Designing Permission and Access States

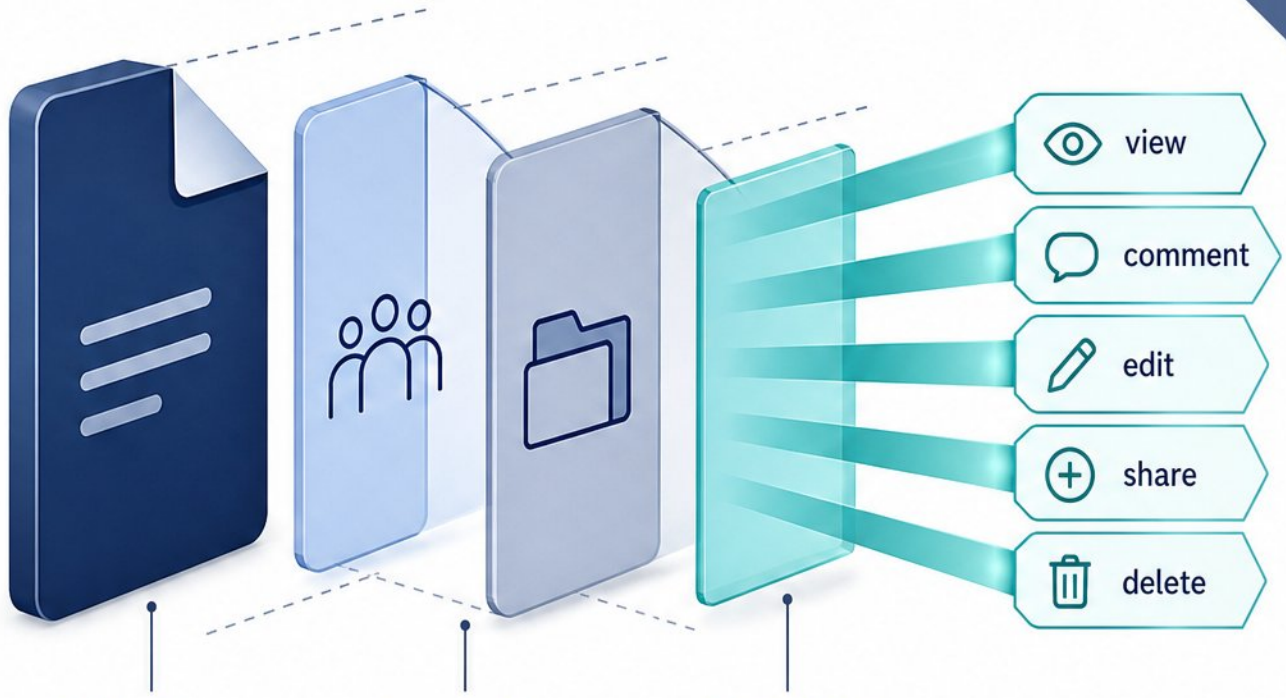
- Learn to clearly communicate what users can see, do, and request
- Good permission design builds trust, safety, and usability
- Shift from engineering permissions to designing guiding experiences
- Core concepts: authentication, authorization, states, and access levels

Why Bad Permissions Break Trust

- Poor design causes frustration, support floods, uninstalls, and security incidents
- "Permission regret": users grant access blindly, later feel misled and unsafe
- Dark patterns destroy trust: repeated prompts, hidden deny buttons, launch-time walls
- BeReal's fatigue-based consent bombards users who decline targeted advertising

Core Vocabulary: The Verbs and Nouns of Access

- Subjects (users), objects (resources), and actions (verbs like view, edit, delete, share)
- Access levels: None, View, Comment, Edit, Admin mapped to CRUD (Create, Read, Update, Delete)
- Standard naming: resource:action pattern (e.g., `reports:view`, `users:invite`)
- Visibility ('can see') vs. capability ('can do'); UI affordances must reflect both



Visibility ('can see')
Determines what is visible to the user.

