

# Runbook Cursor – Multi-dispozitiv (React + Vite + Tailwind + PWA)

Bază confirmată din analiza ta: **Vite + React 18 + React Router 6 + Tailwind 3 + VitePWA**. Mai jos ai pași „copy-paste” pentru Cursor + patch-uri concrete. Rulează pașii în ordine.

## STEP 1 — Normalizează layout & tipografia (fluid)

### 1.1. `tailwind.config.js` — container, breakpoints, ecran „tv”

```
/** @type {import('tailwindcss').Config} */
export default {
  content: [
    './index.html',
    './src/**/*..{js,jsx,ts,tsx}',
  ],
  theme: {
    container: {
      center: true,
      padding: {
        DEFAULT: '1rem',
        sm: '1.25rem',
        md: '1.5rem',
        lg: '2rem',
        xl: '2.5rem',
      },
    },
    extend: {
      screens: {
        xs: '360px',
        '2xl': '1536px',
        // Mod de afișare TV: pointer fin dar fără hover
        tv: { 'raw': '(hover: none) and (pointer: fine)' },
      },
    },
  },
  plugins: [],
}
```

### 1.2. `src/index.css` — tipografie fluidă + utilitare TV/touch

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

```

/* Tipografie fluidă de bază */
@layer base {
  html { font-size: 100%; }
  h1 { font-size: clamp(1.5rem, 2vw + 1rem, 2.25rem); }
  h2 { font-size: clamp(1.25rem, 1.5vw + 0.9rem, 1.75rem); }
  h3 { font-size: clamp(1.125rem, 1vw + 0.85rem, 1.5rem); }
  p, li, input, button { font-size: clamp(1rem, 0.4vw + 0.9rem, 1.125rem); }
}

/* Moduri de interacțiune (setate din index.html sau hook) */
@layer utilities {
  html[data-mode="touch"] .hover-only { display: none !important; }
  html[data-mode="tv"] .hide-on-tv { display: none !important; }
  .tv-safe { padding: var(--tv-safe, 0); }
  .hit-44 { min-width: 44px; min-height: 44px; }
  .hit-64 { min-width: 64px; min-height: 64px; }
}

```

### 1.3. index.html — viewport + detectare mod (touch / tv / desktop)

Plasează imediat înainte de `</head>`:

```

<meta name="viewport" content="width=device-width, initial-scale=1" />
<script>
(function(){
  function setMode(){
    const d = document.documentElement;
    if (matchMedia('(hover: none) and (pointer: fine)').matches)
      d.setAttribute('data-mode', 'tv');
    else if (matchMedia('(hover: none) and (pointer: coarse)').matches)
      d.setAttribute('data-mode', 'touch');
    else d.setAttribute('data-mode', 'desktop');
  }
  setMode();
  matchMedia('(hover: none) and (pointer: fine)').addEventListener('change',
    setMode);
  matchMedia('(hover: none) and (pointer:
coarse)').addEventListener('change', setMode);
})();
</script>

```

### 1.4. Normalizează container-ele & grilele

- Înlocuiește `max-w-7xl` + padding repetat cu `container`.
- Folosește grilă fluidă auto-fit acolo unde e posibil:

```

{/* ex. într-o pagină de carduri */}
<div className="container">
  <div className="grid gap-4 [grid-template-columns:repeat(auto-

```

```
fit,minmax(16rem,1fr))]]">
  { /* carduri */ }
</div>
</div>
```

## STEP 2 — MainLayout: navbar fluid + conținut

### 2.1. `src/components/MainLayout.jsx`

- Înlocuiește în navbar `h-16` cu o înălțime fluidă.
- Fă wrapper-ul de pagină să folosească `container`.

```
export default function MainLayout({ children }) {
  return (
    <div className="min-h-screen flex flex-col">
      <header className="sticky top-0 z-50 bg-white/80 backdrop-blur border-b
        min-h-[clamp(56px,6vh,72px)] flex items-center">
        <div className="container flex items-center justify-between">
          { /* logo, user info, logout */ }
        </div>
      </header>
      <main className="flex-1">
        <div className="container py-4 sm:py-6 lg:py-8">
          {children}
        </div>
      </main>
    </div>
  );
}
```

