File: client/src/hooks/use-toast.ts

Type: ts | **Size:** 3895 bytes

```
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
    }
  | {
      type: ActionType["DISMISS_TOAST"]
      toastId?: ToasterToast["id"]
    }
  | {
      type: ActionType["REMOVE_TOAST"]
      toastId?: ToasterToast["id"]
    }
interface State {
  toasts: ToasterToast[]
}
const toastTimeouts = new Map>()
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
      // ! Side effects ! - This could be extracted into a dismissToast() action,
      // but I'll keep it here for simplicity
      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
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(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 }
```