**🧭 1. Core Idea: Dynamic, Data-Driven RBAC Template**

Think of it as **plug-and-play modules**:

1. **Menu / Module Registry**
   * Every feature/module you build registers itself with a **menu key** and a set of possible **action keys**.
   * Example structure:
2. Module: Loans
3. menuKey: "loan"
4. actions: ["create", "edit", "delete", "approve", "view"]
5. Module: Customers
6. menuKey: "customer"
7. actions: ["create", "edit", "delete", "view"]
8. **Roles Table**
   * Roles are just **collections of permission keys**.
   * No need to hardcode per module; any module added later can just use the same role → permission system.
9. **Role-Permission Assignment GUI**
   * Dynamically generate checkboxes from the module registry.
   * When a new module is added, the GUI automatically shows its actions without any code change.
10. **Route Guard / Middleware**
    * Generic guard checks:
      + Does the user have the **menu key**? → allow route
      + Does the user have the **action key**? → enable button / allow API
    * No changes needed for new modules — just add them to the registry.

**🧱 2. Example Conceptual Flow**

User logs in → get role permissions

Frontend:

Sidebar → render only menus where user has menuKey

Pages → enable buttons based on action keys

Backend:

Middleware → check route’s menuKey & actionKey against user permissions

* New module = add **menu key + actions** in registry
* Role assignment GUI → automatically shows new actions
* Route guard + API middleware → automatically enforce access

✅ **No need to touch core logic** when adding new modules.

**🧩 3. How to Structure This Template**

1. **Module Registry Table / Config**
   * menuKey
   * displayName
   * actions (array)
2. **Roles Table**
   * roleName
   * description
3. **Role-Permission Table**
   * roleId
   * menuKey
   * actionKey
4. **Users Table**
   * userId
   * roleId
   * tenantId (if applicable)
5. **Frontend**
   * Dynamic sidebar + buttons
   * Route guard that reads menuKey/actionKey
6. **Backend**
   * Middleware that reads menuKey/actionKey

**🧠 4. How Extending Works**

1. Create new module → define menuKey + actions
2. Add module to **registry**
3. Role GUI automatically shows checkboxes for this module
4. Route guard and API middleware automatically enforce permissions
5. Done — no changes in core RBAC logic

**🔑 5. Benefits of This Template**

* ✅ **Scalable** → add unlimited modules without touching core RBAC
* ✅ **Flexible** → any combination of permissions per role
* ✅ **Secure** → frontend and backend enforcement uses same data
* ✅ **Future-proof** → works for multi-tenant SaaS

**A. Module / Feature Registry**

This is your **single source of truth** for every module/feature in the system.

* Each module registers:
  + menuKey → unique key for the module
  + displayName → human-readable name
  + actions → array of action keys (create, edit, delete, etc.)

**Example:**

Module: Loans

menuKey: "loan"

displayName: "Loans"

actions: ["create", "edit", "delete", "approve"]

Module: Customers

menuKey: "customer"

displayName: "Customers"

actions: ["create", "edit", "delete"]

**B. Roles**

* Each role is a **collection of permissions**, not tied to specific modules in code.
* Roles table:

roleId | roleName | description

1 | Tenant Admin | Full access to tenant

2 | Loan Officer | Can manage loans, view customers

3 | IT Support | Can view tenants and system users

**C. Role-Permissions Table**

* Connects **roles** with **modules and actions** dynamically.

roleId | menuKey | actionKey

2 | "loan" | "create"

2 | "loan" | "edit"

2 | "customer" | "view"

3 | "systemUser" | "view"

3 | "systemUser" | "edit"

**D. Users Table**

* Assign a role to each user, along with tenant association:

userId | username | roleId | tenantId

101 | JohnDoe | 2 | 1

102 | JaneIT | 3 | null

**E. Frontend Flow**

1. On login → fetch user permissions (menuKey + actionKeys)
2. **Sidebar / Menu:** render only menu items where menuKey exists
3. **Route Guard:** allow route if menuKey exists
4. **Page / Buttons:** enable buttons based on action keys
5. **Add New Module:** update registry → GUI automatically shows checkboxes → roles can assign permissions

