import \* as React from "react"

import type {

ToastActionElement,

ToastProps,

} from "@/components/ui/toast"

const TOAST\_LIMIT = 1

const TOAST\_REMOVE\_DELAY = 1000000

type ToasterToast = ToastProps & {

id: string

title?: React.ReactNode

description?: React.ReactNode

action?: ToastActionElement

}

const actionTypes = {

ADD\_TOAST: "ADD\_TOAST",

UPDATE\_TOAST: "UPDATE\_TOAST",

DISMISS\_TOAST: "DISMISS\_TOAST",

REMOVE\_TOAST: "REMOVE\_TOAST",

} as const

let count = 0

function genId() {

count = (count + 1) % Number.MAX\_SAFE\_INTEGER

return count.toString()

}

type ActionType = typeof actionTypes

type Action =

| {

type: ActionType["ADD\_TOAST"]

toast: ToasterToast

}

| {

type: ActionType["UPDATE\_TOAST"]

toast: Partial<ToasterToast>

}

| {

type: ActionType["DISMISS\_TOAST"]

toastId?: ToasterToast["id"]

}

| {

type: ActionType["REMOVE\_TOAST"]

toastId?: ToasterToast["id"]

}

interface State {

toasts: ToasterToast[]

}

const toastTimeouts = new Map<string, ReturnType<typeof setTimeout>>()

const addToRemoveQueue = (toastId: string) => {

if (toastTimeouts.has(toastId)) {

return

}

const timeout = setTimeout(() => {

toastTimeouts.delete(toastId)

dispatch({

type: "REMOVE\_TOAST",

toastId: toastId,

})

}, TOAST\_REMOVE\_DELAY)

toastTimeouts.set(toastId, timeout)

}

export const reducer = (state: State, action: Action): State => {

switch (action.type) {

case "ADD\_TOAST":

return {

...state,

toasts: [action.toast, ...state.toasts].slice(0, TOAST\_LIMIT),

}

case "UPDATE\_TOAST":

return {

...state,

toasts: state.toasts.map((t) =>

t.id === action.toast.id ? { ...t, ...action.toast } : t

),

}

case "DISMISS\_TOAST": {

const { toastId } = action

if (toastId) {

addToRemoveQueue(toastId)

} else {

state.toasts.forEach((toast) => {

addToRemoveQueue(toast.id)

})

}

return {

...state,

toasts: state.toasts.map((t) =>

t.id === toastId || toastId === undefined

? {

...t,

open: false,

}

: t

),

}

}

case "REMOVE\_TOAST":

if (action.toastId === undefined) {

return {

...state,

toasts: [],

}

}

return {

...state,

toasts: state.toasts.filter((t) => t.id !== action.toastId),

}

}

}

const listeners: Array<(state: State) => void> = []

let memoryState: State = { toasts: [] }

function dispatch(action: Action) {

memoryState = reducer(memoryState, action)

listeners.forEach((listener) => {

listener(memoryState)

})

}

type Toast = Omit<ToasterToast, "id">

function toast({ ...props }: Toast) {

const id = genId()

const update = (props: ToasterToast) =>

dispatch({

type: "UPDATE\_TOAST",

toast: { ...props, id },

})

const dismiss = () => dispatch({ type: "DISMISS\_TOAST", toastId: id })

dispatch({

type: "ADD\_TOAST",

toast: {

...props,

id,

open: true,

onOpenChange: (open) => {

if (!open) dismiss()

},

},

})

return {

id: id,

dismiss,

update,

}

}

function useToast() {

const [state, setState] = React.useState<State>(memoryState)

React.useEffect(() => {

listeners.push(setState)

return () => {

const index = listeners.indexOf(setState)

if (index > -1) {

listeners.splice(index, 1)

}

}

}, [state])

return {

...state,

toast,

dismiss: (toastId?: string) => dispatch({ type: "DISMISS\_TOAST", toastId }),

}

}

export { useToast, toast }