# Proposed RBAC Model for Eden ERP

## Permission Granularity & CRUD Actions

To meet the new requirements, **split broad permissions like “tasks:write” into more granular actions** (Create, Edit, Delete, Comment). This aligns with best practices: for example, deletion is often treated as a separate, elevated permission not given to regular users[[1]](https://www.reddit.com/r/ExperiencedDevs/comments/zd5ami/what_are_the_best_practices_around_developing_rbac/#:~:text=I%20wouldn%27t%20let%20writer%20delete,but%20deletes%20are%20a%20nono). In practice, many systems define distinct permissions for Create, Update (Edit), and Delete – allowing roles that can add and modify entries while **withholding destructive delete rights**[[2]](https://confluence.atlassian.com/adminjiraserver/managing-project-permissions-938847145.html#:~:text=,it%E2%80%99s%20only%20given%20to%20administrators). It’s recommended to avoid “write with exceptions” (i.e. granting tasks:write then disallowing delete) and instead **explicitly grant create/edit vs. delete**. This clarity prevents mistakes and enforces least privilege. Jira’s permission scheme illustrates this approach: it requires a specific “Delete Issues” permission (usually reserved for admins) separate from general edit rights[[2]](https://confluence.atlassian.com/adminjiraserver/managing-project-permissions-938847145.html#:~:text=,it%E2%80%99s%20only%20given%20to%20administrators)[[3]](https://confluence.atlassian.com/adminjiraserver/managing-project-permissions-938847145.html#:~:text=related%20comments%20and%20attachments%20will,it%E2%80%99s%20only%20given%20to%20administrators). In short, **define fine-grained actions** – e.g. tasks:create, tasks:edit, tasks:delete – so you can give the new Contributor role create/edit rights without delete privileges.

For **project management actions**, apply similar granularity. The existing model already separates project-level rights (projects:read/write) from task permissions. Continue this separation so that roles can create or modify tasks *without* being able to create or delete projects. Many real-world platforms enforce this division. For instance, Basecamp distinguishes between internal users who can create projects and outside collaborators who *cannot* create projects or change project settings[[4]](https://3.basecamp-help.com/article/75-permissions#:~:text=This%20person%20doesn%E2%80%99t%20work%20at,Access%20Projects). Similarly, in our ERP, a **Contributor** can handle tasks but would have no project creation or deletion capability. This ensures someone can log and update work items freely while higher-level organizational changes (like creating or archiving projects) remain restricted to admins or project managers.

## Role Naming Conventions

When naming roles, favor clarity and consistency. Roles should be **descriptive of their function or level of access**, rather than tied to an individual’s exact job title[[5]](https://www.myshyft.com/blog/role-based-access-control-2/#:~:text=several%20key%20practices%3A%20establish%20formal,should%20specifically%20examine%20role%20structures). In RBAC design, a role represents a set of permissions aligned with a job function or responsibility (e.g. *Administrator*, *Manager*, *Contributor*)[[6]](https://medium.com/@nocobase/how-to-design-an-rbac-role-based-access-control-system-3b57ca9c6826#:~:text=2). For Eden ERP, the existing names like **ADMIN** and **VIEWER** are clear and generic, which is good. The **OPS** role name is somewhat department-oriented (“Operations”), but it effectively conveys that role’s focus on operational task management. Introducing a new role named **Contributor** fits a permission-based naming style (similar to “Editor” or “Member” in other systems). This generic term emphasizes the role’s abilities (contributing to tasks) rather than a specific job title, making it intuitive across the organization. Many platforms use such naming: for example, Azure’s built-in roles include *Reader*, *Contributor*, *Owner*, which map to permission scopes rather than specific job positions[[7]](https://medium.com/@ajithcrajendran/understanding-the-azure-rbac-structure-7666529209ff#:~:text=Default%20RBAC%20roles). Likewise, tools like ClickUp or Notion use roles like *Admin*, *Member/Contributor*, *Commenter*, *Viewer*, which users find easy to understand.

