Role-Based Access System Documentation

Overview

The Role-Based Access System provides granular permission control for different modules within the application. It supports company-scoped user management where company administrators can create and manage users within their organization, assign specific module permissions, and control access to different parts of the system.

System Architecture

Core Components

1. **Module Permissions**: Granular permissions for different application modules
2. **User Roles**: Different user types with varying levels of access
3. **Permission Guards**: Middleware to enforce access control
4. **Company Admin System**: Company-scoped user management
5. **Permission Conflict Handling**: Graceful handling of permission violations

User Roles

Available Roles

- `user`: Standard user with limited permissions based on assigned modules
- `company-admin`: Can manage users within their company, has COMPANY permission
- `admin`: System administrator with full access to all modules and companies
- `superadmin`: Highest level access with full system control

Role Hierarchy

```
superadmin > admin > company-admin > user
```

Role Access Matrix

Role	Own Company	Other Companies	Create Companies	System Admin
`user`	Limited (based on permissions)			
`company-admin`	Full access			
`admin`				
`superadmin`				

Module Permissions

Available Modules

The system supports the following module permissions:

```
enum ModulePermission {
    SETTINGS = 'settings',
    PURCHASE_REQUEST = 'purchase-request',
    PURCHASE_ORDERS = 'purchase-orders',
    REQUISITION_FOR_QUOTE = 'requisition-for-quote',
    SOURCING = 'sourcing',
    ADMIN = 'admin',
    COMPANY = 'company',
    VERIDION = 'veridion',
    AI = 'ai'
}
```

Module Descriptions

- **`settings`**: Access to company settings, suppliers, categories, etc.
- **`purchase-request`**: Create and manage purchase requests
- **`purchase-orders`**: Create and manage purchase orders
- **`requisition-for-quote`**: Handle RFQ processes
- **`sourcing`**: Access sourcing tools (matching, outreach, quote tracking)
- **`admin`**: Administrative functions (for system admin/superadmin)
- **`company`**: Company management functions (for company-admin)
- **`veridion`**: Veridion integration features
- **`ai`**: AI-powered features

Implementation Guide

1. Protecting Routes with Permissions

Use the `@RequirePermissions()` decorator along with the `PermissionsGuard`:

```
import { Controller, Get, UseGuards } from '@nestjs/common';
import { JwtAuthGuard } from '../auth/guards/jwt-auth.guard';
import { PermissionsGuard } from '../auth/guards/permissions.guard';
import { RequirePermissions } from '../auth/decorators/require-permissions.decorator';
import { ModulePermission } from '../auth/enums/module-permission.enum';

@Controller('settings')
@UseGuards(JwtAuthGuard, PermissionsGuard)
@RequirePermissions(ModulePermission.SETTINGS)
export class SettingsController {

    @Get('suppliers')
```

```
    async getSuppliers() {
        // Only users with SETTINGS permission can access this
        return this.settingsService.getSuppliers();
    }
}
```

2. Multiple Permission Requirements

You can require multiple permissions for a single endpoint:

```
@Get('advanced-sourcing')
@RequirePermissions(ModulePermission.SOURCING, ModulePermission.AI)
async getAdvancedSourcing() {
    // User needs both SOURCING and AI permissions
    return this.sourcingService.getAdvancedFeatures();
}
```

3. Method-Level Permission Override

Override class-level permissions at the method level:

```
@Controller('purchase')
@UseGuards(JwtAuthGuard, PermissionsGuard)
@RequirePermissions(ModulePermission.PURCHASE_REQUEST)
export class PurchaseController {

    @Get()
    async getPurchaseRequests() {
        // Requires PURCHASE_REQUEST permission (from class)
    }

    @Get('orders')
    @RequirePermissions(ModulePermission.PURCHASE_ORDERS)
    async getPurchaseOrders() {
        // Overrides class permission, requires PURCHASE_ORDERS
    }
}
```

Company Admin System

The company admin system allows for hierarchical user management where:

1. **Super Admin/Admin** can create companies and manage all users across all companies
2. **Company Admin** can only manage users within their own company
3. **Users** have limited access based on their assigned module permissions

