Frontend Product Requirements Document

Task Management & Collaboration Platform

Prepared for: Google Jules (Antigravity)

Document Version: 1.0

Date: February 2026

Frontend Stack	Next.js 15 (App Router) | JavaScript | Tailwind CSS | shadcn/ui | Axios | TanStack React Query
Backend API Base URL	`http://localhost:8080/api` (dev) | configurable via `NEXT_PUBLIC_API_BASE_URL`
Design Style	Modern, minimal, sleek — neutral palette (slate/gray) with blue accent
Deployment Target	Vercel
Auth	No auth in v1 — user is selected from dropdown as 'active user' to simulate session

1. Project Overview

This document defines all frontend requirements for the Task Management & Collaboration Platform. It is written specifically for an AI coding agent (Google Jules / Antigravity) and must be followed precisely. The agent should generate a complete, working Next.js application based solely on the specifications below.

1.1 What the App Does

- Users can create and manage projects
- Each project has an owner and team members
- Tasks are created inside projects and assigned to members
- Tasks have statuses (TODO, IN_PROGRESS, DONE) and priorities (LOW, MEDIUM, HIGH, URGENT)
- Team members can comment on tasks
- Labels can be created and applied to tasks for categorisation
- All data is persisted in the Spring Boot backend via REST API

1.2 Guiding Design Principles

- Minimal — no clutter, only what is needed is shown
- Functional — every UI element has a clear purpose
- Fast — use React Query caching; show skeletons not blank screens
- Consistent — use shadcn/ui component library everywhere
- Responsive — works on 1280px+ desktop and 375px+ mobile

1.3 Colour & Typography Tokens

Token	Value	Usage
Primary	#1D4ED8	Buttons, links, active states, badges
Background	#F8FAFC	Page background
Surface	#FFFFFF	Cards, modals, sidebars
Border	#E5E7EB	All borders and dividers

Text Primary	#111827	Headings, important labels
Text Secondary	#6B7280	Subtitles, metadata, placeholders
Danger	#DC2626	Delete actions, error states
Success	#16A34A	Done status, success toasts
Warning	#D97706	IN_PROGRESS status, medium priority
Font	Inter (Google Fonts)	All text throughout the app

2. Project Structure

The application uses the Next.js 15 App Router. All code lives under src/. Strict folder conventions must be followed.

```
taskmanager-frontend/
├── public/
│   └── favicon.ico
├── src/
│   ├── app/                          # Next.js App Router pages
│   │   ├── layout.js                 # Root layout (providers, navbar)
│   │   ├── page.js                   # Dashboard / home redirect
│   │   ├── globals.css               # Tailwind base + custom CSS vars
│   │   ├── users/
│   │   │   └── page.js               # User management page
│   │   ├── projects/
│   │   │   ├── page.js               # Projects list page
│   │   │   └── [id]/
│   │   │       └── page.js           # Project detail page
│   │   ├── tasks/
│   │   │   └── [id]/
│   │   │       └── page.js           # Task detail page
│   │   └── labels/
│   │       └── page.js               # Labels management page
│   ├── components/
│   │   ├── ui/                       # shadcn/ui (auto-generated, do not
edit)
│   │   ├── layout/
│   │   │   ├── Navbar.jsx
│   │   │   ├── Sidebar.jsx
│   │   │   └── ActiveUserSelector.jsx
│   │   ├── users/
│   │   │   ├── UserTable.jsx
│   │   │   ├── UserForm.jsx
│   │   │   └── UserRoleBadge.jsx
│   │   ├── projects/
│   │   │   ├── ProjectCard.jsx
│   │   │   ├── ProjectGrid.jsx
│   │   │   ├── ProjectForm.jsx
│   │   │   ├── ProjectHeader.jsx
│   │   │   └── MemberList.jsx
```

```
│   │   ├── tasks/
│   │   │   ├── TaskCard.jsx
│   │   │   ├── TaskBoard.jsx        # Kanban columns
│   │   │   ├── TaskForm.jsx
│   │   │   ├── TaskDetail.jsx
│   │   │   ├── TaskStatusBadge.jsx
│   │   │   └── TaskPriorityBadge.jsx
│   │   ├── comments/
│   │   │   ├── CommentList.jsx
│   │   │   ├── CommentItem.jsx
│   │   │   └── CommentForm.jsx
│   │   ├── labels/
│   │   │   ├── LabelBadge.jsx
│   │   │   ├── LabelSelector.jsx
│   │   │   └── LabelForm.jsx
│   │   └── shared/
│   │       ├── PageHeader.jsx       # Consistent page title + actions bar
│   │       ├── EmptyState.jsx       # Illustrated empty states
│   │       ├── LoadingSkeleton.jsx  # Skeleton loaders
│   │       ├── ConfirmDialog.jsx    # Delete confirmation
│   │       └── ErrorBanner.jsx      # Inline error display
│   ├── lib/
│   │   ├── api/
│   │   │   ├── client.js            # Axios instance
│   │   │   ├── users.js
│   │   │   ├── projects.js
│   │   │   ├── tasks.js
│   │   │   ├── comments.js
│   │   │   └── labels.js
│   │   ├── hooks/
│   │   │   ├── useUsers.js
│   │   │   ├── useProjects.js
│   │   │   ├── useTasks.js
│   │   │   ├── useComments.js
│   │   │   └── useLabels.js
│   │   └── utils.js                 # formatDate, cn(), constants
│   └── providers/
│       └── QueryProvider.jsx        # TanStack Query + Toaster
├── .env.local
├── components.json                  # shadcn config
├── next.config.js
├── tailwind.config.js
```

```
└── package.json
```

