🏗️ 1RM App – Cursor Build Procedure (End-to-End)

Goal: Start from an empty repo and, by following these steps + prompts inside Cursor, end up with a working **1RM Prediction Beta 1** React Native app.

Assumptions:

- Stack: **React Native + TypeScript** (Expo or bare RN is fine; choose one)
- Platform priority: **Mobile first** (iOS/Android)
- Scope: Exactly the **Beta 1 spec** we defined (bench only, local storage, history graph, etc.)

PHASE 0 – Project Setup

Step 0.1 – Create Project (outside Cursor or via Cursor terminal)

In Cursor's terminal:

- If using Expo:

```
npx create-expo-app 1rm-prediction --template tabs@latest
cd 1rm-prediction
```

Or use your preferred RN template.

> You don't need Cursor's AI yet, just the terminal.

Step 0.2 – Open Project in Cursor

- Open the new project folder in Cursor.
- Make sure it's recognized as a TypeScript React Native project.

PHASE 1 – Basic Structure & Navigation

Step 1.1 – Define Folder Structure

Goal: Create `/src` and subfolders.

Prompt to Cursor:

> Context: 1RM Prediction Beta 1 mobile app. React Native + TypeScript.
>
> Goal: Set up basic folder structure.
>
> Constraints: Do not modify any existing code logic, only create folders and minimal index files where needed.
>
> Instructions:
>
> - Create `/src` with subfolders: `components`, `screens`, `hooks`, `logic`, `storage`, `types`, `utils`.
> - If needed, create minimal placeholder files so TypeScript compiles.
>
>   Output: Only show the list of created files and their minimal contents.

Check:

- `/src` exists with the subfolders.

Step 1.2 – Add React Navigation

Goal: Basic stack/tab navigation.

Prompt:

> Context: 1RM Prediction Beta 1 app.
>
> Goal: Add React Navigation for screen-to-screen navigation.
>
> Constraints:
>
> - Use the recommended navigation library for React Native (React Navigation).
> - Do not create any screens yet, just the navigation setup.
>
>   Instructions:

> - Install required deps in package.json (list them, I will install via terminal).
> - Create `/src/navigation/AppNavigator.tsx` with a basic stack or tab navigator.
> - Wire it into `App.tsx`.
> - Modify ONLY files necessary for navigation.
>
>   Output: Show updated `App.tsx` and new navigation file. No explanations.

Then install whatever packages it lists via terminal.

Check:

- App builds and runs with placeholder screens (if any) or basic navigator.

PHASE 2 – Types & Models

Step 2.1 – Create Core Types

Goal: Define your main data models in one place.

Prompt:

> Context: 1RM Prediction Beta 1 app.
>
> Goal: Define TypeScript interfaces for core entities.
>
> Constraints: Types only, no logic or storage.
>
> Instructions:
>
> - Create file `/src/types/models.ts`.
> - Add interfaces:
>   - `UserProfile` with age, gender, bodyweight.
>   - `BenchSet` with id, timestamp, weight, reps, RIR.
>   - `TestedOneRM` with id, timestamp, weight.
>   - `OneRMEstimate` with id, date, estimated1RM, uncertainty, confidence.
>
>   Output: Show only `/src/types/models.ts`.

Check:

- Types match the concepts we defined.

PHASE 3 – Storage Layer (Local Only)

You're defining how data is stored/retrieved, but still no algorithm.

Step 3.1 – Profile Storage

Prompt:

> Context: 1RM app.
>
> Goal: Implement local storage for `UserProfile`.
>
> Constraints:
>
> - Use AsyncStorage (or chosen local storage library).
> - All storage code lives in `/src/storage`.
> - No UI code.
>
>   Instructions:
> - Create `/src/storage/profileStorage.ts`.
> - Export functions: `getProfile()`, `saveProfile(profile: UserProfile)`.
> - Use the `UserProfile` type from `/src/types/models.ts`.
> - Handle missing profile gracefully (return `null` or similar).
>
>   Output: Show only `/src/storage/profileStorage.ts`.

Step 3.2 – Bench Set Storage

Prompt:

> Context: 1RM app.
>
> Goal: Implement storage for bench sets.
>
> Constraints: Storage only, no UI.
>
> Instructions:
>
> - Create `/src/storage/setsStorage.ts`.

> - Use `BenchSet` type.
> - Functions:
>   - `getAllBenchSets()`: returns array of `BenchSet`.
>   - `addBenchSet(set: BenchSet)`: saves a new set.
>   - Optional: `getBenchSetsSince(date: Date)`.
>
>   Output: Only `/src/storage/setsStorage.ts`.

Step 3.3 – Tested 1RM Storage

Prompt:

> Context: 1RM app.
>
> Goal: Implement storage for tested 1RMs.
>
> Constraints: Storage only.
>
> Instructions:
>
> - Create `/src/storage/tested1RMStorage.ts`.
> - Use `TestedOneRM` type.
> - Functions:
>   - `getAllTestedOneRMs()`.
>   - `addTestedOneRM(test: TestedOneRM)`.
>   - `getLastTestedOneRM()`.
>
>   Output: Only `/src/storage/tested1RMStorage.ts`.