Access Control Flow

`User Request → Authentication Check → Permission Check → Company Scope Check → Allow/Deny`

Permission Inheritance

- **Super Admin/Admin**: Automatic access to all modules and companies
- **Company Admin**: Automatic access to all modules within their company (COMPANY permission)
- **User**: Access only to explicitly assigned modules within their company

Permission Validation

Automatic Permission Checking

The `PermissionsGuard` automatically validates permissions:

1. **Admin/Company-Admin**: Automatically granted access to all modules
2. **Regular Users**: Checked against their `modulePermissions` array
3. **Missing Permissions**: Returns `403 Forbidden` with descriptive error message

Manual Permission Checking

For custom permission logic, inject the user's permissions:

```
@Get('custom-feature')
async getCustomFeature(@GetUser() user: any) {
    const userPermissions = user.modulePermissions || [];

    if (!userPermissions.includes(ModulePermission.SETTINGS)) {
        // Handle missing permission gracefully
        return { autoFillEnabled: false, data: [] };
    }

    // User has permission, enable full features
    return { autoFillEnabled: true, data: this.getFullData() };
}
```

Permission Conflict Handling

Auto-Fill Scenarios

When users lack required permissions for auto-fill features:

```
// In your service
async getFormData(user: any) {
    const hasSettingsAccess = user.modulePermissions?.includes(ModulePermission.SETTINGS);
```

```
    if (!hasSettingsAccess) {
        // Disable auto-fill, return empty form
        return {
            autoFillEnabled: false,
            suppliers: [],
            categories: []
        };
    }

    // Enable auto-fill with settings data
    return {
        autoFillEnabled: true,
        suppliers: await this.getSuppliers(),
        categories: await this.getCategories()
    };
}
```

Graceful Feature Degradation

Handle missing permissions without breaking the user experience:

```
@Get('dashboard')
async getDashboard(@GetUser() user: any) {
    const permissions = user.modulePermissions || [];
    const dashboard = { modules: {} };

    if (permissions.includes(ModulePermission.PURCHASE_REQUEST)) {
        dashboard.modules.purchaseRequests = await this.getPurchaseRequests();
    }

    if (permissions.includes(ModulePermission.SOURCING)) {
        dashboard.modules.sourcing = await this.getSourcingData();
    }

    return dashboard;
}
```

User Schema Structure

User Document Fields

```
interface User {
    email: string;
    password: string;
    name: string;
    isEmailVerified: boolean;
    authProvider: string;
```

```
    role: string; // 'user', 'admin', 'superadmin', 'company-admin'

    // Company-related fields
    companyId?: ObjectId; // Reference to company
    modulePermissions: string[]; // Array of allowed modules
    createdBy?: ObjectId; // Reference to company-admin who created this user
}
```

Example User Document

```
{
    "_id": ObjectId("..."),
    "email": "user@company.com",
    "name": "John Doe",
    "role": "user",
    "companyId": ObjectId("company_id"),
    "modulePermissions": ["settings", "purchase-request"],
    "createdBy": ObjectId("admin_user_id"),
    "isEmailVerified": true,
    "authProvider": "email"
}
```

API Endpoints

Company Management Endpoints (Admin/SuperAdmin only)

Method	Endpoint	Description	Required Permission
POST	`/companies`	Create a new company with admin	`ADMIN`
GET	`/companies`	Get all companies	`ADMIN`
GET	`/companies/:companyId/validate-admin`	Validate company admin	`ADMIN`

Create Company with Admin **POST** `/companies`

Headers:

```
Authorization: Bearer <admin_jwt_token>
Content-Type: application/json
```

Request Body:

```
{
    "name": "Acme Corporation",
    "adminEmail": "admin@acme.com",
    "adminName": "John Admin",
```

```
    "adminPassword": "securePassword123",
    "isActive": true
}
```

