Layout

// app/\_layout.tsx

import { Stack } from 'expo-router'

import { TamaguiProvider, Theme } from 'tamagui'

import tamagui from '../tamagui.config'

export default function Layout() {

  return (

    <TamaguiProvider config={tamagui}>

      {/\* thème global (change en "dark" si tu veux forcer le sombre) \*/}

      <Theme name="light">

        <Stack screenOptions={{ headerShown: false }}>

          {/\* Accueil (hero + bouton ENTRER) \*/}

          <Stack.Screen name="index" />

          {/\* Sélection des ingrédients \*/}

          <Stack.Screen

            name="ingredients"

            options={{ headerShown: true, title: 'Ingrédients' }}

          />

          {/\* Écran convertisseur (kawaii) \*/}

          <Stack.Screen

            name="convertisseur"

            options={{ headerShown: true, title: 'Convertisseur' }}

          />

        </Stack>

      </Theme>

    </TamaguiProvider>

  )

}

Convertisseur

import { router } from 'expo-router'

import React from 'react'

import { StyleSheet, Text, TextInput, TouchableOpacity, View } from 'react-native'

export default function ConvertisseurScreen() {

  return (

    <View style={styles.container}>

      <Text style={styles.title}>✨ Convertisseur Kawaii 💖</Text>

      <View style={styles.card}>

        <Text style={styles.label}>Valeur à convertir</Text>

        <TextInput style={styles.input} placeholder="ex: 2 c. à s." placeholderTextColor="#ff8fcd" />

        <Text style={styles.label}>Ingrédient</Text>

        <TextInput style={styles.input} placeholder="ex: farine" placeholderTextColor="#ff8fcd" />

        <TouchableOpacity style={styles.cta} onPress={() => {}}>

          <Text style={styles.ctaText}>💫 Convertir</Text>

        </TouchableOpacity>

      </View>

      <TouchableOpacity onPress={() => router.back()} style={{ marginTop: 16 }}>

        <Text style={{ color: '#ff4fa2', fontWeight: '700' }}>⬅️ Retour</Text>

      </TouchableOpacity>

    </View>

  )

}

const styles = StyleSheet.create({

  container: { flex: 1, backgroundColor: '#FFEEFC', alignItems: 'center', justifyContent: 'center', padding: 20 },

  title: { fontSize: 26, fontWeight: '900', color: '#FF4FA2', marginBottom: 20, textShadowColor: '#fff', textShadowOffset: { width: 1, height: 1 }, textShadowRadius: 3 },

  card: { width: '92%', backgroundColor: '#fff', borderRadius: 24, padding: 20, shadowColor: '#FF8FCD', shadowOpacity: 0.2, shadowRadius: 10, elevation: 8 },

  label: { fontSize: 15, fontWeight: '700', color: '#FF69B4', marginBottom: 6 },

  input: { backgroundColor: '#FFF0FA', borderRadius: 14, borderWidth: 2, borderColor: '#FFB6F9', padding: 12, fontSize: 16, color: '#FF4FA2', marginBottom: 14 },

  cta: { backgroundColor: '#FF92E0', borderRadius: 999, paddingVertical: 14, alignItems: 'center', elevation: 4, shadowColor: '#FF4FA2', shadowOpacity: 0.3, shadowRadius: 6 },

  ctaText: { color: '#fff', fontWeight: '900', fontSize: 18 },

})

Index

// app/index.tsx

import { router } from 'expo-router'

import React from 'react'

import { Image, StyleSheet, Text, TouchableOpacity, useWindowDimensions, View } from 'react-native'

export default function HomeScreen() {

  const { height, width } = useWindowDimensions()

  // image = ~60% de la hauteur écran, bornée

  const IMG\_H = Math.round(Math.min(540, Math.max(320, height \* 0.60)))

  const go = () => router.push('/ingredients') // nouvelle destination

  return (

    <View style={styles.container}>

      {/\* IMAGE – hauteur explicite pour éviter toute disparition \*/}

      <TouchableOpacity style={[styles.imageWrapper, { height: IMG\_H }]} onPress={go} activeOpacity={0.9}>

        <Image

          // ⚠️ adapte l’extension si nécessaire (.webp/.png/.jpg)

          source={require('../assets/hero-splash.webp')}

          style={styles.image}

          resizeMode="contain"   // ne coupe ni haut ni bas

        />

      </TouchableOpacity>

      {/\* BOUTON – plus haut (moins d’espace bas) \*/}

      <TouchableOpacity style={styles.button} onPress={go} activeOpacity={0.9}>

        <Text style={styles.buttonText}>ENTRER</Text>

      </TouchableOpacity>

    </View>

  )

}

const ORANGE\_FOND = '#F7A14D'

const BEIGE\_BTN   = '#FBE9D0'

const BRUN        = '#6B3E1D'

const styles = StyleSheet.create({

  container: {

    flex: 1,

    backgroundColor: ORANGE\_FOND,

    alignItems: 'center',

    justifyContent: 'center',

    paddingHorizontal: 20,

    paddingTop: 24,

    paddingBottom: 12, // 👈 moins d’espace bas

  },

  imageWrapper: {

    width: '100%',

    marginTop: 8,

    marginBottom: 12,  // 👈 bouton remonte

    borderRadius: 16,

    overflow: 'hidden',

  },

  image: {

    width: '100%',

    height: '100%',

  },

  button: {

    backgroundColor: BEIGE\_BTN,

    borderRadius: 28,

    borderWidth: 2,

    borderColor: BRUN,

    paddingVertical: 10,     // fin

    paddingHorizontal: 56,   // allongé

    // remonte visuellement le bouton

    marginBottom: 8,

    // petite ombre douce

    shadowColor: '#000',

    shadowOpacity: 0.12,

    shadowRadius: 10,

    elevation: 6,

  },

  buttonText: {

    fontSize: 18,

    fontWeight: '800',

    color: BRUN,

    letterSpacing: 0.5,

  },

})

Ingredients

// app/ingredients.tsx

import { router } from 'expo-router'

import React, { useMemo, useState } from 'react'

import { ScrollView, StyleSheet, Text, TextInput, TouchableOpacity, View } from 'react-native'

const DB = [

  'Farine','Sucre','Sel','Beurre','Huile','Lait',

  'Riz','Pâtes','Cacao','Amande','Noisette',

  'Carotte','Pomme','Poire','Oignon','Ail','Tomate',

  'Pommes de terre','Courgette','Poivron','Citron'

]

export default function IngredientsScreen() {

  const [q, setQ] = useState('')

  const [sel, setSel] = useState<string[]>([])

  const list = useMemo(() => {

    const s = q.trim().toLowerCase()

    return s ? DB.filter(x => x.toLowerCase().includes(s)) : DB

  }, [q])

  const toggle = (name: string) =>

    setSel(prev => prev.includes(name) ? prev.filter(x => x !== name) : [...prev, name])

  const validate = () => {

    // plus tard: router.push({ pathname:'/convertisseur', params:{ items: JSON.stringify(sel) }})

    router.push('/convertisseur') // pour l’instant on enchaîne vers l’écran suivant

  }

  return (

    <View style={st.container}>

      <Text style={st.title}>🧺 Ingrédients</Text>

      <View style={st.card}>

        <TextInput

          style={st.input}

          value={q}

          onChangeText={setQ}

          placeholder="Rechercher un ingrédient…"

          placeholderTextColor="#ff8fcd"

        />

        <ScrollView contentContainerStyle={st.chipsWrap} keyboardShouldPersistTaps="handled">

          {list.map((name) => {

            const on = sel.includes(name)

            return (

              <TouchableOpacity

                key={name}

                onPress={() => toggle(name)}

                activeOpacity={0.9}

                style={[st.chip, on && st.chipOn]}

              >

                <Text style={[st.chipText, on && st.chipTextOn]}>{name}</Text>

              </TouchableOpacity>

            )

          })}

        </ScrollView>

        <TouchableOpacity

          onPress={validate}

          disabled={sel.length === 0}

          activeOpacity={0.9}

          style={[st.cta, sel.length === 0 && { opacity: 0.5 }]}

        >

          <Text style={st.ctaText}>Valider ({sel.length})</Text>

        </TouchableOpacity>

      </View>

    </View>

  )

}

const st = StyleSheet.create({

  container: { flex: 1, backgroundColor: '#FFEEFC', padding: 18, paddingTop: 36 },

  title: { fontSize: 26, fontWeight: '900', color: '#FF4FA2', marginBottom: 14 },

  card: {

    flex: 1, backgroundColor: '#fff', borderRadius: 22, padding: 14,

    shadowColor: '#FF8FCD', shadowOpacity: 0.18, shadowRadius: 10, elevation: 6

  },

  input: {

    backgroundColor: '#FFF0FA', borderRadius: 14, borderWidth: 2, borderColor: '#FFB6F9',

    paddingHorizontal: 14, paddingVertical: 10, fontSize: 16, color: '#FF4FA2', marginBottom: 10

  },

  chipsWrap: { flexDirection: 'row', flexWrap: 'wrap', gap: 10, paddingVertical: 6 },

  chip: {

    paddingVertical: 10, paddingHorizontal: 14,

    backgroundColor: '#FFE4F6', borderRadius: 999, borderWidth: 2, borderColor: '#FFB6F9'

  },

  chipOn: { backgroundColor: '#FF92E0', borderColor: '#FF4FA2' },

  chipText: { color: '#FF4FA2', fontWeight: '700' },

  chipTextOn: { color: '#fff' },

  cta: {

    marginTop: 14, backgroundColor: '#FF92E0', borderRadius: 999, alignItems: 'center',

    paddingVertical: 14, shadowColor: '#FF4FA2', shadowOpacity: 0.28, shadowRadius: 6, elevation: 5

  },

  ctaText: { color: '#fff', fontSize: 18, fontWeight: '900' },

})

Results

import { router, useLocalSearchParams } from 'expo-router'

import React, { useMemo } from 'react'

import { Button, Card, Separator, Text, XStack, YStack } from 'tamagui'