Check:

- All three storage modules exist and compile.

PHASE 4 – Hooks for Data Access

Step 4.1 – Hook for Profile

Prompt:

> Context: 1RM app.
>
> Goal: Create a React hook to manage `UserProfile`.
>
> Constraints:
>
> - Hook only, no UI.
> - Uses `profileStorage`.
>
>   Instructions:
> - Create `/src/hooks/useUserProfile.ts`.
> - Hook API: `const { profile, loading, saveProfile } = useUserProfile();`
>
>   Output: Only `/src/hooks/useUserProfile.ts`.

Step 4.2 – Hook for Bench Sets

Prompt:

> Context: 1RM app.
>
> Goal: Create a hook to manage bench sets.
>
> Constraints:
>
> - Use `setsStorage`.
>
>   Instructions:
> - Create `/src/hooks/useBenchSets.ts`.
> - API: `const { sets, addSet, refresh } = useBenchSets();`
>
>   Output: Only `/src/hooks/useBenchSets.ts`.

Step 4.3 – Hook for Tested 1RMs

Prompt:

> Context: 1RM app.
>
> Goal: Hook to manage tested 1RMs.

> Instructions:
>
> - Create `/src/hooks/useTestedOneRMs.ts`.
> - API: `const { tests, lastTest, addTest, refresh } = useTestedOneRMs();`
>
>   Output: Only `/src/hooks/useTestedOneRMs.ts`.

PHASE 5 – Logic: Estimation & Categories (Structure First)

Step 5.1 – Estimator Skeleton

Prompt:

> Context: 1RM Prediction Beta 1.
>
> Goal: Skeleton for 1RM estimation logic. No math yet.
>
> Constraints:
>
> - Pure functions only.
> - No side effects.
> - Lives in `/src/logic`.
>
>   Instructions:
> - Create `/src/logic/estimator.ts`.
> - Export functions:
>   - `estimateOneRMFromSet(set: BenchSet): number` (placeholder implementation).
>   - `estimateBaselineOneRM(sets: BenchSet[], tests: TestedOneRM[]): OneRMEstimate | null` (placeholder implementation).
> - Use types from `/src/types/models.ts`.
>
>   Output: Only `/src/logic/estimator.ts`.

We'll fill the logic later.

Step 5.2 – Strength Category Skeleton

Prompt:

> Context: 1RM app.
>
> Goal: Provide a way to map 1RM/bodyweight to a category.
>
> Constraints:
>
> - No exact tables yet, just structure.
>
>   Instructions:
> - Create `/src/logic/strengthCategories.ts`.
> - Export:
>   - `type StrengthCategory = 'Novice' | 'Intermediate' | 'Advanced' | 'Elite';`
>   - `getStrengthCategory(estimated1RM: number, bodyweight: number, gender: UserProfile['gender']): StrengthCategory;`
> - Implementation can use simple placeholder logic for now.
>
>   Output: Only `/src/logic/strengthCategories.ts`.

Step 5.3 – Personalization Skeleton

Prompt:

> Context: 1RM app.
>
> Goal: Skeleton for personalization after first tested 1RM.
>
> Instructions:
>
> - Create `/src/logic/personalization.ts`.
> - Export:
>   - `type UserCalibration = { multiplier: number };`
>   - `deriveCalibration(sets: BenchSet[], tests: TestedOneRM[]): UserCalibration;`
>   - `applyCalibration(rawEstimate: number, calibration: UserCalibration): number;`
> - Initial implementation can be identity (no-op).
>
>   Output: Only `/src/logic/personalization.ts`.

PHASE 6 – Screens (UI) & Wiring

Now you start actually seeing the app.

Step 6.1 – Onboarding / Profile Screen

Prompt:

> Context: 1RM Prediction Beta 1.
>
> Goal: Create a simple Profile/Onboarding screen.
>
> Constraints:
>
> - Minimal UI (text inputs for age, gender select, bodyweight).
> - Uses `useUserProfile` hook.
> - No navigation setup changes except registering the screen.
>
>   Instructions:
> - Create `/src/screens/ProfileScreen.tsx`.
> - Allow viewing and editing profile.
> - Save profile via `saveProfile`.
>
>   Output: Only `/src/screens/ProfileScreen.tsx`.

Then separately:

> Modify ONLY /src/navigation/AppNavigator.tsx to include ProfileScreen as one of the screens.

Step 6.2 – Log Bench Session Screen

Prompt:

> Context: 1RM app.
>
> Goal: Create Log Bench Session screen.
>
> Constraints:
>
> - Minimal UI: fields for weight, reps, RIR, button to “Add Set”.

> - Uses `useBenchSets`.
> - Displays list of sets for today.
>
>   Instructions:
> - Create `/src/screens/LogBenchScreen.tsx`.
> - On submit, call `addSet`.
>
>   Output: Only `/src/screens/LogBenchScreen.tsx`.