Response (201):

```
{
    "company": {
        "_id": "64f8a1b2c3d4e5f6a7b8c9d0",
        "name": "Acme Corporation",
        "adminUserId": "64f8a1b2c3d4e5f6a7b8c9d1",
        "isActive": true,
        "createdAt": "2024-01-15T10:30:00.000Z",
        "updatedAt": "2024-01-15T10:30:00.000Z"
    },
    "admin": {
        "_id": "64f8a1b2c3d4e5f6a7b8c9d1",
        "email": "admin@acme.com",
        "name": "John Admin",
        "role": "company-admin",
        "companyId": "64f8a1b2c3d4e5f6a7b8c9d0",
        "modulePermissions": ["company"],
        "isEmailVerified": true,
        "createdAt": "2024-01-15T10:30:00.000Z"
    }
}
```

Get All Companies **GET** `/companies`

Headers:

`Authorization: Bearer <admin_jwt_token>`

Response (200):

```
[
    {
        "_id": "64f8a1b2c3d4e5f6a7b8c9d0",
        "name": "Acme Corporation",
        "adminUserId": {
            "_id": "64f8a1b2c3d4e5f6a7b8c9d1",
            "name": "John Admin",
            "email": "admin@acme.com"
        },
        "isActive": true,
        "createdAt": "2024-01-15T10:30:00.000Z"
    }
]
```

Company Admin Endpoints

Method	Endpoint	Description	Required Permission
POST	`/company-admin/:companyId/users`	Create company user	ADMIN or COMPANY
GET	`/company-admin/:companyId/users`	Get paginated list of company users	ADMIN or COMPANY
PATCH	`/company-admin/:companyId/users/:userId/permissions`	Update user permissions	ADMIN or COMPANY
DELETE	`/company-admin/:companyId/users/:userId`	Delete company user	ADMIN or COMPANY
GET	`/company-admin/:companyId/users/:userId/access-validation`	Validate user access	ADMIN or COMPANY

Create Company User **POST** `/company-admin/:companyId/users`

Headers:

```
Authorization: Bearer <company_admin_jwt_token>
Content-Type: application/json
```

Request Body:

```
{
    "email": "newuser@company.com",
    "name": "New User",
    "password": "securePassword123",
    "modulePermissions": ["settings", "purchase-request", "sourcing"]
}
```

Response (201):

```
{
    "_id": "64f8a1b2c3d4e5f6a7b8c9d2",
    "email": "newuser@company.com",
    "name": "New User",
    "role": "user",
    "companyId": "64f8a1b2c3d4e5f6a7b8c9d0",
    "modulePermissions": ["settings", "purchase-request", "sourcing"],
    "createdBy": "64f8a1b2c3d4e5f6a7b8c9d1",
    "isEmailVerified": true,
    "createdAt": "2024-01-15T11:00:00.000Z"
}
```

Get Company Users (Paginated) **GET** `/company-admin/:companyId/users?page=1&limit=10`

Headers:

```
Authorization: Bearer <company_admin_jwt_token>
```

Response (200):


```
{
    "users": [
        {
            "_id": "64f8a1b2c3d4e5f6a7b8c9d2",
            "email": "user1@company.com",
            "name": "User One",
            "role": "user",
            "modulePermissions": ["settings", "purchase-request"],
            "createdBy": {
                "_id": "64f8a1b2c3d4e5f6a7b8c9d1",
                "name": "John Admin",
                "email": "admin@acme.com"
            },
            "createdAt": "2024-01-15T11:00:00.000Z"
        },
        {
            "_id": "64f8a1b2c3d4e5f6a7b8c9d3",
            "email": "user2@company.com",
            "name": "User Two",
            "role": "user",
            "modulePermissions": ["sourcing", "ai"],
            "createdBy": {
                "_id": "64f8a1b2c3d4e5f6a7b8c9d1",
                "name": "John Admin",
                "email": "admin@acme.com"
            },
            "createdAt": "2024-01-15T12:00:00.000Z"
        }
    ],
    "total": 25,
    "page": 1,
    "limit": 10,
    "totalPages": 3
}
```


Update User Permissions **PATCH** `/company-admin/:companyId/users/:userId/permissions`

Headers:

```
Authorization: Bearer <company_admin_jwt_token>
Content-Type: application/json
```

Request Body:

```
{
    "modulePermissions": ["settings", "purchase-request", "sourcing", "ai"]
}
```

