Payroll SaaS - Information Architecture (IA)

Branch: `expenses-structure`
Last Updated: December 18, 2025

Table of Contents

Project Overview

This is a **Deel-like payroll and contractor management SaaS platform** built with Next.js, featuring:

- **Multi-tenant architecture** with role-based access control (RBAC)
- **Contract management** (MSA/SOW structure)
- **Invoice workflow** with state machine transitions
- **Timesheet tracking** with expense management
- **Payment processing** with margin calculations
- **Document management** with S3 storage
- **Audit logging** for all critical actions

Core Entities

- **User**: All people in the system (contractors, clients, admins, agencies)
- **Role**: Dynamic roles with granular permissions
- **Contract**: MSA (Master Service Agreement) or SOW (Statement of Work)
- **Invoice**: Generated from timesheets or manually created
- **Timesheet**: Time tracking with expenses
- **Payment**: Payment tracking and confirmation
- **Company**: Client companies or agencies
- **Bank**: Bank account information

Tech 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 (for local state)
- Jotai 2.6.0 (atomic state)
- **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 (@aws-sdk/client-s3)
- **Email**: SendGrid (@sendgrid/mail)
- **SMS**: Twilio
- **Queue**: BullMQ 5.36.1 + Redis (Upstash)
- **Logging**: Winston 3.18.3

Development

- **Language**: TypeScript 5.9.3
- **Linting**: ESLint + Prettier
- **Package Manager**: npm

Architecture Patterns

1. Multi-Tenant Architecture

Every entity is scoped to a `tenantId`. All database queries must filter by tenant.

```
// Example: Tenant-scoped query
const invoices = await prisma.invoice.findMany({
  where: { tenantId: ctx.tenantId },
});
```

2. RBAC (Role-Based Access Control)

- **Roles** have **Permissions**

- **Users** have one **Role**
- Permissions follow the pattern: `resource.action.scope`
- Resource: `invoice`, `contract`, `user`, etc.
- Action: `create`, `read`, `update`, `delete`, `approve`, etc.
- Scope: `global`, `own`, `tenant`, `page`

3. State Machine Workflows

Critical entities (Invoice, Timesheet, Payment, Payslip, Remittance) use state machines for workflow transitions.

Example: Invoice State Machine

```
draft → submitted → under_review → approved → sent → marked_paid_by_agency → pay-
ment_received
```

4. Ownership Model

Many entities track ownership:
- `createdBy`: User who created the entity
- `ownerId`: User who owns the entity
- Permissions can be scoped to `own` (only access your own resources)

5. Audit Logging

All critical actions are logged to `AuditLog` table with:
- User info (snapshot)
- Action type
- Entity type and ID
- Metadata (JSON)
- IP address and user agent

Database Schema

Core Models

User

```
model User {
  id                  String  @id @default(cuid())
  tenantId            String
  email               String
  passwordHash        String
  roleId              String
  name                String?
  profilePictureUrl   String?
  phone               String?

  // Ownership
  createdBy           String?

  // Relations
  tenant              Tenant
  role                Role
  contractParticipants ContractParticipant[]
  invoicesSent        Invoice[] @relation("InvoiceSender")
  invoicesReceived    Invoice[] @relation("InvoiceReceiver")
  timesheets          Timesheet[]
  expenses            Expense[]
  // ... many more relations
}
```

Role

```
model Role {
  id          String  @id @default(cuid())
  tenantId    String
  name        String  // e.g., "CONTRACTOR", "ADMIN", "CLIENT"
  displayName String
  level       Int     @default(0)
  homePath    String  @default("/dashboard")

  // Relations
  users           User[]
  rolePermissions RolePermission[]
}
```

Permission

```
model Permission {
  id          String  @id @default(cuid())
  resource    String  // e.g., "invoice"
  action      String  // e.g., "create"
  key         String  @unique // e.g., "invoice.create.own"
  scope       String  @default("global") // "global", "own", "tenant"
  displayName String

  rolePermissions RolePermission[]
}
```

Contract

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

  // MSA/SOW structure
  type     String  @default("sow") // "msa" | "sow"
  parentId String? // null if MSA, else parent MSA id

  status      String  @default("draft")
  title       String?
  description String?

  // Financial
  rate           Decimal?
  rateType       String?
  currencyId     String?
  margin         Decimal?
  marginType     String?
  marginPaidBy   String?
  paymentModel   PaymentModel? // GROSS, PAYROLL, PAYROLL_WE_PAY, SPLIT

  // Relations
  participants ContractParticipant[]
  invoices     Invoice[]
  timesheets   Timesheet[]
  documents    ContractDocument[]
}
```

ContractParticipant

```
model ContractParticipant {
  id         String  @id @default(cuid())
  contractId String
  userId     String?
  companyId  String?

  role String // "contractor", "client", "agency", "payroll", "admin", "approver"

  // Signature
  requiresSignature Boolean   @default(false)
  signedAt          DateTime?
  signatureUrl      String?

  contract Contract
  user     User?
  company  Company?
}
```

Invoice

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

  // Ownership
  createdBy  String?
  senderId   String? // User sending the invoice
  receiverId String? // User receiving the invoice (pays)

  // Status & Workflow
  status        String @default("draft")
  workflowState String @default("draft")

  // Financial
  amount           Decimal
  currencyId       String?
  taxAmount        Decimal @default(0)
  totalAmount      Decimal @default(0)
  marginAmount     Decimal?
  marginPercentage Decimal?
  marginPaidBy     String? // "client", "agency", "contractor"
  baseAmount       Decimal?

  // Payment tracking
  agencyMarkedPaidAt DateTime?
  agencyMarkedPaidBy String?
  paymentReceivedAt  DateTime?
  paymentReceivedBy  String?

  // Dates
  issueDate DateTime @default(now())
  dueDate   DateTime
  paidDate  DateTime?

  // Relations
  tenant       Tenant
  contract     Contract?
  timesheet    Timesheet?
  sender       User?
  receiver     User?
  margin       Margin?
  lineItems    InvoiceLineItem[]
  documents    InvoiceDocument[]
  payments     Payment[]
  stateHistory EntityStateHistory[]
}
```

Margin

```
model Margin {
  id         String @id @default(cuid())
  invoiceId  String @unique
  contractId String

  marginType       MarginType // FIXED, VARIABLE, CUSTOM
  marginPercentage Decimal?
  marginAmount     Decimal?
  calculatedMargin Decimal

  // Override tracking
  isOverridden Boolean   @default(false)
  overriddenBy String?
  overriddenAt DateTime?
  notes        String?

  invoice  Invoice
  contract Contract
}
```

Timesheet

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

  submittedBy String
  contractId  String?

