import state from './state';

import { onAction } from './actions';

import { getNode } from './utils';

import { SwalOptions } from './options';

import { CANCEL\_KEY } from './options/buttons';

import CLASS\_NAMES from './class-list';

const { MODAL, BUTTON, OVERLAY } = CLASS\_NAMES;

const onTabAwayLastButton = (e: KeyboardEvent): void => {

e.preventDefault();

setFirstButtonFocus();

};

const onTabBackFirstButton = (e: KeyboardEvent): void => {

e.preventDefault();

setLastButtonFocus();

};

const onKeyUp = (e: KeyboardEvent):void => {

if (!state.isOpen) return;

switch (e.key) {

case "Escape": return onAction(CANCEL\_KEY);

}

};

const onKeyDownLastButton = (e: KeyboardEvent): void => {

if (!state.isOpen) return;

switch (e.key) {

case "Tab": return onTabAwayLastButton(e);

}

};

const onKeyDownFirstButton = (e: KeyboardEvent): void => {

if (!state.isOpen) return;

if (e.key === "Tab" && e.shiftKey) {

return onTabBackFirstButton(e);

}

};

/\*

\* Set default focus on Confirm-button

\*/

const setFirstButtonFocus = (): void => {

const button:HTMLElement = getNode(BUTTON);

if (button) {

button.tabIndex = 0;

button.focus();

}

};

const setLastButtonFocus = (): void => {

const modal: HTMLElement = getNode(MODAL);

const buttons: NodeListOf<Element> = modal.querySelectorAll(`.${BUTTON}`);

const lastIndex: number = buttons.length - 1;

const lastButton: any = buttons[lastIndex];

if (lastButton) {

lastButton.focus();

}

};

const setTabbingForLastButton = (buttons: NodeListOf<Element>): void => {

const lastIndex: number = buttons.length - 1;

const lastButton: Element = buttons[lastIndex];

lastButton.addEventListener('keydown', onKeyDownLastButton);

};

const setTabbingForFirstButton = (buttons: NodeListOf<Element>): void => {

const firstButton: Element = buttons[0];

firstButton.addEventListener('keydown', onKeyDownFirstButton);

};

const setButtonTabbing = (): void => {

const modal: HTMLElement = getNode(MODAL);

const buttons: NodeListOf<Element> = modal.querySelectorAll(`.${BUTTON}`);

if (!buttons.length) return;

setTabbingForLastButton(buttons);

setTabbingForFirstButton(buttons);

};

const onOutsideClick = (e: MouseEvent): void => {

const overlay: HTMLElement = getNode(OVERLAY);

// Don't trigger for children:

if (overlay !== e.target) return;

return onAction(CANCEL\_KEY);

};

const setClickOutside = (allow: boolean): void => {

const overlay: HTMLElement = getNode(OVERLAY);

overlay.removeEventListener('click', onOutsideClick);

if (allow) {

overlay.addEventListener('click', onOutsideClick);

}

};

const setTimer = (ms: number): void => {

if (state.timer) {

clearTimeout(state.timer);

}

if (ms) {

state.timer = window.setTimeout(() => {

return onAction(CANCEL\_KEY);

}, ms);

}

};

const addEventListeners = (opts: SwalOptions):void => {

if (opts.closeOnEsc) {

document.addEventListener('keyup', onKeyUp);

} else {

document.removeEventListener('keyup', onKeyUp);

}

/\* So that you don't accidentally confirm something

\* dangerous by clicking enter

\*/

if (opts.dangerMode) {

setFirstButtonFocus();

} else {

setLastButtonFocus();

}

setButtonTabbing();

setClickOutside(opts.closeOnClickOutside);

setTimer(opts.timer);

};

export default addEventListeners;