FlowTrack Component Blueprints (MVP)

> These blueprints align with your **revised PRD** and **data model** (sessions, physio_logs, flow_recipe_items).
> All examples assume **Next.js App Router**, **React**, **Tailwind**, and **Supabase** with a thin `/lib/db/*` layer as in your technical architecture.

---

1. Dashboard Components

1.1 `DashboardPage`

Purpose
Top-level page for `/dashboard`. Fetches data (server component) and composes `DashboardMetrics`, `DashboardCharts` (v2+ / placeholder), and quick actions.

Props

As a server component it will typically **not** receive props directly, but you can define a type if you pass pre-fetched data into child components:

```
interface DashboardPageProps { userId: string; }
```

State

- None in the server component itself.
- Any filter state lives in an optional client child (`DashboardFilters`).

Handlers / Logic

- On render (server-side):
  - Use server Supabase client to fetch:
    - Today's latest physio log (`physio_logs` , by `user_id` + `date`).
    - Today's sessions and last 7 days of sessions (`sessions`).
  - Aggregate metrics:
    - `todaySessionCount`
    - `todayAvgFlow`
    - `last7DaysSessionCount`
    - `last7DaysAvgFlow`
- Pass aggregated data as props to `DashboardMetrics` and `DashboardCharts` .

Data Bindings

- Fetch via a helper in `/lib/db`:

```
const supabase = createServerSupabase(); const today = /* format YYYY-MM-DD */; const latestPhysio = await getLatestPhysioForToday(userId); const sessionsLast7Days = await getSessionsForLast7Days(userId);
```

Lifecycle

- Runs only on server render / route navigation; no client lifecycle.

Sample JSX Skeleton

```
// app/dashboard/page.tsx import { DashboardMetrics } from '@/components/dashboard/DashboardMetrics'; import { DashboardCharts } from '@/components/dashboard/DashboardCharts'; import { QuickActions } from '@/components/dashboard/QuickActions'; import { getDashboardData } from '@/lib/db/dashboard'; export default async function DashboardPage() { const { userId, todayPhysio, todayStats, weekStats } = await getDashboardData(); return ( <main className="mx-auto max-w-4xl px-4 py-6 space-y-6"> <header className="flex items-center justify-between"> <h1 className="text-2xl font-semibold">Dashboard</h1> </header> <DashboardMetrics todayPhysio={todayPhysio} todayStats={todayStats} weekStats={weekStats} /> <QuickActions /> {/* v2+: charts, or simple placeholder */} <DashboardCharts sessionsSummary={weekStats.sessionsByDate} /> </main> ); }
```

1.2 `DashboardMetrics`

Purpose

Show today’s key stats and a small weekly summary. This is the main “at-a-glance” view.

PRD

Props

```
interface TodayStats { sessionCount: number; avgFlow: number | null; totalDurationMinutes: number; } interface WeekStats { sessionCount: number; avgFlow: number | null; sessionsByDate: { date: string; sessionCount: number; avgFlow: number | null }[]; } interface TodayPhysio { energy?: number | null; mood?: number | null; focus_clarity?: number | null; stress?: number | null; context?: string | null; } interface DashboardMetricsProps { todayPhysio: TodayPhysio | null; todayStats: TodayStats; weekStats: WeekStats; }
```

State

- No internal state; purely presentational.

Handlers / Logic

- None; just renders data.

Data Bindings

- All data passed from `DashboardPage`. No direct Supabase calls.

Lifecycle

- Pure render component.