3. Design System & Component Rules

3.1 Layout Shell

Every page uses a fixed top navbar and a fixed left sidebar. Page content scrolls inside the remaining space.

Zone	Description
Navbar (top, h-14)	Logo left | App name | Active user selector right
Sidebar (left, w-56)	Navigation links: Dashboard, Projects, Users, Labels
Content area	Padded container (max-w-7xl, px-6, py-8), scrollable

3.2 Navbar — Navbar.jsx

- Fixed top, height 56px, white background, bottom border (border-b border-gray-200)
- Left: small blue square icon + 'TaskManager' text in font-semibold
- Right: ActiveUserSelector component (dropdown of all users — simulates logged-in user)
- The selected user's ID is stored in React context and used as authorId / actorId in all POST/PATCH requests

3.3 Sidebar — Sidebar.jsx

- Fixed left, width 224px, white background, right border
- Navigation items with lucide-react icons:
- LayoutDashboard — Dashboard (/)
- FolderKanban — Projects (/projects)
- Users — Users (/users)
- Tag — Labels (/labels)
- Active link: bg-blue-50 text-blue-700 font-medium, left blue bar (border-l-2 border-blue-700)
- Inactive: text-gray-600 hover:bg-gray-50

3.4 shadcn/ui Components to Install

Component	Used In
button	All action buttons
input	All text inputs
label	Form field labels
textarea	Description fields
select	Role, status, priority, assignee dropdowns
dialog	Create/Edit modals
table	Users table
card	Project cards, task cards
badge	Role badges, status badges, label chips
toast / toaster	Success and error notifications
dropdown-menu	Actions menus (edit/delete)
separator	Section dividers
avatar	User avatars (initials fallback)
skeleton	Loading placeholder states
scroll-area	Scrollable comment list
popover	Label selector popover on task detail
command	Searchable member/label picker
tooltip	Icon button tooltips

3.5 Status & Priority Badge Colours

Value	Badge Colour Class
TODO	bg-slate-100 text-slate-700
IN_PROGRESS	bg-amber-100 text-amber-700
DONE	bg-green-100 text-green-700
LOW	bg-slate-100 text-slate-600
MEDIUM	bg-amber-100 text-amber-700

HIGH	bg-red-100 text-red-700
ADMIN	bg-purple-100 text-purple-700
MEMBER	bg-blue-100 text-blue-700
VIEWER	bg-gray-100 text-gray-600

3.6 Shared Components Spec

PageHeader.jsx

- Props: title (string), subtitle (string, optional), actions (ReactNode, optional)
- Renders: large h1 title, muted subtitle below, actions slot on far right
- Adds a bottom separator below the header block

EmptyState.jsx

- Props: icon (lucide component), title, description, action (ReactNode, optional)
- Centred layout with large muted icon, title, description, optional CTA button
- Used when lists are empty (no projects, no tasks, etc.)