**Example:**

Permissions: ["loan:create", "loan:edit", "customer:view"]

Sidebar shows: Loans, Customers

Buttons:

- Loan page → show Create/Edit buttons

- Customer page → view only

**F. Backend Flow**

1. Middleware checks:
   * Does user have menuKey to access the route?
   * Does user have actionKey to perform the API action?
2. If missing → return 403
3. Works automatically for **new modules**, since the permission keys are stored dynamically in DB

**G. Adding New Modules / Extending**

1. Add module to **Module Registry** with menuKey + actions
2. Role GUI automatically shows new actions
3. Route guard and API middleware automatically enforce permissions
4. Done — no need to touch core logic

**🧭 1. System Space RBAC**

*Controls the SaaS platform itself, all tenants, system users, billing, and global modules.*

| **Role** | **Description** | **Menu Keys** | **Action Keys** |
| --- | --- | --- | --- |
| **System Super Admin** | Full control over the platform | All menus (dashboard, tenant, systemUser, billing, audit) | All actions (create, edit, delete, view, update) |
| **System Admin / Staff** | Manage tenants and system users | dashboard, tenant, systemUser | create, edit, delete, view |
| **System Support / IT Support** | Provides support for tenants | dashboard, tenant, systemUser | view, edit (limited) |
| **System Auditor / Compliance** | Audits and views system activities | dashboard, tenant, systemUser, billing, audit | view |
| **Billing Manager** | Handles subscription & billing | dashboard, billing, tenant | view, update |

**Notes:**

* Menu key presence = view/access to page/module
* Action keys control buttons and API actions (create, edit, delete)
* Backend route guards enforce both menu + action keys
* Frontend controls sidebar/menu and button visibility

**🏢 2. Tenant Space RBAC**

*Controls users and actions inside a tenant organization, isolated per tenant.*

| **Role** | **Description** | **Menu Keys** | **Action Keys** |
| --- | --- | --- | --- |
| **Tenant Admin / Owner** | Full control of their tenant | All tenant menus (dashboard, loan, customer, payment, appraisal, user, report, settings) | All actions (create, edit, delete, approve, export) |
| **Branch Manager** | Manages branch operations | dashboard, loan, customer, report | view, create, edit, approve |
| **Loan Officer** | Handles loans and customers | dashboard, loan, customer | view, create, edit |
| **Cashier / Teller** | Processes payments and renewals | dashboard, loan, payment | view, receive, refund |
| **Appraiser** | Evaluates collateral items | dashboard, appraisal | view, create, edit, approve |
| **Auditor / Accountant** | Reviews all financial transactions | dashboard, loan, customer, payment, report | view, export |
| **Viewer / Read-Only** | Can view data only | dashboard, loan, customer, payment, report | view |

**Notes:**

* Tenant roles **cannot affect other tenants**
* Menu keys control page visibility + route guard
* Action keys control buttons/API calls
* Adding new modules: system registers the menu & actions → tenant can assign to roles

**🧩 3. Key Principles for Both Spaces**

1. **Unified permission system**
   * Menu key = page/module access
   * Action keys = buttons/actions on page/API calls
2. **Frontend enforcement**
   * Sidebar and menu rendering → menu keys
   * Button enable/disable → action keys
3. **Backend enforcement**
   * Route guard/middleware → menu key + action key
   * Tenant isolation → ensure users cannot access data outside their tenant
4. **Extensibility**
   * Add new module → define menuKey + actions → automatically available in tenant role GUI and enforced by route guards

✅ **Outcome:**

* System space handles platform-wide control and sensitive modules
* Tenant space handles organization-level control with isolated RBAC
* Same RBAC framework applies to both spaces → reusable, scalable, and secure

A screenshot of a computer screen

AI-generated content may be incorrect.