Sample JSX Skeleton

```
export function DashboardMetrics({ todayPhysio, todayStats, weekStats, }: DashboardMetricsProps) { return ( <section className="space-y-4"> {/* Today */} <div className="rounded-xl border bg-white p-4 shadow-sm"> <div className="mb-2 flex items-center justify-between"> <h2 className="text-lg font-semibold">Today</h2> </div> <div className="grid grid-cols-2 gap-4 md:grid-cols-4"> <MetricTile label="Sessions" value={todayStats.sessionCount} /> <MetricTile label="Avg Flow" value={todayStats.avgFlow != null ? todayStats.avgFlow.toFixed(1) : '-'} /> <MetricTile label="Focus (min)" value={todayStats.totalDurationMinutes} /> <MetricTile label="Energy" value={todayPhysio?.energy ?? '-'} subtitle={todayPhysio?.context ?? ''} /> </div> </div> {/* Last 7 days */} <div className="rounded-xl border bg-white p-4 shadow-sm"> <div className="mb-2 flex items-center justify-between"> <h2 className="text-lg font-semibold">Last 7 Days</h2> </div> <div className="grid grid-cols-2 gap-4 md:grid-cols-3"> <MetricTile label="Sessions" value={weekStats.sessionCount} /> <MetricTile label="Avg Flow" value={weekStats.avgFlow != null ? weekStats.avgFlow.toFixed(1) : '-'} /> </div> </div> </section> ); } function MetricTile({ label, value, subtitle, }: { label: string; value: string | number; subtitle?: string; }) { return ( <div className="flex flex-col rounded-lg bg-gray-50 px-3 py-2"> <span className="text-xs font-medium text-gray-500">{label}</span> <span className="text-lg font-semibold text-gray-900">{value}</span> {subtitle && ( <span className="text-xs text-gray-400 truncate">{subtitle}</span> )} </div> ); }
```

1.3 `DashboardCharts`

Purpose
Optional, simplified charts showing flow over time and maybe flow vs physio (v2+). For MVP, can be a placeholder or very simple bar/line representation.

Props

```
interface SessionsByDate { date: string; // yyyy-mm-dd sessionCount: number; avgFlow: number | null; } interface DashboardChartsProps { sessionsSummary: SessionsByDate[]; }
```

State

- MVP: none.
- v2+: local state for chart type / time range.

Handlers / Logic

- None for MVP.

Data Bindings

- Data comes from `DashboardPage` aggregation.

Lifecycle

- Pure render.

Sample JSX Skeleton

```
export function DashboardCharts({ sessionsSummary }: DashboardChartsProps) { if (!sessionsSummary.length) { return ( <section className="rounded-xl border bg-white p-4 text-sm text-gray-500"> Not enough data for charts yet. Log some sessions to see trends. </section> ); } return ( <section className="rounded-xl border bg-white p-4 space-y-3"> <h2 className="text-lg font-semibold">Flow Over Time</h2> <div className="flex gap-2 overflow-x-auto"> {sessionsSummary.map((d) => ( <div key={d.date} className="flex flex-col items-center text-xs text-gray-500" > <div className="mb-1 flex h-16 w-6 items-end justify-center rounded bg-gray-100"> <div className="w-4 rounded bg-gray-800" style={{ height: `${(d.avgFlow ?? 0) * 10}%`, // simple bar }} /> </div> <span>{d.date.slice(5)}</span> <span className="font-medium text-gray-700"> {d.avgFlow != null ? d.avgFlow.toFixed(1) : '–'} </span> </div> ))} </div> </section> ); }
```

1.4 (Optional) `DashboardFilters`

Purpose
Allows filtering dashboard metrics (e.g., time range) in v2+. For v1 you can skip or keep as a simple stub.

Props

```
interface DashboardFiltersProps { value: '7d' | '30d'; onChange: (value: '7d' | '30d') => void; }
```

State

- None (pure controlled component).

Handlers / Logic

- Calls `onChange` when the filter changes.

Sample JSX Skeleton

```
export function DashboardFilters({ value, onChange }: DashboardFiltersProps) { return ( <div className="flex gap-2 text-xs"> {['7d', '30d'].map((v) => ( <button key={v} className={`rounded-full border px-3 py-1 ${ value === v ? 'bg-gray-900 text-white' : 'bg-white text-gray-700' }`} onClick={() => onChange(v as '7d' | '30d')} > {v === '7d' ? 'Last 7 days' : 'Last 30 days'} </button> ))} </div> ); }
```

---

2. New Session (Session Wizard) Components

2.1 `NewSessionPage`

Purpose
Route component (`/sessions/new`) that renders the `SessionWizard` and ensures the user is authenticated.

Props

Typically none, Next.js page.

State

- None; all wizard state lives in `SessionWizard` .

Handlers / Logic

- Optionally handle redirect after successful save via query param / router.

