Build a Modern Client Portal (with PM integration)

You are an expert full-stack engineer + product designer. Build a production-ready **Client Portal** with a clean, modern UI. The portal must integrate with an internal **Project Management (PM)** service (can be implemented in-repo as a microservice) and enforce strict role-based visibility.

Goals (TL;DR)

- **Communities List on sidebar** (multi-community tenants).
- **Auto-create a Project in PM when a client completes onboarding.**
- **Project Tracker** view for each client.
- **Task Raise** (client can create tasks) synced to PM.
- **Client can only see tasks they created in the portal.**
- **PM shows both client-made and internal-made tasks** (with provenance).
- **Responsive layout with left sidebar + top navbar.**
- **Dashboard**: compact, information-dense overview of all relevant data at a glance.

Use a modern, good-looking UI (glassmorphism-lite, subtle shadows, rounded corners, excellent empty states). Accessibility first.

Tech Stack (Frontend-only for now)

- **App**: Next.js 14 (App Router) + TypeScript
- **UI**: Tailwind CSS, shadcn/ui, Radix Primitives, lucide-react icons
- **State/Data**: TanStack Query (server cache), React Hook Form + Zod for client validation
- **Routing**: App Router with parallel routes for modals (e.g., TaskCreateDialog)
- **Auth (placeholder)**: JWT bearer token from Strapi (to be wired later); for now, mock via MSW
- **APIs**: Strapi REST (later GraphQL optional). Create a thin `strapiClient` wrapper.
- **Testing**: Playwright (UX flows) + Vitest/Testing Library

---

Information Architecture

Top-Level Areas

1. **Dashboard** (compact overview)
2. **Projects** → Project Tracker
3. **Tasks** → Client Task Raise + list (client-visible subset)
4. **Communities** (left sidebar list & switcher)
5. **Onboarding** (first-run flow)
6. **Settings** (profile, notifications, API keys)

Navigation

- **Left Sidebar**: tenant switcher (community list), primary nav (Dashboard, Projects, Tasks, Communities, Settings), footer with help & status.
- **Top Navbar**: global search, quick create (Task), notifications, user menu, context breadcrumbs.

Onboarding Flow (use our prior flow as baseline)

> Treat onboarding as a guided wizard collecting: Organization details, Primary contact, Community selection, Team invites, Initial project intent. On completion, fire `OnboardingCompleted` → auto-create a Project in PM and seed a default task list.

Steps: 1. **Org & Contact**: org name, domain, contact name/email, logo. 2. **Community Selection**: choose one or more communities the client belongs to. 3. **Invite Team**: optional invites (emails → role `CLIENT`). 4. **Initial Project Setup**: short form with project name, goal, target dates. 5. **Review & Finish** → trigger project auto-create.

Empty states should guide users to create their first task or visit the project tracker.

Expected API Shapes (for Strapi later)

> These interfaces describe what the frontend expects from Strapi collections. Use them for typing and mock data now. Adjust if your Strapi content-types differ.

```
// Common
export type ID = string;
export type ISODate = string; // e.g., 2025-09-30T10:00:00.000Z

export enum Role { CLIENT = 'CLIENT', INTERNAL = 'INTERNAL', ADMIN = 'ADMIN' }
export enum ProjectStatus { ACTIVE='ACTIVE', ON_HOLD='ON_HOLD',
COMPLETED='COMPLETED', ARCHIVED='ARCHIVED' }
export enum TaskStatus { OPEN='OPEN', IN_PROGRESS='IN_PROGRESS',
BLOCKED='BLOCKED', DONE='DONE', CANCELED='CANCELED' }
export enum TaskSource { CLIENT_PORTAL='CLIENT_PORTAL',
INTERNAL_PM='INTERNAL_PM', IMPORT='IMPORT', API='API' }

export interface UserDTO {
  id: ID; name?: string; email: string; role: Role; avatarUrl?: string;
}
export interface CommunityDTO {
  id: ID; name: string; slug: string;
}
export interface ProjectDTO {
  id: ID; communityId: ID; name: string; status: ProjectStatus; startDate?:
ISODate; targetDate?: ISODate; createdById?: ID;
}
export interface TaskDTO {
  id: ID; projectId: ID; title: string; description?: string; status:
TaskStatus; source: TaskSource; createdById: ID; assigneeId?: ID; createdAt:
ISODate;
}
export interface OnboardingDTO {
  id: ID; userId: ID; communityIds: ID[]; payload: Record<string, unknown>;
completedAt?: ISODate;
}
```

Strapi Collections (planned) - `users` (with role field) - `communities` (name, slug, members via relation) - `projects` (name, community, status, dates, createdBy) - `tasks` (project, title, description, status, source, createdBy, assignee) - `onboardings` (user, communities, payload, completedAt)

Permissions & Visibility (Critical)

- **Client users** can:
- View **only** tasks where `Task.createdById == session.user.id` **OR** tasks explicitly shared to them (`TaskVisibility` extension below, optional).
- Create tasks in communities where they are members.
- View their own projects (community-scoped) and project tracker summary.
- **Internal/Admin** can see all tasks and projects within their communities; Admins can see all tenants.

Optional extension for sharing:

```
model TaskVisibility {
  id      String @id @default(cuid())
  taskId  String
  userId  String
  task    Task   @relation(fields: [taskId], references: [id])
  user    User   @relation(fields: [userId], references: [id])
  @@unique([taskId, userId])
}
```

API enforcement: All list/read endpoints must filter by community membership and the visibility rules above. Add server-side Zod refinements to prevent over-posting.

