 1000,
//   marginPercentage: 10,
//   totalAmount: 11000,
//   contractorAmount: 10000,
//   clientAmount: 11000,
//   agencyAmount: 1000,
//   breakdown: [...]
// }
```

State History and Audit Trails

Every state transition is logged in `EntityStateHistory`:
- Complete audit trail for compliance
- Actor information (who made the change)
- Reason for transition
- Metadata for additional context

Access state history:

```
const history = await StateTransitionService.getStateHistory(
  WorkflowEntityType.TIMESHEET,
  entityId,
  tenantId
)
```

RBAC Integration

All workflow actions are protected by permissions:
- Permissions follow the pattern: `resource.action.scope`
- Scopes: `own` (user's own resources) or `global` (all resources)
- State machines validate permissions before allowing transitions

Configuration

Tenant-Level Settings

Configure workflow behavior per tenant:

```
await prisma.tenantWorkflowSettings.create({
  data: {
    tenantId,
    requireApprovalForTimesheets: true,
    autoSendInvoiceOnApproval: false,
    defaultMarginPaidBy: 'client',
    allowAdminModifyAmounts: true,
    requireApprovalAfterModify: true
  }
})
```

Custom Workflow Templates

Define custom workflow templates:

```
await prisma.workflowTemplate.create({
  data: {
    tenantId,
    name: 'Fast Track Approval',
    entityType: 'timesheet',
    states: [...],
    transitions: [...],
    permissions: {...}
  }
})
```

Migration Guide

1. **Run Prisma Migrations**
   `bash`
   `  npx prisma migrate dev --name add_workflow_system`

2. **Seed Permissions**
   Add new workflow permissions to your permission seeding script

3. **Update Role Assignments**
   Assign appropriate workflow permissions to roles

4. **Test Workflows**
   Test each workflow end-to-end

Best Practices

1. **Always Use State Machines**: Don't bypass the workflow system - it ensures consistency and auditability
2. **Check Permissions**: Use `getAvailableActions` to show only allowed actions to users
3. **Provide Reasons**: Always provide reasons for rejections and changes
4. **Monitor State History**: Use state history for debugging and compliance

5. **Configure Per Tenant**: Customize workflows based on tenant requirements

Troubleshooting

Common Issues

1. **Transition Not Allowed**
   - Check user permissions
   - Verify current state allows the transition
   - Review state machine definition

2. **Margin Calculation Errors**
   - Verify contract has margin settings
   - Check marginPaidBy value is valid
   - Ensure baseAmount is positive

3. **State History Missing**
   - Verify transaction completed successfully
   - Check EntityStateHistory records are being created

Future Enhancements

- Configurable approval chains (multi-step approvals)
- Automated workflow triggers (webhooks)
- Workflow analytics dashboard
- Custom validation rules per tenant
- Email notifications on state changes

Support

For questions or issues with the workflow system:
1. Review state machine definitions in `lib/workflows/`
2. Check service implementations in `lib/services/`
3. Consult the state history for debugging
4. Review permissions in `lib/permissions.ts`