Invoices Module Documentation

Overview

The Invoices module provides comprehensive invoice management with line items, auto-generation from contracts, status tracking, and financial reporting capabilities.

Features

1. Invoice Management

- **CRUD Operations**: Create, read, update, and delete invoices
- **Line Items**: Support for detailed line item breakdowns
- **Auto-Generation**: Generate invoices automatically from contracts
- **Status Tracking**: Track invoice through its lifecycle
- **Financial Reports**: Comprehensive statistics and reporting

2. Invoice Status

The invoice supports the following statuses:

- **draft**: Invoice is being prepared
- **sent**: Invoice has been sent to the client
- **paid**: Invoice has been paid
- **overdue**: Invoice is past due date and unpaid
- **cancelled**: Invoice was cancelled

3. Line Items

Each invoice can have multiple line items with:
- Description
- Quantity
- Unit Price
- Amount (calculated)

4. Auto-Calculations

The system automatically calculates:
- Line item amounts (quantity × unit price)
- Subtotal (sum of all line items)
- Tax amount (based on VAT rate)
- Total amount (subtotal + tax)

API Endpoints

Invoice Operations

Get All Invoices

```
api.invoice.getAll.useQuery({
  status?: string,
  contractId?: string,
  limit?: number, // default: 50
  offset?: number // default: 0
})
// Returns: { invoices, total }
```

Get Invoice by ID

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

Create Invoice

```
api.invoice.create.useMutation({
  contractId: string,
  invoiceNumber?: string, // auto-generated if not provided
  amount: number,
  currency?: string, // default: "USD"
  taxAmount?: number,
  issueDate: Date,
  dueDate: Date,
  description?: string,
  notes?: string,
  status?: InvoiceStatus,
  lineItems?: LineItem[]
})
```

Update Invoice

```
api.invoice.update.useMutation({
  id: string,
  amount?: number,
  taxAmount?: number,
  issueDate?: Date,
  dueDate?: Date,
  description?: string,
  notes?: string,
  status?: InvoiceStatus,
  lineItems?: LineItem[]
})
```

Update Status

```
api.invoice.updateStatus.useMutation({
  id: string,
  status: InvoiceStatus
})
// Automatically sets sentDate or paidDate based on status
```

Delete Invoice

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

Auto-Generation

Generate from Contract

```
api.invoice.generateFromContract.useMutation({
  contractId: string,
  period?: {
    month: number, // 1-12
    year: number
  }
})
```

This automatically:
- Retrieves contract rate and details
- Generates unique invoice number
- Calculates amounts based on rate type
- Applies VAT if configured
- Creates line items
- Sets appropriate due date

Generate Invoice Number

```
api.invoice.generateInvoiceNumber.useMutation()
// Returns: { invoiceNumber: string } // Format: INV-YYYYMM-XXXX
```

Reporting & Analytics

Get Statistics

```
api.invoice.getStats.useQuery()
// Returns: {
//   total: number,
//   paid: number,
//   overdue: number,
//   totalAmount: number,
//   paidAmount: number,
//   overdueAmount: number
// }
```

Get Overdue Invoices

```
api.invoice.getOverdue.useQuery()
```

Get Invoices by Contract

```
api.invoice.getByContractId.useQuery({ contractId: string })
```

Permissions

The following permissions are required:

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

Database Schema

Invoice Table

```
model Invoice {
  id            String
  tenantId      String
  contractId    String
  invoiceNumber String?
  status        String
  amount        Decimal
  currency      String
  taxAmount     Decimal
  totalAmount   Decimal
  issueDate     DateTime
  dueDate       DateTime
  paidDate      DateTime?
  sentDate      DateTime?
  description   String?
  notes         String?
  createdById   String?

  lineItems     InvoiceLineItem[]
}
```

InvoiceLineItem Table

```
model InvoiceLineItem {
  id          String
  invoiceId   String
  description String
  quantity    Decimal
  unitPrice   Decimal
  amount      Decimal
}
```

Usage Examples

Creating a Simple Invoice

```
const createInvoice = api.invoice.create.useMutation()

await createInvoice.mutateAsync({
  contractId: "contract-id",
  amount: 5000,
  taxAmount: 250,
  issueDate: new Date(),
  dueDate: new Date(Date.now() + 30 * 24 * 60 * 60 * 1000), // 30 days
  description: "Monthly services - January 2025",
  status: "draft"
})
```

Creating an Invoice with Line Items

```
const createInvoice = api.invoice.create.useMutation()

await createInvoice.mutateAsync({
  contractId: "contract-id",
  amount: 7500,
  taxAmount: 375,
  issueDate: new Date(),
  dueDate: calculateDueDate(new Date(), 30),
  lineItems: [
    {
      description: "Software Development (160 hours)",
      quantity: 160,
      unitPrice: 50,
      amount: 8000
    },
    {
      description: "Code Review (20 hours)",
      quantity: 20,
      unitPrice: 40,
      amount: 800
    }
  ],
  status: "draft"
})
```

Auto-Generating from Contract

```
const generateInvoice = api.invoice.generateFromContract.useMutation()

await generateInvoice.mutateAsync({
  contractId: "contract-id",
  period: {
    month: 1, // January
    year: 2025
  }
})
```