// Même base que dans select (tu peux later factoriser dans /constants)

const DB: Record<string, {

  label: string

  gPerTbsp: number

  gPerTsp: number

  avgUnitG: number

  peeledFactor: number

}> = {

  farine:  { label:'Farine',       gPerTbsp:8,  gPerTsp:3,  avgUnitG:0,   peeledFactor:1.00 },

  sucre:   { label:'Sucre',        gPerTbsp:12, gPerTsp:4,  avgUnitG:0,   peeledFactor:1.00 },

  sel:     { label:'Sel fin',      gPerTbsp:18, gPerTsp:6,  avgUnitG:0,   peeledFactor:1.00 },

  huile:   { label:'Huile',        gPerTbsp:14, gPerTsp:5,  avgUnitG:0,   peeledFactor:1.00 },

  lait:    { label:'Lait',         gPerTbsp:15, gPerTsp:5,  avgUnitG:0,   peeledFactor:1.00 },

  riz:     { label:'Riz cru',      gPerTbsp:12, gPerTsp:4,  avgUnitG:0,   peeledFactor:1.00 },

  beurre:  { label:'Beurre',       gPerTbsp:14, gPerTsp:5,  avgUnitG:0,   peeledFactor:1.00 },

  carotte: { label:'Carotte',      gPerTbsp:0,  gPerTsp:0,  avgUnitG:60,  peeledFactor:0.90 },

  pomme:   { label:'Pomme',        gPerTbsp:0,  gPerTsp:0,  avgUnitG:180, peeledFactor:0.90 },

  oignon:  { label:'Oignon',       gPerTbsp:0,  gPerTsp:0,  avgUnitG:110, peeledFactor:0.88 },

  ail:     { label:'Gousse d’ail', gPerTbsp:0,  gPerTsp:0,  avgUnitG:4,   peeledFactor:0.95 },

}

function Row({ left, right }: { left: string; right: string }) {

  return (

    <XStack ai="center" jc="space-between">

      <Text>{left}</Text>

      <Text fontWeight="700">{right}</Text>

    </XStack>

  )

}

export default function Results() {

  const { items } = useLocalSearchParams<{ items?: string }>()

  const ids: string[] = useMemo(() => {

    try { return items ? JSON.parse(items) : [] } catch { return [] }

  }, [items])

  return (

    <YStack f={1} bg="$background" p="$4" gap="$4">

      <XStack ai="center" jc="space-between">

        <Text fontSize={22} fontWeight="800">Conversions</Text>

        <Button size="$3" variant="outlined" onPress={() => router.back()}>↩︎ Modifier</Button>

      </XStack>

      {ids.length === 0 && (

        <Text>Pas d’ingrédients sélectionnés.</Text>

      )}

      <YStack gap="$4" pb="$6">

        {ids.map((id) => {

          const d = DB[id]

          if (!d) return null

          const tbsp = d.gPerTbsp ? `${d.gPerTbsp} g` : '—'

          const tsp  = d.gPerTsp  ? `${d.gPerTsp} g`  : '—'

          const unit = d.avgUnitG ? `${d.avgUnitG} g` : '—'

          const peeled =

            d.avgUnitG && d.peeledFactor

              ? `${Math.round(d.avgUnitG \* d.peeledFactor)} g`

              : '—'

          return (

            <Card key={id} p="$4" br="$8" bordered elevate>

              <Text fontWeight="800" fontSize={18} mb="$3">{d.label}</Text>

              <YStack gap="$2">

                <Row left="1 c. à soupe ↔ g" right={tbsp} />

                <Row left="1 c. à café ↔ g"  right={tsp} />

                <Row left="1 pièce moyenne"  right={unit} />

                <Row left="Épluché (par pièce)" right={peeled} />

              </YStack>

              <Separator my="$3" />

              {/\* Exemples rapides utiles \*/}

              <YStack gap="$1">

                {!!d.gPerTbsp && <Text>2 c. à soupe ≈ <Text fontWeight="700">{d.gPerTbsp \* 2} g</Text></Text>}

                {!!d.gPerTsp  && <Text>3 c. à café ≈ <Text fontWeight="700">{d.gPerTsp \* 3} g</Text></Text>}

              </YStack>

            </Card>

          )

        })}

      </YStack>

    </YStack>

  )

}

Select

import { router } from 'expo-router'

import React, { useMemo, useState } from 'react'

import { Button, Card, Input, ScrollView, Text, XStack, YStack } from 'tamagui'

// Mini base d’ingrédients (exemples)

const INGREDIENTS = [

  { id: 'farine',      label: 'Farine',        gPerTbsp: 8,  gPerTsp: 3,  avgUnitG: 0,   peeledFactor: 1.00 },

  { id: 'sucre',       label: 'Sucre',         gPerTbsp: 12, gPerTsp: 4,  avgUnitG: 0,   peeledFactor: 1.00 },

  { id: 'sel',         label: 'Sel fin',       gPerTbsp: 18, gPerTsp: 6,  avgUnitG: 0,   peeledFactor: 1.00 },

  { id: 'huile',       label: 'Huile',         gPerTbsp: 14, gPerTsp: 5,  avgUnitG: 0,   peeledFactor: 1.00 },

  { id: 'lait',        label: 'Lait',          gPerTbsp: 15, gPerTsp: 5,  avgUnitG: 0,   peeledFactor: 1.00 },

  { id: 'riz',         label: 'Riz cru',       gPerTbsp: 12, gPerTsp: 4,  avgUnitG: 0,   peeledFactor: 1.00 },

  { id: 'beurre',      label: 'Beurre',        gPerTbsp: 14, gPerTsp: 5,  avgUnitG: 0,   peeledFactor: 1.00 },

  { id: 'carotte',     label: 'Carotte',       gPerTbsp: 0,  gPerTsp: 0,  avgUnitG: 60,  peeledFactor: 0.90 },

  { id: 'pomme',       label: 'Pomme',         gPerTbsp: 0,  gPerTsp: 0,  avgUnitG: 180, peeledFactor: 0.90 },

  { id: 'oignon',      label: 'Oignon',        gPerTbsp: 0,  gPerTsp: 0,  avgUnitG: 110, peeledFactor: 0.88 },

  { id: 'ail',         label: 'Gousse d’ail',  gPerTbsp: 0,  gPerTsp: 0,  avgUnitG: 4,   peeledFactor: 0.95 },

]

export default function Select() {

  const [q, setQ] = useState('')

  const [sel, setSel] = useState<string[]>([])

  const list = useMemo(() => {

    const s = q.trim().toLowerCase()

    return !s ? INGREDIENTS

      : INGREDIENTS.filter(i => i.label.toLowerCase().includes(s) || i.id.includes(s))

  }, [q])

  const toggle = (id: string) =>

    setSel(prev => prev.includes(id) ? prev.filter(x => x !== id) : [...prev, id])

  const goResults = () => {

    const payload = JSON.stringify(sel)

    router.push({ pathname: '/results', params: { items: payload } })

  }

  return (

    <YStack f={1} bg="$background" p="$4" gap="$4">

      <Text fontSize={22} fontWeight="800">Sélection d’ingrédients</Text>

      {/\* Recherche \*/}

      <Input

        placeholder="Rechercher… (ex: farine, sucre, carotte)"

        value={q}

        onChangeText={setQ}

      />

      {/\* Sélection en cours \*/}

      {sel.length > 0 && (

        <XStack flexWrap="wrap" gap="$2">

          {sel.map(id => {

            const it = INGREDIENTS.find(i => i.id === id)!

            return (

              <Button

                key={id}

                size="$3"

                bg="$blue10"

                px="$3"

                br="$8"

                onPress={() => toggle(id)}

              >

                <Text color="white">{it.label} ✕</Text>

              </Button>

            )

          })}

        </XStack>

      )}

      {/\* Liste filtrée \*/}

      <ScrollView style={{ flex: 1 }}>

        <YStack gap="$3" pb="$6">

          {list.map(it => {

            const on = sel.includes(it.id)

            return (

              <Card

                key={it.id}

                p="$3"

                br="$8"

                bordered

                bg={on ? '$blue4' : '$background'}

                pressStyle={{ scale: 0.98 }}

                onPress={() => toggle(it.id)}

              >

                <XStack ai="center" jc="space-between">

                  <Text fontSize={16} fontWeight="700">{it.label}</Text>

                  <Button size="$3" variant={on ? 'solid' : 'outlined'}>

                    {on ? 'Ajouté' : 'Ajouter'}

                  </Button>

                </XStack>

              </Card>

            )

          })}

        </YStack>

      </ScrollView>

      {/\* Valider \*/}

      <Button

        bg="$blue10"

        br="$10"

        px="$6"

        py="$3"

        disabled={sel.length === 0}

        opacity={sel.length === 0 ? 0.5 : 1}

        onPress={goResults}

      >

        <Text color="white" fontWeight="800">Valider ({sel.length})</Text>

      </Button>

    </YStack>

  )

}

App.json

{

  "expo": {

    "name": "Convertisseur",

    "slug": "Convertisseur",

    "version": "1.0.0",

    "orientation": "portrait",

    "icon": "./assets/images/icon.png",

    "scheme": "convertisseur",

    "userInterfaceStyle": "automatic",

    "newArchEnabled": true,

    "ios": {

      "supportsTablet": true

    },

    "android": {

      "adaptiveIcon": {

        "foregroundImage": "./assets/images/adaptive-icon.png",

        "backgroundColor": "#ffffff"

      },

      "edgeToEdgeEnabled": true

    },

    "web": {

      "bundler": "metro",

      "output": "static",

      "favicon": "./assets/images/favicon.png"

    },

    "plugins": [

      "expo-router",

      [

        "expo-splash-screen",

        {

          "image": "./assets/images/splash-icon.png",

          "imageWidth": 200,

          "resizeMode": "contain",

          "backgroundColor": "#ffffff"

        }

      ]

