Concise Explanation:

CONTRACTS.MD = API AGREEMENT

- **What endpoints exist** (names, methods, paths)
- **What data they accept** (request shape)
- **What data they return** (response shape)
- **What errors they throw** (error codes)

This allows 4 devs to work in parallel because:

1. **Frontend** knows exactly what to call → builds UI
2. **Backend** knows exactly what to build → implements logic
3. **No surprises** → integration just works
4. **Type safety** → shared types catch mistakes at compile time

Will Contracts Need Updates?

YES! Contracts will evolve as you build. Here's when:

You MUST update CONTRACTS.md when:

- ✏️ Adding a new endpoint (e.g., `GET /api/admin/reports`)
- ✏️ Changing request/response shape (e.g., adding `birthDate` to Profile)
- ✏️ Adding new data models (e.g., `Notification`, `AdminLog`)
- ✏️ Changing URL paths or methods

MASTER PROMPT: insert at the bottom of all cursor prompts when building.

You are working inside a shared monorepo with multiple developers operating in parallel.

This project is CONTRACT-FIRST.

Before making any change, you MUST:

1) Locate and read `/docs/CONTRACTS.md`.

2) Treat everything in that document as the single source of truth for:

  - API endpoints and their names

  - Request/response shapes

  - Shared TypeScript types

  - Screen/route names

  - Ownership boundaries between app areas

Rules you MUST follow:

- DO NOT rename existing endpoints, routes, or shared types.

- DO NOT change request/response JSON shapes.

- DO NOT add implicit behavior that contradicts documented contracts.

- DO NOT "fix" contracts silently.

If you need something that is NOT currently in the contracts:

- First check whether it can be implemented WITHOUT changing contracts.

- If a contract change is truly necessary:

  1) Update `/docs/CONTRACTS.md` clearly and explicitly.

  2) Explain WHY the change is needed in a short comment or commit message.

  3) Prefer additive changes (new fields/endpoints) over breaking ones.

4) Keep names boring, explicit, and stable.

Implementation expectations:

- Match the documented endpoint names and shapes exactly.
- Use shared types from `packages/shared` instead of redefining them.
- Keep implementations minimal and predictable.
- If something is unclear, assume the contract is correct and the implementation is missing.

When unsure:

- Ask: “What does CONTRACTS.md say?”
- Default to NOT changing contracts.

Your goal is to build features that slot cleanly into an existing system without surprising other developers or breaking parallel work.