Capability ('can do')
Determines what actions the user can perform.

UI affordances must reflect both.

The User's Journey:
From Access Denied to Empowered

- Map user lifecycle states: new, active, guest, deactivated—each encounters distinct permission gates.

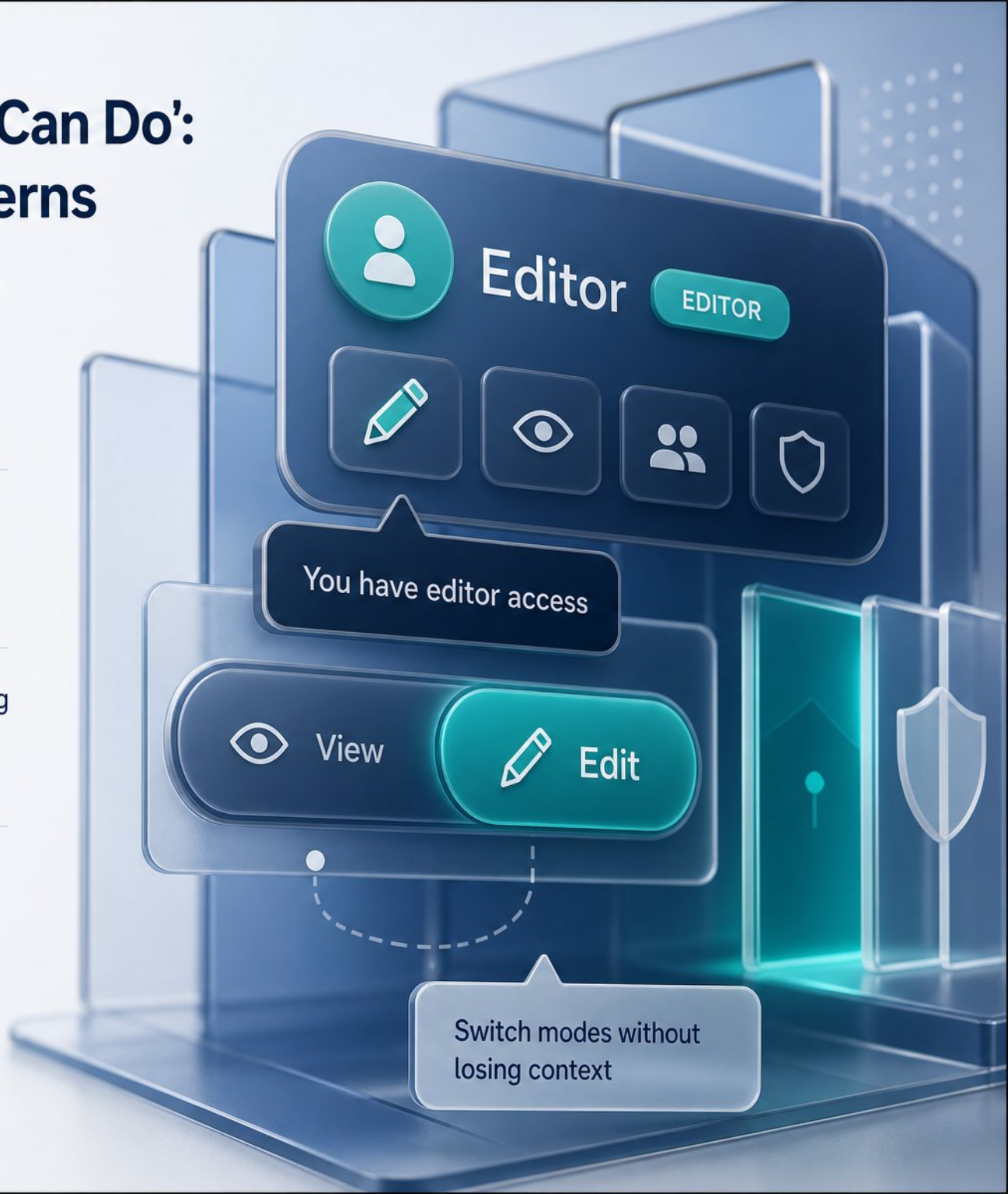
- Design the 'gate' as a conversation starter, not a dead end.