    ],

    "experiments": {

      "typedRoutes": true

    }

  }

}

Babel.config.json

module.exports = function (api) {

  api.cache(true)

  return {

    presets: ['babel-preset-expo'],

    plugins: [

      ['@tamagui/babel-plugin', { config: './tamagui.config.ts' }],

      'react-native-reanimated/plugin' // toujours en dernier

    ],

  }

}

Package.json

{

  "name": "convertisseur",

  "main": "expo-router/entry",

  "version": "1.0.0",

  "scripts": {

    "start": "expo start",

    "reset-project": "node ./scripts/reset-project.js",

    "android": "expo start --android",

    "ios": "expo start --ios",

    "web": "expo start --web",

    "lint": "expo lint"

  },

  "dependencies": {

    "@expo/vector-icons": "^14.1.0",

    "@react-navigation/bottom-tabs": "^7.3.10",

    "@react-navigation/elements": "^2.3.8",

    "@react-navigation/native": "^7.1.6",

    "@tamagui/config": "^1.132.20",

    "expo": "~53.0.20",

    "expo-blur": "~14.1.5",

    "expo-constants": "~17.1.7",

    "expo-font": "~13.3.2",

    "expo-haptics": "~14.1.4",

    "expo-image": "~2.4.0",

    "expo-linking": "~7.1.7",

    "expo-router": "~5.1.4",

    "expo-splash-screen": "~0.30.10",

    "expo-status-bar": "~2.2.3",

    "expo-symbols": "~0.4.5",

    "expo-system-ui": "~5.0.10",

    "expo-web-browser": "~14.2.0",

    "react": "19.0.0",

    "react-dom": "19.0.0",

    "react-native": "0.79.5",

    "react-native-gesture-handler": "~2.24.0",

    "react-native-reanimated": "~3.17.4",

    "react-native-safe-area-context": "5.4.0",

    "react-native-screens": "~4.11.1",

    "react-native-svg": "^15.12.1",

    "react-native-web": "~0.20.0",

    "react-native-webview": "13.13.5",

    "tamagui": "^1.132.20"

  },

  "devDependencies": {

    "@babel/core": "^7.25.2",

    "@tamagui/babel-plugin": "^1.132.20",

    "@types/react": "~19.0.10",

    "eslint": "^9.25.0",

    "eslint-config-expo": "~9.2.0",

    "typescript": "~5.8.3"

  },

  "private": true

}

Tamagui.config.ts

import { config as configBase } from '@tamagui/config/v3'

import { createTamagui } from 'tamagui'

const config = createTamagui(configBase)

export type Conf = typeof config

declare module 'tamagui' {

  interface TamaguiCustomConfig extends Conf {}

}

export default config

Results 250825 10.48

// app/results.tsx

import { router, useLocalSearchParams } from 'expo-router'

import React, { useMemo, useState } from 'react'

import { Image, ScrollView, StyleSheet, Text, TextInput, TouchableOpacity, View } from 'react-native'

import { msToMMSS, useTimer } from '../src/timerContext'

// DB statique

// @ts-ignore

const DB = require('../data/ingredients.json') as any[]

// Map d’images (facultatif)

let IMAGES: Record<string, any> = {}

try {

  // génération dans app/imageMap.ts (cas courant)

  // eslint-disable-next-line @typescript-eslint/no-var-requires

  IMAGES = require('../src/imageMap').IMAGES || {}

} catch {

  try {

    // génération alternative dans src/imageMap.ts

    // eslint-disable-next-line @typescript-eslint/no-var-requires

    IMAGES = require('../src/imageMap').IMAGES || {}

  } catch {}

}

// -------- Types (commentaires sans caractères spéciaux pour eviter warnings) --------

type Item = {

  id: string

  label: string

  avg\_unit\_g: number | null

  peeled\_yield: number | null

  juice\_ml\_per\_unit: number | null

  lgth\_g: number | null

  tbsp\_g: number | null

  tsp\_g: number | null

  density\_g\_ml?: number | null

  // Pâtes

  psta\_wter?: number | null

  psta\_slt?: number | null

  // Œufs (remplis si la colonne "egg" existe dans ton CSV)

  egg?: boolean | null

  egg\_s?: number | null   // poids moyen 1 œuf petit (g)

  egg\_m?: number | null   // poids moyen 1 œuf moyen (g)

  egg\_l?: number | null   // poids moyen 1 œuf gros  (g)

  whte\_pctge?: number | null // fraction du blanc (ex: 0.6)

  ylw\_pctge?: number | null  // fraction du jaune (ex: 0.4)

}

// -------- Helpers --------

const num = (s: string) => {

  const n = Number((s ?? '').toString().replace(',', '.'))

  return isNaN(n) ? 0 : n

}

function fmt(n: number, digits = 2) {

  if (!isFinite(n)) return '0'

  const r = Number(n.toFixed(digits))

  const s = String(r)

  return /\.\d+$/.test(s) ? s.replace(/\.?0+$/, '') : s

}

function fmtAllUnits(grams: number) {

  const g = Math.max(0, grams || 0)

  const mg = Math.round(g \* 1000)

  const kg = g / 1000

  return `${fmt(g)} g  |  ${fmt(mg, 0)} mg  |  ${fmt(kg, 3)} kg`

}

// Convertit potentiellement string -> number, accepte virgule

function toNumMaybe(v: any): number | null {

  if (v === undefined || v === null || v === '') return null

  const n = Number(String(v).replace(',', '.'))

  return Number.isFinite(n) ? n : null

}

/\*\* Champ avec rappel compact (apparait a droite quand une valeur est presente) \*/

function InputWithEcho(props: {

  value: string

  onChangeText: (t: string) => void

  placeholder: string

  echoLabel: string

  keyboardType?: 'default' | 'numeric'

}) {

  const { value, onChangeText, placeholder, echoLabel, keyboardType = 'numeric' } = props

  return (

    <View style={st.inputWrap}>

      <TextInput

        style={st.input}

        keyboardType={keyboardType}

        value={value}

        onChangeText={onChangeText}

        placeholder={placeholder}

        placeholderTextColor="#ff8fcd"

      />

      {!!value && (

        <Text numberOfLines={1} ellipsizeMode="tail" style={st.echo}>

          {echoLabel}: {value}

        </Text>

      )}

    </View>

  )

}

export default function Results() {

  const { running, remainingMs } = useTimer()

  const { items } = useLocalSearchParams<{ items?: string }>()

  const ids: string[] = useMemo(() => {

    try { return items ? JSON.parse(items) : [] } catch { return [] }

  }, [items])

  const data: Item[] = useMemo(() => {

    const map = Object.fromEntries((DB as Item[]).map(d => [d.id, d]))

    return ids.map(id => map[id]).filter(Boolean)

  }, [ids])

  return (

    <ScrollView

      style={st.container}

      contentContainerStyle={{ padding: 16, paddingTop: 28 }}

    >

      <View style={st.headerRow}>

        <Text style={st.h1}>Convertisseurs</Text>

        <View style={{ flexDirection: 'row', gap: 12 }}>

          <TouchableOpacity onPress={() => router.push('/timer')}>

            <Text style={st.link}>⏱️ Minuteur</Text>

          </TouchableOpacity>

          <TouchableOpacity onPress={() => router.back()}>

            <Text style={st.link}>↩︎ Modifier</Text>

          </TouchableOpacity>

        </View>

      </View>

      {running && (

        <TouchableOpacity

          onPress={() => router.push('/timer')}

          style={st.timerBanner}

          activeOpacity={0.9}

        >

          <Text style={st.timerBannerText}>⏱ Temps restant : {msToMMSS(remainingMs)} — toucher pour ouvrir</Text>

        </TouchableOpacity>

      )}

      {data.length === 0 && <Text>Aucun ingrédient sélectionné.</Text>}

      {data.map(d => (

        <IngredientCard key={d.id} d={d} />

      ))}

    </ScrollView>

  )

}

function Row({ left, right }: { left: string; right: string }) {

  return (

    <View style={st.row}>

      <Text style={st.k}>{left}</Text>

      <Text style={st.v}>{right}</Text>

    </View>

  )

}

function IngredientCard({ d }: { d: Item }) {

  // Etats saisies

  // Epluchage (g)

  const [qtyEpl, setQtyEpl] = useState('')

  const [qtyNon, setQtyNon] = useState('')

  // Quantite <-> Poids

  const [countNon, setCountNon] = useState('')

  const [countEpl, setCountEpl] = useState('')

  // Jus

  const [countJuice, setCountJuice] = useState('')

  const [volMl, setVolMl] = useState('')

  // Taille

  const [lengthCm, setLengthCm] = useState('')

  const [lenWeightG, setLenWeightG] = useState('')

  // Cuilleres

  const [tsp, setTsp] = useState('')

  const [tbsp, setTbsp] = useState('')

  const [weightToSpoons, setWeightToSpoons] = useState('')

  // Pates

  const [pastaG, setPastaG] = useState('')

  const [waterL, setWaterL] = useState('')

  // Œufs

  const [eggSize, setEggSize] = useState<'S' | 'M' | 'L'>('M')

  const [eggTotalG, setEggTotalG]   = useState('') // Blanc+Jaune voulu (g)

  const [eggWhitesG, setEggWhitesG] = useState('') // Blancs voulus (g)

  const [eggYolksG, setEggYolksG]   = useState('') // Jaunes voulus (g)

  const [eggCount, setEggCount]     = useState('') // Nb d'œufs

  // Constantes

  const density = d.density\_g\_ml ?? 1

  const tsp\_g = d.tsp\_g ?? (d.tbsp\_g ? d.tbsp\_g / 3 : null)

  const tbsp\_g = d.tbsp\_g ?? (tsp\_g ? tsp\_g \* 3 : null)

  // on considère que le module est visible si AU MOINS une des deux valeurs est renseignée

  const pastaW = toNumMaybe(d.psta\_wter)

  const pastaS = toNumMaybe(d.psta\_slt)

  const hasPasta = (pastaW !== null) || (pastaS !== null)

  const whitesPct = d.whte\_pctge ?? 0.6

  const yolkPct   = d.ylw\_pctge  ?? 0.4

  const eggW =

    eggSize === 'S' ? (d.egg\_s ?? 0) :

    eggSize === 'L' ? (d.egg\_l ?? 0) :