LoadingSkeleton.jsx

- Exports: TableSkeleton, CardGridSkeleton, DetailSkeleton
- Use shadcn Skeleton component with appropriate height/width placeholders

ConfirmDialog.jsx

- Props: open, onOpenChange, title, description, onConfirm, loading
- Uses shadcn Dialog with a red destructive Confirm button
- Always used before delete operations

4. Pages Specification

4.1 Dashboard — /

The landing page. Provides a quick stats overview and shortcuts to recent projects.

Layout

- PageHeader: title='Dashboard', subtitle='Overview of your workspace'
- Stats row: 4 stat cards in a grid (grid-cols-4)
- Below: 2-column grid — recent projects (left) + quick task summary (right)

Stat Cards (4 cards)

Card	Data Source	Icon
Total Projects	GET /api/projects — count array length	FolderKanban
Total Tasks	GET /api/tasks — count array length	CheckSquare
Tasks In Progress	GET /api/tasks — filter status=IN_PROGRESS	Clock
Total Users	GET /api/users — count array length	Users

Recent Projects Section

- Title: 'Recent Projects' | Link: 'View all' (navigates to /projects)
- Shows last 4 projects (slice from projects array)
- Each row: project title, owner name, task count badge, created date
- Click on project title navigates to /projects/[id]

My Tasks Section

- Title: 'My Tasks' — tasks where assignee.id === activeUser.id
- Shows up to 5 tasks
- Each row: task title, status badge, priority badge, project name
- Click navigates to /tasks/[id]

4.2 Projects List — /projects

Displays all projects as a card grid. Supports filtering and project creation.

Layout

- PageHeader: title='Projects' | action button: 'New Project' (opens create dialog)
- Filter bar below header: search input (filter by title client-side) + no other filters needed
- ProjectGrid component: responsive grid (grid-cols-1 sm:grid-cols-2 lg:grid-cols-3)
- Each cell: ProjectCard component
- Empty state if no projects: EmptyState with FolderPlus icon

ProjectCard.jsx

- White card with subtle shadow and hover:shadow-md transition
- Card top: coloured accent bar (5px, blue)
- Title: font-semibold text-gray-900
- Description: 2-line clamp, text-sm text-gray-500
- Meta row: owner avatar + name | member count (Users icon) | task count (CheckSquare icon)
- Footer: start date – end date | Actions dropdown (Edit, Delete) — top-right of card
- Click on card body (not dropdown) navigates to /projects/[id]

ProjectForm.jsx (Create / Edit Modal)

- Dialog title: 'Create Project' or 'Edit Project'
- Fields:
- Title — Input, required
- Description — Textarea, optional
- Owner — Select of all users (shows user name)
- Start Date — Input type='date'
- End Date — Input type='date'
- Buttons: Cancel | Save (primary blue)
- On submit: POST /api/projects or PUT /api/projects/{id}

API Calls for This Page

Action	Method + URL
Load projects	`GET /api/projects`
Create project	`POST /api/projects`
Update project	`PUT /api/projects/{id}`
Delete project	`DELETE /api/projects/{id}`

4.3 Project Detail — /projects/[id]

The richest page. Shows project info, member management, and the task Kanban board.

Layout (top to bottom)

- ProjectHeader component (top)
- Two-tab layout: 'Board' tab (default) | 'Members' tab
- Board tab: TaskBoard component
- Members tab: MemberList component

ProjectHeader.jsx

- Project title (text-2xl font-bold) + Edit button (pencil icon, icon-only)
- Description in muted text below title
- Meta chips row: owner avatar+name | start–end dates | member count | task count
- 'Add Task' button (primary) on the right — opens TaskForm dialog

TaskBoard.jsx — Kanban Board

- Three columns side by side: TODO | IN_PROGRESS | DONE
- Column header: status label + count badge (e.g. 'TODO 3')
- Each column: vertical stack of TaskCard components
- Empty column: small dashed placeholder box
- '+ Add Task' text button at bottom of TODO column