  // Period
  startDate DateTime
  endDate   DateTime

  // Status & Workflow
  status        String @default("draft")
  workflowState String @default("draft")

  // Totals
  totalHours    Decimal
  baseAmount    Decimal? // Work amount (hours × rate)
  marginAmount  Decimal? // Calculated margin (HIDDEN from contractors)
  totalExpenses Decimal? @default(0)
  totalAmount   Decimal? // Final total (baseAmount + marginAmount + totalExpenses)
  currency      String?

  // Admin modifications
  adminModifiedAmount   Decimal?
  adminModificationNote String?
  modifiedBy            String?

  // Relations
  tenant    Tenant
  submitter User
  contract  Contract?
  entries   TimesheetEntry[]
  invoice   Invoice?
  documents TimesheetDocument[]
  expenses  Expense[]
}
```

Payment

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

  invoiceId    String?
  expenseId    String?

  createdBy String

  amount          Decimal
  currency        String
  status          String // pending, received, confirmed, completed, failed
  workflowState   String @default("pending")
  paymentMethod   String

  // Payment tracking
  amountReceived Decimal?
  receivedBy     String?
  receivedAt     DateTime?
  confirmedBy    String?
  confirmedAt    DateTime?

  // Relations
  tenant  Tenant
  invoice Invoice?
  expense Expense?
}
```

Company

```
model Company {
  id       String @id @default(cuid())
  tenantId String
  name     String

  // Ownership
  ownerType String  @default("tenant")
  ownerId   String?
  createdBy String?

  bankId String?
  bank   Bank?

  // Contact & Address
  contactPerson String?
  contactEmail  String?
  contactPhone  String?
  address1      String?
  city          String?
  countryId     String?

  // Relations
  companyUsers         CompanyUser[]
  contractParticipants ContractParticipant[]
}
```

Bank

```
model Bank {
  id            String  @id @default(cuid())
  tenantId      String
  name          String
  accountNumber String?
  swiftCode     String?
  iban          String?
  address       String?

  createdBy String?
  status    String @default("active")

  companies Company[]
  contracts Contract[]
}
```

Enums

```
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
}

enum MarginType {
  FIXED      // Fixed amount margin
  VARIABLE   // Percentage-based margin
  CUSTOM     // Custom calculation
}

enum PaymentMethodType {
  BANK_ACCOUNT
  CREDIT_CARD
  DEBIT_CARD
  PAYPAL
  STRIPE
  WISE
  REVOLUT
  OTHER
}
```

Permission System (RBAC)

Permission Structure

Permissions follow the pattern: `resource.action.scope`

File: `server/rbac/permissions.ts`

```
export enum Resource {
  USER = "user",
  ROLE = "role",
  CONTRACT = "contract",
  INVOICE = "invoice",
  TIMESHEET = "timesheet",
  PAYMENT = "payment",
  EXPENSE = "expense",
  // ... more
}

export enum Action {
  CREATE = "create",
  READ = "read",
  UPDATE = "update",
  DELETE = "delete",
  LIST = "list",
  APPROVE = "approve",
  REJECT = "reject",
  SEND = "send",
  PAY = "pay",
  // ... more
}

export enum PermissionScope {
  GLOBAL = "global",   // Access all resources in tenant
  OWN = "own",         // Access only own resources
  TENANT = "tenant",  // Tenant-level access
  PAGE = "page",      // Page access permission
}
```

Example Permissions

```
// Invoice permissions
"invoice.create.own"      // Create own invoices
"invoice.read.own"        // Read own invoices
"invoice.list.global"     // List all invoices
"invoice.approve.global"  // Approve any invoice
"invoice.pay.global"      // Mark any invoice as paid
```

Checking Permissions

In tRPC procedures:

```
import { hasPermission, hasAnyPermission } from "../trpc";

// Single permission
.use(hasPermission("invoice.approve.global"))

// Any of multiple permissions
.use(hasAnyPermission(["invoice.read.own", "invoice.list.global"]))
```

In React components:

```tsx
import { usePermissions } from "@/hooks/use-permissions";

const { hasPermission } = usePermissions();

if (hasPermission("invoice.approve.global")) {
  // Show approve button
}
```

In RouteGuard:

```tsx
<RouteGuard permissions={["invoice.read.own", "invoice.list.global"]}>
  <InvoicesPageContent />
</RouteGuard>
```

Permission Helpers

File: `server/rbac/permissions.ts`

```tsx
// Check if user has permission with context (ownership)
hasPermissionWithContext(
  user: UserContext,
  resource: Resource,
  action: Action,
  resourceContext?: ResourceContext
): boolean

// Filter resources by permission
filterResourcesByPermission<T>(
  user: UserContext,
  resources: T[],
  resource: Resource,
  action: Action
): T[]
```

Workflow State Machines

Invoice State Machine

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

States

```
export enum InvoiceState {
  DRAFT = 'draft',
  SUBMITTED = 'submitted',
  PENDING_MARGIN_CONFIRMATION = 'pending_margin_confirmation',
  UNDER_REVIEW = 'under_review',
  APPROVED = 'approved',
  REJECTED = 'rejected',
  SENT = 'sent',
  MARKED_PAID_BY_AGENCY = 'marked_paid_by_agency',
  PAYMENT_RECEIVED = 'payment_received',
  PAID = 'paid',
  OVERDUE = 'overdue',
  CANCELLED = 'cancelled',
  CHANGES_REQUESTED = 'changes_requested',
}
```

Workflow Flow

```
1. DRAFT
   ↓ (submit)
2. SUBMITTED
   ↓ (review)
3. UNDER_REVIEW
   ↓ (approve)
4. APPROVED
   ↓ (send)
5. SENT
   ↓ (mark_paid_by_agency)
6. MARKED_PAID_BY_AGENCY
   ↓ (mark_payment_received)
7. PAYMENT_RECEIVED (FINAL)
```

Alternative paths:

- `SUBMITTED → PENDING_MARGIN_CONFIRMATION → UNDER_REVIEW` (for auto-generated invoices)
- `UNDER_REVIEW → CHANGES_REQUESTED → SUBMITTED` (request changes)
- `UNDER_REVIEW → REJECTED` (reject)
- `SENT → PAID` (legacy direct payment)

Permissions Required

```
export const InvoicePermissions = {
  SUBMIT_OWN: 'invoice.submit.own',
  REVIEW_ALL: 'invoice.review.global',
  APPROVE_ALL: 'invoice.approve.global',
  REJECT_ALL: 'invoice.reject.global',
  SEND_ALL: 'invoice.send.global',
  MODIFY_ALL: 'invoice.modify.global',
  MARK_PAID_ALL: 'invoice.mark_paid.global',
}
```

State Transition Service

File: `lib/services/StateTransitionService.ts`

```
// Execute a state transition
await StateTransitionService.executeTransition({
  entityType: WorkflowEntityType.INVOICE,
  entityId: invoiceId,
  action: WorkflowAction.APPROVE,
  userId: ctx.session.user.id,
  tenantId: ctx.tenantId,
  reason: "Approved by admin",
  metadata: { notes: "Looks good" },
});
```

Other State Machines

- **Timesheet**: `lib/workflows/timesheet-state-machine.ts`
- **Payment**: `lib/workflows/payment-state-machine.ts`
- **Payslip**: `lib/workflows/payslip-state-machine.ts`
- **Remittance**: `lib/workflows/remittance-state-machine.ts`

API Structure (tRPC)

tRPC Setup

File: `server/api/trpc.ts`

```
// Create context with session and tenant
export const createTRPCContext = async (opts: { headers: Headers }) => {
  const session = await getServerAuthSession();
  const tenantId = await getTenantId(session);

  return {
    session,
    tenantId,
    prisma,
  };
};

// Middleware for tenant-scoped procedures
export const tenantProcedure = publicProcedure.use(async ({ ctx, next }) => {
  if (!ctx.session || !ctx.tenantId) {
    throw new TRPCError({ code: "UNAUTHORIZED" });
  }
  return next({ ctx });
});

// Permission middleware
export const hasPermission = (permission: string) =>
  async ({ ctx, next }: any) => {
    if (!ctx.session.user.permissions.includes(permission)) {
      throw new TRPCError({ code: "FORBIDDEN" });
    }
    return next();
  };
```

Router Structure

File: `server/api/root.ts`

```
export const appRouter = createTRPCRouter({
  auth: authRouter,
  user: userRouter,
  role: roleRouter,
  contract: contractRouter,
  simpleContract: simpleContractRouter,
  invoice: invoiceRouter,
  timesheet: timesheetRouter,
  payment: paymentRouter,
  expense: expenseRouter,
  bank: bankRouter,
  company: companyRouter,
  // ... more routers
});
```

Invoice Router Example

File: `server/api/routers/invoice.ts`

```
export const invoiceRouter = createTRPCRouter({

  // List all invoices (global permission)
  getAll: tenantProcedure
    .use(hasAnyPermission([P.LIST_GLOBAL, P.READ_OWN]))
    .input(z.object({
      status: z.string().optional(),
      limit: z.number().default(50),
    }))
    .query(async ({ ctx, input }) => {
      const isGlobal = ctx.session.user.permissions.includes(P.LIST_GLOBAL);

      const where: any = { tenantId: ctx.tenantId };

      // OWN scope: filter by createdBy or receiverId
      if (!isGlobal) {
        where.OR = [
          { createdBy: ctx.session.user.id },
          { receiverId: ctx.session.user.id },
        ];
      }

      return ctx.prisma.invoice.findMany({ where });
    }),

  // Get single invoice
  getById: tenantProcedure
    .use(hasAnyPermission([P.LIST_GLOBAL, P.READ_OWN]))
    .input(z.object({ id: z.string() }))
    .query(async ({ ctx, input }) => {
      const invoice = await ctx.prisma.invoice.findFirst({
        where: { id: input.id, tenantId: ctx.tenantId },
        include: {
          lineItems: true,
          sender: true,
          receiver: true,
          contract: { include: { participants: true, bank: true } },
          margin: ctx.session.user.permissions.includes(P.LIST_GLOBAL),
        },
      });

      // Security check for OWN scope
      if (!ctx.session.user.permissions.includes(P.LIST_GLOBAL)) {
        if (invoice.createdBy !== ctx.session.user.id &&
            invoice.receiverId !== ctx.session.user.id) {
          throw new TRPCError({ code: "FORBIDDEN" });
        }
      }

      return invoice;
    }),

  // Workflow actions
  approveInvoiceWorkflow: tenantProcedure
    .use(hasPermission(P.APPROVE_GLOBAL))
    .input(z.object({ id: z.string(), notes: z.string().optional() }))
    .mutation(async ({ ctx, input }) => {
      const result = await StateTransitionService.executeTransition({
        entityType: WorkflowEntityType.INVOICE,
        entityId: input.id,
        action: WorkflowAction.APPROVE,
        userId: ctx.session.user.id,
```

```
        tenantId: ctx.tenantId,
        reason: input.notes,
      });

      if (!result.success) {
        throw new TRPCError({ code: "BAD_REQUEST", message: result.errors.join(', ') }
);
      }

      return result.entity;
    }),

  // ... more procedures
});
```

Client-Side Usage

File: `app/(dashboard)/(modules)/invoices/page.tsx`

```
import { api } from "@/lib/trpc";

function InvoicesPage() {
  const utils = api.useUtils();

  // Query
  const { data, isLoading } = api.invoice.getAll.useQuery({ limit: 50 });

  // Mutation
  const approveMutation = api.invoice.approveInvoiceWorkflow.useMutation({
    onSuccess: () => {
      toast.success("Invoice approved!");
      utils.invoice.getAll.invalidate();
    },
    onError: (err) => toast.error(err.message),
  });

  const handleApprove = (id: string) => {
    approveMutation.mutate({ id, notes: "Approved" });
  };

  return (
    <div>
      {data?.invoices.map(invoice => (
        <div key={invoice.id}>
          {invoice.invoiceNumber}
          <button onClick={() => handleApprove(invoice.id)}>Approve</button>
        </div>
      ))}
    </div>
  );
}
```