                      (d.egg\_m ?? 0)

  return (

    <View style={st.card}>

      {/\* Titre + image \*/}

      <View style={{ flexDirection: 'row', alignItems: 'center' }}>

        <Text style={[st.h2, { flex: 1 }]}>{d.label}</Text>

        {IMAGES[d.id] && (

          <Image source={IMAGES[d.id]} style={{ width: 44, height: 44, marginLeft: 8 }} resizeMode="contain" />

        )}

      </View>

      {/\* Infos cles \*/}

      {(d.avg\_unit\_g || d.peeled\_yield || d.juice\_ml\_per\_unit) && (

        <View style={st.section}>

          <Text style={st.sTitle}>Infos clés</Text>

          {d.avg\_unit\_g ? <Row left="Poids moyen (1 pièce)" right={`${fmt(d.avg\_unit\_g)} g`} /> : null}

          {d.peeled\_yield && d.avg\_unit\_g

            ? <Row left={`Poids épluché (×${fmt(d.peeled\_yield)})`} right={`${fmt(d.avg\_unit\_g \* d.peeled\_yield)} g`} />

            : null}

          {d.juice\_ml\_per\_unit

            ? <Row left="Jus moyen (1 pièce)" right={`${fmt(d.juice\_ml\_per\_unit)} ml (≈ ${fmt(d.juice\_ml\_per\_unit \* density)} g)`} />

            : null}

        </View>

      )}

      {/\* Epluché ⇆ Non épluché (grammes) \*/}

      {d.peeled\_yield ? (

        <View style={st.section}>

          <Text style={st.sTitle}>Épluché <Text style={st.arrow}>⇆</Text> Non épluché</Text>

          <InputWithEcho

            value={qtyEpl}

            onChangeText={setQtyEpl}

            placeholder="Quantité épluchée (g)"

            echoLabel="Épluché (g)"

          />

          <Row left="Quantité non épluchée" right={fmtAllUnits(num(qtyEpl) / (d.peeled\_yield || 1))} />

          <InputWithEcho

            value={qtyNon}

            onChangeText={setQtyNon}

            placeholder="Quantité non épluchée (g)"

            echoLabel="Non épl. (g)"

          />

          <Row left="Quantité épluchée" right={fmtAllUnits(num(qtyNon) \* (d.peeled\_yield || 1))} />

        </View>

      ) : null}

      {/\* Quantité ⇆ Poids \*/}

      {d.avg\_unit\_g ? (

        <View style={st.section}>

          <Text style={st.sTitle}>Quantité <Text style={st.arrow}>⇆</Text> Poids</Text>

          <InputWithEcho

            value={countNon}

            onChangeText={setCountNon}

            placeholder="Pièces non épl. (ex: 3)"

            echoLabel="Pièces non épl."

          />

          <Row left="Poids non épluché" right={fmtAllUnits(num(countNon) \* (d.avg\_unit\_g || 0))} />

          {d.peeled\_yield

            ? <Row left="Poids épluché" right={fmtAllUnits(num(countNon) \* (d.avg\_unit\_g || 0) \* (d.peeled\_yield || 1))} />

            : null}

          <InputWithEcho

            value={countEpl}

            onChangeText={setCountEpl}

            placeholder="Pièces épl. (ex: 3)"

            echoLabel="Pièces épl."

          />

          <Row left="Poids non épluché" right={fmtAllUnits(num(countEpl) \* (d.avg\_unit\_g || 0))} />

          {d.peeled\_yield

            ? <Row left="Poids épluché" right={fmtAllUnits(num(countEpl) \* (d.avg\_unit\_g || 0) \* (d.peeled\_yield || 1))} />

            : null}

        </View>

      ) : null}

      {/\* Jus \*/}

      {d.juice\_ml\_per\_unit ? (

        <View style={st.section}>

          <Text style={st.sTitle}>Quantité <Text style={st.arrow}>⇆</Text> Jus</Text>

          <InputWithEcho

            value={countJuice}

            onChangeText={setCountJuice}

            placeholder="Nombre de pièces (ex: 2 citrons)"

            echoLabel="Pièces"

          />

          <Row

            left="Volume"

            right={`${fmt(num(countJuice) \* (d.juice\_ml\_per\_unit || 0))} ml  |  ${fmt(num(countJuice) \* (d.juice\_ml\_per\_unit || 0) / 10)} cl  |  ${fmt(num(countJuice) \* (d.juice\_ml\_per\_unit || 0) / 1000)} l`}

          />

          <InputWithEcho

            value={volMl}

            onChangeText={setVolMl}

            placeholder="Volume ou poids voulu (ml ou g)"

            echoLabel="Voulu"

          />

          <Row left="Nombre de pièces estimé" right={`${fmt(Math.ceil(num(volMl) / (d.juice\_ml\_per\_unit || 1)))} `} />

        </View>

      ) : null}

      {/\* Taille ⇆ Poids \*/}

      {d.lgth\_g ? (

        <View style={st.section}>

          <Text style={st.sTitle}>Taille <Text style={st.arrow}>⇆</Text> Poids</Text>

          <InputWithEcho

            value={lengthCm}

            onChangeText={setLengthCm}

            placeholder="Longueur (cm)"

            echoLabel="Longueur (cm)"

          />

          <Row left="Poids estimé" right={`${fmt(num(lengthCm) \* (d.lgth\_g || 0))} g`} />

          <InputWithEcho

            value={lenWeightG}

            onChangeText={setLenWeightG}

            placeholder="Poids (g)"

            echoLabel="Poids (g)"

          />

          <Row left="Longueur estimée" right={`${fmt(num(lenWeightG) / (d.lgth\_g || 1))} cm`} />

        </View>

      ) : null}

      {/\* Cuillères ⇆ Poids \*/}

      {(tbsp\_g || tsp\_g) ? (

        <View style={st.section}>

          <Text style={st.sTitle}>Cuillères <Text style={st.arrow}>⇆</Text> Poids</Text>

          <InputWithEcho

            value={tsp}

            onChangeText={setTsp}

            placeholder="Cuillères à café (ex: 2)"

            echoLabel="c. à café"

          />

          <Row left="Poids" right={fmtAllUnits(num(tsp) \* (tsp\_g || 0))} />

          <InputWithEcho

            value={tbsp}

            onChangeText={setTbsp}

            placeholder="Cuillères à soupe (ex: 2)"

            echoLabel="c. à soupe"

          />

          <Row left="Poids" right={fmtAllUnits(num(tbsp) \* (tbsp\_g || 0))} />

          <Text style={[st.sTitle, { marginTop: 10 }]}>Poids <Text style={st.arrow}>⇆</Text> Cuillères</Text>

          <InputWithEcho

            value={weightToSpoons}

            onChangeText={setWeightToSpoons}

            placeholder="Poids (g) — ex: 15"

            echoLabel="Poids (g)"

          />

          <Row

            left="Équivalent"

            right={`${tsp\_g ? `${fmt(num(weightToSpoons) / tsp\_g, 2)} c. à café` : '— c. à café'}   |   ${tbsp\_g ? `${fmt(num(weightToSpoons) / tbsp\_g, 2)} c. à soupe` : '— c. à soupe'}`}

          />

        </View>

      ) : null}

      {/\* ŒUFS \*/}

{(d.egg || d.egg\_s || d.egg\_m || d.egg\_l) ? (

  <View style={st.section}>

    <Text style={st.sTitle}>Œufs</Text>

    {/\* Infos clés \*/}

    <View style={{ marginBottom: 6 }}>

      <Row left="Œuf petit" right="inférieur à 50 g (S)" />

      <Row left="Œuf moyen" right="entre 50 et 60 g (M)" />

      <Row left="Œuf gros"  right="entre 60 et 70 g (L)" />

    </View>

    <View style={{ marginBottom: 6 }}>

      <Text style={[st.k, { marginBottom: 2 }]}>Cuisson (départ eau bouillante)</Text>

      <Row left="Œufs pochés"     right="2 min" />

      <Row left="Œufs à la coque" right="3 min" />

      <Row left="Œufs durs"       right="9 min" />

    </View>

    {/\* Choix S / M / L \*/}

    <View style={st.eggSizeWrap}>

      {(['S','M','L'] as const).map(sz => {

        const on = eggSize === sz

        return (

          <TouchableOpacity

            key={sz}

            onPress={() => setEggSize(sz)}

            activeOpacity={0.9}

            style={[st.eggBtn, on && st.eggBtnOn]}

          >

            <Text style={[st.eggBtnTxt, on && st.eggBtnTxtOn]}>{sz}</Text>

          </TouchableOpacity>

        )

      })}

    </View>

    {/\* 1) Poids ⇆ Quantité : Blanc+Jaune \*/}

    <Text style={st.sTitle}>Poids <Text style={st.arrow}>⇆</Text> Quantité</Text>

    <InputWithEcho

      value={eggTotalG}

      onChangeText={setEggTotalG}

      placeholder="Pds voulu Blanc+Jaune (g)"

      echoLabel="Poids total (g)"

    />

    <Row

      left="Nombre d'œufs estimés"

      right={

        eggW > 0

          ? fmt(Math.ceil(num(eggTotalG) / eggW)) // arrondi vers le haut si tu préfères

          : '—'

      }

    />

    {/\* 2) Poids ⇆ Quantité : Blancs \*/}

    <InputWithEcho

      value={eggWhitesG}

      onChangeText={setEggWhitesG}

      placeholder="Poids voulu Blancs (g)"

      echoLabel="Blancs (g)"

    />

    <Row

      left="Nombre d'œufs estimés"

      right={

        (eggW > 0 && whitesPct > 0)

          ? fmt(Math.ceil(num(eggWhitesG) / (eggW \* whitesPct)))

          : '—'

      }

    />

    {/\* 3) Poids ⇆ Quantité : Jaunes \*/}

    <InputWithEcho

      value={eggYolksG}

      onChangeText={setEggYolksG}

      placeholder="Poids voulu Jaune (g)"

      echoLabel="Jaunes (g)"