Then:

> Modify ONLY /src/navigation/AppNavigator.tsx to hook in LogBenchScreen as a screen/tab.

Step 6.3 – Home / Dashboard Screen

Prompt:

> Context: 1RM app.
>
> Goal: Create HomeScreen that displays current baseline 1RM and category.
>
> Constraints:
>
> - Minimal UI.
> - Uses:
>   - `useUserProfile`
>   - `useBenchSets`
>   - `useTestedOneRMs`
>   - functions from `estimator` and `strengthCategories`
>
>   Instructions:
> - Create `/src/screens/HomeScreen.tsx`.
> - On render, compute:
>   - `baselineEstimate = estimateBaselineOneRM(sets, tests)`
>   - `strengthCategory` using `getStrengthCategory`.

> - Show:
>   - estimated 1RM with uncertainty
>   - category
>   - date of last tested 1RM (if any)
>   - a short text like “Based on your recent bench sets.”
>
>   Output: Only `/src/screens/HomeScreen.tsx`.

Then wire HomeScreen into navigator.

Step 6.4 – History / Graph Screen

You’ll need a chart library (like `victory-native` or `react-native-svg-charts`).

Prompt:

> Context: 1RM app.
>
> Goal: Create a HistoryScreen showing 90-day baseline 1RM trend.
>
> Constraints:
>
> - Minimal line chart.
> - Use one chart library (you pick the simplest).
> - Data comes from bench sets + tested 1RMs using estimator.
>
>   Instructions:
> - Tell me which chart library to install and how.
> - Create `/src/screens/HistoryScreen.tsx`.
> - Prepare data: for each day in last 90 days, compute or approximate a baseline 1RM (can reuse `estimateBaselineOneRM` or use a helper).
> - Plot baseline 1RM vs date.
>
>   Output: Only `/src/screens/HistoryScreen.tsx`.

Then install chart deps via terminal and hook screen into navigator.

PHASE 7 – Fill In Estimation Logic

Now that UI & structure exist, make the estimator actually do what we planned.

Step 7.1 – Per-Set 1RM Estimation

Prompt:

> Modify ONLY /src/logic/estimator.ts.
>
> Goal: Implement `estimateOneRMFromSet(set: BenchSet)` using a simple reps + RIR based formula.
>
> Constraints:
>
> - Use a standard rep-max formula adjusted so:
>   - effectiveRepsToFailure = reps + RIR
>   - Map that to 1RM estimate.
> - Keep it simple and well-commented.
>
>   Output: Show only `/src/logic/estimator.ts`.

Step 7.2 – Baseline 1RM (60/90 Days, Hard Reset)

Prompt:

> Modify ONLY /src/logic/estimator.ts.
>
> Goal: Implement `estimateBaselineOneRM(sets, tests)`.
>
> Requirements:
>
> - Use only bench sets from last 90 days.
> - Weight last 60 days more than older sets.
> - Use per-set estimates as input.
> - If there is at least one tested 1RM:
>   - Use the most recent one as a hard anchor (baseline near that value).
> - Compute:
>   - estimated1RM (number)

> - uncertainty (e.g., based on spread of per-set estimates)
> - confidence (e.g., “low” if few recent sets or long break)
>
> Output: Updated `/src/logic/estimator.ts` only.

Step 7.3 – Personalization (Mild Calibration)

Prompt:

> Modify ONLY /src/logic/personalization.ts and /src/logic/estimator.ts if needed.
>
> Goal: Implement mild user calibration after first tested 1RM.
>
> Requirements:
>
> - `deriveCalibration`:
>   - Compare predicted vs actual around tested 1RMs.
>   - Derive a simple multiplier (e.g. `actual / predicted`).
> - `applyCalibration`:
>   - Multiply raw estimate by this factor.
> - Integrate calibration into `estimateBaselineOneRM`.
>
>   Output: Show changed files only.

Step 7.4 – Strength Categories

Prompt:

> Modify ONLY /src/logic/strengthCategories.ts.
>
> Goal: Implement simple strength category thresholds based on 1RM/bodyweight and gender.
>
> Requirements:
>
> - Use 1RM-to-bodyweight ratio.
> - Define basic ranges for Novice, Intermediate, Advanced, Elite (can be approximate).

> Output: Only `/src/logic/strengthCategories.ts`.

PHASE 8 – Polish & Manual Testing

Step 8.1 – Flow Check

Manually go through on device/emulator:

- Set up profile
- Log several bench sets
- Optionally add a tested 1RM
- Return to Home → verify:
  - 1RM shown
  - category shown
  - uncertainty looks reasonable
- Check History screen → line is plotted

If something breaks, ask Cursor to fix **only the relevant file**.

Step 8.2 – Friend/Athlete Testing

- Install the app on a few phones (or have them use your device).
- Have them:
  - log 1–2 weeks of sets
  - test a true 1RM
- Manually compare:
  - app's estimate vs their actual max

Keep notes somewhere else (e.g. a simple table in Notion).