UI Component Patterns

Component Structure

```
components/
├── ui/                    # Base UI components (shadcn/ui)
│   ├── button.tsx
│   ├── dialog.tsx
│   ├── table.tsx
│   └── ...
├── guards/                # Permission guards
│   ├── RouteGuard.tsx
│   └── PermissionGuard.tsx
├── modals/                # Entity modals
│   ├── invoice-modal.tsx
│   └── ...
├── invoices/              # Invoice-specific components
│   ├── InvoiceReviewModal.tsx
│   ├── MarginConfirmationCard.tsx
│   └── PaymentTrackingCard.tsx
├── workflow/              # Workflow components
│   ├── WorkflowStatusBadge.tsx
│   ├── WorkflowActionButtons.tsx
│   └── MarginCalculationDisplay.tsx
└── shared/                # Shared components
    ├── empty-state.tsx
    └── loading-state.tsx
```

RouteGuard Pattern

File: `components/guards/RouteGuard.tsx`

```tsx
interface RouteGuardProps {
  permissions: string[];
  children: React.ReactNode;
}

export function RouteGuard({ permissions, children }: RouteGuardProps) {
  const { data: session, status } = useSession();

  if (status === "loading") {
    return <LoadingState />;
  }

  if (!session) {
    redirect("/auth/login");
  }

  const userPermissions = session.user.permissions || [];
  const hasAccess = permissions.some(p => userPermissions.includes(p));

  if (!hasAccess) {
    return <ForbiddenPageContent />;
  }

  return <>{children}</>;
}
```

Usage:

```tsx
export default function InvoicesPage() {
  return (
    <RouteGuard permissions={["invoice.read.own", "invoice.list.global"]}>
      <InvoicesPageContent />
    </RouteGuard>
  );
}
```

Modal Pattern

File: `components/modals/invoice-modal.tsx`

```tsx
interface InvoiceModalProps {
  open: boolean;
  onOpenChange: (open: boolean) => void;
  invoice?: Invoice;
  readOnly?: boolean;
}

export function InvoiceModal({ open, onOpenChange, invoice, readOnly }: InvoiceModal-
Props) {
  const form = useForm<InvoiceFormData>({
    resolver: zodResolver(invoiceSchema),
    defaultValues: invoice || {},
  });

  const createMutation = api.invoice.create.useMutation({
    onSuccess: () => {
      toast.success("Invoice created");
      onOpenChange(false);
    },
  });

  const onSubmit = (data: InvoiceFormData) => {
    createMutation.mutate(data);
  };

  return (
    <Dialog open={open} onOpenChange={onOpenChange}>
      <DialogContent>
        <DialogHeader>
          <DialogTitle>{invoice ? "Edit" : "Create"} Invoice</DialogTitle>
        </DialogHeader>
        <Form {...form}>
          <form onSubmit={form.handleSubmit(onSubmit)}>
            {/* Form fields */}
          </form>
        </Form>
      </DialogContent>
    </Dialog>
  );
}
```

Table Pattern

```
<Table>
  <TableHeader>
    <TableRow>
      <TableHead>Invoice #</TableHead>
      <TableHead>Amount</TableHead>
      <TableHead>Status</TableHead>
      <TableHead className="text-right">Actions</TableHead>
    </TableRow>
  </TableHeader>
  <TableBody>
    {invoices.map(invoice => (
      <TableRow key={invoice.id}>
        <TableCell>{invoice.invoiceNumber}</TableCell>
        <TableCell>${invoice.totalAmount}</TableCell>
        <TableCell><StatusBadge status={invoice.status} /></TableCell>
        <TableCell className="text-right">
          <Button variant="ghost" size="icon">
            <Eye className="h-4 w-4" />
          </Button>
        </TableCell>
      </TableRow>
    ))}
  </TableBody>
</Table>
```

Workflow Components

File: `components/workflow/WorkflowActionButtons.tsx`

```
interface WorkflowActionButtonsProps {
  entityType: WorkflowEntityType;
  currentState: string;
  entityId: string;
  onActionComplete?: () => void;
}

export function WorkflowActionButtons({
  entityType,
  currentState,
  entityId,
  onActionComplete,
}: WorkflowActionButtonsProps) {
  const { hasPermission } = usePermissions();

  // Get allowed transitions based on current state and permissions
  const allowedActions = getWorkflowActions(entityType, currentState, permissions);

  return (
    <div className="flex gap-2">
      {allowedActions.map(action => (
        <Button
          key={action.name}
          onClick={() => handleAction(action)}
          variant={action.variant}
        >
          {action.label}
        </Button>
      ))}
    </div>
  );
}
```

File Organization

Directory Structure

```
payroll-saas/
├── app/                              # Next.js App Router
│   ├── (dashboard)/                  # Dashboard layout group
│   │   ├── (modules)/                # Feature modules
│   │   │   ├── invoices/
│   │   │   │   ├── page.tsx          # Invoice list page
│   │   │   │   └── [id]/
│   │   │   │       └── page.tsx      # Invoice detail page
│   │   │   ├── contracts/
│   │   │   ├── timesheets/
│   │   │   ├── payments/
│   │   │   └── ...
│   │   ├── home/
│   │   └── layout.tsx                # Dashboard layout
│   ├── api/
│   │   ├── auth/[...nextauth]/       # NextAuth API route
│   │   ├── trpc/[trpc]/              # tRPC API route
│   │   └── upload/                   # File upload route
│   ├── auth/                         # Auth pages
│   │   ├── login/
│   │   └── signin/
│   ├── globals.css
│   └── layout.tsx                    # Root layout
│
├── server/                           # Backend code
│   ├── api/
│   │   ├── routers/                  # tRPC routers
│   │   │   ├── invoice.ts
│   │   │   ├── contract.ts
│   │   │   ├── timesheet.ts
│   │   │   └── ...
│   │   ├── root.ts                   # Root router
│   │   └── trpc.ts                   # tRPC setup
│   ├── rbac/
│   │   └── permissions.ts            # Permission definitions
│   ├── helpers/                      # Helper functions
│   └── validators/                   # Zod schemas
│
├── lib/                              # Shared utilities
│   ├── workflows/                    # State machines
│   │   ├── invoice-state-machine.ts
│   │   ├── timesheet-state-machine.ts
│   │   └── types.ts
│   ├── services/                     # Business logic services
│   │   ├── StateTransitionService.ts
│   │   ├── MarginCalculationService.ts
│   │   ├── MarginService.ts
│   │   └── PaymentWorkflowService.ts
│   ├── auth.ts                       # NextAuth config
│   ├── db.ts                         # Prisma client
│   ├── trpc.ts                       # tRPC client
│   ├── utils.ts                      # Utility functions
│   ├── permissions.ts                # Permission helpers
│   └── audit.ts                      # Audit logging
│
├── components/                       # React components
│   ├── ui/                           # Base UI components
│   ├── guards/                       # Permission guards
│   ├── modals/                       # Entity modals
│   ├── invoices/                     # Invoice components
│   ├── workflow/                     # Workflow components
│   ├── layout/                       # Layout components
```

```
│   └── shared/                    # Shared components
│
├── hooks/                         # Custom React hooks
│   ├── use-permissions.ts
│   ├── use-debounce.ts
│   └── ...
│
├── prisma/
│   ├── schema.prisma              # Database schema
│   └── migrations/                # Database migrations
│
├── scripts/
│   └── seed.ts                    # Database seeding
│
├── types/
│   └── next-auth.d.ts             # NextAuth type extensions
│
├── .env.example
├── next.config.js
├── tailwind.config.ts
├── tsconfig.json
└── package.json
```