- Use access-denied as a recovery tool: confirm account, explain why in plain language, offer next step.

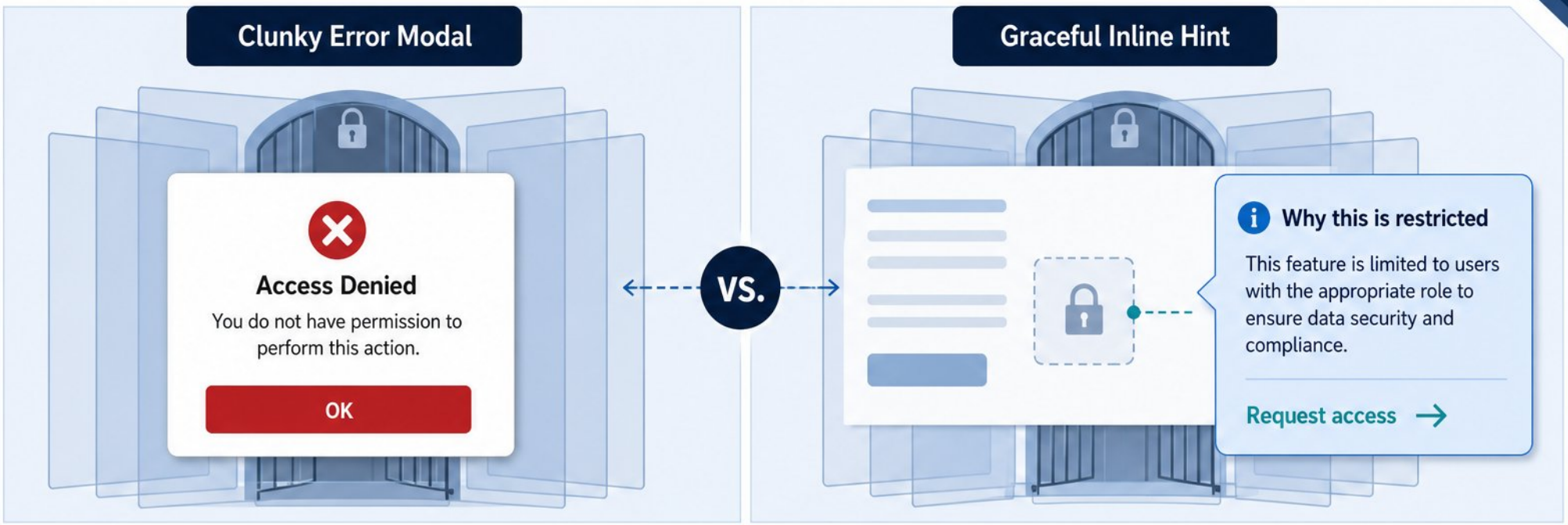
- Progressive disclosure: reveal locked features before they're needed to set expectations and invite upgrade or request.

Communicating the 'Can Do': Visual and Text Patterns

- Use iconography, badges, and tooltips to signal edit, view, or manage access
- Write concise microcopy like ‘You have editor access’ instead of technical jargon
- Design smooth mode switching between view-only and edit states without context loss
- Audit UI hints: cursor changes, disabled states, and never hide critical controls



