import { isPlainObject, throwErr } from '../utils';

export interface ButtonOptions {

visible?: boolean,

text?: string,

value?: any,

className?: string | Array<string>,

closeModal?: boolean,

};

export interface ButtonList {

[buttonNamespace: string]: ButtonOptions | boolean,

};

export const CONFIRM\_KEY = 'confirm';

export const CANCEL\_KEY = 'cancel';

const defaultButton: ButtonOptions = {

visible: true,

text: null,

value: null,

className: '',

closeModal: true,

};

const defaultCancelButton: ButtonOptions = Object.assign({},

defaultButton, {

visible: false,

text: "Cancel",

value: null,

}

);

const defaultConfirmButton: ButtonOptions = Object.assign({},

defaultButton, {

text: "OK",

value: true,

}

);

export const defaultButtonList: ButtonList = {

cancel: defaultCancelButton,

confirm: defaultConfirmButton,

};

const getDefaultButton = (key: string): ButtonOptions => {

switch (key) {

case CONFIRM\_KEY:

return defaultConfirmButton;

case CANCEL\_KEY:

return defaultCancelButton;

default:

// Capitalize:

const text = key.charAt(0).toUpperCase() + key.slice(1);

return Object.assign({}, defaultButton, {

text,

value: key,

});

}

};

const normalizeButton = (key: string, param: string | object | boolean): ButtonOptions => {

const button: ButtonOptions = getDefaultButton(key);

/\*

\* Use the default button + make it visible

\*/

if (param === true) {

return Object.assign({}, button, {

visible: true,

});

}

/\* Set the text of the button: \*/

if (typeof param === "string") {

return Object.assign({}, button, {

visible: true,

text: param,

});

}

/\* A specified button should always be visible,

\* unless "visible" is explicitly set to "false"

\*/

if (isPlainObject(param)) {

return Object.assign({

visible: true,

}, button, param);

}

return Object.assign({}, button, {

visible: false,

});

};

const normalizeButtonListObj = (obj: any): ButtonList => {

let buttons: ButtonList = {};

for (let key of Object.keys(obj)) {

const opts: any = obj[key];

const button: ButtonOptions = normalizeButton(key, opts);

buttons[key] = button;

}

/\*

\* We always need a cancel action,

\* even if the button isn't visible

\*/

if (!buttons.cancel) {

buttons.cancel = defaultCancelButton;

}

return buttons;

};

const normalizeButtonArray = (arr: any[]): ButtonList => {

let buttonListObj: ButtonList = {};

switch (arr.length) {

/\* input: ["Accept"]

\* result: only set the confirm button text to "Accept"

\*/

case 1:

buttonListObj[CANCEL\_KEY] = Object.assign({}, defaultCancelButton, {

visible: false,

});

break;

/\* input: ["No", "Ok!"]

\* result: Set cancel button to "No", and confirm to "Ok!"

\*/

case 2:

buttonListObj[CANCEL\_KEY] = normalizeButton(CANCEL\_KEY, arr[0]);

buttonListObj[CONFIRM\_KEY] = normalizeButton(CONFIRM\_KEY, arr[1]);

break;

default:

throwErr(`Invalid number of 'buttons' in array (${arr.length}).

If you want more than 2 buttons, you need to use an object!`);

}

return buttonListObj;

};

export const getButtonListOpts = (opts: string | object | boolean): ButtonList => {

let buttonListObj: ButtonList = defaultButtonList;

if (typeof opts === "string") {

buttonListObj[CONFIRM\_KEY] = normalizeButton(CONFIRM\_KEY, opts);

} else if (Array.isArray(opts)) {

buttonListObj = normalizeButtonArray(opts);

} else if (isPlainObject(opts)) {

buttonListObj = normalizeButtonListObj(opts);

} else if (opts === true) {

buttonListObj = normalizeButtonArray([true, true]);

} else if (opts === false) {

buttonListObj = normalizeButtonArray([false, false]);

} else if (opts === undefined) {

buttonListObj = defaultButtonList;

}

return buttonListObj;

};