Key Files

Authentication

- `lib/auth.ts` - NextAuth configuration
- `app/api/auth/[...nextauth]/route.ts` - NextAuth API route
- `types/next-auth.d.ts` - Session type extensions

Database

- `prisma/schema.prisma` - Database schema
- `lib/db.ts` - Prisma client singleton

API

- `server/api/trpc.ts` - tRPC setup and middleware
- `server/api/root.ts` - Root router
- `server/api/routers/*.ts` - Feature routers

Permissions

- `server/rbac/permissions.ts` - Permission definitions
- `lib/permissions.ts` - Permission helper functions
- `hooks/use-permissions.ts` - Permission hook

Workflows

- `lib/workflows/*-state-machine.ts` - State machine definitions
- `lib/services/StateTransitionService.ts` - State transition logic

UI

- `components/ui/*.tsx` - Base UI components (shadcn/ui)
- `components/guards/*.tsx` - Permission guards
- `components/workflow/*.tsx` - Workflow components

Key Features & Workflows

1. Invoice Workflow

Creating an Invoice

Manual Creation:
1. User clicks “Create Invoice” button
2. `InvoiceModal` opens with form
3. User fills in details (contract, amount, line items)
4. Form submits to `api.invoice.create`
5. Invoice created with status `draft`

Auto-Creation from Timesheet:
1. Timesheet is approved
2. System automatically creates invoice
3. Invoice status: `submitted`
4. Workflow state: `pending_margin_confirmation`

Invoice Review & Approval

States:
1. **DRAFT** - Being created by contractor
2. **SUBMITTED** - Submitted for review
3. **PENDING_MARGIN_CONFIRMATION** - Admin needs to confirm margin
4. **UNDER_REVIEW** - Admin is reviewing
5. **APPROVED** - Ready to send
6. **SENT** - Sent to client
7. **MARKED_PAID_BY_AGENCY** - Agency marked as paid
8. **PAYMENT_RECEIVED** - Payment confirmed received

Actions:
- `submit` - Contractor submits invoice
- `review` - Admin starts review
- `confirm_margin` - Admin confirms margin calculation
- `approve` - Admin approves invoice
- `reject` - Admin rejects invoice
- `request_changes` - Admin requests changes
- `send` - Admin sends invoice to client
- `mark_paid_by_agency` - Agency marks as paid
- `mark_payment_received` - Admin confirms payment received

Permissions:
- `invoice.submit.own` - Submit own invoices
- `invoice.review.global` - Review any invoice
- `invoice.approve.global` - Approve any invoice
- `invoice.send.global` - Send any invoice
- `invoice.pay.global` - Mark as paid

Margin Calculation

File: `lib/services/MarginCalculationService.ts`

```
// Calculate margin for an invoice
const marginResult = await MarginCalculationService.calculateMargin({
  baseAmount: 1000,
  marginType: MarginType.VARIABLE,
  marginPercentage: 10,
  marginPaidBy: MarginPaidBy.CLIENT,
});

// Result:
// {
//   baseAmount: 1000,
//   marginAmount: 100,
//   totalWithMargin: 1100,
//   marginPaidBy: "client"
// }
```

Margin Paid By:
- **CLIENT**: Client pays the margin (invoice total = base + margin)
- **AGENCY**: Agency absorbs the margin (invoice total = base)
- **CONTRACTOR**: Contractor pays the margin (invoice total = base - margin)

Payment Tracking

Two-Step Payment Process:

1. **Agency Marks as Paid**
   - Agency user clicks “Mark as Paid”
   - Invoice state: `MARKED_PAID_BY_AGENCY`
   - Fields updated:
     - `agencyMarkedPaidAt`
     - `agencyMarkedPaidBy`
2. **Admin Confirms Payment Received**
   - Admin user clicks “Confirm Payment Received”
   - Invoice state: `PAYMENT_RECEIVED`
   - Fields updated:
     - `paymentReceivedAt`
     - `paymentReceivedBy`

API Endpoints:

```
// Agency marks as paid
api.invoice.markAsPaid.mutate({
  id: invoiceId,
  paymentMethod: "bank_transfer",
  transactionId: "TXN123",
  notes: "Paid via wire transfer",
});

// Admin confirms payment received
api.invoice.confirmPaymentReceived.mutate({
  id: invoiceId,
  amountReceived: 1100,
  notes: "Payment confirmed in bank account",
});
```

2. Contract Management

Contract Types

MSA (Master Service Agreement):
- Parent contract that defines terms
- Can have multiple SOWs
- `type: "msa"`, `parentId: null`

SOW (Statement of Work):
- Child contract under an MSA
- Specific work engagement
- `type: "sow"`, `parentId: <msa_id>`

Contract Participants

Roles:
- `CONTRACTOR` - Person doing the work
- `CLIENT` - Company/person paying for work
- `AGENCY` - Agency managing the relationship
- `PAYROLL` - Payroll partner
- `ADMIN` - Admin managing the contract
- `APPROVER` - Person who approves timesheets/invoices

Example:

```
// Create contract with participants
await prisma.contract.create({
  data: {
    type: "sow",
    parentId: msaId,
    title: "Website Development",
    participants: {
      create: [
        { userId: contractorId, role: "CONTRACTOR" },
        { companyId: clientCompanyId, role: "CLIENT" },
        { userId: agencyId, role: "AGENCY" },
      ],
    },
  },
});
```