Sample JSX Skeleton

```
// app/sessions/new/page.tsx import { SessionWizard } from '@/components/sessions/SessionWizard'; export default function NewSessionPage() { return ( <main className="mx-auto flex max-w-xl flex-col gap-4 px-4 py-6"> <h1 className="text-2xl font-semibold">New Session</h1> <SessionWizard /> </main> ); }
```

2.2 `SessionWizard` (SPECIAL FOCUS)

Purpose

Client component that manages the entire Typeform-style multi-step flow: pre-session questions, timer, post-session questions, and final save to Supabase.

Props

```
interface SessionWizardProps { // in v1 can be empty; later you might pass defaults }
```

State

```
type SessionWizardStepId = | 'pre-activity' | 'pre-energy' | 'pre-mood' | 'timer' | 'post-flow' | 'post-notes'; interface SessionFormData { activity?: string; pre_energy?: number; pre_mood?: number; startTime?: Date; endTime?: Date; flow_rating?: number; notes?: string; } const [currentStepIndex, setCurrentStepIndex] = useState<number>(0); const [formData, setFormData] = useState<SessionFormData>({}); const [isSaving, setIsSaving] = useState(false); const [error, setError] = useState<string | null>(null);
```

Steps Definition

```
const steps: { id: SessionWizardStepId; label: string }[] = [ { id: 'pre-activity', label: 'Activity' }, { id: 'pre-energy', label: 'Energy' }, { id: 'pre-mood', label: 'Mood' }, { id: 'timer', label: 'Deep work' }, { id: 'post-flow', label: 'Flow rating' }, { id: 'post-notes', label: 'Reflection' }, ];
```

Key Handlers / Logic

- `handleNextStep()`:
  - Validate current step if needed (e.g., `flow_rating` required before leaving that step).
  - Increment `currentStepIndex`.
- `handlePrevStep()`:
  - Decrement `currentStepIndex` (except when at 0).
- `handleUpdateField(field, value)`:
  - Update `formData`.
- `handleStartSession()`:
  - Set `formData.startTime = new Date()`.
  - Move to next step (`timer`).
- `handleEndSession()`:
  - Set `formData.endTime = new Date()`.
  - Move to next step.

- `handleSubmit()`:
  - Validate that `startTime`, `endTime`, `flow_rating` exist.
  - Build payload for `sessions` table:
    - `date` from `startTime`
    - `start_time`, `end_time`
    - `duration_seconds`
    - `activity`
    - `flow_rating`
    - `notes`
  - Call `insertSession(payload)`.
  - Redirect to `/sessions` or `/dashboard`.

Data Bindings

- Use client Supabase instance or a `/lib/db/sessions.ts` function that encapsulates `supabase.from('sessions').insert(...)`.

```
const payload = { user_id, date: startTime.toISOString().slice(0, 10), start_time: startTime.toISOString(), end_time: endTime.toISOString(), duration_seconds: Math.floor( (endTime.getTime() - startTime.getTime()) / 1000 ), activity: formData.activity ?? 'Untitled session', flow_rating: formData.flow_rating!, notes: formData.notes ?? '', };
```

Lifecycle

- On mount: nothing heavy. Optionally preload suggested activity names (autocomplete) via a small supabase call.
- On unmount: nothing required in v1.