Response (200):

```
{
    "_id": "64f8a1b2c3d4e5f6a7b8c9d2",
    "email": "user@company.com",
    "name": "User Name",
    "role": "user",
    "companyId": "64f8a1b2c3d4e5f6a7b8c9d0",
    "modulePermissions": ["settings", "purchase-request", "sourcing", "ai"],
    "createdBy": "64f8a1b2c3d4e5f6a7b8c9d1",
    "updatedAt": "2024-01-15T13:00:00.000Z"
}
```

Delete Company User **DELETE** `/company-admin/:companyId/users/:userId`

Headers:

`Authorization: Bearer <company_admin_jwt_token>`

Response (204): No Content

Validate User Access **GET** `/company-admin/:companyId/users/:userId/access-validation`

Headers:

`Authorization: Bearer <company_admin_jwt_token>`

Response (200):

```
{
    "hasAccess": true
}
```

Error Handling

Common Error Responses

400 Bad Request - Validation Failed

```
{
    "statusCode": 400,
    "message": [
        "email must be an email",
        "password must be longer than or equal to 8 characters"
    ],
    "error": "Bad Request"
}
```

401 Unauthorized - Invalid Token


```
{
    "statusCode": 401,
    "message": "Invalid token",
    "error": "Unauthorized"
}
```


403 Forbidden - Missing Permissions


```
{
    "statusCode": 403,
    "message": "Access denied. Required permissions: [settings]. User has: [purchase-request
    "error": "Forbidden"
}
```


403 Forbidden - Company Access Denied


```
{
    "statusCode": 403,
    "message": "Access denied: User does not belong to your company",
    "error": "Forbidden"
}
```


403 Forbidden - Not Company Admin


```
{
    "statusCode": 403,
    "message": "User is not authorized for company operations",
    "error": "Forbidden"
}
```


404 Not Found - User Not Found


```
{
    "statusCode": 404,
    "message": "User not found",
    "error": "Not Found"
}
```


404 Not Found - Company Not Found


```
{
    "statusCode": 404,
    "message": "Company not found",
    "error": "Not Found"
}
```

409 Conflict - User Already Exists

```
{
    "statusCode": 409,
    "message": "User with this email already exists",
    "error": "Conflict"
}
```

409 Conflict - Company Already Exists

```
{
    "statusCode": 409,
    "message": "Company with this name already exists",
    "error": "Conflict"
}
```

Best Practices

1. Permission Assignment

- **Principle of Least Privilege**: Only assign necessary permissions
- **Role-Based Assignment**: Group permissions logically by job function
- **Regular Review**: Periodically review and update user permissions

2. Error Handling

- **Graceful Degradation**: Disable features instead of showing errors when possible
- **Clear Messages**: Provide descriptive error messages for permission violations
- **Logging**: Log permission violations for security monitoring

3. Frontend Integration

```
// Check permissions in frontend
const userPermissions = user.modulePermissions || [];
const hasSettingsAccess = userPermissions.includes('settings');

// Conditionally render UI elements
{hasSettingsAccess && (
    <SettingsPanel />
)}

// Disable features gracefully
<AutoFillButton disabled={!hasSettingsAccess} />
```

4. Testing Permissions

```
// Test permission enforcement
describe('PermissionsGuard', () => {
    it('should allow access with correct permissions', async () => {
        const user = {
            role: 'user',
            modulePermissions: ['settings']
        };
        // Test access granted
    });

    it('should deny access without permissions', async () => {
        const user = {
            role: 'user',
            modulePermissions: []
        };
        // Test access denied
    });
});
```

Security Considerations

1. Permission Validation

- Always validate permissions on the server side
- Never rely solely on frontend permission checks
- Use the `PermissionsGuard` for all protected routes

2. Company Isolation

- Company admins can only manage users within their company
- Data is automatically scoped to the user's company
- Cross-company access is prevented at the service level

3. Audit Logging

- All permission changes are logged for audit purposes
- User creation, updates, and deletions are tracked
- Logs include admin details, timestamps, and affected users

