Full Stack Snippets.

From Chris' Full Stack Blog.

Client

mergeArrays.ts

typescript

```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

```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"
})
```

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.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

```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,
})
```

updateArray.js

javascript

```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.ts

typescript

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

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

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

  return didMount
}
```

Usage

typescript

```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 <></>
}
```

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.ts

typescript

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

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

Usage

typescript

```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!</>
}
```

useAppSelector.js

javascript

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

useAppDispatch.ts

typescript

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

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

Usage

typescript

```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 handleButtonClick = () => {
        dispatch(someReduxAction())
    }

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

useAppDispatch.js

javascript

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

sendSlackMessage.ts

typescript

```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

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

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

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!")
```

Backend

JavaScript (Node.js)

C#

PatchFiltererService

Filter out unwanted properties from your models on the server side in .NET.

From post: C# .NET Core and TypeScript: Using Generics and LINQ to Secure and Filter Operations on Your JSONPatchDocuments

`PatchFiltererService.cs`
csharp

```csharp
using System;
using System.Linq;
using Microsoft.AspNetCore.JsonPatch;

namespace JsonPatchFilterExample.Services
{
    // a security filter for JSON patch filter operations
    // see the full blog post at https://chrisfrew.in/blog/filtering-json-
patch-in-c-sharp/
    public static class PatchFiltererService
    {
        public static JsonPatchDocument<T> ApplyAttributeFilterToPatch<T, TU>
(JsonPatchDocument<T> patch)
```

```csharp
        where T : class
        where TU : Attribute
        {
            // Get path for all attributes of type TU that are in type T
            var allowedPaths = typeof(T)
                .GetProperties()
                .Where(x => x.GetCustomAttributes(false).OfType<TU>().Any())
                .Select(x => x.Name);

            // Now build a new JSONPatchDocument based on properties in T that
were found above
            var filteredPatch = new JsonPatchDocument<T>();
            patch.Operations.ForEach(x =>
            {
                if (allowedPaths.Contains(x.path))
                {
                    filteredPatch.Operations.Add(x);
                }
            });

            return filteredPatch;
        }
    }
}
```

Usage

csharp

```csharp
using System;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.JsonPatch;
using JsonPatchFilterExample.Services;
using JsonPatchFilterExample.Models;
using System.ComponentModel.DataAnnotations;
using Microsoft.Extensions.FileProviders;
using System.IO;
using Newtonsoft.Json;

namespace JsonPatchFilterExample.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class WidgetController : ControllerBase
    {
        [HttpPatch("{id}")]
        public ActionResult Patch(Guid id, [FromBody]
JsonPatchDocument<WidgetModel> patch)
        {
            try
            {
                // For now, load the widget from the json file - ideally this
would be retrieved via a repository from a database
                var physicalProvider = new
PhysicalFileProvider(Directory.GetCurrentDirectory());
                var jsonFilePath = Path.Combine(physicalProvider.Root,
"App_Data", "ExampleWidget.json");
                var item = new WidgetModel();
                using (var reader = new StreamReader(jsonFilePath))
                {
                    var content = reader.ReadToEnd();
                    item = JsonConvert.DeserializeObject<WidgetModel>(content);
                }
                if (item.Id != id || patch == null)
                {
                    return NotFound();
                }

                // Create a new patch to match only the type and attributes
passed
                patch =
PatchFiltererService.ApplyAttributeFilterToPatch<WidgetModel,
StringLengthAttribute>(patch);

                // Apply the patch!
                patch.ApplyTo(item);

                // Update updated time - normally would be handled in a
repository
                item.Updated = DateTime.Now;

                // Update the item - ideally this would also be done with a
repository via an 'Update' method
                // write JSON directly to a file
                var json = JsonConvert.SerializeObject(item);

                //write string to file
                System.IO.File.WriteAllText(jsonFilePath, json);

                return Ok();
            }
            catch
            {
                return UnprocessableEntity();
            }
        }
    }
}
```

AssertPropertiesAreNonNullService

Assert that all required, or simple all properties on your objects are not null.

From post: Recursively Assert All Properties Are Non-null Using Reflection

AssertPropertiesAreNonNullService.cs

csharp

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using Shouldly;

namespace AssertPropertiesAreNonNullExample {
    public static class AssertPropertiesAreNonNullService
    {
        // Asserts that all required properties (via the 'Required' attribute)
be non null
        // Optionally include all properties if desired
        private static void AssertPropertiesAreNonNull<T>(T obj, bool
onlyRequiredProperties = true)
        {
            if (obj == null)
            {
                return;
            }

            var objType = obj.GetType();

