Full Stack Snippets.

From Chris' Full Stack Blog.

mergeArrays.js

javascript

```javascript
export const mergeArrays = (params) => {
    const { mergeArray, existingArray, matchKey } = params
    return existingArray.map((existingItem) => {
        const match = mergeArray.find(
            (mergeItem) => mergeItem[matchKey] === existingItem[matchKey]
        )
        if (match) {
            return Object.assign(existingItem, match)
        }
        return existingItem
    })
}
```

Usage

javascript

```javascript
// Given interface IFile:
export interface IFile {
    fileLabel: string
    code: string
}

// and interface IEditorSetting:
export interface IEditorSetting extends IFile {
    isActive: boolean
}

// and array editorSettingsState, which is of type Array<IEditorSetting>:
const editorSettingsState: Array<IEditorSetting> = [
    {
        fileLabel: 'myJSFile.js',
        code: '// some JS comment',
        isActive: false
    },
    {
        fileLabel: 'myHTMLFile.html',
        code: '<h1>hello world</h1>',
        isActive: true
    },
    {
        fileLabel: 'myCSSFile.css',
        code: 'h1 { color: red; }',
        isActive: false
    }
]

// and some incoming files from an API or similar:
const files: Array<IFile> = [
    {
        fileLabel: 'myJSFile.js',
        code: '// awesome server generated code'
    },
    {
        fileLabel: 'myHTMLFile.js',
        code: '<h1>awesome generated code</h1>'
    },
    {
        fileLabel: 'myCSSFile.css',
        code: 'h1 { color: blue; font-weight: bold; }'
    },
]

// This will return a new array of type Array<IEditorSetting>,
// with the code updated the code for all files WITHOUT changing the isActive
property (since isActive is not in IFile)
const mergedArray = mergeArrays({
    mergeArray: files,
    existingArray: editorSettingsState,
    matchKey: "fileLabel"
})
```

updateArray.js

javascript

```
export const updateArray = (options) => {
    const {
        array,
        testKey,
        testValue,
        updateKey,
        updateValue,
        testFailValue,
    } = options
    return array.map((item) => {
        if (item[testKey] === testValue) {
            item[updateKey] = updateValue
        } else if (testFailValue !== undefined) {
            item[updateKey] = testFailValue
        }
        return item
    })
}
```

Usage

javascript

```javascript
import { updateArray } from "../../../../frontend/typescript/utils/updateArray"

// Given interface IEditorSetting:
export default interface IEditorSetting {
    fileLabel: string
    code: string
    isActive: boolean
}

// and array editorSettingsState, which is of type Array<IEditorSetting>:
const editorSettingsState: Array<IEditorSetting> = [
    {
        fileLabel: 'myJSFile.js',
        code: '// some JS comment',
        isActive: false
    },
    {
        fileLabel: 'myHTMLFile.html',
        code: '<h1>hello world</h1>',
        isActive: true
    },
    {
        fileLabel: 'myCSSFile.css',
        code: 'h1 { color: red; }',
        isActive: false
    }
]
```

```
const code = "<p>some new HTML code for the html editor</p>"

// This will return a new array of type Array<IEditorSetting>,
// with the code updated the code ONLY for the editor(s) which isActive = true
const updatedArray = updateArray({
    array: editorSettingsState,
    testKey: "isActive",
    testValue: true,
    updateKey: "code",
    updateValue: code,
})
```

useDidMount.js

javascript

```javascript
import { useState, useEffect } from 'react'

export const useDidMount = () => {
  const [didMount, setDidMount] = useState(false)

  useEffect(() => {
    setDidMount(true)
  }, [])

  return didMount
}
```

Usage

javascript

```javascript
import * as React from "react"
import { useDidMount } from "./hooks/useDidMount"

export function ExampleComponent() {
    const didMount = useDidMount()

    if (didMount) {
        console.log(
            "I am mounted! Things like the DOM and window are available! Or,
you could run some animation you were waiting to run!"
        )
    }

    return <></>
}
```

useAppSelector.js

javascript

```javascript
// This hook only makes sense to use in TypeScript code :(
```

useAppDispatch.js

javascript

```javascript
// This hook only makes sense to use in TypeScript code :(
```

sendSlackMessage.js

javascript

```javascript
export const sendSlackMessage = (message) => {
    fetch(process.env.SLACK_WEBHOOK_URL, {
        method: "POST",
        headers: {
            "Content-Type": "application/json",
        },
        body: JSON.stringify({
            text: message,
        }),
    })
}
```

Usage

javascript

```javascript
import { sendSlackMessage } from "./sendSlackMessage";

// Send the message!
sendSlackMessage("Hello world!")
```