    />

    <Row

      left="Nombre d'œufs estimés"

      right={

        (eggW > 0 && yolkPct > 0)

          ? fmt(Math.ceil(num(eggYolksG) / (eggW \* yolkPct)))

          : '—'

      }

    />

    {/\* 4) Quantité → Poids (g) \*/}

    <InputWithEcho

      value={eggCount}

      onChangeText={setEggCount}

      placeholder="Nombre d'œufs (ex: 2)"

      echoLabel="Œufs"

    />

    {(() => {

      const n = num(eggCount)

      const total = n \* eggW

      const whites = total \* (whitesPct || 0)

      const yolks  = total \* (yolkPct || 0)

      return (

        <>

          <Row left="Poids estimé (Blanc+Jaune)" right={`${fmt(total)} g`} />

          <Row left="Poids estimé (Blanc)"       right={`${fmt(whites)} g`} />

          <Row left="Poids estimé (Jaune)"       right={`${fmt(yolks)} g`} />

        </>

      )

    })()}

  </View>

) : null}

      {/\* Pates \*/}

      {/\* Pâtes → section si au moins une valeur est définie \*/}

{hasPasta && (

  <View style={st.section}>

    <Text style={st.sTitle}>Pâtes <Text style={st.arrow}>⇆</Text> Eau & Sel</Text>

    {/\* Quantité de pâtes → eau + sel \*/}

    <InputWithEcho

      value={pastaG}

      onChangeText={setPastaG}

      placeholder="Qtité de pâtes (g)"

      echoLabel="Pâtes (g)"

    />

    {(() => {

      const g = num(pastaG)

      const L = g \* (pastaW ?? 0)

      const cl = L \* 10

      const ml = L \* 1000

      const saltG = (pastaS ?? 0) \* g

      return (

        <>

          <Row left="Quantité d'eau" right={`${fmt(L, 3)} l  |  ${fmt(cl, 1)} cl  |  ${fmt(ml, 0)} ml`} />

          <Row left="Quantité de sel" right={fmtAllUnits(saltG)} />

        </>

      )

    })()}

    {/\* Quantité d'eau → sel \*/}

    <InputWithEcho

      value={waterL}

      onChangeText={setWaterL}

      placeholder="Quantité d'eau (l)"

      echoLabel="Eau (l)"

    />

    {(() => {

      const L2 = num(waterL)

      const saltG2 = L2 \* (pastaS ?? 0) \* 100

      return <Row left="Quantité de sel" right={fmtAllUnits(saltG2)} />

    })()}

    {/\* 👉 Message si une des 2 valeurs manque \*/}

    {(pastaW === null || pastaS === null) && (

      <Text style={st.tip}>

        Conseil : renseigne à la fois <Text style={{fontWeight: 'bold'}}>psta\_wter</Text> (L/g)

        et <Text style={{fontWeight: 'bold'}}>psta\_slt</Text> (g pour 100 g d'eau) dans le CSV pour tout activer.

      </Text>

    )}

  </View>

)}

{/\* 👉 Si aucune valeur n'est renseignée, afficher une note (hors section) \*/}

{!hasPasta && (

  <Text style={[st.tip, { marginTop: 6 }]}>

    Ajoute <Text style={{fontWeight: 'bold'}}>psta\_wter</Text> et/ou <Text style={{fontWeight: 'bold'}}>psta\_slt</Text> dans ton CSV pour activer le module “Pâtes”.

  </Text>

)}

    </View>

  )

}

// -------- Styles --------

// -------- Styles --------

const st = StyleSheet.create({

  container: { flex: 1, backgroundColor: '#FFEEFC' },

  headerRow: { flexDirection: 'row', justifyContent: 'space-between', alignItems: 'center', marginBottom: 12 },

  h1: { fontSize: 24, fontWeight: '900', color: '#FF4FA2' },

  link: { color: '#7c3aed', fontWeight: '700' },

  card: { backgroundColor: '#fff', borderRadius: 18, padding: 14, marginBottom: 14, shadowColor: '#FF8FCD', shadowOpacity: 0.16, shadowRadius: 8, elevation: 5 },

  h2: { fontSize: 18, fontWeight: '900', color: '#FF4FA2', marginBottom: 8 },

  timerBanner: {

    backgroundColor: '#FF92E0',

    borderRadius: 16,

    paddingVertical: 10,

    paddingHorizontal: 12,

    marginBottom: 10,

    shadowColor: '#FF4FA2',

    shadowOpacity: 0.2,

    shadowRadius: 6,

    elevation: 5,

  },

  timerBannerText: {

    color: '#fff',

    fontWeight: '900',

    textAlign: 'center',

  },

  section: { marginTop: 8 },

  sTitle: { fontWeight: '800', marginBottom: 6, color: '#444' },

  arrow: { fontSize: 18, fontWeight: '900', color: '#FF4FA2' },

  row: { flexDirection: 'row', justifyContent: 'space-between', alignItems: 'center', marginVertical: 3 },

  k: { color: '#555' },

  v: { fontWeight: '800', color: '#222' },

  inputWrap: { position: 'relative', marginTop: 6 },

  input: {

    backgroundColor: '#FFF0FA',

    borderRadius: 14,

    borderWidth: 2,

    borderColor: '#FFB6F9',

    paddingHorizontal: 12,

    paddingVertical: 10,

    fontSize: 16,

    color: '#FF4FA2',

  },

  echo: {

    position: 'absolute',

    right: 10,

    bottom: 8,

    fontSize: 12,

    fontStyle: 'italic',

    color: '#9a3aa5',

    maxWidth: '55%',

  },

  tip: { marginTop: 6, color: '#6b7280', fontStyle: 'italic' },

  eggSizeWrap: {

  flexDirection: 'row',

  gap: 8,

  marginBottom: 8,

},

eggBtn: {

  paddingVertical: 6,

  paddingHorizontal: 12,

  borderRadius: 999,

  borderWidth: 2,

  borderColor: '#FFB6F9',

  backgroundColor: '#FFE4F6',

},

eggBtnOn: {

  backgroundColor: '#FF92E0',

  borderColor: '#FF4FA2',

},

eggBtnTxt: {

  color: '#FF4FA2',

  fontWeight: '800',

  fontSize: 16,

},

eggBtnTxtOn: {

  color: '#fff',

},

});

Results 250825 11.14

// app/results.tsx

import { router, useLocalSearchParams } from 'expo-router'

import React, { useMemo, useState } from 'react'

import { Image, ScrollView, StyleSheet, Text, TextInput, TouchableOpacity, View } from 'react-native'

import { msToMMSS, useTimer } from '../src/timerContext'

// DB statique

// @ts-ignore

const DB = require('../data/ingredients.json') as any[]

// Map d’images (facultatif)

let IMAGES: Record<string, any> = {}

try {

  // cas courant (généré par le script)

  // eslint-disable-next-line @typescript-eslint/no-var-requires

  IMAGES = require('./imageMap').IMAGES || {}

} catch {

  try {

    // alternative

    // eslint-disable-next-line @typescript-eslint/no-var-requires

    IMAGES = require('../src/imageMap').IMAGES || {}

  } catch {}

}

// -------- Types --------

type Item = {

  id: string

  label: string

  avg\_unit\_g: number | null

  peeled\_yield: number | null

  juice\_ml\_per\_unit: number | null

  lgth\_g: number | null

  tbsp\_g: number | null

  tsp\_g: number | null

  density\_g\_ml?: number | null

  // Pâtes

  psta\_wter?: number | null   // litres d’eau par gramme de pâtes

  psta\_slt?: number | null    // grammes de sel par 100 g d’eau

  // Œufs

  egg\_s?: number | null

  egg\_m?: number | null

  egg\_l?: number | null

  whte\_pctge?: number | null  // proportion blanc (0..1)

  ylw\_pctge?: number | null   // proportion jaune (0..1)

}

// -------- Helpers --------

const num = (s: string) => {

  const n = Number((s ?? '').toString().replace(',', '.'))

  return isNaN(n) ? 0 : n

}

function fmt(n: number, digits = 2) {

  if (!isFinite(n)) return '0'

  const r = Number(n.toFixed(digits))

  const s = String(r)

  return /\.\d+$/.test(s) ? s.replace(/\.?0+$/, '') : s

}

function fmtAllUnits(grams: number) {

  const g = Math.max(0, grams || 0)

  const mg = Math.round(g \* 1000)

  const kg = g / 1000

  return `${fmt(g)} g  |  ${fmt(mg, 0)} mg  |  ${fmt(kg, 3)} kg`

}

// Convertit potentiellement string/unknown -> number, accepte virgule

function toNumMaybe(v: any): number | null {

  if (v === undefined || v === null || v === '') return null

  const n = Number(String(v).replace(',', '.'))

  return Number.isFinite(n) ? n : null

}

/\*\* Champ avec rappel compact (apparaît à droite quand une valeur est présente) \*/

function InputWithEcho(props: {

  value: string

  onChangeText: (t: string) => void

  placeholder: string

  echoLabel: string

  keyboardType?: 'default' | 'numeric'

}) {

  const { value, onChangeText, placeholder, echoLabel, keyboardType = 'numeric' } = props

  return (

    <View style={st.inputWrap}>

      <TextInput

        style={st.input}

        keyboardType={keyboardType}

        value={value}

        onChangeText={onChangeText}

        placeholder={placeholder}

        placeholderTextColor="#ff8fcd"

      />

      {!!value && (

        <Text numberOfLines={1} ellipsizeMode="tail" style={st.echo}>

          {echoLabel}: {value}

        </Text>

      )}

    </View>

  )

}

export default function Results() {

  const { running, remainingMs } = useTimer()

  const { items } = useLocalSearchParams<{ items?: string }>()

  const ids: string[] = useMemo(() => {

    try { return items ? JSON.parse(items) : [] } catch { return [] }

  }, [items])

  const data: Item[] = useMemo(() => {

    const map = Object.fromEntries((DB as Item[]).map(d => [d.id, d]))

    return ids.map(id => map[id]).filter(Boolean)