Sample JSX Skeleton

```
'use client'; import { useState } from 'react'; import { WizardStep } from './WizardStep'; import { WizardProgress } from './WizardProgress'; import { insertSession } from '@/lib/db/sessions'; import { useRouter } from 'next/navigation'; export function SessionWizard({}: SessionWizardProps) { const router = useRouter(); const steps = [ { id: 'pre-activity', label: 'Activity' }, { id: 'pre-energy', label: 'Energy' }, { id: 'pre-mood', label: 'Mood' }, { id: 'timer', label: 'Deep work' }, { id: 'post-flow', label: 'Flow rating' }, { id: 'post-notes', label: 'Reflection' }, ] as const; type StepId = (typeof steps)[number]['id']; const [currentStepIndex, setCurrentStepIndex] = useState(0); const [formData, setFormData] = useState<SessionFormData>({}); const [isSaving, setIsSaving] = useState(false); const [error, setError] = useState<string | null>(null); const currentStep = steps[currentStepIndex]; function updateField<K extends keyof SessionFormData>(field: K, value: SessionFormData[K]) {
```

```
setFormData((prev) => ({ ...prev, [field]: value })); } function handleNext() { // Simple validation for required fields on certain steps if (currentStep.id === 'post-flow' && !formData.flow_rating) { setError('Please rate your flow (0-10).'); return; } setError(null); setCurrentStepIndex((i) => Math.min(i + 1, steps.length - 1)); } function handlePrev() { setError(null); setCurrentStepIndex((i) => Math.max(i - 1, 0)); } function handleStartSession() { updateField('startTime', new Date()); handleNext(); // move to timer step } function handleEndSession() { updateField('endTime', new Date()); handleNext(); // move to post-flow step } async function handleSubmit() { if (!formData.startTime || !formData.endTime || !formData.flow_rating) { setError('Missing timing or flow rating.'); return; } try { setIsSaving(true); setError(null); const startTime = formData.startTime; const endTime = formData.endTime; const duration_seconds = Math.floor( (endTime.getTime() - startTime.getTime()) / 1000 ); await insertSession({ date: startTime.toISOString().slice(0, 10), start_time: startTime.toISOString(), end_time: endTime.toISOString(), duration_seconds, activity: formData.activity ?? 'Untitled session', flow_rating: formData.flow_rating, notes: formData.notes ?? '', }); router.push('/sessions'); } catch (err: any) { setError(err.message ?? 'Failed to save session.'); } finally { setIsSaving(false); } } return ( <div className="flex flex-col gap-4 rounded-xl border bg-white p-4 shadow-sm"> <WizardProgress steps={steps} currentIndex={currentStepIndex} /> {error && ( <p className="text-sm text-red-600"> {error} </p> )} <WizardStep step={currentStep} formData={formData} onChange={updateField} onStartSession={handleStartSession} onEndSession={handleEndSession} /> <div className="mt-4 flex justify-between"> <button type="button" className="text-sm text-gray-500" onClick={handlePrev} disabled={currentStepIndex === 0} > Back </button> {currentStep.id === 'post-notes' ? ( <button type="button" className="rounded-full bg-gray-900 px-4 py-2 text-sm font-medium text-white" onClick={handleSubmit} disabled={isSaving} > {isSaving ? 'Saving…' : 'Save Session'} </button> ) : ( <button type="button" className="rounded-full bg-gray-900 px-4 py-2 text-sm font-medium text-white" onClick={currentStep.id === 'timer' ? handleEndSession : handleNext} > {currentStep.id === 'timer' ? 'End Session' : 'Next'} </button> )} </div> </div> ); }
```

2.3 `WizardStep`

Purpose

Render a single step of the wizard. Different UI depending on `step.id` (pre-session, timer, post-session).

Props

```
interface WizardStepProps { step: { id: SessionWizardStepId; label: string }; formData: SessionFormData; onChange: <K extends keyof SessionFormData>( field: K,
```

`value: SessionFormData[K] ) => void; onStartSession: () => void; onEndSession: () => void; }`

State

- Optionally local state for timer display if you want to show live seconds (or you can delegate to `SessionTimer`).

Handlers / Logic

- Renders appropriate view based on `step.id`.

Sample JSX Skeleton (pre + post example)

```
export function WizardStep({ step, formData, onChange, onStartSession, onEndSession, }: WizardStepProps) { if (step.id === 'pre-activity') { return ( <div className="space-y-4"> <p className="text-sm text-gray-600">What are you working on?</p> <input className="w-full rounded-lg border px-3 py-2 text-sm" placeholder="e.g. Write FlowTrack PRD" value={formData.activity ?? ''} onChange={(e) => onChange('activity', e.target.value)} /> <button type="button" className="ml-auto rounded-full bg-gray-900 px-4 py-2 text-sm font-medium text-white" onClick={onStartSession} > Begin deep work </button> </div> ); } if (step.id === 'pre-energy') { return ( <div className="space-y-4"> <p className="text-sm text-gray-600">How is your energy right now?</p> <div className="flex flex-wrap gap-2"> {Array.from({ length: 11 }).map((_, i) => ( <button key={i} type="button" className={`h-8 w-8 rounded-full text-xs ${ formData.pre_energy === i ? 'bg-gray-900 text-white' : 'bg-gray-100 text-gray-700' }`} onClick={() => onChange('pre_energy', i)} > {i} </button> ))} </div> </div> ); } if (step.id === 'timer') { return ( <div className="space-y-4 text-center"> <p className="text-sm text-gray-600">You're in a focused block. Stay with it.</p> {/* You could embed a <SessionTimer /> here */} <div className="text-4xl font-semibold tracking-tight">00:00</div> </div> ); } if (step.id === 'post-flow') { return ( <div className="space-y-4"> <p className="text-sm text-gray-600"> How strong was your flow in this session? </p> <div className="flex flex-wrap gap-2"> {Array.from({ length: 11 }).map((_, i) => ( <button key={i} type="button" className={`h-8 w-8 rounded-full text-xs ${ formData.flow_rating === i ? 'bg-gray-900 text-white' : 'bg-gray-100 text-gray-700' }`} onClick={() => onChange('flow_rating', i)} > {i} </button> ))} </div> </div> ); } if (step.id === 'post-notes') { return ( <div className="space-y-4"> <p className="text-sm text-gray-600"> Any quick notes? What helped or hurt your focus? </p> <textarea className="w-full rounded-lg border px-3 py-2 text-sm" rows={4} value={formData.notes ?? ''} onChange={(e) => onChange('notes', e.target.value)} /> </div> ); } return null; }
```