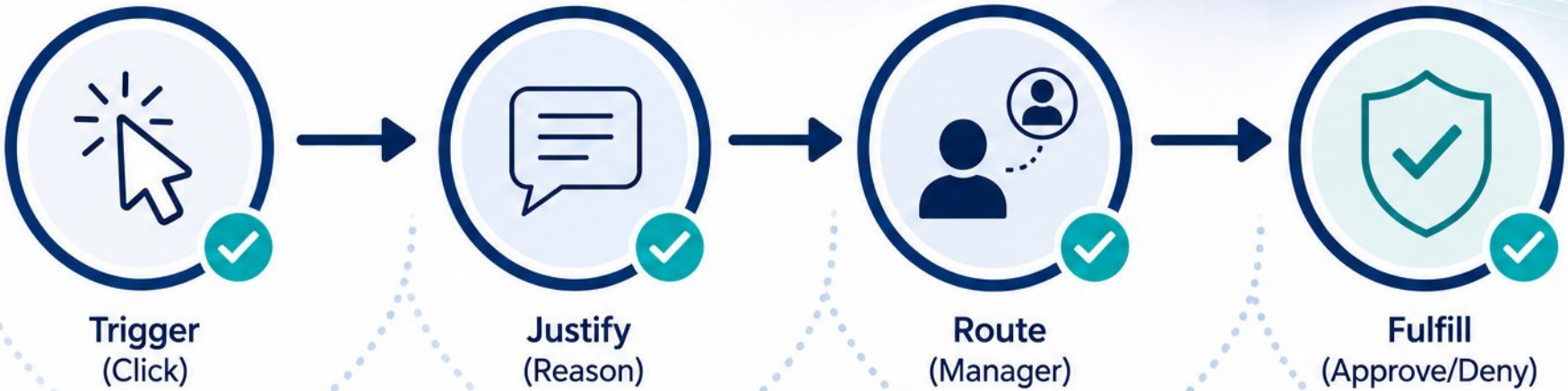
Communicating the 'Cannot Do': Graceful Restrictions



- Distinguish between role-based and state-based restrictions for accurate guidance
- Use non-intrusive tooltips and inline messages instead of disruptive error modals
- Apply the 'Why' pattern: explain why a feature is restricted to reduce frustration
- Keep critical navigation always visible—avoid the 'mystery meat' anti-pattern
- Write neutral, non-blaming access-denied messages with clear next actions