  }, [ids])

  return (

    <ScrollView

      style={st.container}

      contentContainerStyle={{ padding: 16, paddingTop: 28 }}

    >

      <View style={st.headerRow}>

        <Text style={st.h1}>Convertisseurs</Text>

        <View style={{ flexDirection: 'row', gap: 12 }}>

          <TouchableOpacity onPress={() => router.push('/timer')}>

            <Text style={st.link}>⏱️ Minuteur</Text>

          </TouchableOpacity>

          <TouchableOpacity onPress={() => router.back()}>

            <Text style={st.link}>↩︎ Modifier</Text>

          </TouchableOpacity>

        </View>

      </View>

      {running && (

        <TouchableOpacity

          onPress={() => router.push('/timer')}

          style={st.timerBanner}

          activeOpacity={0.9}

        >

          <Text style={st.timerBannerText}>⏱ Temps restant : {msToMMSS(remainingMs)} — toucher pour ouvrir</Text>

        </TouchableOpacity>

      )}

      {data.length === 0 && <Text>Aucun ingrédient sélectionné.</Text>}

      {data.map(d => (

        <IngredientCard key={d.id} d={d} />

      ))}

    </ScrollView>

  )

}

function Row({ left, right }: { left: string; right: string }) {

  return (

    <View style={st.row}>

      <Text style={st.k}>{left}</Text>

      <Text style={st.v}>{right}</Text>

    </View>

  )

}

function IngredientCard({ d }: { d: Item }) {

  // États saisies

  // Épluchage (g)

  const [qtyEpl, setQtyEpl] = useState('')

  const [qtyNon, setQtyNon] = useState('')

  // Quantité <-> Poids

  const [countNon, setCountNon] = useState('')

  const [countEpl, setCountEpl] = useState('')

  // Jus

  const [countJuice, setCountJuice] = useState('')

  const [volMl, setVolMl] = useState('')

  // Taille

  const [lengthCm, setLengthCm] = useState('')

  const [lenWeightG, setLenWeightG] = useState('')

  // Cuillères

  const [tsp, setTsp] = useState('')

  const [tbsp, setTbsp] = useState('')

  const [weightToSpoons, setWeightToSpoons] = useState('')

  // Pâtes

  const [pastaG, setPastaG] = useState('')

  const [waterL, setWaterL] = useState('')

  // Œufs

  const [eggSize, setEggSize] = useState<'S' | 'M' | 'L'>('M')

  const [eggWholeG, setEggWholeG] = useState('')

  const [eggWhiteG, setEggWhiteG] = useState('')

  const [eggYolkG, setEggYolkG] = useState('')

  const [eggCount, setEggCount] = useState('')

  // Constantes

  const density = d.density\_g\_ml ?? 1

  const tsp\_g = d.tsp\_g ?? (d.tbsp\_g ? d.tbsp\_g / 3 : null)

  const tbsp\_g = d.tbsp\_g ?? (tsp\_g ? tsp\_g \* 3 : null)

  // Pâtes

  const pastaW = toNumMaybe(d.psta\_wter)

  const pastaS = toNumMaybe(d.psta\_slt)

  const hasPasta = (pastaW !== null) || (pastaS !== null)

  // Œufs (normalisation)

  const eggS = toNumMaybe(d.egg\_s)

  const eggM = toNumMaybe(d.egg\_m)

  const eggL = toNumMaybe(d.egg\_l)

  const whitePct = toNumMaybe(d.whte\_pctge) ?? 0.6

  const yolkPct  = toNumMaybe(d.ylw\_pctge) ?? 0.4

  const hasEgg = eggS !== null || eggM !== null || eggL !== null

  const eggUnit = () => {

    if (eggSize === 'S') return eggS ?? eggM ?? eggL ?? null

    if (eggSize === 'L') return eggL ?? eggM ?? eggS ?? null

    return eggM ?? eggS ?? eggL ?? null

  }

  return (

    <View style={st.card}>

      {/\* Titre + image \*/}

      <View style={{ flexDirection: 'row', alignItems: 'center' }}>

        <Text style={[st.h2, { flex: 1 }]}>{d.label}</Text>

        {IMAGES[d.id] && (

          <Image source={IMAGES[d.id]} style={{ width: 44, height: 44, marginLeft: 8 }} resizeMode="contain" />

        )}

      </View>

      {/\* Infos clés \*/}

      {(d.avg\_unit\_g || d.peeled\_yield || d.juice\_ml\_per\_unit) && (

        <View style={st.section}>

          <Text style={st.sTitle}>Infos clés</Text>

          {d.avg\_unit\_g ? <Row left="Poids moyen (1 pièce)" right={`${fmt(d.avg\_unit\_g)} g`} /> : null}

          {d.peeled\_yield && d.avg\_unit\_g

            ? <Row left={`Poids épluché (×${fmt(d.peeled\_yield)})`} right={`${fmt(d.avg\_unit\_g \* d.peeled\_yield)} g`} />

            : null}

          {d.juice\_ml\_per\_unit

            ? <Row left="Jus moyen (1 pièce)" right={`${fmt(d.juice\_ml\_per\_unit)} ml (≈ ${fmt(d.juice\_ml\_per\_unit \* density)} g)`} />

            : null}

        </View>

      )}

      {/\* Épluché ⇆ Non épluché (grammes) \*/}

      {d.peeled\_yield ? (

        <View style={st.section}>

          <Text style={st.sTitle}>Épluché <Text style={st.arrow}>⇆</Text> Non épluché</Text>

          <InputWithEcho

            value={qtyEpl}

            onChangeText={setQtyEpl}

            placeholder="Quantité épluchée (g)"

            echoLabel="Épluché (g)"

          />

          <Row left="Quantité non épluchée" right={fmtAllUnits(num(qtyEpl) / (d.peeled\_yield || 1))} />

          <InputWithEcho

            value={qtyNon}

            onChangeText={setQtyNon}

            placeholder="Quantité non épluchée (g)"

            echoLabel="Non épl. (g)"

          />

          <Row left="Quantité épluchée" right={fmtAllUnits(num(qtyNon) \* (d.peeled\_yield || 1))} />

        </View>

      ) : null}

      {/\* Quantité ⇆ Poids \*/}

      {d.avg\_unit\_g ? (

        <View style={st.section}>

          <Text style={st.sTitle}>Quantité <Text style={st.arrow}>⇆</Text> Poids</Text>

          <InputWithEcho

            value={countNon}

            onChangeText={setCountNon}

            placeholder="Pièces non épl. (ex: 3)"

            echoLabel="Pièces non épl."

          />

          <Row left="Poids non épluché" right={fmtAllUnits(num(countNon) \* (d.avg\_unit\_g || 0))} />

          {d.peeled\_yield

            ? <Row left="Poids épluché" right={fmtAllUnits(num(countNon) \* (d.avg\_unit\_g || 0) \* (d.peeled\_yield || 1))} />

            : null}

          <InputWithEcho

            value={countEpl}

            onChangeText={setCountEpl}

            placeholder="Pièces épl. (ex: 3)"

            echoLabel="Pièces épl."

          />

          <Row left="Poids non épluché" right={fmtAllUnits(num(countEpl) \* (d.avg\_unit\_g || 0))} />

          {d.peeled\_yield

            ? <Row left="Poids épluché" right={fmtAllUnits(num(countEpl) \* (d.avg\_unit\_g || 0) \* (d.peeled\_yield || 1))} />

            : null}

        </View>

      ) : null}

      {/\* Jus \*/}

      {d.juice\_ml\_per\_unit ? (

        <View style={st.section}>

          <Text style={st.sTitle}>Quantité <Text style={st.arrow}>⇆</Text> Jus</Text>

          <InputWithEcho

            value={countJuice}

            onChangeText={setCountJuice}

            placeholder="Nombre de pièces (ex: 2 citrons)"

            echoLabel="Pièces"

          />

          <Row

            left="Volume"

            right={`${fmt(num(countJuice) \* (d.juice\_ml\_per\_unit || 0))} ml  |  ${fmt(num(countJuice) \* (d.juice\_ml\_per\_unit || 0) / 10)} cl  |  ${fmt(num(countJuice) \* (d.juice\_ml\_per\_unit || 0) / 1000)} l`}

          />

          <InputWithEcho

            value={volMl}

            onChangeText={setVolMl}

            placeholder="Volume ou poids voulu (ml ou g)"

            echoLabel="Voulu"

          />

          <Row left="Nombre de pièces estimé" right={`${fmt(Math.ceil(num(volMl) / (d.juice\_ml\_per\_unit || 1)))} `} />

        </View>

      ) : null}

      {/\* Taille ⇆ Poids \*/}

      {d.lgth\_g ? (

        <View style={st.section}>

          <Text style={st.sTitle}>Taille <Text style={st.arrow}>⇆</Text> Poids</Text>

          <InputWithEcho

            value={lengthCm}

            onChangeText={setLengthCm}

            placeholder="Longueur (cm)"

            echoLabel="Longueur (cm)"

          />

          <Row left="Poids estimé" right={`${fmt(num(lengthCm) \* (d.lgth\_g || 0))} g`} />

          <InputWithEcho

            value={lenWeightG}

            onChangeText={setLenWeightG}

            placeholder="Poids (g)"

            echoLabel="Poids (g)"

          />

          <Row left="Longueur estimée" right={`${fmt(num(lenWeightG) / (d.lgth\_g || 1))} cm`} />

        </View>

      ) : null}

      {/\* Cuillères ⇆ Poids \*/}

      {(tbsp\_g || tsp\_g) ? (

        <View style={st.section}>

          <Text style={st.sTitle}>Cuillères <Text style={st.arrow}>⇆</Text> Poids</Text>

          <InputWithEcho

            value={tsp}

            onChangeText={setTsp}

            placeholder="Cuillères à café (ex: 2)"

            echoLabel="c. à café"

          />

          <Row left="Poids" right={fmtAllUnits(num(tsp) \* (tsp\_g || 0))} />

          <InputWithEcho

            value={tbsp}

            onChangeText={setTbsp}

            placeholder="Cuillères à soupe (ex: 2)"

            echoLabel="c. à soupe"