2.4 `WizardProgress`

Purpose
Show multi-step progress (dots or bar) and step label. This supports the Typeform-feel with clear progress.

PRD

Props

```
interface WizardProgressProps { steps: { id: SessionWizardStepId; label: string } []; currentIndex: number; }
```

State

- None.

Sample JSX

```
export function WizardProgress({ steps, currentIndex }: WizardProgressProps) { return ( <div className="flex items-center justify-between text-xs text-gray-500"> <div className="flex gap-1"> {steps.map((step, idx) => ( <span key={step.id} className={`h-1.5 w-6 rounded-full ${ idx <= currentIndex ? 'bg-gray-900' : 'bg-gray-200' }`} /> ))} </div> <span className="text-xs"> Step {currentIndex + 1} of {steps.length} </span> </div> ); }
```

2.5 (Optional) `SessionSummary`

Purpose
Show a simple summary before final save. MVP can skip, but easy to add.

---

3. Physio Log Components

3.1 `PhysioLogPage`

Purpose
Route for `/physio/new`. Renders `PhysioForm` (Typeform-style or compact form as in PRD).

PRD

Sample JSX

```
// app/physio/new/page.tsx import { PhysioForm } from '@/components/physio/PhysioForm'; export default function PhysioLogPage() { return
```

```
( <main className="mx-auto flex max-w-xl flex-col gap-4 px-4 py-6"> <h1 className="text-2xl font-semibold">Log Physio</h1> <PhysioForm /> </main> ); }
```

3.2 `PhysioForm`

Purpose
Client component capturing daily physiology (HRV, sleep, etc., or just the 4–5 fields from revised PRD: energy, mood, focus_clarity, stress, context).

PRD

Props

```
interface PhysioFormProps { // optional initial values, v2+ }
```

State

```
interface PhysioFormData { energy?: number; mood?: number; focus_clarity?: number; stress?: number; context?: string; } const [data, setData] = useState<PhysioFormData>({}); const [stepIndex, setStepIndex] = useState(0); // if Typeform style const [isSaving, setIsSaving] = useState(false); const [error, setError] = useState<string | null>(null);
```

Handlers

- `updateField(field, value)`
- `handleNext() / handlePrev()` if using one-question-per-screen.
- `handleSubmit()`:
  - Build payload for `physio_logs`.
  - Insert via Supabase.
  - Redirect to `/dashboard`.