**Why not use job titles?** Tying roles to job titles (e.g. *Technician*, *Coordinator*) can become messy if titles change or if multiple titles share the same access needs. Instead, define roles by access level or function. For example, *Contributor* (or “Team Member”) could cover all regular employees who need to create and update tasks, regardless of whether they’re a plumber, technician, or coordinator. This keeps the RBAC scheme **flexible and scalable**. Ensure role names are consistent (don’t mix naming styles) and clearly communicate the role’s scope. As one best-practice guide puts it, use naming conventions that reflect the role’s function and make its permissions obvious to users and admins[[5]](https://www.myshyft.com/blog/role-based-access-control-2/#:~:text=several%20key%20practices%3A%20establish%20formal,should%20specifically%20examine%20role%20structures). In summary, **prefer permission-level names** (Admin, Contributor, Viewer) or clearly defined function names, and avoid overly granular role proliferation. Each role should correspond to a broad category of access that many users can share (avoiding one-role-per-user scenarios)[[8]](https://medium.com/@nocobase/how-to-design-an-rbac-role-based-access-control-system-3b57ca9c6826#:~:text=Recommendation%3A%20Keep%20the%20number%20of,for%20a%20type%20of%20user).

## Task vs. Comment Permissions

**Should commenting be a separate permission?** In many systems, **yes – commenting is treated as distinct from full edit rights**. This allows, for example, a user who cannot edit a task’s core details to still comment or add updates. Notion, for instance, has a **“Can comment”** access level that lets a user discuss or add notes without editing the page’s content[[9]](https://www.notion.com/help/sharing-and-permissions#:~:text=filters,here%20%E2%86%92). Jira likewise separates the “Add Comments” permission from “Edit Issues” to let some users comment on an issue even if they can’t modify other fields[[10]](https://confluence.atlassian.com/adminjiraserver/managing-project-permissions-938847145.html#:~:text=Add%20comments).

For Eden ERP, it’s wise to introduce a permission like comments:write (or include it as its own subset of task permissions). The new Contributor role would then have tasks:create, tasks:edit, and comments:write – meaning they can create tasks, update task details, and comment – but **no delete powers**. By separating comments, you also enable a potential future scenario of a “Commenter” role (read + comment only, no task edits) if needed, similar to Notion’s setup. If you determine that every role who can edit tasks will inherently be allowed to comment, you could fold commenting into general task write access; however, separating it provides more flexibility. Given the common pattern that *commenting is a non-destructive action that many users may need even with limited edit rights*, a **dedicated comments permission** is recommended for clarity.

## Role Definitions and New “Contributor” Role

Taking the above into account, here’s a **recommended roles and permissions schema** for Eden ERP:

* **ADMIN** – Full access to everything. This role retains projects:read, projects:write (which would include creating/editing/deleting projects), and all task permissions (tasks:read, tasks:create, tasks:edit, tasks:delete, comments:write). Admins can also manage users/roles if applicable. Essentially, Admin = superuser. *(No changes needed, except to explicitly include the new granular perms.)*
* **OPS (Operations Manager/Coordinator)** – Access to all task operations but not projects. Under the granular scheme, Ops would have: tasks:read, tasks:create, tasks:edit, **possibly tasks:delete**, and comments:write. They do **not** have any project creation or edit rights. In practice, you might choose to **forbid Ops from deleting tasks as well**, if deletion is considered an admin-only action. (If so, Ops would simply omit the tasks:delete permission while still being able to create and modify tasks.) The key is that Ops can fully manage tasks (assign them, update statuses, etc.) but cannot start new projects or modify project settings. This mirrors a project coordinator role in many systems – they handle day-to-day work items, but **cannot delete core records without escalation**[[1]](https://www.reddit.com/r/ExperiencedDevs/comments/zd5ami/what_are_the_best_practices_around_developing_rbac/#:~:text=I%20wouldn%27t%20let%20writer%20delete,but%20deletes%20are%20a%20nono). Adjust the deletion right based on your comfort: some organizations allow trusted managers to delete tasks; others require all deletions to go through Admin to prevent data loss. Given the principle of least privilege, it’s safer to exclude deletion by default and handle exceptions via Admin.
* **CONTRIBUTOR (Regular Employee)** – This is the new role for the majority of staff (e.g. field technicians, crew members). Contributors can **view all tasks, create new tasks, edit/update existing tasks, and add comments**, but cannot delete tasks or projects. In permission terms: tasks:read, tasks:create, tasks:edit, comments:write. **No** tasks:delete (and of course no project permissions). This allows regular employees to log work (e.g. create a task for a job done or a problem found), attach photos or notes (comments), and update statuses or details on tasks assigned to them. They would not be able to remove tasks – if a task is completed or added in error, they might mark it complete or request a manager/admin to delete or archive it. This aligns with real-world patterns: field techs create tasks (and perhaps upload photos/comments), coordinators review and might edit or close tasks, but **neither typically deletes tasks** in order to maintain records[[1]](https://www.reddit.com/r/ExperiencedDevs/comments/zd5ami/what_are_the_best_practices_around_developing_rbac/#:~:text=I%20wouldn%27t%20let%20writer%20delete,but%20deletes%20are%20a%20nono). By giving Contributors create/edit rights, we empower them to do their jobs without risking accidental data loss of entire tasks or projects. (If your system distinguishes editing one’s own tasks vs any tasks, you could further refine that – though often it’s acceptable for all contributors to update any task details since they’re all within the team. Fine-tuning by task assignment can also be handled via business rules or future ABAC if needed.)
* **VIEWER** – A read-only role. Viewers get tasks:read (and possibly projects:read if you want them to see project info or task groupings). They cannot modify or create anything. Viewers likely cannot comment either, since commenting is a form of writing. This role is for cases like executives or external auditors/clients who may need visibility into tasks/status but should not make any changes. In many systems “Can view” is the lowest level, with no commenting or editing allowed[[9]](https://www.notion.com/help/sharing-and-permissions#:~:text=filters,here%20%E2%86%92). If you wanted a role that can comment but not create tasks, you could introduce a **“Commenter”** role (tasks read + comment only), but unless you have that use case, it’s not necessary now. For now, Viewer remains strictly read-only (no comments:write, no task write).

**Table: Roles and Permissions** (summary)

| **Role** | **Projects** | **Tasks** | **Comments** |
| --- | --- | --- | --- |
| **Admin** | projects:read/write (all) | tasks:read/create/edit/delete (all actions on tasks) | comments:write (all) |
| **Ops** | *no project access* | tasks:read/create/edit (**no delete** by default) | comments:write |
| **Contributor** | *no project access* | tasks:read/create/edit (**no delete**) | comments:write |
| **Viewer** | (maybe projects:read only) | tasks:read only (no create/edit/delete) | *no* (no commenting) |

In this schema, the new **Contributor** role is essentially a limited version of Ops – capable of contributing content but lacking any dangerous permissions. The **Ops** role could be reserved for team leads or coordinators who might have one extra ability (if any) like deleting or bulk-editing tasks, but as noted, you might keep deletion exclusive to Admin anyway. Both Ops and Contributor share the core “task contributor” abilities, so in practice you might even merge these if there isn’t a strong distinction – but presumably Ops has some additional project oversight in your system (perhaps Ops can change task assignment across all users, etc., while Contributors might be intended to mainly manage their own tasks).