Payment Models

GROSS:
- Client pays gross amount directly to contractor
- No payroll partner involved

PAYROLL:
- Payroll partner handles all payments
- Client pays payroll partner
- Payroll partner pays contractor

PAYROLL_WE_PAY:
- We pay contractor upfront
- Payroll partner reimburses us later

SPLIT:
- Payment split across multiple methods
- Multiple bank accounts/payment methods

3. Timesheet Workflow

Timesheet Submission

1. Contractor creates timesheet
2. Adds time entries (date, hours, description)
3. Adds expenses (optional)
4. Submits for approval
5. Admin reviews and approves/rejects
6. On approval, invoice is auto-created

Timesheet Structure

```
{
  totalHours: 40,
  baseAmount: 4000,        // 40 hours × $100/hr
  marginAmount: 400,       // 10% margin (HIDDEN from contractor)
  totalExpenses: 100,      // Expenses
  totalAmount: 4500,       // baseAmount + marginAmount + totalExpenses

  entries: [
    { date: "2025-01-01", hours: 8, description: "Development" },
    { date: "2025-01-02", hours: 8, description: "Testing" },
  ],

  expenses: [
    { amount: 50, description: "Software license" },
    { amount: 50, description: "Travel" },
  ],
}
```

Important: Margin is HIDDEN from contractors. They only see `baseAmount + totalExpenses`.

4. Bank Account Management

Bank Account Linking

User Bank Accounts:
- Users can add their own bank accounts
- Used for receiving payments
- Stored in `PaymentMethod` table with `type: BANK_ACCOUNT`

Company Bank Accounts:
- Companies can have linked bank accounts
- Stored in `Bank` table
- Linked to `Company` via `bankId`

Contract Bank Accounts:
- Contracts can specify a bank account
- Used for invoice payments
- Linked via `contract.bankId`

Displaying Bank Information

In Invoice Display:

```
// Get bank info from contract
const bankInfo = invoice.contract?.bank;

if (bankInfo) {
  // Show bank details
  return (
    <div>
      <p>Bank: {bankInfo.name}</p>
      <p>Account: {bankInfo.accountNumber}</p>
      <p>SWIFT: {bankInfo.swiftCode}</p>
      <p>IBAN: {bankInfo.iban}</p>
    </div>
  );
} else {
  return <p>No bank account linked</p>;
}
```

5. Document Management

Document Upload

S3 Storage:
- Files uploaded to AWS S3
- Presigned URLs for secure access
- File metadata stored in `Document` table

Document Types:
- Contract documents
- Invoice documents
- Timesheet documents (expenses)
- User documents (ID, etc.)

Example:

```
// Upload document
const document = await prisma.document.create({
  data: {
    tenantId,
    entityType: "invoice",
    entityId: invoiceId,
    s3Key: "invoices/INV-123/receipt.pdf",
    fileName: "receipt.pdf",
    mimeType: "application/pdf",
    fileSize: 12345,
    uploadedBy: userId,
  },
});

// Link to invoice
await prisma.invoiceDocument.create({
  data: {
    invoiceId,
    documentId: document.id,
    category: "receipt",
    description: "Payment receipt",
  },
});
```

How to Add New Features

Adding a New Entity

1. Define Database Schema

File: `prisma/schema.prisma`

```
model MyEntity {
  id       String @id @default(cuid())
  tenantId String
  name     String
  status   String @default("active")

  // Ownership
  createdBy String

  // Dates
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt

  // Relations
  tenant Tenant @relation(fields: [tenantId], references: [id], onDelete: Cascade)

  @@index([tenantId])
  @@index([createdBy])
  @@map("my_entities")
}
```

Run migration:

```
npx prisma migrate dev --name add_my_entity
```

2. Add Permissions

File: `server/rbac/permissions.ts`

```
export enum Resource {
  // ... existing
  MY_ENTITY = "my_entity",
}

// Add permissions
createPermission(
  Resource.MY_ENTITY,
  Action.LIST,
  PermissionScope.GLOBAL,
  "View all entities",
  "List all entities in tenant",
  PermissionCategory.BUSINESS
),
createPermission(
  Resource.MY_ENTITY,
  Action.CREATE,
  PermissionScope.OWN,
  "Create entities",
  "Create new entities",
  PermissionCategory.BUSINESS
),
// ... more permissions
```

3. Create tRPC Router

File: `server/api/routers/myEntity.ts`

```
import { z } from "zod";
import { createTRPCRouter, tenantProcedure, hasPermission } from "../trpc";

const P = {
  LIST_GLOBAL: "my_entity.list.global",
  CREATE_OWN: "my_entity.create.own",
  UPDATE_OWN: "my_entity.update.own",
  DELETE_GLOBAL: "my_entity.delete.global",
};

export const myEntityRouter = createTRPCRouter({

  getAll: tenantProcedure
    .use(hasPermission(P.LIST_GLOBAL))
    .query(async ({ ctx }) => {
      return ctx.prisma.myEntity.findMany({
        where: { tenantId: ctx.tenantId },
        orderBy: { createdAt: "desc" },
      });
    }),

  create: tenantProcedure
    .use(hasPermission(P.CREATE_OWN))
    .input(z.object({
      name: z.string(),
    }))
    .mutation(async ({ ctx, input }) => {
      return ctx.prisma.myEntity.create({
        data: {
          tenantId: ctx.tenantId,
          name: input.name,
          createdBy: ctx.session.user.id,
        },
      });
    }),

  // ... more procedures
});
```

Add to root router:

File: `server/api/root.ts`

```
import { myEntityRouter } from "./routers/myEntity";

export const appRouter = createTRPCRouter({
  // ... existing routers
  myEntity: myEntityRouter,
});
```