TaskCard.jsx

- White card, rounded-lg, border, hover:shadow-sm, cursor-pointer
- Top row: priority badge (left) | actions dropdown (right) — Edit, Delete
- Task title: font-medium text-sm
- Labels row: coloured LabelBadge chips (if any)
- Footer: assignee avatar+name (left) | due date with CalendarIcon (right, red if overdue)
- Click on card (not dropdown) navigates to /tasks/[taskId]

MemberList.jsx

- Section title: 'Team Members' + 'Add Member' button
- Table: Avatar | Name | Email | Role badge | Remove button
- Remove button: only visible if activeUser is owner or ADMIN
- Add Member: opens Command popover — searchable list of all users, click to add

API Calls for This Page

Action	Method + URL
Load project details	`GET /api/projects/{id}`
Load project tasks	`GET /api/projects/{id}/tasks`
Load project members	`GET /api/projects/{id}/members`
Create task	`POST /api/projects/{id}/tasks`
Add member	`POST /api/projects/{id}/members/{userId}`
Remove member	`DELETE /api/projects/{id}/members/{userId}`
Update project	`PUT /api/projects/{id}`
Delete project	`DELETE /api/projects/{id}`
Delete task	`DELETE /api/tasks/{id}`

4.4 Task Detail — /tasks/[id]

Full-page view of a single task with all metadata, label management, and comments.

Layout — Two Column (lg:grid-cols-3)

- Left column (2/3 width): task title, description, comments section
- Right column (1/3 width): metadata panel (status, priority, assignee, labels, dates)

Left Column

- Back link: '← Back to project' (navigates to /projects/[task.project.id])
- Task title: text-2xl font-bold, editable inline (click to edit, blur to save)
- Description: rendered as plain text, 'Edit' button opens textarea
- Separator
- Comments section heading: 'Comments' + count
- CommentList: scrollable list of CommentItem
- CommentForm: textarea + 'Post Comment' button at bottom

Right Column — Metadata Panel

- White card with padding, sticky top-8
- Each field row: label (text-xs uppercase text-gray-400) + value below
- Status: Select (TODO / IN_PROGRESS / DONE) — saves on change via PATCH /api/tasks/{id}/status
- Priority: Select (LOW / MEDIUM / HIGH / URGENT) — saves on update
- Assignee: Select of project members — saves via PATCH /api/tasks/{id}/assign/{userId}
- Labels: chips with X to remove + Popover button to add more
- Due Date: date display — red if past due
- Project: link to /projects/[id]
- Created At: formatted date

CommentItem.jsx

- Avatar (initials) + author name + relative timestamp (e.g. '2 hours ago')
- Comment content text
- Edit / Delete icons — only visible if comment.author.id === activeUser.id

- Edit: inline textarea replaces content, Save / Cancel buttons

API Calls for This Page

Action	Method + URL
Load task details	`GET /api/tasks/{id}`
Update task	`PUT /api/tasks/{id}`
Update status	`PATCH /api/tasks/{id}/status`
Assign user	`PATCH /api/tasks/{id}/assign/{userId}`
Add label	`POST /api/tasks/{taskId}/labels/{labelId}`
Remove label	`DELETE /api/tasks/{taskId}/labels/{labelId}`
Load comments	`GET /api/tasks/{id}/comments`
Post comment	`POST /api/tasks/{id}/comments`
Edit comment	`PUT /api/comments/{id}`
Delete comment	`DELETE /api/comments/{id}`
Delete task	`DELETE /api/tasks/{id}`

4.5 Users — /users

Admin-style page to view and manage all platform users.

Layout

- PageHeader: title='Users' | action: 'New User' button
- Search input to filter table by name or email (client-side)
- UserTable component

UserTable.jsx

- Columns: Avatar | Name | Email | Role | Created | Actions
- Avatar: circle with user initials, bg-blue-100 text-blue-700
- Role: UserRoleBadge (coloured badge)
- Created: formatted date
- Actions column: Edit icon button | Delete icon button
- Delete triggers ConfirmDialog before calling DELETE /api/users/{id}

UserForm.jsx

- Dialog: 'Create User' or 'Edit User'
- Fields: Name (Input, required) | Email (Input type=email, required) | Role (Select)
- Role options: ADMIN | MEMBER | VIEWER
- On create: POST /api/users
- On edit: PUT /api/users/{id}

API Calls for This Page

Action	Method + URL
Load users	`GET /api/users`
Create user	`POST /api/users`
Update user	`PUT /api/users/{id}`
Delete user	`DELETE /api/users/{id}`

4.6 Labels — /labels

Manage reusable labels that can be applied to any task.

