import { SwalParams } from '../../core';

import {

throwErr,

isPlainObject,

ordinalSuffixOf,

} from '../utils';

import {

ButtonList,

getButtonListOpts,

defaultButtonList

} from './buttons';

import {

getContentOpts,

ContentOptions,

} from './content';

import {

DEPRECATED\_OPTS,

logDeprecation,

} from './deprecations';

/\*

\* The final object that we transform the given params into

\*/

export interface SwalOptions {

title: string,

text: string,

icon: string,

buttons: ButtonList | Array<string | boolean>,

content: ContentOptions,

className: string,

closeOnClickOutside: boolean,

closeOnEsc: boolean,

dangerMode: boolean,

timer: number,

};

const defaultOpts: SwalOptions = {

title: null,

text: null,

icon: null,

buttons: defaultButtonList,

content: null,

className: null,

closeOnClickOutside: true,

closeOnEsc: true,

dangerMode: false,

timer: null,

};

/\*

\* Default options customizeable through "setDefaults":

\*/

let userDefaults: SwalOptions = Object.assign({}, defaultOpts);

export const setDefaults = (opts: object): void => {

userDefaults = Object.assign({}, defaultOpts, opts);

};

/\*

\* Since the user can set both "button" and "buttons",

\* we need to make sure we pick one of the options

\*/

const pickButtonParam = (opts: any): object => {

const singleButton: string|object = opts && opts.button;

const buttonList: object = opts && opts.buttons;

if (singleButton !== undefined && buttonList !== undefined) {

throwErr(`Cannot set both 'button' and 'buttons' options!`);

}

if (singleButton !== undefined) {

return {

confirm: singleButton,

};

} else {

return buttonList;

}

};

// Example 0 -> 1st

const indexToOrdinal = (index: number): string => ordinalSuffixOf(index + 1);

const invalidParam = (param: any, index: number): void => {

throwErr(`${indexToOrdinal(index)} argument ('${param}') is invalid`);

};

const expectOptionsOrNothingAfter = (index: number, allParams: SwalParams): void => {

let nextIndex = (index + 1);

let nextParam = allParams[nextIndex];

if (!isPlainObject(nextParam) && nextParam !== undefined) {

throwErr(`Expected ${indexToOrdinal(nextIndex)} argument ('${nextParam}') to be a plain object`);

}

};

const expectNothingAfter = (index: number, allParams: SwalParams): void => {

let nextIndex = (index + 1);

let nextParam = allParams[nextIndex];

if (nextParam !== undefined) {

throwErr(`Unexpected ${indexToOrdinal(nextIndex)} argument (${nextParam})`);

}

};

/\*

\* Based on the number of arguments, their position and their type,

\* we return an object that's merged into the final SwalOptions

\*/

const paramToOption = (opts: any, param: any, index: number, allParams: SwalParams): object => {

const paramType = (typeof param);

const isString = (paramType === "string");

const isDOMNode = (param instanceof Element);

if (isString) {

if (index === 0) {

// Example: swal("Hi there!");

return {

text: param,

};

}

else if (index === 1) {

// Example: swal("Wait!", "Are you sure you want to do this?");

// (The text is now the second argument)

return {

text: param,

title: allParams[0],

};

}

else if (index === 2) {

// Example: swal("Wait!", "Are you sure?", "warning");

expectOptionsOrNothingAfter(index, allParams);

return {

icon: param,

};

}

else {

invalidParam(param, index);

}

}

else if (isDOMNode && index === 0) {

// Example: swal(<DOMNode />);

expectOptionsOrNothingAfter(index, allParams);

return {

content: param,

};

}

else if (isPlainObject(param)) {

expectNothingAfter(index, allParams);

return param;

}

else {

invalidParam(param, index);

}

};

/\*

\* No matter if the user calls swal with

\* - swal("Oops!", "An error occurred!", "error") or

\* - swal({ title: "Oops!", text: "An error occurred!", icon: "error" })

\* ... we always want to transform the params into the second version

\*/

export const getOpts = (...params: SwalParams): SwalOptions => {

let opts = <any>{};

params.forEach((param, index) => {

let changes: object = paramToOption(opts, param, index, params);

Object.assign(opts, changes);

});

// Since Object.assign doesn't deep clone,

// we need to do this:

let buttonListOpts = pickButtonParam(opts);

opts.buttons = getButtonListOpts(buttonListOpts);

delete opts.button;

opts.content = getContentOpts(opts.content);

const finalOptions: SwalOptions = Object.assign({}, defaultOpts, userDefaults, opts);

// Check if the users uses any deprecated options:

Object.keys(finalOptions).forEach(optionName => {

if (DEPRECATED\_OPTS[optionName]) {

logDeprecation(optionName);

}

});

return finalOptions;

};