## Future-Proofing for Growth

To keep the RBAC model scalable, design it so new roles or modules can be added **without starting from scratch**. A few considerations:

* **Department-Specific Roles:** As Eden ERP expands, you might integrate modules for Accounting, HR, etc. This could necessitate roles like “Accounting Clerk” or “Sales Manager” with specialized permissions (e.g. access to financial records or CRM data in the ERP). One approach is to **layer department access on top of base roles**. For example, a user could have the Contributor role for general tasks, plus an “Accounting” role granting them finance module permissions. In classic RBAC, roles can be defined by department (Sales, Finance, Operations) or by function (Data Entry, Manager, etc.)[[11]](https://medium.com/@nocobase/how-to-design-an-rbac-role-based-access-control-system-3b57ca9c6826#:~:text=Common%20ways%20to%20define%20roles%3A) – often a combination is used. The key is to avoid explosion of very specific roles for every tiny variation. Instead, consider a **role hierarchy or multiple-role assignment**: Each role remains a general profile of permissions, and a user can hold more than one role if they straddle departments. For instance, a **“Field Technician”** role could be essentially the same as the Contributor (tasks create/edit) role. A **“Coordinator – Operations”** role might inherit Contributor permissions but also allow viewing all tasks in the Ops department or running certain reports. Using multi-role assignments (one user, many roles) allows mixing and matching permissions as needed[[12]](https://www.reddit.com/r/ExperiencedDevs/comments/zd5ami/what_are_the_best_practices_around_developing_rbac/#:~:text=to%20accomodate%20this,the%20changing%20the%20existing%20roles). For now, it’s enough to design roles for the known needs (Admin, Ops, Contributor, Viewer), but keep the structure flexible so you can introduce, say, an **“Accounting Viewer”** or **“Inventory Manager”** role later.
* **Resource-Based Separation:** Ensure your permissions are categorized by resource type (as we did with tasks vs projects). This modular approach means if tomorrow you add a **Work Orders module or an Invoicing module**, you can define new permissions (e.g. invoices:read/write) and create a role (e.g. “Billing Staff”) who has those without affecting the task/project roles. Eden ERP should maintain separate permission sets for each major feature (tasks, projects, comments, files, etc.), which can then be grouped into roles as needed. This makes the RBAC schema easy to extend – new resource or department needs = new permissions and possibly new roles, without changing the fundamentals for existing roles.
* **Role Hierarchies and Overrides:** It may be useful to implement role hierarchy (inheritance) if the system grows complex. For example, you could define **Viewer < Contributor < Admin** as a hierarchy where higher roles automatically include the lower roles’ permissions. This can simplify management (e.g. Admin inherently can do what a Contributor can, plus more). Some enterprise ERP systems use hierarchical roles or **role groups** to reflect organizational structure[[13]](https://stackoverflow.com/questions/32605252/rbac-roles-vs-business-functions#:~:text=London,Temporary)[[14]](https://stackoverflow.com/questions/32605252/rbac-roles-vs-business-functions#:~:text=London,building). However, hierarchy can also be managed simply by assignment (e.g. just give Admin all the perms, no need to formally tie it to Contributor role). If considering department overrides, one strategy is **policy-based exceptions** (a bit like ABAC) – for instance, generally no one can delete tasks, but maybe you allow a **specific override** for a “Project Owner” of a certain project. These kinds of special cases can often be handled outside of RBAC (via workflow or approval processes) to keep RBAC roles clean. A best practice from the security community is to keep roles broad and manage any unusual access via controlled exceptions or custom roles as needed[[12]](https://www.reddit.com/r/ExperiencedDevs/comments/zd5ami/what_are_the_best_practices_around_developing_rbac/#:~:text=to%20accomodate%20this,the%20changing%20the%20existing%20roles).
* **Real-World Patterns:** Learn from how other platforms group permissions. We’ve already drawn parallels to Jira (distinct issue actions), Notion (comment vs edit roles), Basecamp (project creation limited to certain user types), and ClickUp (which allows setting users as read-only, comment-only, edit, or full). It’s common that **normal team members can create and update content but not delete or administer**. For example, in Jira a “Developer” might create and edit issues but only a Project Lead or Admin can delete issues[[3]](https://confluence.atlassian.com/adminjiraserver/managing-project-permissions-938847145.html#:~:text=related%20comments%20and%20attachments%20will,it%E2%80%99s%20only%20given%20to%20administrators). In Basecamp, regular employees can create tasks/todos and collaborate, but only admins/owners can fully delete or restore things from the trash[[15]](https://3.basecamp-help.com/article/75-permissions#:~:text=,factor%20authentication)[[16]](https://3.basecamp-help.com/article/75-permissions#:~:text=,place%20it%20in%20the%20trash). Our proposed Contributor role is analogous to a “member” or “user” role in those systems: empowered to do daily work, while critical or irreversible actions are reserved. This pattern minimizes risk from mistakes or malicious actions (e.g. an angry employee cannot wipe out data because they lack delete rights[[1]](https://www.reddit.com/r/ExperiencedDevs/comments/zd5ami/what_are_the_best_practices_around_developing_rbac/#:~:text=I%20wouldn%27t%20let%20writer%20delete,but%20deletes%20are%20a%20nono)).