Layout

- PageHeader: title='Labels' | action: 'New Label' button
- Responsive grid of label cards (grid-cols-2 md:grid-cols-3 lg:grid-cols-4)

Label Card

- White card, left coloured border (4px, label.color)
- Label name in font-medium
- Colour swatch circle showing label.color
- Edit icon + Delete icon (top right)

LabelForm.jsx

- Dialog: 'Create Label' or 'Edit Label'
- Fields: Name (Input, required) | Color (Input type=color + hex text input)
- Preview chip showing name on selected colour

API Calls for This Page

Action	Method + URL
Load labels	`GET /api/labels`
Create label	`POST /api/labels`
Delete label	`DELETE /api/labels/{id}`

5. Full API Reference

Backend base URL: NEXT_PUBLIC_API_BASE_URL (default: http://localhost:8080/api)

All request/response bodies are JSON. Content-Type: application/json on all POST/PUT/PATCH.

5.1 Users API

Method	URL	Request Body	Response
GET	/api/users	—	Array of UserResponse
GET	/api/users/{id}	—	UserResponse
POST	/api/users	{ name, email, role }	201 UserResponse
PUT	/api/users/{id}	{ name, email, role }	UserResponse
DELETE	/api/users/{id}	—	204 No Content

UserResponse shape

```
{ id, name, email, role, createdAt }
```

5.2 Projects API

Method	URL	Request Body	Response
GET	/api/projects	—	Array of ProjectResponse
GET	/api/projects/{id}	—	ProjectResponse
POST	/api/projects	{ title, description, ownerId, startDate, endDate }	201 ProjectResponse
PUT	/api/projects/{id}	{ title, description, ownerId,	ProjectResponse

		startDate, endDate }	
DELETE	/api/projects/{id}	—	204 No Content
POST	/api/projects/{id}/members/{userId}	—	200 OK
DELETE	/api/projects/{id}/members/{userId}	—	200 OK
GET	/api/projects/{id}/members	—	Array of UserResponse
GET	/api/projects/{id}/tasks	—	Array of TaskResponse

ProjectResponse shape

```
{ id, title, description, startDate, endDate, createdAt,
  owner: { id, name, email, role },
  members: [ UserResponse ],
  tasks: [ TaskResponse ] }
```

5.3 Tasks API

Method	URL	Request Body	Response
GET	/api/tasks	—	Array of TaskResponse
GET	/api/tasks/{id}	—	TaskResponse
GET	/api/projects/{id}/tasks	—	Array of TaskResponse
POST	/api/projects/{id}/tasks	{ title, description, status, priority, dueDate, assigneeId }	201 TaskResponse
PUT	/api/tasks/{id}	{ title, description, status, priority, dueDate, assigneeId }	TaskResponse
PATCH	/api/tasks/{id}/status	{ status }	TaskResponse

PATCH	/api/tasks/{id}/assign/{userId}	—	TaskResponse
DELETE	/api/tasks/{id}	—	204 No Content
POST	/api/tasks/{taskId}/labels/{labelId}	—	200 OK
DELETE	/api/tasks/{taskId}/labels/{labelId}	—	200 OK

TaskResponse shape

```
{ id, title, description, status, priority, dueDate, createdAt,
  project: { id, title },
  assignee: { id, name, email } | null,
  labels: [ LabelResponse ],
  comments: [ CommentResponse ] }
```

5.4 Comments API

Method	URL	Request Body	Response
GET	/api/tasks/{id}/comments	—	Array of CommentResponse
POST	/api/tasks/{id}/comments	{ content, authorId }	201 CommentResponse
PUT	/api/comments/{id}	{ content }	CommentResponse
DELETE	/api/comments/{id}	—	204 No Content

CommentResponse shape

```
{ id, content, createdAt,
  author: { id, name, email },
  task: { id, title } }
```

5.5 Labels API

Method	URL	Request Body	Response
GET	/api/labels	—	Array of LabelResponse

POST	/api/labels	{ name, color }	201 LabelResponse
DELETE	/api/labels/{id}	—	204 No Content