          />

          <Row left="Poids" right={fmtAllUnits(num(tbsp) \* (tbsp\_g || 0))} />

          <Text style={[st.sTitle, { marginTop: 10 }]}>Poids <Text style={st.arrow}>⇆</Text> Cuillères</Text>

          <InputWithEcho

            value={weightToSpoons}

            onChangeText={setWeightToSpoons}

            placeholder="Poids (g) — ex: 15"

            echoLabel="Poids (g)"

          />

          <Row

            left="Équivalent"

            right={`${tsp\_g ? `${fmt(num(weightToSpoons) / tsp\_g, 2)} c. à café` : '— c. à café'}   |   ${tbsp\_g ? `${fmt(num(weightToSpoons) / tbsp\_g, 2)} c. à soupe` : '— c. à soupe'}`}

          />

        </View>

      ) : null}

      {/\* Œufs \*/}

      {hasEgg && (

        <View style={st.section}>

          <Text style={st.sTitle}>Œufs</Text>

          {/\* Infos clés \*/}

          <Row left="Œuf petit" right="&lt; 50 g (S)" />

          <Row left="Œuf moyen" right="50–60 g (M)" />

          <Row left="Œuf gros" right="60–70 g (L)" />

          <Text style={[st.sTitle, { marginTop: 6 }]}>Cuisson (départ eau bouillante)</Text>

          <Row left="Pochés" right="2 min" />

          <Row left="À la coque" right="3 min" />

          <Row left="Durs" right="9 min" />

          {/\* Choix S / M / L \*/}

          <View style={st.segmentWrap}>

            {(['S', 'M', 'L'] as const).map(size => {

              const on = eggSize === size

              return (

                <TouchableOpacity

                  key={size}

                  onPress={() => setEggSize(size)}

                  activeOpacity={0.9}

                  style={[st.segmentBtn, on && st.segmentOn]}

                >

                  <Text style={[st.segmentTxt, on && st.segmentTxtOn]}>{size}</Text>

                </TouchableOpacity>

              )

            })}

          </View>

          {/\* 1) Poids ⇆ Quantité : Pds voulu Blanc+Jaune (g) \*/}

          <Text style={[st.sTitle, { marginTop: 8 }]}>

            Poids <Text style={st.arrow}>⇆</Text> Quantité

          </Text>

          <InputWithEcho

            value={eggWholeG}

            onChangeText={setEggWholeG}

            placeholder="Pds voulu Blanc+Jaune (g)"

            echoLabel="Blanc+Jaune (g)"

          />

          {(() => {

            const w = num(eggWholeG)

            const unit = eggUnit()

            const eggs = unit ? w / unit : 0

            return <Row left="Nombre d'œufs estimés" right={`${Math.ceil(eggs)} œufs`} />

          })()}

          {/\* 2) Poids voulu Blancs (g) → nb d'œufs \*/}

          <InputWithEcho

            value={eggWhiteG}

            onChangeText={setEggWhiteG}

            placeholder="Poids voulu Blancs (g)"

            echoLabel="Blancs (g)"

          />

          {(() => {

            const w = num(eggWhiteG)

            const unit = eggUnit()

            const eggs = unit ? w / (unit \* whitePct) : 0

            return <Row left="Nombre d'œufs estimés" right={`${Math.ceil(eggs)} œufs`} />

          })()}

          {/\* 3) Poids voulu Jaune (g) → nb d'œufs \*/}

          <InputWithEcho

            value={eggYolkG}

            onChangeText={setEggYolkG}

            placeholder="Poids voulu Jaune (g)"

            echoLabel="Jaune (g)"

          />

          {(() => {

            const w = num(eggYolkG)

            const unit = eggUnit()

            const eggs = unit ? w / (unit \* yolkPct) : 0

            return <Row left="Nombre d'œufs estimés" right={`${Math.ceil(eggs)} œufs`} />

          })()}

          {/\* 4) Nombre d'œufs → poids estimé \*/}

          <InputWithEcho

            value={eggCount}

            onChangeText={setEggCount}

            placeholder="Nombre d'œufs (ex: 2)"

            echoLabel="Œufs"

          />

          {(() => {

            const n = num(eggCount)

            const unit = eggUnit() ?? 0

            const whole = unit \* n

            const whites = whole \* whitePct

            const yolks  = whole \* yolkPct

            return (

              <>

                <Row left="Blanc+Jaune" right={`${fmt(whole)} g`} />

                <Row left="Blanc"       right={`${fmt(whites)} g`} />

                <Row left="Jaune"       right={`${fmt(yolks)} g`} />

              </>

            )

          })()}

        </View>

      )}

      {/\* Pâtes \*/}

      {hasPasta && (

        <View style={st.section}>

          <Text style={st.sTitle}>Pâtes <Text style={st.arrow}>⇆</Text> Eau & Sel</Text>

          {/\* Qté de pâtes (g) \*/}

          <InputWithEcho

            value={pastaG}

            onChangeText={setPastaG}

            placeholder="Qtité de pâtes (g)"

            echoLabel="Pâtes (g)"

          />

          {(() => {

            const g = num(pastaG)

            const L = g \* (pastaW ?? 0) // litres d’eau

            const cl = L \* 10

            const ml = L \* 1000

            const saltG = (pastaS ?? 0) \* g // g de sel = psta\_slt \* g de pâtes

            return (

              <>

                <Row left="Quantité d'eau" right={`${fmt(L, 3)} l  |  ${fmt(cl, 1)} cl  |  ${fmt(ml, 0)} ml`} />

                <Row left="Quantité de sel" right={fmtAllUnits(saltG)} />

              </>

            )

          })()}

          {/\* Qté d’eau (l) → sel \*/}

          <InputWithEcho

            value={waterL}

            onChangeText={setWaterL}

            placeholder="Quantité d'eau (l)"

            echoLabel="Eau (l)"

          />

          {(() => {

            const L2 = num(waterL)

            const saltG2 = L2 \* (pastaS ?? 0) \* 100 // g de sel = eau(l) \* psta\_slt \* 100

            return <Row left="Quantité de sel" right={fmtAllUnits(saltG2)} />

          })()}

          {(pastaW === null || pastaS === null) && (

            <Text style={st.tip}>

              Conseil : renseigne <Text style={{ fontWeight: 'bold' }}>psta\_wter</Text> (L/g)

              et <Text style={{ fontWeight: 'bold' }}>psta\_slt</Text> (g / 100 g d'eau) dans le CSV pour tout activer.

            </Text>

          )}

        </View>

      )}

    </View>

  )

}

// -------- Styles --------

const st = StyleSheet.create({

  container: { flex: 1, backgroundColor: '#FFEEFC' },

  headerRow: { flexDirection: 'row', justifyContent: 'space-between', alignItems: 'center', marginBottom: 12 },

  h1: { fontSize: 24, fontWeight: '900', color: '#FF4FA2' },

  link: { color: '#7c3aed', fontWeight: '700' },

  card: { backgroundColor: '#fff', borderRadius: 18, padding: 14, marginBottom: 14, shadowColor: '#FF8FCD', shadowOpacity: 0.16, shadowRadius: 8, elevation: 5 },

  h2: { fontSize: 18, fontWeight: '900', color: '#FF4FA2', marginBottom: 8 },

  // Bandeau minuteur

  timerBanner: {

    backgroundColor: '#FF92E0',

    borderRadius: 16,

    paddingVertical: 10,

    paddingHorizontal: 12,

    marginBottom: 10,

    shadowColor: '#FF4FA2',

    shadowOpacity: 0.2,

    shadowRadius: 6,

    elevation: 5,

  },

  timerBannerText: { color: '#fff', fontWeight: '900', textAlign: 'center' },

  // Sections & lignes

  section: { marginTop: 8 },

  sTitle: { fontWeight: '800', marginBottom: 6, color: '#444' },

  arrow: { fontSize: 18, fontWeight: '900', color: '#FF4FA2' },

  row: { flexDirection: 'row', justifyContent: 'space-between', alignItems: 'center', marginVertical: 3 },

  k: { color: '#555' },

  v: { fontWeight: '800', color: '#222' },

  // Inputs + écho

  inputWrap: { position: 'relative', marginTop: 6 },

  input: {

    backgroundColor: '#FFF0FA',

    borderRadius: 14,

    borderWidth: 2,

    borderColor: '#FFB6F9',

    paddingHorizontal: 12,

    paddingVertical: 10,

    fontSize: 16,

    color: '#FF4FA2',

  },

  echo: {

    position: 'absolute',

    right: 10,

    bottom: 8,

    fontSize: 12,

    fontStyle: 'italic',

    color: '#9a3aa5',

    maxWidth: '55%',

  },

  // Segmented control S/M/L

  segmentWrap: { flexDirection: 'row', gap: 8, marginTop: 8 },

  segmentBtn: {

    paddingVertical: 8,

    paddingHorizontal: 14,

    borderRadius: 999,

    borderWidth: 2,

    borderColor: '#FFB6F9',

    backgroundColor: '#FFE4F6',

  },

  segmentOn: {

    backgroundColor: '#FF92E0',

    borderColor: '#FF4FA2',

  },

  segmentTxt: { color: '#FF4FA2', fontWeight: '800' },

  segmentTxtOn: { color: '#fff' },

  tip: { marginTop: 6, color: '#6b7280', fontStyle: 'italic' },

})

App.timer

// app/timer.tsx

import { useKeepAwake } from 'expo-keep-awake'; // npm i expo-keep-awake (facultatif mais pratique)

import { router } from 'expo-router'

import React, { useMemo, useRef } from 'react'

import { StyleSheet, Text, TextInput, TouchableOpacity, View } from 'react-native'

import { msToMMSS, useTimer } from '../src/timerContext'

function clamp(n: number, min: number, max: number) {

  return Math.max(min, Math.min(max, n))

}

export default function TimerScreen() {

  useKeepAwake() // évite que l’écran se mette en veille pendant le minuteur

  const { running, remainingMs, durationMs, start, pause, reset, setRemaining } = useTimer()