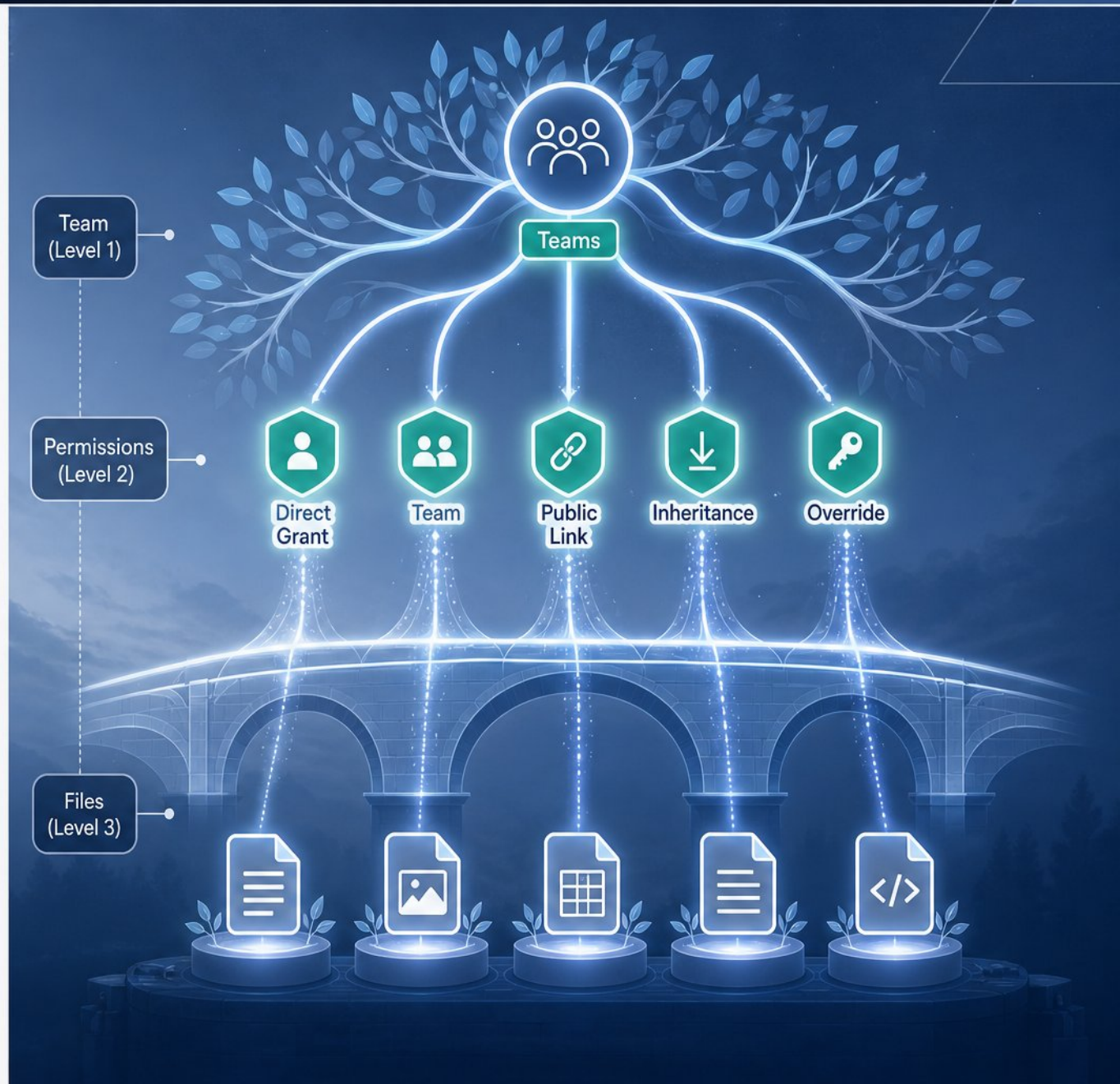
Designing the ‘Request Access’ Flow

- Four components: trigger, justification, routing, and fulfillment
- Auto-capture context: page area, timestamp, reference ID
- Short request form with an optional reason field
- Clear status tracking: pending, approved, denied
- Approver side: one-click approve/deny with reason templates