In summary, the RBAC model for Eden ERP should use **granular permissions** (especially separating deletes and comments), **clear role names** that indicate scope (Contributor, Viewer, etc.), and **roles tailored to use-cases** (Admin for full control, Ops for task management, Contributor for staff, Viewer for read-only). The new **Contributor role** will fulfill the requirement: *view all tasks, create and edit tasks, comment on tasks, but not delete anything*. This schema is intuitive (maps to what employees actually do), and it’s scalable. As the company grows or the ERP adds modules, you can create new roles or combine roles for department-specific needs without breaking the existing setup. By following the principle of least privilege and referencing proven patterns from platforms like Jira, Notion, ClickUp, and Basecamp, Eden ERP’s access control will remain **secure, easy to understand, and adaptable** to future needs[[3]](https://confluence.atlassian.com/adminjiraserver/managing-project-permissions-938847145.html#:~:text=related%20comments%20and%20attachments%20will,it%E2%80%99s%20only%20given%20to%20administrators)[[5]](https://www.myshyft.com/blog/role-based-access-control-2/#:~:text=several%20key%20practices%3A%20establish%20formal,should%20specifically%20examine%20role%20structures).

**Sources:**

* NocoBase Blog – *How to Design an RBAC System* (Jul 2025) – on defining roles by function vs. org structure[[11]](https://medium.com/@nocobase/how-to-design-an-rbac-role-based-access-control-system-3b57ca9c6826#:~:text=Common%20ways%20to%20define%20roles%3A)[[8]](https://medium.com/@nocobase/how-to-design-an-rbac-role-based-access-control-system-3b57ca9c6826#:~:text=Recommendation%3A%20Keep%20the%20number%20of,for%20a%20type%20of%20user)
* Reddit discussion – *Best practices around developing RBAC* – emphasizes treating “delete” as a special permission[[1]](https://www.reddit.com/r/ExperiencedDevs/comments/zd5ami/what_are_the_best_practices_around_developing_rbac/#:~:text=I%20wouldn%27t%20let%20writer%20delete,but%20deletes%20are%20a%20nono)
* Atlassian Jira Documentation – *Project Permissions* – example of separate Create, Edit, Delete, and Comment permissions[[2]](https://confluence.atlassian.com/adminjiraserver/managing-project-permissions-938847145.html#:~:text=,it%E2%80%99s%20only%20given%20to%20administrators)[[10]](https://confluence.atlassian.com/adminjiraserver/managing-project-permissions-938847145.html#:~:text=Add%20comments)
* Notion Help Center – *Sharing & Permissions* – demonstrates a distinct “Can comment” role vs. edit/view[[9]](https://www.notion.com/help/sharing-and-permissions#:~:text=filters,here%20%E2%86%92)
* Basecamp Help – *User Types and Permissions* – shows that outside collaborators/clients can’t create or delete projects[[4]](https://3.basecamp-help.com/article/75-permissions#:~:text=This%20person%20doesn%E2%80%99t%20work%20at,Access%20Projects)[[16]](https://3.basecamp-help.com/article/75-permissions#:~:text=,place%20it%20in%20the%20trash)
* Shyft (RBAC in scheduling apps) – stresses role names reflecting functions, not individuals[[5]](https://www.myshyft.com/blog/role-based-access-control-2/#:~:text=several%20key%20practices%3A%20establish%20formal,should%20specifically%20examine%20role%20structures).