This will:

1. Fetch contract details (rate, type, VAT, etc.)

2. Generate a unique invoice number
3. Calculate amount based on rate type
4. Apply tax if VAT rate is configured
5. Create appropriate line items
6. Set due date based on contract's `invoiceDueDays`

Marking Invoice as Paid

```
const updateStatus = api.invoice.updateStatus.useMutation()

await updateStatus.mutateAsync({
  id: "invoice-id",
  status: "paid"
})
// Automatically sets paidDate to current date
```

Getting Financial Overview

```
const { data: stats } = api.invoice.getStats.useQuery()

// Display:
// - Total invoices: stats.total
// - Paid invoices: stats.paid
// - Overdue invoices: stats.overdue
// - Total revenue: stats.totalAmount
// - Collected revenue: stats.paidAmount
// - Outstanding: stats.overdueAmount
```

Handling Overdue Invoices

```
const { data: overdueInvoices } = api.invoice.getOverdue.useQuery()

// Send reminders for each overdue invoice
overdueInvoices?.forEach(invoice => {
  sendReminder(invoice.contract.contractor.email, invoice)
})
```

Helper Functions

The module provides several helper functions in `lib/types/invoices.ts`:

Generate Invoice Number

```
import { generateInvoiceNumber } from "@/lib/types/invoices"

const number = generateInvoiceNumber()
// Returns: "INV-202501-000123"
```

Calculate Due Date

```
import { calculateDueDate } from "@/lib/types/invoices"

const dueDate = calculateDueDate(new Date(), 30) // 30 days from now
```

Check if Overdue

```
import { isInvoiceOverdue } from "@/lib/types/invoices"

const overdue = isInvoiceOverdue(invoice.dueDate, invoice.status)
```

Get Status Label & Color

```
import {
  getInvoiceStatusLabel,
  getInvoiceStatusColor
} from "@/lib/types/invoices"

const label = getInvoiceStatusLabel(invoice.status) // "Paid"
const color = getInvoiceStatusColor(invoice.status) // "bg-green-100 text-green-800"
```

Calculate Amounts

```
import {
  calculateLineItemAmount,
  calculateInvoiceTotal
} from "@/lib/types/invoices"

const lineItemAmount = calculateLineItemAmount(10, 50) // 500
const total = calculateInvoiceTotal(1000, 50) // 1050
```

Auto Status Updates

The system can automatically update invoice status:

```
import { getAutoInvoiceStatus } from "@/lib/types/invoices"

// Automatically marks as overdue if past due date
const newStatus = getAutoInvoiceStatus(
  currentStatus,
  dueDate,
  paidDate
)
```

Invoice Number Format

Invoice numbers follow the format: **INV-YYYYMM-NNNN**

- `INV` : Prefix
- `YYYYMM` : Year and month (e.g., 202501 for January 2025)
- `NNNN` : Sequential number within the month (padded to 4 digits)

Examples:
- `INV-202501-0001`
- `INV-202501-0042`
- `INV-202512-1234`

Best Practices

1. **Always generate invoice numbers** using the provided utility
2. **Set appropriate due dates** based on contract terms
3. **Include detailed line items** for transparency
4. **Auto-generate from contracts** when possible for consistency
5. **Update status promptly** when invoices are sent or paid
6. **Monitor overdue invoices** regularly
7. **Calculate totals correctly** including tax
8. **Keep audit trails** of all changes

Validation Rules

- Amount must be positive
- Due date should be after issue date
- Status transitions should be logical (draft → sent → paid)
- Line item amounts must equal total amount
- Invoice number must be unique per tenant
- Tax amount must be non-negative

Error Handling

Handle these common errors:

```
try {
  await createInvoice.mutateAsync(...)
} catch (error) {
  if (error.data?.code === "NOT_FOUND") {
    toast.error("Contract not found")
  } else if (error.data?.code === "BAD_REQUEST") {
    toast.error("Invalid invoice data")
  }
}
```

Integration with Contracts

Invoices are tightly integrated with contracts:

- Each invoice must be linked to a contract
- Contract rate and terms determine invoice amounts
- Contract VAT rate applies to invoices
- Contract due days determine invoice due date
- Contract currency is used for invoices

Reporting Capabilities

Generate various reports using the provided queries:

Revenue Report

```
const stats = api.invoice.getStats.useQuery()
// Total revenue, paid revenue, outstanding
```

Overdue Report

```
const overdue = api.invoice.getOverdue.useQuery()
// All overdue invoices with details
```

Contract-specific Report

```
const invoices = api.invoice.getByContractId.useQuery({
  contractId: "contract-id"
})
// All invoices for a specific contract
```

Filtered Report

```
const invoices = api.invoice.getAll.useQuery({
  status: "paid",
  limit: 100,
  offset: 0
})
// Paginated results with filters
```

Future Enhancements

- [ ] PDF invoice generation
- [ ] Email sending integration
- [ ] Payment gateway integration
- [ ] Recurring invoice automation
- [ ] Multi-currency support
- [ ] Invoice templates
- [ ] Bulk operations
- [ ] Advanced reporting
- [ ] Export to accounting software
- [ ] Payment reminders