4. Create UI Components

File: `components/modals/my-entity-modal.tsx`

```
"use client";

import { api } from "@/lib/trpc";
import { Dialog, DialogContent, DialogHeader, DialogTitle } from "@/components/ui/dia-
log";
import { Button } from "@/components/ui/button";
import { Input } from "@/components/ui/input";
import { Label } from "@/components/ui/label";
import { useForm } from "react-hook-form";
import { zodResolver } from "@hookform/resolvers/zod";
import { z } from "zod";
import { toast } from "sonner";

const schema = z.object({
  name: z.string().min(1, "Name is required"),
});

type FormData = z.infer<typeof schema>;

interface MyEntityModalProps {
  open: boolean;
  onOpenChange: (open: boolean) => void;
  entity?: any;
}

export function MyEntityModal({ open, onOpenChange, entity }: MyEntityModalProps) {
  const utils = api.useUtils();

  const form = useForm<FormData>({
    resolver: zodResolver(schema),
    defaultValues: entity || { name: "" },
  });

  const createMutation = api.myEntity.create.useMutation({
    onSuccess: () => {
      toast.success("Entity created");
      utils.myEntity.getAll.invalidate();
      onOpenChange(false);
    },
    onError: (err) => toast.error(err.message),
  });

  const onSubmit = (data: FormData) => {
    createMutation.mutate(data);
  };

  return (
    <Dialog open={open} onOpenChange={onOpenChange}>
      <DialogContent>
        <DialogHeader>
          <DialogTitle>{entity ? "Edit" : "Create"} Entity</DialogTitle>
        </DialogHeader>
        <form onSubmit={form.handleSubmit(onSubmit)} className="space-y-4">
          <div>
            <Label htmlFor="name">Name</Label>
            <Input id="name" {...form.register("name")} />
            {form.formState.errors.name && (
              <p className="text-sm text-red-500">{form.formState.errors.name.message}
</p>
            )}
          </div>
          <Button type="submit" disabled={createMutation.isPending}>
```

```
            {createMutation.isPending ? "Creating..." : "Create"}
          </Button>
        </form>
      </DialogContent>
    </Dialog>
  );
}
```

5. Create Page

File: `app/(dashboard)/(modules)/my-entities/page.tsx`

```
"use client";

import { useState } from "react";
import { api } from "@/lib/trpc";
import { RouteGuard } from "@/components/guards/RouteGuard";
import { PageHeader } from "@/components/ui/page-header";
import { Button } from "@/components/ui/button";
import { Card, CardContent } from "@/components/ui/card";
import { Table, TableBody, TableCell, TableHead, TableHeader, TableRow } from "@/com-
ponents/ui/table";
import { MyEntityModal } from "@/components/modals/my-entity-modal";
import { Plus, Edit, Trash2 } from "lucide-react";
import { toast } from "sonner";

function MyEntitiesPageContent() {
  const [modalOpen, setModalOpen] = useState(false);
  const [editingEntity, setEditingEntity] = useState<any>(null);

  const { data, isLoading } = api.myEntity.getAll.useQuery();
  const utils = api.useUtils();

  const deleteMutation = api.myEntity.delete.useMutation({
    onSuccess: () => {
      toast.success("Entity deleted");
      utils.myEntity.getAll.invalidate();
    },
  });

  return (
    <div className="space-y-6">
      <PageHeader title="My Entities" description="Manage your entities">
        <Button onClick={() => setModalOpen(true)}>
          <Plus className="mr-2 h-4 w-4" /> Create Entity
        </Button>
      </PageHeader>

      <Card>
        <CardContent className="p-0">
          <Table>
            <TableHeader>
              <TableRow>
                <TableHead>Name</TableHead>
                <TableHead>Status</TableHead>
                <TableHead className="text-right">Actions</TableHead>
              </TableRow>
            </TableHeader>
            <TableBody>
              {data?.map(entity => (
                <TableRow key={entity.id}>
                  <TableCell>{entity.name}</TableCell>
                  <TableCell>{entity.status}</TableCell>
                  <TableCell className="text-right">
                    <Button
                      variant="ghost"
                      size="icon"
                      onClick={() => setEditingEntity(entity)}
                    >
                      <Edit className="h-4 w-4" />
                    </Button>
                    <Button
                      variant="ghost"
                      size="icon"
```

```
                      onClick={() => deleteMutation.mutate({ id: entity.id })}
                    >
                      <Trash2 className="h-4 w-4" />
                    </Button>
                  </TableCell>
                </TableRow>
              ))}
            </TableBody>
          </Table>
        </CardContent>
      </Card>

      <MyEntityModal
        open={modalOpen || !!editingEntity}
        onOpenChange={(open) => {
          if (!open) {
            setModalOpen(false);
            setEditingEntity(null);
          }
        }}
        entity={editingEntity}
      />
    </div>
  );
}

export default function MyEntitiesPage() {
  return (
    <RouteGuard permissions={["my_entity.list.global"]}>
      <MyEntitiesPageContent />
    </RouteGuard>
  );
}
```

Adding a Workflow State Machine

1. Define States and Transitions

File: `lib/workflows/my-entity-state-machine.ts`

```
import {
  WorkflowEntityType,
  WorkflowAction,
  StateMachineDefinition,
  StateDefinition,
  TransitionDefinition,
} from './types';

export enum MyEntityState {
  DRAFT = 'draft',
  SUBMITTED = 'submitted',
  APPROVED = 'approved',
  REJECTED = 'rejected',
}

const states: StateDefinition[] = [
  {
    name: MyEntityState.DRAFT,
    displayName: 'Draft',
    description: 'Entity is being created',
    isInitial: true,
    allowedActions: [WorkflowAction.SUBMIT],
  },
  {
    name: MyEntityState.SUBMITTED,
    displayName: 'Submitted',
    description: 'Entity submitted for approval',
    allowedActions: [WorkflowAction.APPROVE, WorkflowAction.REJECT],
  },
  {
    name: MyEntityState.APPROVED,
    displayName: 'Approved',
    description: 'Entity approved',
    isFinal: true,
    allowedActions: [],
  },
  {
    name: MyEntityState.REJECTED,
    displayName: 'Rejected',
    description: 'Entity rejected',
    isFinal: true,
    allowedActions: [],
  },
];

const transitions: TransitionDefinition[] = [
  {
    from: MyEntityState.DRAFT,
    to: MyEntityState.SUBMITTED,
    action: WorkflowAction.SUBMIT,
    requiredPermissions: ['my_entity.submit.own'],
  },
  {
    from: MyEntityState.SUBMITTED,
    to: MyEntityState.APPROVED,
    action: WorkflowAction.APPROVE,
    requiredPermissions: ['my_entity.approve.global'],
  },
  {
    from: MyEntityState.SUBMITTED,
    to: MyEntityState.REJECTED,
    action: WorkflowAction.REJECT,
```

```
    requiredPermissions: ['my_entity.reject.global'],
  },
];

export const myEntityStateMachineDefinition: StateMachineDefinition = {
  entityType: WorkflowEntityType.MY_ENTITY,
  states,
  transitions,
  initialState: MyEntityState.DRAFT,
};
```