Sample JSX (compact version)

```
'use client'; import { useState } from 'react'; import { insertPhysioLog } from '@/lib/db/physio'; import { useRouter } from 'next/navigation'; export function PhysioForm({}: PhysioFormProps) { const router = useRouter(); const [data, setData] = useState<PhysioFormData>({}); const [isSaving, setIsSaving] = useState(false); const [error, setError] = useState<string | null>(null); function updateField<K extends keyof PhysioFormData>(field: K, value: PhysioFormData[K]) { setData((prev) => ({ ...prev, [field]: value })); } async function handleSubmit(e: React.FormEvent) { e.preventDefault(); try { setIsSaving(true); setError(null); await insertPhysioLog({ energy: data.energy ?? null, mood: data.mood ?? null, focus_clarity: data.focus_clarity ?? null, stress: data.stress ?? null, context:
```

```
data.context?.trim() || null, }); router.push('/dashboard'); } catch (err: any) { setError(err.message ?? 'Failed to save physio.'); } finally { setIsSaving(false); } } return ( <form onSubmit={handleSubmit} className="space-y-4 rounded-xl border bg-white p-4 shadow-sm" > <FieldSlider label="Energy" value={data.energy} onChange={(v) => updateField('energy', v)} /> <FieldSlider label="Mood" value={data.mood} onChange={(v) => updateField('mood', v)} /> <FieldSlider label="Focus / Clarity" value={data.focus_clarity} onChange={(v) => updateField('focus_clarity', v)} /> <FieldSlider label="Stress" value={data.stress} onChange={(v) => updateField('stress', v)} /> <div> <label className="mb-1 block text-xs font-medium text-gray-600"> Context (one word, optional) </label> <input className="w-full rounded-lg border px-3 py-2 text-sm" placeholder="tired, gym, fasted…" value={data.context ?? ''} onChange={(e) => updateField('context', e.target.value)} /> </div> {error && <p className="text-sm text-red-600">{error}</p>} <button type="submit" className="w-full rounded-full bg-gray-900 px-4 py-2 text-sm font-medium text-white" disabled={isSaving} > {isSaving ? 'Saving…' : 'Save Physio'} </button> </form> ); } function FieldSlider({ label, value, onChange, }: { label: string; value?: number; onChange: (v: number) => void; }) { return ( <div> <label className="mb-1 block text-xs font-medium text-gray-600"> {label} </label> <div className="flex flex-wrap gap-2"> {Array.from({ length: 11 }).map((_, i) => ( <button key={i} type="button" className={`h-8 w-8 rounded-full text-xs ${ value === i ? 'bg-gray-900 text-white' : 'bg-gray-100 text-gray-700' }`} onClick={() => onChange(i)} > {i} </button> ))} </div> </div> ); }
```

---

4. Sessions List Components

4.1 `SessionsListPage`

Purpose

Route `/sessions` that fetches sessions (server) and renders filters + list grouped by day.

Sample JSX

```
// app/sessions/page.tsx import { getSessionsForList } from '@/lib/db/sessions'; import { SessionsFilterBar } from '@/components/sessions/SessionsFilterBar'; import { SessionCard } from '@/components/sessions/SessionCard'; export default async function SessionsListPage({ searchParams, }: { searchParams: { minFlow?: string; from?: string; to?: string }; }) { const filters = { minFlow: searchParams.minFlow ? Number(searchParams.minFlow) : undefined, from: searchParams.from, to: searchParams.to, }; const groupedSessions = await getSessionsForList(filters); return ( <main className="mx-auto max-w-3xl px-4 py-6 space-y-4"> <h1 className="text-2xl font-semibold">Sessions</h1> <SessionsFilterBar /> <div className="space-y-4"> {groupedSessions.map((group) => ( <section key={group.label} className="space-y-2"> <h2 className="text-xs font-
```

```
medium uppercase text-gray-500"> {group.label} </h2> <div className="space-y-2"> {group.sessions.map((session) => ( <SessionCard key={session.id} session={session} /> )} </div> </section> )} </div> </main> ); }
```

4.2 `SessionsFilterBar`

Purpose
Simple client-side or server-side filter bar for min flow score, date range (MVP).

Props

For a client component:

```
interface SessionsFilterBarProps { initialMinFlow?: number; initialFrom?: string; initialTo?: string; }
```

State

- Controlled inputs for filters.

Sample JSX

```
'use client'; import { useRouter, useSearchParams } from 'next/navigation'; import { useState } from 'react'; export function SessionsFilterBar() { const router = useRouter(); const searchParams = useSearchParams(); const [minFlow, setMinFlow] = useState<number | ''>( searchParams.get('minFlow') ? Number(searchParams.get('minFlow')) : '' ); function applyFilters() { const params = new URLSearchParams(searchParams.toString()); if (minFlow !== '') params.set('minFlow', String(minFlow)); else params.delete('minFlow'); router.push(`/sessions?${params.toString()}`); } return ( <div className="flex items-end gap-3 text-xs"> <div> <label className="mb-1 block text-[11px] font-medium text-gray-600"> Min flow </label> <input type="number" min={0} max={10} className="w-20 rounded border px-2 py-1" value={minFlow} onChange={(e) => setMinFlow(e.target.value === '' ? '' : Number(e.target.value))} /> </div> <button type="button" className="rounded-full bg-gray-900 px-3 py-1 text-xs font-medium text-white" onClick={applyFilters} > Apply </button> </div> ); }
```

4.3 `SessionCard`

Purpose
Display a single session in the list with date, activity, duration, and flow rating.