**🧭 Single users Table Approach**

**Table Structure Example:**

| **Column** | **Description** |
| --- | --- |
| userId | Primary key |
| username | Login name / email |
| password | Hashed password |
| roleId | Role assigned (system role or tenant role) |
| tenantId | NULL for system users, tenant ID for tenant users |
| userType | "system" or "tenant" (optional, can infer from tenantId) |
| status | Active / inactive |

**🏗️ Recommended RBAC Tables**

**1. Users Table (system + tenant users)**

| **Column** | **Description** |
| --- | --- |
| userId | PK |
| username | Email/login |
| password | Hashed password |
| roleId | FK to roles (system or tenant) |
| tenantId | NULL for system users, tenant ID for tenant users |
| status | Active / inactive |
| userType | Optional: "system" / "tenant" |

**2. Roles Table**

| **Column** | **Description** |
| --- | --- |
| roleId | PK |
| roleName | System or Tenant role name |
| description | Optional description |
| space | "system" or "tenant" |

**3. Modules Table (menu keys)**

| **Column** | **Description** |
| --- | --- |
| moduleId | PK |
| menuKey | Unique key (used in route guard + menu rendering) |
| displayName | Human-readable menu name |
| space | "system" or "tenant" |

**4. Role\_Permissions Table**

| **Column** | **Description** |
| --- | --- |
| roleId | FK → roles |
| menuKey | FK → modules (determines page/module access) |
| actionKey | Action permission (create, edit, delete, approve, etc.) |

* **Menu key presence** → page/module access
* **Action key presence** → controls buttons/API actions

**5. Tenants Table**

| **Column** | **Description** |
| --- | --- |
| tenantId | PK |
| name | Tenant organization name |
| status | Active/Inactive |

**🧩 How It Works**

1. **Login**
   * Determine userType and fetch roleId → get permissions
2. **Frontend**
   * Sidebar: render menu if menuKey exists
   * Buttons: enable/disable based on actionKey
3. **Backend**
   * Middleware checks menuKey for route access
   * Checks actionKey for API actions
   * Checks tenantId to enforce tenant isolation

**🔑 Why This Is Best**

* **Single users table** simplifies everything
* **Roles + Role\_Permissions** are reusable for system and tenants
* **Modules table** allows plug-and-play for new features
* **Action keys** control buttons and API, no need for separate “view” key
* Fully extendable: new modules, actions, tenants, or roles → no core logic changes

**🏗️ Recommended RBAC Tables**

**1. Users Table (system + tenant users)**

| **Column** | **Description** |
| --- | --- |
| userId | PK |
| username | Email/login |
| password | Hashed password |
| roleId | FK to roles (system or tenant) |
| tenantId | NULL for system users, tenant ID for tenant users |
| status | Active / inactive |
| userType | Optional: "system" / "tenant" |

**2. Roles Table**

| **Column** | **Description** |
| --- | --- |
| roleId | PK |
| roleName | System or Tenant role name |
| description | Optional description |
| space | "system" or "tenant" |

**3. Modules Table (menu keys)**

| **Column** | **Description** |
| --- | --- |
| moduleId | PK |
| menuKey | Unique key (used in route guard + menu rendering) |
| displayName | Human-readable menu name |
| space | "system" or "tenant" |

**4. Role\_Permissions Table**

| **Column** | **Description** |
| --- | --- |
| roleId | FK → roles |
| menuKey | FK → modules (determines page/module access) |
| actionKey | Action permission (create, edit, delete, approve, etc.) |

* **Menu key presence** → page/module access
* **Action key presence** → controls buttons/API actions

**5. Tenants Table**

| **Column** | **Description** |
| --- | --- |
| tenantId | PK |
| name | Tenant organization name |
| status | Active/Inactive |

**🧩 How It Works**

1. **Login**
   * Determine userType and fetch roleId → get permissions
2. **Frontend**
   * Sidebar: render menu if menuKey exists
   * Buttons: enable/disable based on actionKey
3. **Backend**
   * Middleware checks menuKey for route access
   * Checks actionKey for API actions
   * Checks tenantId to enforce tenant isolation

**🔑 Why This Is Best**

* **Single users table** simplifies everything
* **Roles + Role\_Permissions** are reusable for system and tenants
* **Modules table** allows plug-and-play for new features
* **Action keys** control buttons and API, no need for separate “view” key
* Fully extendable: new modules, actions, tenants, or roles → no core logic changes