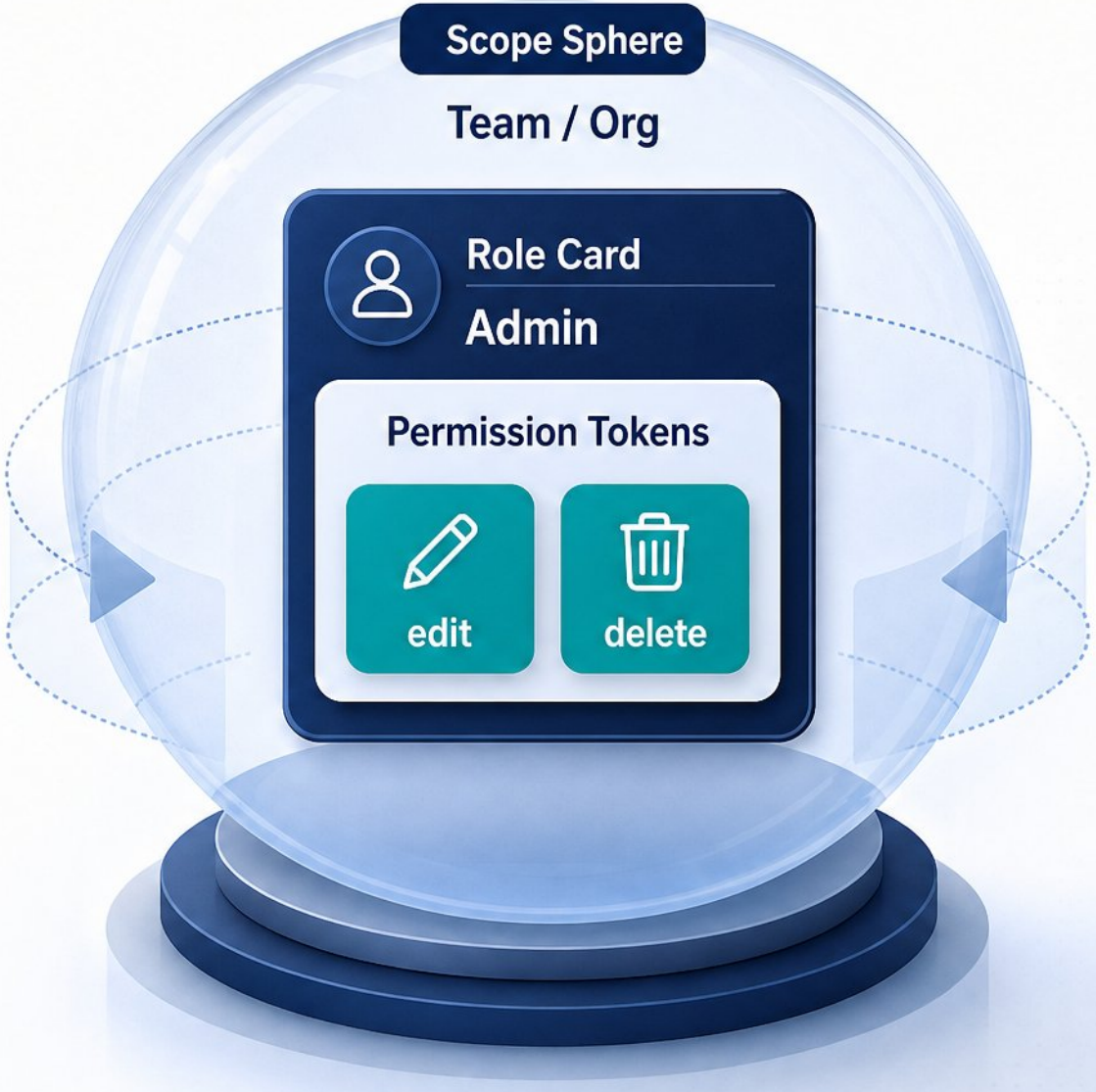
Multi-Level Access: Groups, Inheritance, and Overrides

- **Inheritance in plain language:** folder access flows down to files inside
- **Permission cards aggregate access sources:** direct grant, team, public link, inheritance