Props

```
interface Session { id: string; date: string; activity: string; duration_seconds: number; flow_rating: number; } interface SessionCardProps { session: Session; onClick?: () => void; }
```

State

- None.

Sample JSX

```
export function SessionCard({ session, onClick }: SessionCardProps) { const minutes = Math.round(session.duration_seconds / 60); return ( <button type="button" onClick={onClick} className="w-full rounded-lg border bg-white p-3 text-left shadow-sm hover:bg-gray-50" > <div className="flex items-center justify-between"> <span className="text-xs text-gray-500"> {session.date} </span> <span className="text-xs font-semibold text-gray-900"> Flow {session.flow_rating}/10 </span> </div> <div className="mt-1 text-sm font-medium text-gray-900"> {session.activity} </div> <div className="mt-1 text-xs text-gray-500"> Duration: {minutes} min </div> </button> ); }
```

---

5. Flow Recipe Components

5.1 `FlowRecipePage`

Purpose
Route `/flow-recipe` showing the active recipe items with add/edit/delete, plus a list of versions if you decide to surface them.

Sample JSX

```
// app/flow-recipe/page.tsx import { getActiveFlowRecipeItems } from '@/lib/db/flowRecipe'; import { FlowRecipeEditor } from '@/components/recipe/FlowRecipeEditor'; export default async function FlowRecipePage() { const items = await getActiveFlowRecipeItems(); return ( <main className="mx-auto max-w-2xl px-4 py-6 space-y-4"> <h1 className="text-2xl font-semibold">Flow Recipe</h1> <FlowRecipeEditor initialItems={items} /> </main> ); }
```

5.2 `FlowRecipeEditor`

Purpose
Client component to view, add, edit, and delete active recipe items.

Props

`interface FlowRecipeItem { id: string; title: string; notes: string | null; order_index: number; } interface FlowRecipeEditorProps { initialItems: FlowRecipeItem[]; }`

State

`const [items, setItems] = useState<FlowRecipeItem[]>(initialItems); const [isSaving, setIsSaving] = useState(false); const [error, setError] = useState<string | null>(null); const [editingItem, setEditingItem] = useState<FlowRecipeItem | null>(null);`

Handlers

- `handleAdd()` → open empty `FlowRecipeItemForm`.
- `handleEdit(item)` → open form with item.
- `handleDelete(id)` → call delete helper, update state.
- `handleSave(item)` → insert or update via Supabase, update state.

Sample JSX

`'use client'; import { useState } from 'react'; import { FlowRecipeItemForm } from './FlowRecipeItemForm'; import { insertFlowRecipeItem, updateFlowRecipeItem, deleteFlowRecipeItem, } from '@/lib/db/flowRecipe'; export function FlowRecipeEditor({ initialItems }: FlowRecipeEditorProps) { const [items, setItems] = useState(initialItems); const [editingItem, setEditingItem] = useState<FlowRecipeItem | null>(null); const [isSaving, setIsSaving] = useState(false); const [error, setError] = useState<string | null>(null); async function handleDelete(id: string) { setIsSaving(true); try { await deleteFlowRecipeItem(id); setItems((prev) => prev.filter((i) => i.id !== id)); } catch (err: any) { setError(err.message ?? 'Failed to delete item.'); } finally { setIsSaving(false); } } async function handleSave(input: { id?: string; title: string; notes?: string }) { setIsSaving(true); setError(null); try { if (input.id) { const updated = await updateFlowRecipeItem(input.id, { title: input.title, notes: input.notes ?? null, }); setItems((prev) => prev.map((i) => (i.id === updated.id ? updated : i))); } else { const created = await insertFlowRecipeItem({ title: input.title, notes: input.notes ?? null, order_index: items.length, }); setItems((prev) => [...prev, created]); } setEditingItem(null); } catch (err: any) { setError(err.message ?? 'Failed to save item.'); } finally { setIsSaving(false); } } return (<section className="space-y-3 rounded-xl border bg-white p-4 shadow-sm"> {error && <p className="text-sm text-red-600">{error}</p>} {items.length === 0 && (<p className="text-sm text-gray-600"> Your Flow Recipe is empty. Add 1–3 habits that help you focus. </p>)} <ul className="space-y-2"> {items.map((item) => (<FlowRecipeVersionItem key={item.id} item={item} onEdit={() => setEditingItem(item)} onDelete={() => handleDelete(item.id)} />))} <button type="button" className="mt-2 rounded-full bg-gray-900 px-4 py-2 text-sm font-medium text-white" onClick={() => setEditingItem({ id: '', title: '', notes: '',`

```
order_index: items.length })} > + Add item </button> {editingItem && ( <FlowRecipeItemForm initialItem={editingItem.id ? editingItem : undefined} onCancel={() => setEditingItem(null)} onSave={handleSave} isSaving={isSaving} /> )} </section> ); }
```