[[1]](https://www.reddit.com/r/ExperiencedDevs/comments/zd5ami/what_are_the_best_practices_around_developing_rbac/#:~:text=I%20wouldn%27t%20let%20writer%20delete,but%20deletes%20are%20a%20nono) [[12]](https://www.reddit.com/r/ExperiencedDevs/comments/zd5ami/what_are_the_best_practices_around_developing_rbac/#:~:text=to%20accomodate%20this,the%20changing%20the%20existing%20roles) What are the best practices around developing RBAC : r/ExperiencedDevs

<https://www.reddit.com/r/ExperiencedDevs/comments/zd5ami/what_are_the_best_practices_around_developing_rbac/>

[[2]](https://confluence.atlassian.com/adminjiraserver/managing-project-permissions-938847145.html#:~:text=,it%E2%80%99s%20only%20given%20to%20administrators) [[3]](https://confluence.atlassian.com/adminjiraserver/managing-project-permissions-938847145.html#:~:text=related%20comments%20and%20attachments%20will,it%E2%80%99s%20only%20given%20to%20administrators) [[10]](https://confluence.atlassian.com/adminjiraserver/managing-project-permissions-938847145.html#:~:text=Add%20comments) Managing project permissions | Administering Jira applications Data Center 11.1 | Atlassian Documentation

<https://confluence.atlassian.com/adminjiraserver/managing-project-permissions-938847145.html>

[[4]](https://3.basecamp-help.com/article/75-permissions#:~:text=This%20person%20doesn%E2%80%99t%20work%20at,Access%20Projects) [[15]](https://3.basecamp-help.com/article/75-permissions#:~:text=,factor%20authentication) [[16]](https://3.basecamp-help.com/article/75-permissions#:~:text=,place%20it%20in%20the%20trash) Permissions - Basecamp Help

<https://3.basecamp-help.com/article/75-permissions>

[[5]](https://www.myshyft.com/blog/role-based-access-control-2/#:~:text=several%20key%20practices%3A%20establish%20formal,should%20specifically%20examine%20role%20structures) Role-Based Access Control For Mobile Scheduling Security - myshyft.com

<https://www.myshyft.com/blog/role-based-access-control-2/>

[[6]](https://medium.com/@nocobase/how-to-design-an-rbac-role-based-access-control-system-3b57ca9c6826#:~:text=2) [[8]](https://medium.com/@nocobase/how-to-design-an-rbac-role-based-access-control-system-3b57ca9c6826#:~:text=Recommendation%3A%20Keep%20the%20number%20of,for%20a%20type%20of%20user) [[11]](https://medium.com/@nocobase/how-to-design-an-rbac-role-based-access-control-system-3b57ca9c6826#:~:text=Common%20ways%20to%20define%20roles%3A) How to Design an RBAC (Role-Based Access Control) System | by NocoBase | Medium

<https://medium.com/@nocobase/how-to-design-an-rbac-role-based-access-control-system-3b57ca9c6826>

[[7]](https://medium.com/@ajithcrajendran/understanding-the-azure-rbac-structure-7666529209ff#:~:text=Default%20RBAC%20roles) Understanding the Azure RBAC structure | by Ajith Rajendran | Medium

<https://medium.com/@ajithcrajendran/understanding-the-azure-rbac-structure-7666529209ff>

[[9]](https://www.notion.com/help/sharing-and-permissions#:~:text=filters,here%20%E2%86%92) Sharing & permissions – Notion Help Center

<https://www.notion.com/help/sharing-and-permissions>

[[13]](https://stackoverflow.com/questions/32605252/rbac-roles-vs-business-functions#:~:text=London,Temporary) [[14]](https://stackoverflow.com/questions/32605252/rbac-roles-vs-business-functions#:~:text=London,building) RBAC Roles vs Business functions - Stack Overflow

<https://stackoverflow.com/questions/32605252/rbac-roles-vs-business-functions>