## STEP 3 — Butoane & target-uri tactil $\geq 44$ px (și mod TV)

### 3.1. `src/components/ui/Button.jsx`

```
import React from 'react';

function cn(...a){ return a.filter(Boolean).join(' '); }

export function Button({ as:Tag='button', variant='primary', size='md',
iconOnly=false, className='', ...props }) {
  const base = 'inline-flex items-center justify-center rounded-xl font-
medium focus-visible:outline outline-2 outline-offset-2 outline-sky-500
disabled:opacity-50 disabled:pointer-events-none';
  const sizes = {
    sm: iconOnly ? 'hit-44 text-sm' : 'hit-44 px-3 py-2 text-sm',
```



```


md: iconOnly ? 'hit-44'      : 'hit-44 px-4 py-2',
lg: iconOnly ? 'hit-48 text-lg' : 'min-h-[48px] px-5 py-3 text-lg',
tv: iconOnly ? 'hit-64 text-xl' : 'hit-64 px-6 py-4 text-xl',
};
const variants = {
  primary: 'bg-sky-600 text-white hover:bg-sky-700 active:bg-sky-800',
  ghost: 'bg-transparent hover:bg-slate-100 active:bg-slate-200',
  outline: 'border border-slate-300 hover:bg-slate-50 active:bg-slate-100',
};
return (
  <Tag
    className={cn(base, sizes[size] || sizes.md, variants[variant] ||
variants.primary, className)}
    {...props}
  />
);
}

export function IconButton({ label, size='md', className='', ...props }){
  return (
    <Button aria-label={label} title={label} iconOnly size={size}
className={cn('aspect-square', className)} {...props} />
  );
}

```

### 3.2. Folosește `IconButton` pentru emoji/icoane

Ex.: în tabele/liste, înlocuiește , , etc. cu:

```
<IconButton label="Închide" onClick={...}> </IconButton>
```

## STEP 4 — Tooltips/Popover touch-friendly (fără „hover-only”)

### 4.1. `src/hooks/useInteractionMode.js`

```

import { useEffect, useState } from 'react';
export function useInteractionMode(){
  const [mode, setMode] = useState('desktop');
  useEffect(()=>{
    const d = document.documentElement;
    function apply(){
      const m = window.matchMedia('(hover: none) and (pointer:
fine)').matches ? 'tv'
      : window.matchMedia('(hover: none) and (pointer: coarse)').matches ?
'touch'
      : 'desktop';
      setMode(m); d.setAttribute('data-mode', m);
    }
  });
}

```

```

    }
    apply();
    const a = [
      window.matchMedia('(hover: none) and (pointer: fine)'),
      window.matchMedia('(hover: none) and (pointer: coarse)')
    ];
    a.forEach(x=>x.addEventListener('change', apply));
    return ()=>a.forEach(x=>x.removeEventListener('change', apply));
  },[]);
  return mode;
}

```

#### 4.2. `src/components/ui/Tooltip.tsx|jsx` (schemă simplă)

```

import { useState } from 'react';
import { useInteractionMode } from '@hooks/useInteractionMode';

export function Tooltip({ trigger, content }){
  const mode = useInteractionMode();
  const [open,setOpen] = useState(false);
  const isHover = mode === 'desktop';
  return (
    <span className="relative inline-block">
      <span
        onMouseEnter={isHover ? ()=>setOpen(true) : undefined}
        onMouseLeave={isHover ? ()=>setOpen(false) : undefined}
        onClick={!isHover ? ()=>setOpen(v=>!v) : undefined}
        className={isHover ? '' : 'hit-44'}
        role="button"
        aria-haspopup="dialog"
        aria-expanded={open}
      >{trigger}</span>
      {open && (
        <span className="absolute z-50 mt-2 max-w-xs rounded-lg bg-slate-900
text-white p-2 text-sm shadow-lg">
          {content}
        </span>
      )}
    </span>
  );
}

```

## STEP 5 — TV „safe area” + preset de dimensiuni

### 5.1. `src/styles/tv.css`

```
html[data-mode="tv"] { --tv-safe: 5vmin; }
html[data-mode="tv"] body { font-size: 18px; }
```

Importă fișierul în `src/main.jsx` sau în `index.css`:

```
import '@/styles/tv.css';
```

## 5.2. Wrapper `SafeArea`

`src/components/SafeArea.jsx`

```
export default function SafeArea({ className='', children }){
  return <div className={`tv-safe ${className}`}>{children}</div>;
}
```

Folosește în layout-uri full-bleed:

```
<SafeArea>
  {/* conținut */}
</SafeArea>
```

---

# STEP 6 — Navigație spațială (telecomandă)

## 6.1. `src/hooks/useSpatialNavigation.js`

```
import { useEffect } from 'react';

const DIR = { ArrowUp: 'up', ArrowDown: 'down', ArrowLeft: 'left',
ArrowRight: 'right' };

function center(el){ const r = el.getBoundingClientRect(); return { x:
r.left + r.width/2, y: r.top + r.height/2 }; }

function inDir(from, to, dir){
  const dx = to.x - from.x, dy = to.y - from.y;
  if (dir==='left') return dx < 0 && Math.abs(dx) > Math.abs(dy);
  if (dir==='right') return dx > 0 && Math.abs(dx) > Math.abs(dy);
  if (dir==='up') return dy < 0 && Math.abs(dy) > Math.abs(dx);
  if (dir==='down') return dy > 0 && Math.abs(dy) > Math.abs(dx);
}

export function useSpatialNavigation(containerRef, { onBack }={}){
  useEffect(()=>{
    const el = containerRef?.current; if(!el) return;