- **Conflict resolution rules:** most permissive usually wins; clearly surface effective permissions
- **Visualize access trees with simple badges** and scope labels, never with dense matrix grids



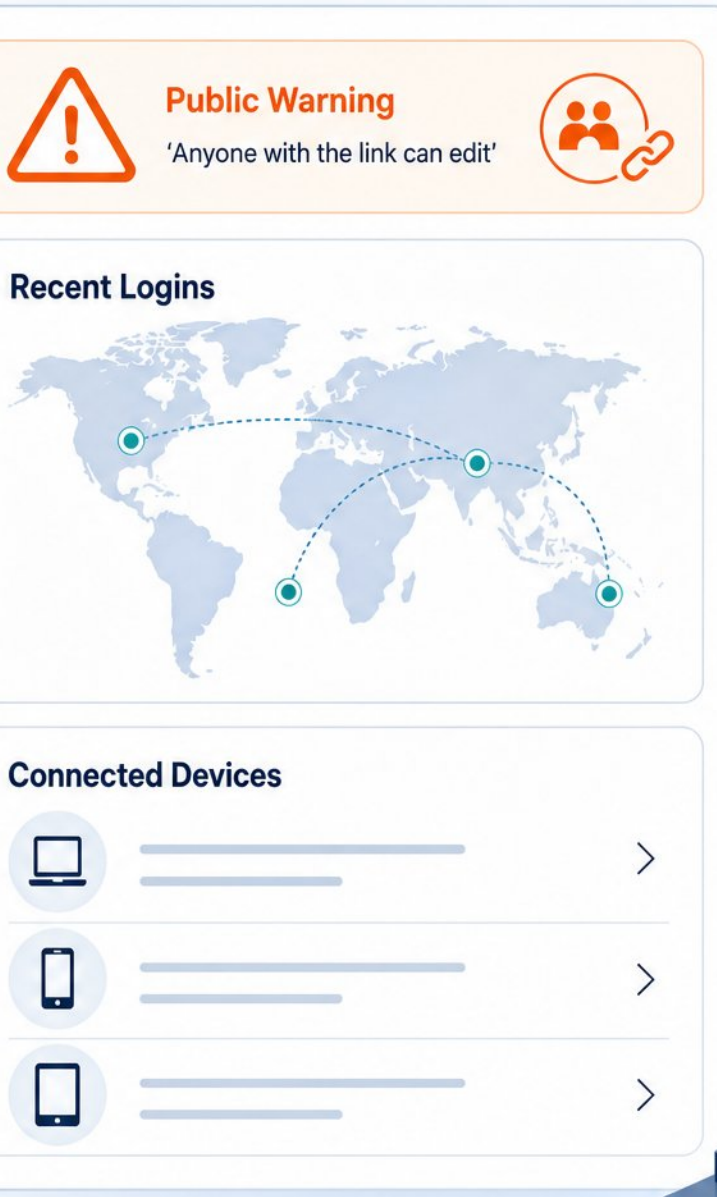
Roles, Permissions, and Scope: A Practical Model