Troubleshooting

Common Issues

1. **Permission Denied Errors**
    - Check user's `modulePermissions` array
    - Verify the correct permission is required in the decorator

- Ensure `PermissionsGuard` is applied to the route

2. **Auto-Fill Not Working**
   - Verify user has `settings` permission
   - Check permission conflict handling in the service
   - Ensure graceful degradation is implemented
3. **Company Admin Cannot Create Users**
   - Verify user role is `company-admin`
   - Check if company exists and is linked to the admin
   - Ensure admin is authenticated properly

Debug Commands

```
# Check user permissions in database
db.users.findOne({email: "user@company.com"}, {modulePermissions: 1, role: 1, companyId: 1})

# Check company admin setup
db.companies.findOne({adminUserId: ObjectId("admin_id")})

# View audit logs
db.auditlogs.find({adminId: "admin_id"}).sort({createdAt: -1})
```

Real-World Usage Examples

Example 1: Company Setup Flow

1. **Super Admin creates a company:**

```
curl -X POST http://localhost:3000/companies \
  -H "Authorization: Bearer <super_admin_token>" \
  -H "Content-Type: application/json" \
  -d '{
    "name": "Tech Solutions Inc",
    "adminEmail": "admin@techsolutions.com",
    "adminName": "Sarah Johnson",
    "adminPassword": "SecurePass123!",
    "isActive": true
  }'
```

2. **Company Admin creates users:**

```
curl -X POST http://localhost:3000/company-admin/64f8a1b2c3d4e5f6a7b8c9d0/users \
  -H "Authorization: Bearer <company_admin_token>" \
  -H "Content-Type: application/json" \
  -d '{
    "email": "procurement@techsolutions.com",
    "name": "Mike Procurement",
    "password": "UserPass123!",
```

```
    "modulePermissions": ["purchase-request", "purchase-orders", "settings"]
  }'
```

Example 2: Permission-Based Feature Access

```
// In your controller
@Get('dashboard')
@UseGuards(JwtAuthGuard)
async getDashboard(@GetUser() user: any) {
    const permissions = user.modulePermissions || [];
    const dashboard: any = { modules: {} };

    // Only show purchase requests if user has permission
    if (permissions.includes('purchase-request')) {
        dashboard.modules.purchaseRequests = await this.getPurchaseRequests(user.companyId);
    }

    // Only show sourcing data if user has permission
    if (permissions.includes('sourcing')) {
        dashboard.modules.sourcing = await this.getSourcingData(user.companyId);
    }

    return dashboard;
}
```

Example 3: Multi-Company Admin Access

```
# Super Admin can access any company's users
curl -X GET http://localhost:3000/company-admin/64f8a1b2c3d4e5f6a7b8c9d0/users \
  -H "Authorization: Bearer <super_admin_token>"

# Company Admin can only access their own company's users
curl -X GET http://localhost:3000/company-admin/64f8a1b2c3d4e5f6a7b8c9d0/users \
  -H "Authorization: Bearer <company_admin_token>"
```

Integration Guide

Frontend Integration

```
// Check user permissions in React/Vue/Angular
const UserDashboard = ({ user }) => {
    const hasSettingsAccess = user.modulePermissions?.includes('settings');
    const hasSourcingAccess = user.modulePermissions?.includes('sourcing');

    return (
        <div>
            {hasSettingsAccess && <SettingsPanel />}
```

```
            {hasSourcingAccess && <SourcingTools />}
            {!hasSettingsAccess && <div>Contact admin for settings access</div>}
        </div>
    );
};
```

Backend Service Integration

```
// In your service
@Injectable()
export class PurchaseService {
    async createPurchaseRequest(userId: string, data: any) {
        const user = await this.userService.findById(userId);

        if (!user.modulePermissions.includes('purchase-request')) {
            throw new ForbiddenException('Purchase request permission required');
        }

        // Create purchase request scoped to user's company
        return this.purchaseModel.create({
            ...data,
            companyId: user.companyId,
            createdBy: userId
        });
    }
}
```

This documentation provides a comprehensive guide to using the role-based access system. For additional support or questions, refer to the source code in the `src/auth` and `src/company` directories.