```

```

    function focusables(){ return Array.from(el.querySelectorAll('[data-
focusable="true"]:not([disabled])')); }
    function keydown(e){
      if (e.key in DIR){
        e.preventDefault();
        const items = focusables(); if(!items.length) return;
        const current = document.activeElement &&
el.contains(document.activeElement) ? document.activeElement : items[0];
        const from = center(current);
        const dir = DIR[e.key];
        const candidates = items.filter(x=>x!==current).map(x=>({ el:x,
c:center(x) })))
          .filter(({c})=>inDir(from, c, dir))
          .sort((a,b)=>{
            const da = Math.hypot(a.c.x-from.x, a.c.y-from.y);
            const db = Math.hypot(b.c.x-from.x, b.c.y-from.y);
            return da - db;
          });
        (candidates[0]?.el || items[0]).focus();
      } else if (e.key === 'Enter') {
        e.preventDefault(); document.activeElement?.click?();
      } else if (e.key === 'Escape' || e.key === 'Backspace') {
        onBack?();
      }
    }
  }
  el.addEventListener('keydown', keydown);
  return ()=>el.removeEventListener('keydown', keydown);
}, [containerRef, onBack]);
}

```

## 6.2. Exemplu de utilizare pe un grid de carduri

```

import { useRef, useEffect } from 'react';
import { useSpatialNavigation } from '@hooks/useSpatialNavigation';

export function CardsGrid({ items }){
  const ref = useRef(null);
  useSpatialNavigation(ref);
  useEffect(()=>{ ref.current?.focus(); },[]);

  return (
    <div ref={ref} tabIndex={0} className="outline-none grid gap-4 [grid-
template-columns:repeat(auto-fit,minmax(16rem,1fr))]">
      {items.map((it, i)=> (
        <button
          key={it.id || i}
          data-focusable="true"
          className="rounded-2xl border p-4 text-left focus-visible:outline
outline-2 outline-offset-2 outline-sky-500 tv:hit-64"
          onClick={()=>{/* open */}}

```

```

    >
    <h3 className="text-lg font-semibold mb-1">{it.title}</h3>
    <p className="text-slate-600">{it.subtitle}</p>
  </button>
  )})
</div>
);
}

```

Notă: prefixul `tv:` devine disponibil după modificarea `tailwind.config.js` (screen „tv”).


## STEP 7 — Fixuri punctuale identificate în analiză

**Căutări & înlocuiri** (la nivel de repo; verifică vizual înainte de commit):

1) Înălțime navbar fixă: - **Find:** `h-16` - **Replace:** `min-h-[clamp(56px,6vh,72px)]`

2) Padding repetat pe containere: - **Find (regex):** `px-4\s+sm:px-6\s+lg:px-8` - **Replace:** `container`

3) Grid-uri rigide: - **Find:** `grid-cols-1 md:grid-cols-2` - **Replace:** `[grid-template-columns:repeat(auto-fit,minmax(16rem,1fr))]`

4) Butoane mici (icon-only,  emoji): - Înlocuiește cu `<IconButton label="...">...</IconButton>`; menține aria-label clar.

5) Signature Canvas hardcodat: - **În componentă** setează stiluri CSS:

```

<div className="w-full max-w-[600px] h-[clamp(200px,30vh,300px)]">
  <canvas className="w-full h-full block rounded-xl border" /* ... */ />
</div>

```

6) Modale `w-[95vw] max-w-4xl`: - Adaugă varianta TV: `tv:w-[90vw] tv:max-w-[1600px]`.