  // réglage local

  const minsRef = useRef('3')

  const secsRef = useRef('00')

  const setMins = (t: string) => { minsRef.current = t }

  const setSecs = (t: string) => { secsRef.current = t }

  const totalMs = useMemo(() => {

    const m = clamp(Number(minsRef.current.replace(',', '.')) || 0, 0, 999)

    const s = clamp(Number(secsRef.current.replace(',', '.')) || 0, 0, 59)

    return (m \* 60 + s) \* 1000

  }, [running, remainingMs]) // recalcul léger

  const display = msToMMSS(running ? remainingMs : (durationMs || totalMs))

  // Progress simple (barre)

  const progress = durationMs > 0 ? 1 - remainingMs / durationMs : 0

  const pct = Math.max(0, Math.min(1, progress))

  return (

    <View style={st.container}>

      <View style={st.header}>

        <TouchableOpacity onPress={() => router.back()}>

          <Text style={st.link}>↩︎ Retour</Text>

        </TouchableOpacity>

        <Text style={st.title}>⏱️ Minuteur</Text>

        <View style={{ width: 60 }} />

      </View>

      <View style={st.timeBox}>

        <Text style={st.timeText}>{display}</Text>

        <View style={st.progressTrack}>

          <View style={[st.progressFill, { width: `${pct \* 100}%` }]} />

        </View>

        <Text style={st.subTime}>

          {running ? 'En cours…' : (durationMs > 0 ? 'En pause' : 'Prêt')}

        </Text>

      </View>

      <View style={st.settingsCard}>

        <Text style={st.sectionTitle}>Réglage</Text>

        <View style={st.inputsRow}>

          <View style={st.inputWrap}>

            <Text style={st.label}>Minutes</Text>

            <TextInput

              style={st.input}

              keyboardType="numeric"

              defaultValue="3"

              onChangeText={t => setMins(String(t).replace(/[^\d,\.]/g, ''))}

              placeholder="0"

              placeholderTextColor="#ff8fcd"

              maxLength={3}

            />

          </View>

          <View style={st.inputWrap}>

            <Text style={st.label}>Secondes</Text>

            <TextInput

              style={st.input}

              keyboardType="numeric"

              defaultValue="00"

              onChangeText={t => setSecs(String(t).replace(/[^\d,\.]/g, ''))}

              placeholder="0"

              placeholderTextColor="#ff8fcd"

              maxLength={2}

            />

          </View>

        </View>

        <View style={st.presetRow}>

          {[1, 3, 5, 10].map(m => (

            <TouchableOpacity

              key={m}

              style={st.pill}

              onPress={() => {

                minsRef.current = String(m)

                secsRef.current = '00'

                if (!running) setRemaining(m \* 60 \* 1000)

              }}

            >

              <Text style={st.pillText}>{m} min</Text>

            </TouchableOpacity>

          ))}

        </View>

      </View>

      <View style={st.actionsRow}>

        {!running ? (

          <TouchableOpacity style={[st.btn, st.btnPrimary]} onPress={() => start(totalMs)} activeOpacity={0.9}>

            <Text style={st.btnText}>Démarrer</Text>

          </TouchableOpacity>

        ) : (

          <TouchableOpacity style={[st.btn, st.btnWarn]} onPress={pause} activeOpacity={0.9}>

            <Text style={st.btnText}>Pause</Text>

          </TouchableOpacity>

        )}

        <TouchableOpacity

          style={[st.btn, st.btnGhost]}

          onPress={() => reset()}

          onLongPress={() => { minsRef.current = '0'; secsRef.current = '00'; reset() }}

          activeOpacity={0.9}

        >

          <Text style={st.btnGhostText}>Réinitialiser</Text>

        </TouchableOpacity>

      </View>

    </View>

  )

}

const st = StyleSheet.create({

  container: { flex: 1, backgroundColor: '#FFEEFC', padding: 16, paddingTop: 28 },

  header: { flexDirection: 'row', alignItems: 'center', justifyContent: 'space-between', marginBottom: 12 },

  link: { color: '#7c3aed', fontWeight: '700' },

  title: { fontSize: 22, fontWeight: '900', color: '#FF4FA2' },

  timeBox: {

    backgroundColor: '#fff',

    borderRadius: 26,

    paddingVertical: 30,

    alignItems: 'center',

    justifyContent: 'center',

    shadowColor: '#FF8FCD',

    shadowOpacity: 0.18,

    shadowRadius: 10,

    elevation: 6,

    marginBottom: 14,

  },

  timeText: {

    fontSize: 72,                 // 👈 plus grand

    fontWeight: '900',

    color: '#FF4FA2',

    letterSpacing: 1,

    textShadowColor: '#ffd6ef',

    textShadowOffset: { width: 1, height: 1 },

    textShadowRadius: 3,

  },

  subTime: { marginTop: 8, color: '#7c7c8a', fontWeight: '700' },

  // barre de progression

  progressTrack: {

    width: '86%',

    height: 10,

    backgroundColor: '#FFE4F6',

    borderRadius: 999,

    overflow: 'hidden',

    marginTop: 14,

  },

  progressFill: {

    height: '100%',

    backgroundColor: '#FF92E0',

  },

  settingsCard: {

    backgroundColor: '#fff',

    borderRadius: 22,

    padding: 14,

    shadowColor: '#FF8FCD',

    shadowOpacity: 0.16,

    shadowRadius: 8,

    elevation: 5,

    marginBottom: 12,

  },

  sectionTitle: { fontWeight: '800', color: '#444', marginBottom: 8 },

  inputsRow: { flexDirection: 'row', gap: 12 },

  inputWrap: { flex: 1 },

  label: { fontSize: 13, fontWeight: '700', color: '#FF69B4', marginBottom: 6 },

  input: {

    backgroundColor: '#FFF0FA',

    borderRadius: 14,

    borderWidth: 2,

    borderColor: '#FFB6F9',

    paddingHorizontal: 12,

    paddingVertical: 10,

    fontSize: 20,                 // 👈 un poil plus grand

    color: '#FF4FA2',

    textAlign: 'center',

  },

  presetRow: { flexDirection: 'row', flexWrap: 'wrap', gap: 8, marginTop: 10 },

  pill: {

    backgroundColor: '#FFE4F6',

    borderColor: '#FFB6F9',

    borderWidth: 2,

    borderRadius: 999,

    paddingHorizontal: 12,

    paddingVertical: 8,

  },

  pillText: { color: '#FF4FA2', fontWeight: '800' },

  actionsRow: { flexDirection: 'row', gap: 10, marginTop: 8 },

  btn: {

    flex: 1,

    borderRadius: 999,

    paddingVertical: 14,

    alignItems: 'center',

    shadowColor: '#FF4FA2',

    shadowOpacity: 0.25,

    shadowRadius: 6,

    elevation: 5,

  },

  btnPrimary: { backgroundColor: '#FF92E0' },

  btnWarn: { backgroundColor: '#f59e0b' },

  btnText: { color: '#fff', fontWeight: '900', fontSize: 16 },

  btnGhost: { backgroundColor: '#fff', borderWidth: 2, borderColor: '#FFB6F9' },

  btnGhostText: { color: '#FF4FA2', fontWeight: '900', fontSize: 16 },

})

Timer.context

// src/timerContext.tsx

import React, { createContext, useContext, useEffect, useMemo, useState } from 'react'

type TimerCtx = {

  running: boolean

  remainingMs: number

  durationMs: number

  start: (totalMs: number) => void

  pause: () => void

  reset: () => void

  setRemaining: (ms: number) => void

}

const Ctx = createContext<TimerCtx | null>(null)

export function msToMMSS(ms: number) {

  const total = Math.max(0, Math.floor(ms / 1000))

  const m = Math.floor(total / 60)

  const s = total % 60

  const mm = String(m).padStart(2, '0')

  const ss = String(s).padStart(2, '0')

  return `${mm}:${ss}`

}

export function TimerProvider({ children }: { children: React.ReactNode }) {

  const [running, setRunning] = useState(false)

  const [targetTs, setTargetTs] = useState<number | null>(null)

  const [remainingMs, setRemainingMs] = useState(0)

  const [durationMs, setDurationMs] = useState(0)

  // tick

  useEffect(() => {

    let raf: number | null = null

    const loop = () => {

      if (!running || targetTs == null) return

      const now = Date.now()

      const left = Math.max(0, targetTs - now)

      setRemainingMs(left)

      if (left === 0) {

        setRunning(false)

        setTargetTs(null)

      } else {

        raf = requestAnimationFrame(loop)

      }

    }

    if (running) raf = requestAnimationFrame(loop)

    return () => { if (raf) cancelAnimationFrame(raf) }

  }, [running, targetTs])

  const start = (totalMs: number) => {

    if (!Number.isFinite(totalMs) || totalMs <= 0) return

    const now = Date.now()

    setTargetTs(now + totalMs)

    setRemainingMs(totalMs)  // affichage immédiat

    setDurationMs(totalMs)   // pour la barre de progression

    setRunning(true)

  }

  const pause = () => {

    if (!running) return

    setRunning(false)

    if (targetTs) {

      const left = Math.max(0, targetTs - Date.now())

      setRemainingMs(left)

      setTargetTs(null)

    }

  }

  const reset = () => {

    setRunning(false)

    setTargetTs(null)

    setRemainingMs(0)

    setDurationMs(0)

  }

  /\*\* Permet d’ajuster la valeur affichée (presets, etc.) sans relancer \*/

  const setRemaining = (ms: number) => {

    setRemainingMs(Math.max(0, ms | 0))

    setDurationMs((d) => (d > 0 ? d : Math.max(0, ms | 0)))

    setTargetTs(null)

    setRunning(false)

  }

  const value = useMemo(() => ({

    running,

    remainingMs,

    durationMs,

    start,

    pause,

    reset,

    setRemaining,

  }), [running, remainingMs, durationMs])

  return <Ctx.Provider value={value}>{children}</Ctx.Provider>

}

export function useTimer() {

  const ctx = useContext(Ctx)

  if (!ctx) throw new Error('useTimer must be used within <TimerProvider>')

  return ctx

}

↩︎