

Full Stack Snippets.

From [Chris' Full Stack Blog](#).

mergeArrays.ts

typescript

```
export const mergeArrays = <T, U extends T>(params: {
  mergeArray: Array<T>
  existingArray: Array<U>
  matchKey: keyof T
}): Array<U> => {
  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

typescript

```
// 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.ts

typescript

```

export const updateArray = <T, U extends keyof T, V extends keyof T>(params: {
  array: Array<T>
  testKey: keyof T
  testValue: T[U]
  updateKey: keyof T
  updateValue: T[V]
  testFailValue?: T[V]
}): Array<T> => {
  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

typescript

```

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.ts

typescript

```

import { useState, useEffect } from 'react'

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

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

  return didMount
}

```

Usage

typescript

```

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.ts

typescript

```
import { TypedUseSelectorHook, useSelector } from "react-redux";
import { RootState } from "../store";

export const useAppSelector: TypedUseSelectorHook<RootState> = useSelector
```

Usage

typescript

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

export function ExampleComponent() {
  // complexTypedPartOfSlice here will be typed just as defined in the slice.
  // TypeScript also won't complain about state missing a typing,
  // since it's been typed in the definition for useAppSelector!
  const { complexTypedPartOfSlice } = useAppSelector(
    (state) => state.someSliceOfState
  )
  return <>Hello world!</>
}
```

useAppDispatch.ts

typescript

```
import { useDispatch } from "react-redux";
import { AppDispatch } from "../store";

export const useAppDispatch = () => useDispatch<AppDispatch>()
```

Usage

typescript

```
import * as React from "react"
```

```
import { useAppDispatch } from "../hooks/useAppDispatch"

export function ExampleComponent() {
  // here 'dispatch' will have the correct typing depending on
  // which middleware(s) you are using!
  const dispatch = useAppDispatch()

  const handleClick = () => {
    dispatch(someReduxAction())
  }

  return <button onClick={handleButtonClick}>Click me!</button>
}
```

sendSlackMessage.ts

typescript

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

Usage

typescript

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

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