## STEP 8 — PWA: mici îmbunătățiri sigure (fără risc de cache bust mare)

### 8.1. `vite.config.js` — manifest extins (exemplu minimal)

```

import { defineConfig } from 'vite';
import react from '@vitejs/plugin-react';
import { VitePWA } from 'vite-plugin-pwa';

```

```


export default defineConfig({
  plugins: [
    react(),
    VitePWA({
      registerType: 'autoUpdate',
      includeAssets: ['favicon.svg', 'robots.txt', 'apple-touch-icon.png'],
      manifest: {
        name: 'DeCamino',
        short_name: 'DeCamino',
        start_url: '/',
        display: 'standalone',
        background_color: '#ffffff',
        theme_color: '#0ea5e9',
        icons: [
          { src: '/icons/192.png', sizes: '192x192', type: 'image/png' },
          { src: '/icons/512.png', sizes: '512x512', type: 'image/png' }
        ]
      },
      workbox: {
        navigateFallbackDenylist: [/^\/api\/$/],
      }
    })
  ],
});

```

## 8.2. Indicator offline simplu

src/components/OfflineIndicator.jsx

```

import { useEffect, useState } from 'react';
export default function OfflineIndicator(){
  const [off,setOff] = useState(!navigator.onLine);
  useEffect(()=>{
    const on = ()=>setOff(false); const offn = ()=>setOff(true);
    window.addEventListener('online', on);
    window.addEventListener('offline', offn);
    return ()=>{ window.removeEventListener('online', on);
    window.removeEventListener('offline', offn); };
  },[]);
  if(!off) return null;
  return <div className="fixed bottom-4 left-1/2 -translate-x-1/2 rounded-
full bg-amber-500 text-white px-4 py-2 shadow-lg">Ești offline  datele pot
fi limitate</div>;
}

```

Include `OfflineIndicator` în `MainLayout` (sub `<main>` sau la finalul body-ului).

## STEP 9 — Teste rapide & Definition of Done

**Teste manuale:** - Chrome DevTools: emulează iPhone SE (320px), iPad, 1440p, 4K. - `Tab` / `Shift+Tab` prin pagini — focus vizibil, fără capcane. - TV: în DevTools forțează `data-mode="tv"` pe `<html>` și navighează cu săgeți.

**DoD:** - Fără scroll horizontal între **320px–4K**. - Toate controalele au  $\geq 44 \times 44$  (TV:  $\geq 64 \times 64$  pentru butoane primare). - Navigație cu săgeți pe grile/meniuri; `Enter` activează, `Escape/Backspace` dă „înapoi”. - PWA instalabilă; offline shell funcțional; indicator offline vizibil când cade rețeaua.

## PROMPTURI „COPY-PASTE” PENTRU CURSOR


### Prompt A — Aplică STEP 1-2 global

Analizează repo-ul și aplică modificările din „STEP 1” și „STEP 2” din runbook-ul furnizat:

- Actualizează `tailwind.config.js` (container + screens xs, 2xl, tv).
- Adaugă regulile din `src/index.css`.
- Adaugă snippetul de detectare mod în `index.html`.
- Refactorizează `MainLayout.jsx` conform exemplului.
- Înlocuiește containerele custom cu utilitarul Tailwind `container` acolo unde se folosea `max-w-7xl` și paddingurile repetate.

Nu altera logica paginilor. Creează commit separat „feat(responsive): container + fluid nav + typography clamp”.

### Prompt B — Componentizare butoane & icon-buttons

Creează/actualizează `src/components/ui/Button.jsx` cu implementarea din runbook (sizes sm/md/lg/tv, IconButton). Înlocuiește în repo butoanele icon-only (X, , emoji) cu `<IconButton aria-label corespunzător>`. Commit: „feat(ui): touch targets  $\geq 44$ px + IconButton”.

### Prompt C — Tooltips/Popover touch-friendly

Adaugă hook-ul `useInteractionMode` și adaptează componentele Tooltip/Popover să comute pe click în mod touch/tv și pe hover în desktop, conform codului din runbook. Commit: „fix(ux): hover-only eliminat, suport touch/tv”.

### Prompt D — TV safe area + navigație spațială

Adaugă `styles/tv.css`, componenta `SafeArea` și hook-ul `useSpatialNavigation`. Aplică navigația spațială pe cel puțin o pagină cu grid de carduri (ex: Dashboard). Adaugă clase `tv:` (dimensiuni mărite) unde e relevant. Commit: „feat(tv): safe-area + spatial navigation”.

### Prompt E — Fixuri punctuale & signature canvas

Execută căutările/înlocuirile din STEP 7 și fă canvas-ul de semnătură responsiv conform snippetului. Commit: "refactor(responsive): grids, modals, signature canvas".

### Prompt F — PWA polish minim

Actualizează configurația VitePWA din vite.config.js conform exemplului din runbook și adaugă un indicator offline. Commit: "chore(pwa): manifest polish + offline indicator".

---

## Observații finale

- Păstrează commituri mici, tematice, în ordinea pașilor.
- După fiecare pas: rulează Lighthouse (Mobile și Desktop). Ținte: **≥90** la Performance/Best Practices/Accessibility/PWA.
- Pentru TV real: testează pe un Android TV (browser sau WebView) — mod `data-mode=tv` va declanșa stilurile adecvate.