RBAC (Role-Based Access Control) Implementation - Phase 1

Overview

This document describes the comprehensive RBAC system implementation for the Payroll SaaS application. The system provides dynamic role and permission management with a clean, user-friendly interface.

Architecture

Database Schema

The RBAC system uses three main tables:

1. **Role** - Defines user roles within a tenant
   - `id` : Unique identifier
   - `tenantId` : Associated tenant
   - `name` : Role name (unique within tenant)
   - `homePath` : Default landing page for users with this role

2. **Permission** - System-wide permissions
   - `id` : Unique identifier
   - `key` : Permission key (e.g., “users.create”)
   - `description` : Human-readable description

3. **RolePermission** - Many-to-many relationship
   - `roleId` : Role reference
   - `permissionId` : Permission reference

Backend API (tRPC)

Role Router (`/server/api/routers/role.ts`)

Endpoints:

- `getAll()` - Get all roles for current tenant
- `getById(id)` - Get specific role with permissions
- `create(data)` - Create new role with optional permissions
- `update(id, data)` - Update role details
- `delete(id)` - Delete role (checks for active users)
- `assignPermissions(roleId, permissionIds)` - Assign permissions to role
- `getPermissions(roleId)` - Get all permissions for a role
- `getStats()` - Get role statistics

Example Usage:

```
// Create a new role
const role = await api.role.create.mutate({
  name: "Manager",
  homePath: "/admin",
  permissionIds: ["users.view", "users.create"]
})

// Assign permissions
await api.role.assignPermissions.mutate({
  roleId: role.id,
  permissionIds: ["contracts.view", "invoices.view"]
})
```

Permission Router (`/server/api/routers/permission.ts`)

Endpoints:

- `getAll()` - Get all system permissions
- `getById(id)` - Get specific permission
- `getByKeys(keys)` - Get permissions by their keys
- `getMyPermissions()` - Get current user's permissions
- `hasPermission(permission)` - Check if user has a permission
- `hasAnyPermission(permissions)` - Check if user has any of the permissions
- `hasAllPermissions(permissions)` - Check if user has all permissions
- `getGrouped()` - Get permissions grouped by category

Frontend Components

1. Permission Hook (`/hooks/use-permissions.ts`)

A React hook for checking permissions:

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

function MyComponent() {
  const { hasPermission, hasAnyPermission, isSuperAdmin } = usePermissions()

  if (hasPermission("users.create")) {
    // Show create user button
  }

  if (hasAnyPermission(["invoices.view", "contracts.view"])) {
    // Show finance section
  }
}
```

2. Protected Route Component (`/components/rbac/protected-route.tsx`)

Protects entire routes based on permissions:

```tsx
import { ProtectedRoute } from "@/components/rbac"

export default function AdminPage() {
  return (
    <ProtectedRoute permission="admin.access">
      <div>Admin Content</div>
    </ProtectedRoute>
  )
}

// Or with multiple permissions
<ProtectedRoute
  permissions={["users.view", "users.create"]}
  requireAll={true}
>
  <UserManagement />
</ProtectedRoute>
```

3. Can Component (`/components/rbac/can.tsx`)

Conditional rendering based on permissions:

```tsx
import { Can } from "@/components/rbac"

function UserList() {
  return (
    <div>
      <Can permission="users.create">
        <Button>Create User</Button>
      </Can>

      <Can permissions={["users.update", "users.delete"]} requireAll={false}>
        <Button>Manage Users</Button>
      </Can>
    </div>
  )
}
```

4. Dynamic Sidebar (`/lib/dynamicMenuConfig.ts`)

The sidebar automatically filters menu items based on user permissions:

```tsx
// Menu items with permission requirements
const menuItem = {
  label: "Manage Users",
  href: "/admin/users",
  icon: Users,
  permission: "tenant.users.view"  // Only shown if user has this permission
}

// Menu with multiple permissions (OR logic)
const menuItem = {
  label: "Settings",
  href: "/admin/settings",
  permissions: ["settings.view", "tenant.roles.view"],
  requireAll: false  // User needs ANY of these permissions
}
```

5. Role Management Modal (`/components/modals/role-modal.tsx`)

Comprehensive modal for creating and editing roles:

Features:
- Two-tab interface (Basic Info / Permissions)
- Permission grouping by category
- Select/Deselect all for each category
- Real-time permission count
- Role statistics display
- Validation and error handling

Permission Categories

The system organizes permissions into the following categories:

- **tenant** - Tenant management and configuration
- **companies** - Company management
- **agencies** - Agency management
- **contractors** - Contractor management
- **contracts** - Contract management
- **invoices** - Invoice management
- **payroll** - Payroll operations
- **payslip** - Payslip management
- **banks** - Bank account management
- **settings** - System settings
- **onboarding** - Onboarding processes
- **tasks** - Task management
- **leads** - Lead management
- **audit** - Audit log access
- **superadmin** - Super admin operations

Middleware & Authentication

The authentication middleware (`/server/api/trpc.ts`) automatically:

1. Loads user permissions from database
2. Attaches permissions to session
3. Validates permission requirements on protected endpoints
4. Handles SuperAdmin with full access

Usage in tRPC:

```
export const myRouter = createTRPCRouter({
  sensitiveAction: tenantProcedure
    .use(hasPermission(PERMISSION_TREE.users.delete))
    .mutation(async ({ ctx, input }) => {
      // Only users with "users.delete" permission can access this
    })
})
```

Admin Interface

Role Management Page (`/admin/settings/roles`)

Features:
- List all roles with user count and permission count
- Search and filter roles
- Create new roles
- Edit existing roles
- Delete roles (with validation)
- Statistics dashboard

Actions:
- **Create Role**: Opens modal to create new role with permissions
- **Edit Role**: Opens modal to modify role and permissions
- **Delete Role**: Confirms deletion (blocked if users are assigned)

Role Modal Features

1. **Basic Info Tab**
   - Role name (required, unique)
   - Home path (default landing page)
   - Role statistics (users, permissions, created date)

2. **Permissions Tab**
   - Grouped by category
   - Checkbox for each permission
   - Select/Deselect all per category
   - Permission count badge
   - Searchable and scrollable

Security Considerations

1. **Tenant Isolation**: All role and permission checks are tenant-scoped
2. **Permission Validation**: Backend validates all permission checks
3. **Audit Logging**: All role changes are logged
4. **Delete Protection**: Cannot delete roles with active users
5. **SuperAdmin Override**: SuperAdmin has access to everything

Usage Examples

Example 1: Creating a Custom Role

```
// Backend (tRPC mutation)
const customRole = await api.role.create.mutate({
  name: "Finance Manager",
  homePath: "/finance",
  permissionIds: [
    "invoices.view",
    "invoices.create",
    "invoices.update",
    "payroll.view",
    "banks.view"
  ]
})
```

Example 2: Protecting a Component

```
import { Can } from "@/components/rbac"

function InvoiceActions({ invoice }) {
  return (
    <div>
      <Can permission="invoices.view">
        <ViewButton invoice={invoice} />
      </Can>

      <Can permission="invoices.update">
        <EditButton invoice={invoice} />
      </Can>

      <Can permission="invoices.delete">
        <DeleteButton invoice={invoice} />
      </Can>
    </div>
  )
}
```

Example 3: Dynamic Route Protection

```
// In page component
import { ProtectedRoute } from "@/components/rbac"

export default function FinanceDashboard() {
  return (
    <ProtectedRoute
      permissions={["invoices.view", "payroll.view"]}
      requireAll={false}
    >
      <div>
        <h1>Finance Dashboard</h1>
        {/* Content */}
      </div>
    </ProtectedRoute>
  )
}
```

Testing Checklist

- [ ] Create a new role
- [ ] Assign permissions to role
- [ ] Update role permissions
- [ ] Create user with custom role
- [ ] Verify sidebar shows correct menu items
- [ ] Verify protected routes work
- [ ] Verify Can component conditionally renders
- [ ] Try to delete role with active users (should fail)
- [ ] Try to access protected route without permission (should redirect)
- [ ] Verify audit logs are created

Future Enhancements (Phase 2)

1. **Permission Templates**: Pre-defined permission sets for common roles
2. **Role Cloning**: Duplicate existing roles
3. **Permission Search**: Search/filter permissions in modal
4. **Bulk User Assignment**: Assign role to multiple users at once
5. **Permission History**: Track permission changes over time
6. **Advanced Rules**: Time-based permissions, IP restrictions
7. **Row-Level Security**: Filter data based on user attributes

Troubleshooting

User can’t see menu items

- Check user’s role has required permissions
- Verify permissions are correctly assigned to role
- Check `dynamicMenuConfig.ts` has correct permission keys

Permission check failing

- Verify permission key matches exactly (case-sensitive)
- Check user session has loaded permissions
- Confirm permission exists in database

Role creation fails

- Ensure role name is unique within tenant
- Check all permission IDs are valid
- Verify user has `tenant.roles.create` permission

Support

For issues or questions about the RBAC system:
1. Check this documentation
2. Review code comments in implementation files
3. Check audit logs for permission-related errors
4. Contact development team