- Roles group permissions (Owner, Admin, Member, Viewer); start with a small, stable set
- Permissions are specific action grants like `reports:view`, not role checks
- Scope answers where permissions apply: own, team, or org-wide resources
- Flat models cause over-permissioning; separate org-level from resource-level roles



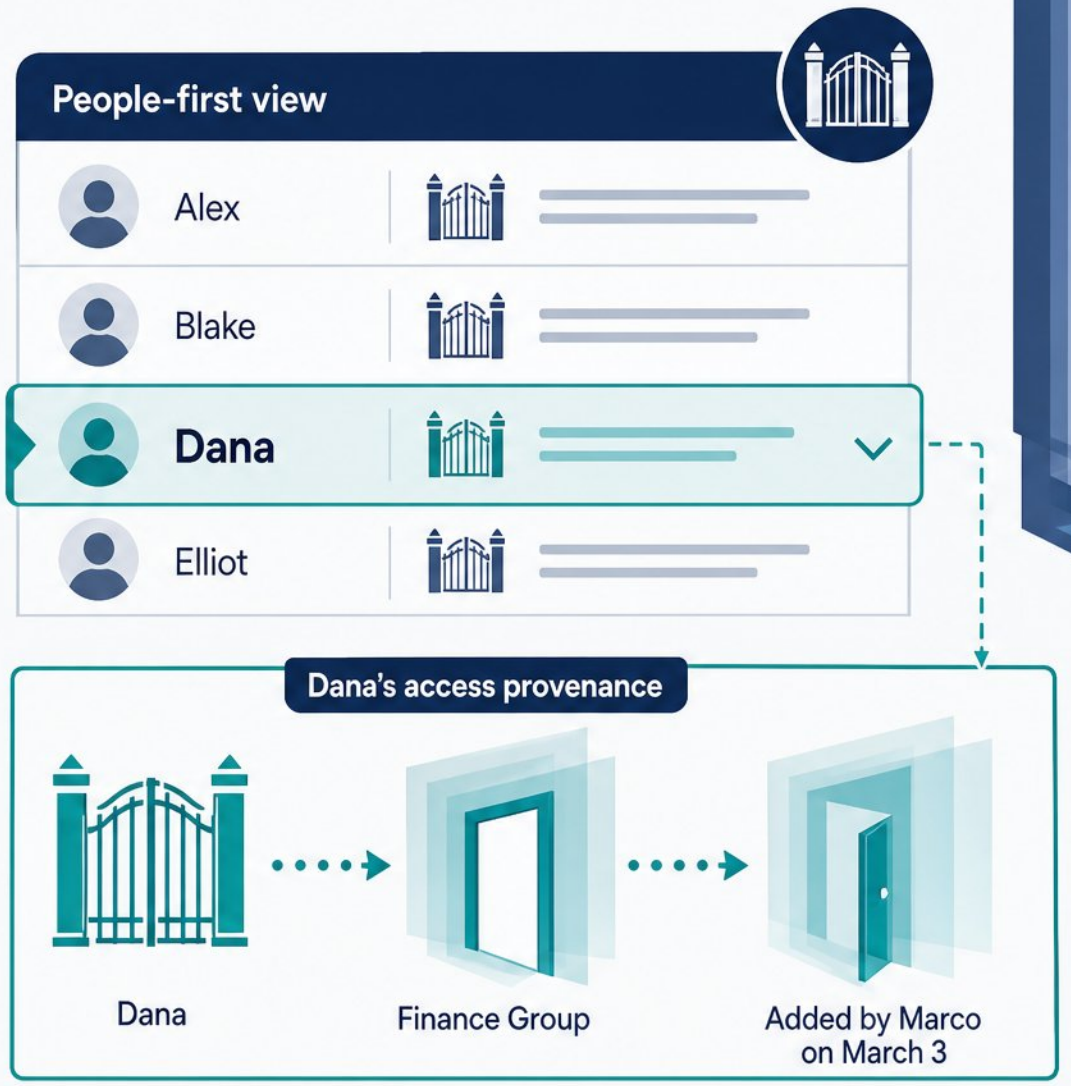
Security Signals and Trust Indicators

- Surface critical info: logins, devices, public visibility without paranoia. [S1, S4]
- Make sharing scope vivid: ‘Anyone with the link can edit’ with icons. [S1, S6]
- Require PIN or re-auth for destructive actions; avoid excessive friction. [S4, S7]
- Escalate warnings for high-risk, inform inline for low-risk to prevent fatigue. [S1, S5]
- Treat geolocation and media metadata as sensitive; disable by default. [S6]



Admin-Side Design: Making Permissions Manageable

- **Lead with people, not matrices:** show user lists with effective access and drill-downs.
- **Always answer:** "What can this person touch?" and "Who can perform this action?"
- **Show before/after states** for edits, and isolate dangerous capabilities.
- **Expose access provenance:** "Dana can export invoices because she's in Finance, added by Marco."



Writing Clear Permission Microcopy

- Structure messages to answer: what happened, why, and what's next
- Replace jargon with plain language (e.g., use "Access denied" not "403")
- Use a neutral tone for denials; be supportive for grants
- Test copy: is it brief, specific, actionable, and consistent?

Practical Strategy: Designing for All States

- For each element, define Owner, Editor, Viewer, Guest, signed-out, and wrong-account states
- Create a permission matrix in design files mapping UI components to roles
- Redesign ambiguous sharing dialogs using role descriptions, scope, and confirmations
- Test permission UX: ask about comprehension, next steps, and trust perception

Owner | Editor | Viewer | Guest | signed-out | wrong-account

UI Component	Owner	Editor	Viewer	Guest	signed-out	wrong-account
View Content	✓	✓	✓	–	–	–
Edit Content	✓	✓	–	–	–	–
Share Content	✓	✓	–	–	–	–
Comment	✓	✓	✓	–	–	–
Download	✓	✓	✓	–	–	–
Delete Content	✓	✓	–	–	–	–

Key Takeaways and Action Plan

- Core framework: roles, permissions, visibility, capability, scope, and recovery paths

- Signals of success: low over-provisioning, fast "who can do what" answers, fewer tickets

- Immediate next step: audit an access-denied page and add a clear "Request Access" path

- Build a simple permission matrix for your next feature

- Resources: toolkit checklists, naming conventions, and admin-side best practices

PersonWise.

PersonWise • AI digital-human course platform

Scan the code or click anywhere to watch this course live, with voice Q&A

personwise.ai/t/5d109de3/public