5.3 `FlowRecipeVersionItem` (really just “item row” in v1)

Purpose

Display a single recipe item row with title, notes preview, and Edit/Delete actions.

Props

```
interface FlowRecipeVersionItemProps { item: FlowRecipeItem; onEdit: () => void; onDelete: () => void; }
```

Sample JSX

```
export function FlowRecipeVersionItem({ item, onEdit, onDelete, }: FlowRecipeVersionItemProps) { return ( <li className="flex items-start justify-between gap-2 rounded-lg border px-3 py-2"> <div className="flex-1"> <div className="text-sm font-medium text-gray-900">{item.title}</div> {item.notes && ( <div className="text-xs text-gray-600 line-clamp-2">{item.notes}</div> )} </div> <div className="flex gap-2 text-xs"> <button type="button" className="text-gray-600 hover:text-gray-900" onClick={onEdit} > Edit </button> <button type="button" className="text-red-600 hover:text-red-700" onClick={onDelete} > Delete </button> </div> </li> ); }
```

5.4 `FlowRecipeItemForm`

Purpose

Inline form for creating or editing a recipe item (title + notes).

Props

```
interface FlowRecipeItemFormProps { initialItem?: { id: string; title: string; notes?: string | null }; onCancel: () => void; onSave: (input: { id?: string; title: string; notes?: string }) => void; isSaving?: boolean; }
```

State

```
const [title, setTitle] = useState(initialItem?.title ?? ''); const [notes, setNotes] = useState(initialItem?.notes ?? '');
```

Sample JSX

```
'use client'; import { useState, useEffect } from 'react'; export function FlowRecipeItemForm({ initialItem, onCancel, onSave, isSaving, }: FlowRecipeItemFormProps) { const [title, setTitle] = useState(initialItem?.title ?? ''); const [notes, setNotes] = useState(initialItem?.notes ?? ''); useEffect(() => { if (initialItem) { setTitle(initialItem.title); setNotes(initialItem.notes ?? ''); } }, [initialItem]); function handleSubmit(e: React.FormEvent) { e.preventDefault(); if (!title.trim()) return; onSave({ id: initialItem?.id, title: title.trim(), notes: notes.trim() }); } return ( <form onSubmit={handleSubmit} className="mt-3 space-y-3 rounded-lg border bg-gray-50 p-3 text-xs" > <div> <label className="mb-1 block text-[11px] font-medium text-gray-600"> Title </label> <input className="w-full rounded border px-2 py-1 text-sm" value={title} onChange={(e) => setTitle(e.target.value)} placeholder="e.g. Morning deep work 7-9am" /> </div> <div> <label className="mb-1 block text-[11px] font-medium text-gray-600"> Notes (optional) </label> <textarea className="w-full rounded border px-2 py-1 text-sm" rows={3} value={notes} onChange={(e) => setNotes(e.target.value)} /> </div> <div className="flex justify-end gap-2"> <button type="button" className="rounded-full border px-3 py-1" onClick={onCancel} > Cancel </button> <button type="submit" disabled={isSaving} className="rounded-full bg-gray-900 px-4 py-1 text-xs font-medium text-white" > {isSaving ? 'Saving…' : 'Save'} </button> </div> </form> ); }
```