            // Get either all or only required properties
            var properties = onlyRequiredProperties ? objType.GetProperties()
                .Where(x =>
x.GetCustomAttributes(false).OfType<RequiredAttribute>().Any()) :
            objType.GetProperties();

            foreach (var property in properties)
            {
                var propValue = property.GetValue(obj, null);
                var elems = propValue as IList<object>;
                // Another layer
                if (elems != null)
                {
                    foreach (var item in elems)
                    {
                        AssertPropertiesAreNonNull(item,
onlyRequiredProperties);
                    }
                }
                else
                {
                    if (property.PropertyType.Assembly == objType.Assembly)
                    {
                        AssertPropertiesAreNonNull(propValue,
onlyRequiredProperties);
                    }
                    // Reached the end of the tree
                    else
                    {
                        propValue.ShouldNotBeNull();
                    }
                }
            }
        }
    }
}
```

Usage

csharp

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using AssertPropertiesAreNonNullService;

public class SomethingNested
{
        [Required]
        public string SomeString { get; set; }

        [Required]
        public int SomeNumber { get; set; }

        public bool SomeBoolean { get; set; }
        public List<string> SomeStringList { get; set; }

}

public class MyWidget
{
        [Required]
        public SomethingNested SomethingNested { get; set; }

        public string SomeString { get; set; }

        public int SomeNumber { get; set; }

        public bool SomeBoolean { get; set; }

        [Required]
        public List<string> SomeStringList { get; set; }
}

public class Program
{
        public static void Main()
        {
                // Declare some object you want to check for null values
                var myWidget = new MyWidget
                {
                        SomethingNested = new SomethingNested
                        {
                                SomeString = null,
                                SomeNumber = 123,
                                SomeBoolean = true,
                                SomeStringList = new List<string> { "a", "b",
null }
                        },
                        SomeString = null,
                        SomeNumber = 123,
                        SomeBoolean = true,
                        SomeStringList = null
                };

                // Only run for required properties of myWidget

AssertPropertiesAreNonNullService.AssertPropertiesAreNonNull(myWidget);

                // Run for ALL properties in myWidget

AssertPropertiesAreNonNullService.AssertPropertiesAreNonNull(myWidget, false);
```

```
        }
}
```

Devops

Bash

---

buildColorPrompt

Letter-level color changes for your bash prompt!

From post: Awesome Colors for Shell Prompts!

buildColorPrompt.sh

bash

```bash
function buildColorPrompt() {

    # I always like showing what directory I am in (special character "\w" in PS1) - store the equivalent in this 'directory' variable
    directory=$(pwd)

    # Modify these to whatever you'd like!
    PROMPT_TEXT="awesome-shell-prompt-colors@awesome-machine [$directory] "

    # Colors seperated by comma - acceptable values are:
    # black, white, red, green, yellow, blue, magenta, cyan, light gray, light red, light green, light yellow, light blue, light magenta, light cyan
    PROMPT_COLORS="red,white,blue"

    # Colors!
    BLACK="\e[30m"
    WHITE="\e[97m"
    RED="\e[31m"
    GREEN="\e[32m"
    YELLOW="\e[33m"
    BLUE="\e[34m"
    MAGENTA="\e[35m"
    CYAN="\e[36m"
    LIGHT_GRAY="\e[37m"
    DARK_GRAY="\e[90m"
    LIGHT_RED="\e[91m"
    LIGHT_GREEN="\e[92m"
    LIGHT_YELLOW="\e[93m"
    LIGHT_BLUE="\e[94m"
    LIGHT_MAGENTA="\e[95m"
    LIGHT_CYAN="\e[96m"

    # End formatting string
    END_FORMATTING="\[\e[0m\]"

    # split PROMPT_COLORS into array
    count=0
    IFS=','
    for x in $PROMPT_COLORS
    do
        colors_array[$count]=$x
        ((count=count+1))
    done
    unset IFS

    # break PROMPT_TEXT into character array
    letters=()
    for (( i=0 ; i < ${#PROMPT_TEXT} ; i++ )) {
        letters[$i]=${PROMPT_TEXT:$i:1}
    }

    # build prompt with colors
    color_index=0
    ps1='\['
    for (( i=0 ; i < ${#letters[@]} ; i++ )) {
        # Determine color in this giant case statement
        color="${colors_array[color_index]}"
        case $color in
            "black")
                COLOR=$BLACK
                ;;
            "red")
                COLOR=$RED
                ;;
            "green")
                COLOR=$GREEN
                ;;
            "yellow")
                COLOR=$YELLOW
                ;;
            "blue")
                COLOR=$BLUE
                ;;
            "magenta")
```

```
                COLOR=$MAGENTA
                ;;
            "cyan")
                COLOR=$CYAN
                ;;
            "light gray")
                COLOR=$LIGHT_GRAY
                ;;
            "dark gray")
                COLOR=$DARK_GRAY
                ;;
            "light red")
                COLOR=$LIGHT_RED
                ;;
            "light green")
                COLOR=$LIGHT_GREEN
                ;;
            "light yellow")
                COLOR=$LIGHT_YELLOW
                ;;
            "light blue")
                COLOR=$LIGHT_BLUE
                ;;
            "light magenta")
                COLOR=$LIGHT_MAGENTA
                ;;
            "light cyan")
                COLOR=$LIGHT_CYAN
                ;;
            "white")
                COLOR=$WHITE
                ;;
            *)
                COLOR=$WHITE
                ;;
        esac

        # add to ps1 var - color, then letter, then the end formatter
        ps1+=$COLOR"${letters[$i]}"

        # reset color index if we are at the end of the color array, otherwise
increment it
        if (( $color_index == ${#colors_array[@]} - 1 ))
        then
            color_index=0
        else
            ((color_index=color_index+1))
        fi
```

```
    }
    ps1+="$END_FORMATTING\]"

    # Finally: set the PS1 variable
    PS1=$ps1
}

# Set the special bash variable PROMPT_COMMAND to our custom function
PROMPT_COMMAND=buildColorPrompt;
```