LabelResponse shape

```
{ id, name, color }
```

6. Data Layer — API & React Query

6.1 Axios Client — src/lib/api/client.js

- Create axios instance with baseURL from NEXT_PUBLIC_API_BASE_URL
- Default header: Content-Type: application/json
- Response interceptor: log errors, extract error.response.data.message for toast

6.2 React Query Configuration

- staleTime: 60 * 1000 (1 minute)
- refetchOnWindowFocus: false
- Wrap app in QueryClientProvider inside QueryProvider.jsx
- QueryProvider must be marked 'use client'

6.3 Query Key Conventions

Data	Query Key
All users	`['users']`
Single user	`['users', id]`
All projects	`['projects']`
Single project	`['projects', id]`
Project tasks	`['projects', id, 'tasks']`
Project members	`['projects', id, 'members']`
All tasks	`['tasks']`
Single task	`['tasks', id]`
Task comments	`['tasks', id, 'comments']`
All labels	`['labels']`

6.4 Mutation Patterns

- Every mutation must call queryClient.invalidateQueries on success
- Every mutation must call toast({ title: 'Error', description, variant: 'destructive' }) on error
- Show success toast after create/update/delete
- Disable submit buttons while mutation is pending (use mutation.isPending)

7. Active User Simulation (No Auth)

Since Spring Security is not yet implemented, the frontend simulates a logged-in user via a global context.

7.1 ActiveUserContext

- Create src/providers/ActiveUserContext.jsx
- Fetches all users from GET /api/users on mount
- Stores selectedUser in state (default: first user in list)
- Exposes: { activeUser, setActiveUser } via useContext

7.2 ActiveUserSelector.jsx

- Shown in Navbar top-right
- Dropdown (shadcn Select or DropdownMenu) listing all users with avatar + name
- On change: updates activeUser in context
- Displays current user avatar + name in closed state

7.3 Usage in API Calls

Action	Where activeUser.id is used
Post a comment	authorId in POST /api/tasks/{id}/comments body
Create a project	ownerId in POST /api/projects body
Authorization checks	Compare with project.owner.id or comment.author.id before showing edit/delete controls

8. Forms & Validation Rules

All forms use controlled React state (no external form library required). Validate on submit.

Form	Field	Validation
UserForm	name	Required, min 3 chars, max 100 chars
UserForm	email	Required, must match email regex
UserForm	role	Required, one of: ADMIN, MEMBER, VIEWER
ProjectForm	title	Required, min 3 chars, max 100 chars
ProjectForm	description	Optional, max 1000 chars
ProjectForm	ownerId	Required, must select a user
ProjectForm	startDate	Optional
ProjectForm	endDate	Optional, must be after startDate if both set
TaskForm	title	Required, min 3 chars, max 200 chars
TaskForm	description	Optional, max 2000 chars
TaskForm	status	Required, default: TODO
TaskForm	priority	Required, default: MEDIUM
TaskForm	dueDate	Optional
TaskForm	assigneeId	Optional, must be a project member if set
CommentForm	content	Required, min 1 char, max 1000 chars
LabelForm	name	Required, min 2 chars, max 30 chars
LabelForm	color	Optional, must be valid hex (#RRGGBB)

Show inline validation errors below each field in text-sm text-red-500.

Disable submit button while mutation is pending.

Reset form state on dialog close.

9. Error & Loading States

9.1 Loading States

Context	Component to Use
Page loading (table)	TableSkeleton — 5 rows of gray skeleton bars
Page loading (cards)	CardGridSkeleton — 6 card-shaped skeletons
Detail page loading	DetailSkeleton — title + side panel placeholders
Button submitting	Spinner icon inside button + disabled
Inline data loading	Small Skeleton inline

9.2 Error States

- If a page-level query fails: show ErrorBanner with error message and 'Retry' button
- If a mutation fails: show destructive toast with backend error message
- If a resource is not found (404): show EmptyState with 'Not Found' message
- Network error: show toast 'Could not reach server. Is the backend running?'

