

Contracts Module Documentation

Overview

The Contracts module provides comprehensive contract lifecycle management with workflow status tracking, document management, and notifications.

Features

1. Contract Management

- **CRUD Operations:** Create, read, update, and delete contracts
- **Workflow Status Management:** Track contracts through various stages
- **Document Management:** Upload and manage contract-related documents
- **Status History:** Complete audit trail of status changes
- **Notifications:** Automated notifications for contract events

2. Workflow States

The contract workflow supports the following states:

- **draft:** Initial state when contract is being prepared
- **pending_agency_sign:** Awaiting agency signature
- **pending_contractor_sign:** Awaiting contractor signature
- **active:** Contract is active and in effect
- **paused:** Temporarily paused
- **completed:** Contract successfully completed
- **cancelled:** Contract was cancelled
- **terminated:** Contract was terminated early

3. Workflow Transitions

Valid state transitions are enforced by the system:

```
draft → pending_agency_sign | cancelled
pending_agency_sign → pending_contractor_sign | cancelled | draft
pending_contractor_sign → active | cancelled | pending_agency_sign
active → paused | completed | terminated
paused → active | terminated
completed → [final state]
cancelled → [final state]
terminated → [final state]
```

4. Document Types

Contracts support multiple document types:

- **contract:** Main contract document
- **amendment:** Contract amendments
- **termination:** Termination documents
- **other:** Other related documents

5. Notification Types

- **signature_request**: Request for signature
- **renewal_reminder**: Contract renewal reminder
- **termination_notice**: Notice of termination
- **status_change**: Status change notification
- **expiration_warning**: Contract expiration warning

API Endpoints

Contract Operations

Get All Contracts

```
api.contract.getAll.useQuery()
```

Get Contract by ID

```
api.contract.getById.useQuery({ id: string })
```

Create Contract

```
api.contract.create.useMutation({
  agencyId: string,
  contractorId: string,
  payrollPartnerId: string,
  // ... other fields
})
```

Update Contract

```
api.contract.update.useMutation({
  id: string,
  // ... fields to update
})
```

Delete Contract

```
api.contract.delete.useMutation({ id: string })
```

Workflow Operations

Update Workflow Status

```
api.contract.updateWorkflowStatus.useMutation({
  id: string,
  workflowStatus: WorkflowStatus,
  reason?: string,
  terminationReason?: string
})
```

Get Status History

```
api.contract.getStatusHistory.useQuery({ contractId: string })
```

Document Operations

Upload Document

```
api.contract.uploadDocument.useMutation({
  contractId: string,
  type: DocumentType,
  name: string,
  fileUrl: string,
  fileSize: number,
  mimeType: string
})
```

Get Documents

```
api.contract.getDocuments.useQuery({ contractId: string })
```

Delete Document

```
api.contract.deleteDocument.useMutation({ id: string })
```

Notification Operations

Create Notification

```
api.contract.createNotification.useMutation({
  contractId: string,
  recipientId: string,
  type: string,
  title: string,
  message: string
})
```

Get Notifications

```
api.contract.getNotifications.useQuery({
  contractId?: string,
  recipientId?: string
})
```

Mark Notification as Read

```
api.contract.markNotificationRead.useMutation({ id: string })
```

Utility Operations

Generate Contract Reference

```
api.contract.generateReference.useMutation()
// Returns: { reference: string } // Format: CTR-YYYYMM-XXXX
```

Get Expiring Contracts

```
api.contract.getExpiringContracts.useQuery({
  days: number // default: 30
})
```

Get Statistics

```
api.contract.getStats.useQuery()
// Returns: { total, active, draft, completed }
```

Permissions

The following permissions are required:

- contracts.view : View contracts
- contracts.create : Create new contracts
- contracts.update : Update existing contracts
- contracts.delete : Delete contracts

Database Schema

Contract Table

```
model Contract {
  id                      String
  tenantId                String
  companyId               String?
  agencyId                String
  contractorId           String
  payrollPartnerId        String
  status                  String
  workflowStatus          String
  terminationReason       String?
  terminatedAt            DateTime?
  terminatedBy             String?
  // ... financial and other fields

  documents               ContractDocument[]
  statusHistory            ContractStatusHistory[]
  notifications            ContractNotification[]
}
```

ContractDocument Table

```
model ContractDocument {
    id      String
    contractId String
    type    String
    name    String
    fileUrl String
    fileSize Int
    mimeType String
    version  Int
    isActive Boolean
    uploadedBy String
    uploadedAt DateTime
}
```

ContractStatusHistory Table

```
model ContractStatusHistory {
    id      String
    contractId String
    fromStatus String?
    toStatus  String
    changedBy String
    changedAt DateTime
    reason   String?
    metadata Json?
}
```

ContractNotification Table

```
model ContractNotification {
    id      String
    contractId String
    recipientId String
    type   String
    title   String
    message String
    sentAt  DateTime
    readAt  DateTime?
}
```

Usage Examples

Creating a Contract with Workflow

```
const createContract = api.contract.create.useMutation()

await createContract.mutateAsync({
  agencyId: "agency-id",
  contractorId: "contractor-id",
  payrollPartnerId: "payroll-id",
  title: "Software Development Contract",
  rate: 5000,
  rateType: "monthly",
  startDate: new Date(),
  workflowStatus: "draft"
})
```

Uploading a Document

```
const uploadDoc = api.contract.uploadDocument.useMutation()

await uploadDoc.mutateAsync({
  contractId: "contract-id",
  type: "contract",
  name: "Signed Contract.pdf",
  fileUrl: "https://storage.example.com/contract.pdf",
  fileSize: 245678,
  mimeType: "application/pdf"
})
```

Updating Workflow Status

```
const updateStatus = api.contract.updateWorkflowStatus.useMutation()

await updateStatus.mutateAsync({
  id: "contract-id",
  workflowStatus: "active",
  reason: "All signatures obtained"
})
```

Checking for Expiring Contracts

```
const { data: expiringContracts } = api.contract.getExpiringContracts.useQuery({
  days: 30 // contracts expiring in next 30 days
})
```

Best Practices

1. **Always validate workflow transitions** before attempting status changes
2. **Upload documents** before changing status to active
3. **Send notifications** when status changes affect other parties
4. **Track history** for all important changes
5. **Check permissions** before allowing operations

6. **Auto-generate contract references** for consistency
7. **Monitor expiring contracts** regularly
8. **Use proper document versioning** when uploading updates

Error Handling

The API returns standard tRPC errors:

- `NOT_FOUND` : Contract or document not found
- `BAD_REQUEST` : Invalid workflow transition or data
- `FORBIDDEN` : Insufficient permissions
- `UNAUTHORIZED` : Not authenticated

Always handle these errors in your UI:

```
try {
  await updateStatus.mutateAsync(...)
} catch (error) {
  if (error.data?.code === "BAD_REQUEST") {
    toast.error("Invalid status transition")
  }
}
```

Future Enhancements

- [] E-signature integration
- [] Contract templates
- [] Bulk operations
- [] Contract renewal automation
- [] Advanced reporting
- [] PDF generation
- [] Email notifications