Usage

bash

```bash
# Assuming the buildColorPrompt function is in your .bash_profile:
# Set the special bash variable PROMPT_COMMAND to our custom function
PROMPT_COMMAND=buildColorPrompt;
```

---

sendSlackMessage

Util function to send a Slack message from bash.

From post: The Last Bitbucket Pipelines Tutorial You'll Ever Need: Mastering CI and CD

`sendSlackMessage.sh`

bash

```bash
function sendSlackMessage {
    curl -X POST -H 'Content-type: application/json' --data '{"text":"$1"}' $2
}
```

Usage

bash

```bash
# the two parameters are 1. the message, and 2. the webhook url
sendSlackMessage "Hello World!" https://yourslackwebhookurl/secret/supersecret
```

---

supercurl

Get detailed network times for a website.

From post: Magento 2 IP Location Detection (GeoIP) and Store Context Control Using the ipstack API

supercurl.sh

bash

```bash
function supercurl() {
    curl -s -w '\nLookup time:\t%{time_namelookup}\nConnect time:\t%{time_connect}\nAppCon time:\t%{time_appconnect}\nRedirect time:\t%{time_redirect}\nPreXfer time:\t%{time_pretransfer}\nStartXfer time:\t%{time_starttransfer}\n\nTotal time:\t%{time_total}\n' -o /dev/null $1
}
```

Usage

bash

```bash
# simply pass the website you want "super" curl as the first argument :)
supercurl google.com

# example output:
# Lookup time:    0.061647
# Connect time:   0.281195
# AppCon time:    0.000000
# Redirect time:  0.000000
# PreXfer time:   0.281248
# StartXfer time: 0.759128

# Total time:     0.761387
```

zsh

---

buildColorPrompt

Letter-level color changes for your zsh prompt!

From post: Awesome Colors for Shell Prompts!

buildColorPrompt.sh

bash

```bash
function buildColorPrompt() {

    # I always like showing what directory I am in
    directory=$(pwd)

    # Modify these to whatever you'd like!
    PROMPT_TEXT="youruser@yourmachine [$directory]"

    # Comma seperated colors - as many or as few as you'd like
    PROMPT_COLORS="15"

    # This will be the color of everything in the input part of the prompt
(here set to 15 = white)
    PROMPT_INPUT_COLOR="15"

    # split PROMPT_COLORS into array
    colors_array=("${(@s/,/)PROMPT_COLORS}") # @ modifier

    # break PROMPT_TEXT into character array
    letters=()
    for (( i=1 ; i < ${#PROMPT_TEXT}+1 ; i++ )) {
        letters[$i]=${PROMPT_TEXT:$i-1:1}
    }

    # build prompt with colors
    color_index=1
    ps1=""
    for (( i=1 ; i < ${#letters[@]}+1 ; i++ )) {
        # Determine color in this giant case statement
        color="${colors_array[color_index]}"

        # add to ps1 var - color, then letter, then the end formatter
        ps1+="%F{$color}${letters[$i]}"

        # reset color index if we are at the end of the color array, otherwise
increment it
        if (( $color_index == ${#colors_array[@]} ))
        then
            color_index=1
        else
            ((color_index=color_index+1))
        fi
    }

    # end color formating
    ps1+="%F{$PROMPT_INPUT_COLOR} %# "

    # Finally: set the PROMPT variable
    PROMPT=$ps1
}
```

```
# set the precmd() hook to our custom function
precmd() {
   buildColorPrompt;
}
```

Usage

bash

```bash
# Assuming the buildColorPrompt function is in your .zprofile:
# Set the precmd() hook to our custom function
precmd() {
   buildColorPrompt;
}
```