9.3 Toast Specification

- Success: title='Success', description='[Entity] [action] successfully', no variant
- Error: title='Error', description=error.response?.data?.message, variant='destructive'
- Position: bottom-right, auto-dismiss after 4 seconds

10. Utilities — src/lib/utils.js

Utility	Description
cn(...inputs)	clsx + tailwind-merge for conditional classNames (already in shadcn)
formatDate(dateStr)	Format ISO date to 'Jan 15, 2026' using Intl.DateTimeFormat
formatRelativeTime(dateStr)	Format to relative time e.g. '2 hours ago' using Intl.RelativeTimeFormat
getInitials(name)	Extract initials from full name e.g. 'John Doe' -> 'JD'
isOverdue(dueDateStr)	Return true if dueDate is in the past and status !== DONE
getStatusColor(status)	Return Tailwind class string for status badge
getPriorityColor(priority)	Return Tailwind class string for priority badge
getRoleColor(role)	Return Tailwind class string for role badge

11. Package Dependencies

Package	Purpose
next	App framework
react	UI library
react-dom	DOM rendering
axios	HTTP client
@tanstack/react-query	Server state management
lucide-react	Icon set
clsx	Conditional classNames
tailwind-merge	Merge Tailwind classes
class-variance-authority	shadcn/ui variant system
next/font (Inter)	Font optimisation

shadcn/ui Components to Install

```
npx shadcn@latest add button input label textarea select dialog
npx shadcn@latest add table card badge toast dropdown-menu
npx shadcn@latest add separator avatar skeleton scroll-area
npx shadcn@latest add popover command tooltip
```

Environment Variables (.env.local)

```
NEXT_PUBLIC_API_BASE_URL=http://localhost:8080/api
```

12. Recommended Implementation Order for Agent

Follow this exact order to allow each step to build on the previous.

1. Install Next.js, Tailwind, shadcn/ui. Run npx shadcn@latest init and install all required components.
2. Create .env.local with NEXT_PUBLIC_API_BASE_URL.
3. Create src/lib/api/client.js (Axios instance).
4. Create all five API modules: users.js, projects.js, tasks.js, comments.js, labels.js.
5. Create src/providers/QueryProvider.jsx and ActiveUserContext.jsx.
6. Create src/app/layout.js with Navbar, Sidebar, providers, and Toaster.
7. Create shared components: PageHeader, EmptyState, LoadingSkeleton, ConfirmDialog, ErrorBanner.
8. Create src/lib/utils.js with all utility functions.
9. Build Users page (/users) — the simplest CRUD page. Get it working end-to-end.
10. Build Labels page (/labels).
11. Build Projects list page (/projects).
12. Build Project Detail page (/projects/[id]) with TaskBoard and MemberList.
13. Build Task Detail page (/tasks/[id]) with metadata panel and comments.
14. Build Dashboard page (/) with stats and summary sections.
15. Final pass: ensure all loading skeletons, empty states, and error toasts are in place.

13. Critical Notes for the AI Agent

Do NOT use TypeScript.

All files must be plain JavaScript (.js / .jsx). Do not add type annotations.

Do NOT use 'use server' on API files.

API calls happen client-side via Axios. Mark all interactive components with 'use client'.

Match backend field names exactly.

The backend returns: ownerId (not owner_id), assigneeId (not assignee_id), createdAt (not created_at). Check Section 5 shapes carefully.

No auth headers needed in v1.

The Axios interceptor should NOT add Authorization headers. That comes in a future phase.

Always use shadcn/ui components.

Do not write raw HTML buttons, inputs, or selects. Use the shadcn equivalents throughout.

Always show loading and error states.

Never render a page without handling isLoading and isError from React Query.

Invalidate queries after mutations.

After every create/update/delete, call queryClient.invalidateQueries with the correct query key (see Section 6.3).

Do NOT hard-code user IDs.

All authorId and actorId values must come from the activeUser context, never hard-coded.

Date format from backend is ISO 8601.

Always parse dates through formatDate() or formatRelativeTime() before displaying.

CORS is already configured in the backend.

Do not add any CORS workarounds in Next.js. Direct Axios calls to localhost:8080 will work.

The app is a Client-Side Rendered SPA, not SSR.

Do not use getServerSideProps, getStaticProps, or any Next.js server actions. All data fetching is done with React Query on the client.

— End of Document —