API Integration (Strapi, mocked for now)

Endpoints (target shapes)

- `GET /api/communities` → `CommunityDTO[]`
- `GET /api/projects?communityId=...` → `ProjectDTO[]`
- `POST /api/projects` → create project (used after onboarding)
- `GET /api/tasks?projectId=...&scope=client|internal` → `TaskDTO[]`
- `POST /api/tasks` → create task (Task Raise)

Visibility Logic (client-side enforcement for now)

- If `session.role === CLIENT`, only display tasks where `createdById === session.user.id` unless a `sharedWith` array exists (future).
- Internal/Admin see all tasks; add filter chips: `All | Client-Made | Internal-Made` using `TaskSource`.

Dev Strategy (no backend yet)

- Use **MSW (Mock Service Worker)** to stub all endpoints above.
- Provide `fixtures/` with realistic datasets across multiple communities; include both client- and internal-made tasks.

UI/UX Requirements (Frontend)

- **Design Language**: sleek, modern, whitespace-efficient. shadcn/ui Cards, Tabs, DataTable. Subtle Framer Motion transitions.
- **Dashboard (compact)** shows, above the fold:
- **My Tasks** (created by me) — top 5, grouped by status (chips).
- **Project Tracker summary** (active projects per community, progress ring, risk badge if deadlines near).
- **Upcoming** (target dates within 14 days) with mini-timeline.
- **Community switcher** chip row.
- **Quick actions**: Raise Task, View Project, Invite teammate.
- **Sidebar**: Communities list (searchable) + primary nav (Dashboard, Projects, Tasks, Communities, Settings). Collapsible on mobile.
- **Top Navbar**: global search (cmd+k), quick add (Task), notifications, user menu, breadcrumbs.
- **Projects**: Tracker page with List + Kanban tabs; progress ring; milestone chips; filters by status and community.
- **Tasks**: Task Raise dialog (title, description, project, attachments placeholder), list with **Provenance** (Client/Internal) and status controls.
- **Communities**: list + details; selecting a community scopes the whole app.
- **Empty States**: friendly + action-oriented. Skeleton loaders.
- **Accessibility**: keyboard nav, focus states, color contrast AA.

Pages & Components (Next.js, frontend only)

```
/app
  /(protected)
    /dashboard (compact cards)
    /projects/[projectId]
    /projects
    /tasks
    /communities
    /onboarding (wizard UI only; emits event to create project)
    /settings
/components
  Sidebar.tsx (communities, nav)
  Topbar.tsx (search, quick add, user menu)
  CommunitySwitcher.tsx
  ProjectTracker.tsx
  TaskList.tsx
  TaskCreateDialog.tsx
  KanbanBoard.tsx
  ProgressRing.tsx
  StatusBadge.tsx
  EmptyState.tsx
  DataTable.tsx
  ScopeGuard.tsx (role-aware wrappers)
/lib
  strapiClient.ts (fetch wrapper)
  api.ts (typed calls)
  session.ts (mocked for now)
  visibility.ts (filters)
  msw/handlers.ts (mock endpoints)
```

Key Implementation Notes (Frontend)

- **Community scoping** via `CommunityContext` persisted to URL and localStorage. All queries include `communityId`.
- **Visibility**: Inject client-side filter for CLIENT role; show provenance via `Task.source` badges. Internal users get scope filter chips.
- **Onboarding**: Wizard captures org/contact/community/project; on finish, call `POST /api/projects` (mocked) and redirect to project page with a toast.
- **Optimistic UX**: On task create, update TaskList immediately; rollback on error.
- **Error & Empty states**: Consistent toasts and banners.
- **Future**: Swap MSW to real Strapi by flipping an env flag and providing the base URL + auth token.

Seed Scripts

Create seeds for: - Communities (2–3 demo tenants) - Internal users + client users - One project per community (auto-created) - Mixed tasks (client vs internal) to test visibility.

Acceptance Criteria (Frontend)

1. Completing onboarding triggers **create project** call and redirects to Project Tracker with a seeded kickoff task (from mocks).
2. **CLIENT** user sees only tasks they created; can raise a task which appears with `source = CLIENT_PORTAL`.
3. **INTERNAL** user can view both client- and internal-made tasks; can filter `All | Client-Made | Internal-Made`.
4. Sidebar lists **Communities**; switching scopes data everywhere.
5. **Dashboard** surfaces compact key info above the fold.
6. Responsive: great on desktop & mobile (collapsible sidebar, sticky topbar).
7. API layer typed to the interfaces; MSW handlers mirror Strapi endpoints.

Developer Tasks Checklist (Frontend)

- [ ] Scaffold Next.js + Tailwind + shadcn/ui + TanStack Query.
- [ ] Implement `strapiClient` and typed API calls (REST).
- [ ] Add MSW handlers and realistic fixtures.
- [ ] Build Onboarding wizard; wire project creation + redirect.
- [ ] Sidebar with Communities; Top navbar with quick add.
- [ ] Dashboard cards, Project tracker (List + Kanban), Tasks CRUD (frontend).
- [ ] Role visibility filters; provenance badges.
- [ ] Playwright e2e for acceptance criteria.

Stretch (if time allows)

- Real-time hints via polling or SSE once Strapi is live.
- File attachments UI with pre-signed URL flow stubbed.
- Notifications center + email templates (front-end views).
- Metrics panel on Project (burndown mock).

Deliverables: Frontend-only Next.js app with mocks, typed API layer, Playwright tests, and a README explaining how to switch to Strapi.