2. Add Workflow Actions to Router

File: `server/api/routers/myEntity.ts`

```
import { StateTransitionService } from "@/lib/services/StateTransitionService";
import { WorkflowEntityType, WorkflowAction } from "@/lib/workflows";

// Add to router
approveEntity: tenantProcedure
  .use(hasPermission("my_entity.approve.global"))
  .input(z.object({ id: z.string(), notes: z.string().optional() }))
  .mutation(async ({ ctx, input }) => {
    const result = await StateTransitionService.executeTransition({
      entityType: WorkflowEntityType.MY_ENTITY,
      entityId: input.id,
      action: WorkflowAction.APPROVE,
      userId: ctx.session.user.id,
      tenantId: ctx.tenantId,
      reason: input.notes,
    });

    if (!result.success) {
      throw new TRPCError({
        code: "BAD_REQUEST",
        message: result.errors.join(', '),
      });
    }

    return result.entity;
  }),
```

Adding a New Permission

1. Add to Permission Definitions

File: `server/rbac/permissions.ts`

```
createPermission(
  Resource.MY_ENTITY,
  Action.CUSTOM_ACTION,
  PermissionScope.GLOBAL,
  "Custom Action",
  "Perform custom action on entities",
  PermissionCategory.BUSINESS
),
```

2. Use in tRPC Procedure

```
customAction: tenantProcedure
  .use(hasPermission("my_entity.custom_action.global"))
  .input(z.object({ id: z.string() }))
  .mutation(async ({ ctx, input }) => {
    // Perform action
  }),
```

3. Check in UI

```
const { hasPermission } = usePermissions();

if (hasPermission("my_entity.custom_action.global")) {
  return <Button onClick={handleCustomAction}>Custom Action</Button>;
}
```

Common Issues & Solutions

Issue: “React child object error with role object”

Problem: Trying to render an object directly in React.

Solution: Extract the specific property you want to display.

```
// ❌ Wrong
<div>{user.role}</div>

// ✅ Correct
<div>{user.role.displayName}</div>
```

Issue: Permission denied errors

Problem: User doesn’t have required permission.

Solution:
1. Check user’s role permissions in database
2. Verify permission key matches exactly
3. Check if using correct scope (global vs own)

```
// Check permissions
const permissions = session?.user?.permissions ?? [];
console.log("User permissions:", permissions);

// Verify permission exists
const hasPermission = permissions.includes("invoice.approve.global");
```

Issue: State transition fails

Problem: Invalid state transition or missing permission.

Solution:
1. Check current state of entity

2. Verify transition exists in state machine
3. Check user has required permission
4. Check transition conditions are met

```
// Debug state transition
const result = await StateTransitionService.executeTransition({
  entityType: WorkflowEntityType.INVOICE,
  entityId: invoiceId,
  action: WorkflowAction.APPROVE,
  userId: ctx.session.user.id,
  tenantId: ctx.tenantId,
});

if (!result.success) {
  console.error("Transition failed:", result.errors);
}
```

Issue: Margin not showing/calculating

Problem: Margin is only visible to admins.

Solution:

1. Check user has `invoice.list.global` permission
2. Verify margin is included in query for admins only

```
// In tRPC query
include: {
  margin: ctx.session.user.permissions.includes("invoice.list.global"),
}
```

Issue: Bank information not displaying

Problem: Bank not linked to contract or company.

Solution:

1. Check if contract has `bankId` set
2. Verify bank exists in database
3. Include bank in query

```
// Include bank in contract query
include: {
  contract: {
    include: {
      bank: true,
    },
  },
}

// Display bank info
if (invoice.contract?.bank) {
  // Show bank details
} else {
  // Show "No bank account linked"
}
```

Development Workflow

Running the Application

```
# Install dependencies
npm install

# Generate Prisma client
npx prisma generate

# Run database migrations
npx prisma migrate dev

# Seed database (optional)
npm run seed

# Start development server
npm run dev
```

Database Commands

```
# Create migration
npx prisma migrate dev --name migration_name

# Reset database (WARNING: deletes all data)
npx prisma migrate reset

# Open Prisma Studio (database GUI)
npx prisma studio

# Generate Prisma client
npx prisma generate
```

Environment Variables

File: `.env`

```
# Database
DATABASE_URL="postgresql://user:password@localhost:5432/payroll_saas"

# NextAuth
NEXTAUTH_URL="http://localhost:3000"
NEXTAUTH_SECRET="your-secret-key"

# AWS S3
AWS_ACCESS_KEY_ID="your-access-key"
AWS_SECRET_ACCESS_KEY="your-secret-key"
AWS_REGION="us-east-1"
AWS_S3_BUCKET="your-bucket-name"

# Email (SendGrid)
SENDGRID_API_KEY="your-sendgrid-key"

# SMS (Twilio)
TWILIO_ACCOUNT_SID="your-twilio-sid"
TWILIO_AUTH_TOKEN="your-twilio-token"

# Redis (Upstash)
UPSTASH_REDIS_REST_URL="your-redis-url"
UPSTASH_REDIS_REST_TOKEN="your-redis-token"
```

Testing

Manual Testing Checklist

Invoice Workflow:
- [ ] Create invoice as contractor
- [ ] Submit invoice for review
- [ ] Review invoice as admin
- [ ] Approve invoice
- [ ] Send invoice
- [ ] Mark as paid by agency
- [ ] Confirm payment received

Permissions:
- [ ] Test with different roles (contractor, admin, client)
- [ ] Verify permission guards work
- [ ] Test OWN vs GLOBAL scope

State Transitions:
- [ ] Test all valid transitions
- [ ] Test invalid transitions (should fail)
- [ ] Verify state history is recorded

Conclusion

This document provides a comprehensive overview of the Payroll SaaS application architecture. Use it as a reference when:

- Adding new features
- Debugging issues
- Understanding workflows
- Implementing permissions
- Working with state machines

For questions or clarifications, refer to the specific files mentioned in each section.

Last Updated: December 18, 2025
Version: 1.0
Branch: expenses-structure