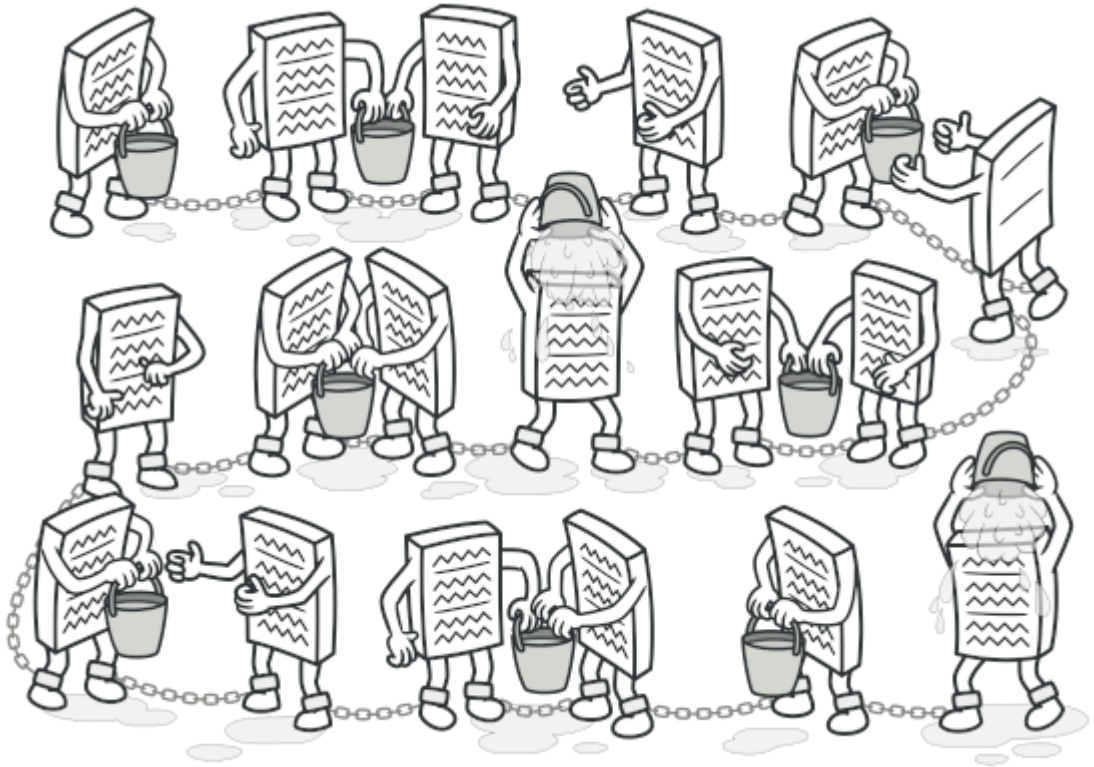
Chain of Responsability (ou Middleware)

Esse é um padrão de construção de aplicações muito utilizado em alguns frameworks, como o express por exemplo.

Esse padrão preza por separar as diversas execuções de uma aplicação em funções diferentes, onde cada função é responsável por uma única tarefa, a essas funções é dado o nome de “handle”.

No Middleware, em vez de fazer uma execução ir de um ponto inicial passando por todas as funções até chegar na última função e entregar sua resposta, isso é feito de maneira diferente. Cada Handle pode ser chamado á partir de outro Handle de acordo com a necessidade, se não for necessário usar um Handle naquela execução ele simplesmente é pulado, e dá lugar a outro Handle.

Isso é possível graças ao padrão de passagem de parâmetros, onde cada Handle recebe o nome da próxima Handle que deverá ser executada juntamente com os parâmetros normais. Ao final de uma Handle á uma função “next()” que executa a função passada como parâmetro, como sendo a próxima função, isso permite que somente as funções necessárias sejam chamadas, em vez de executar um processo inteiro executando funções desnecessárias.

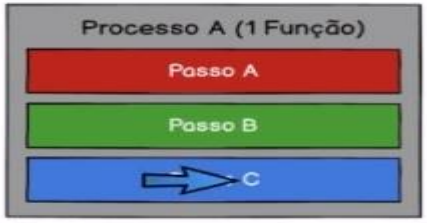


Onde o Middleware não deve ser aplicado?

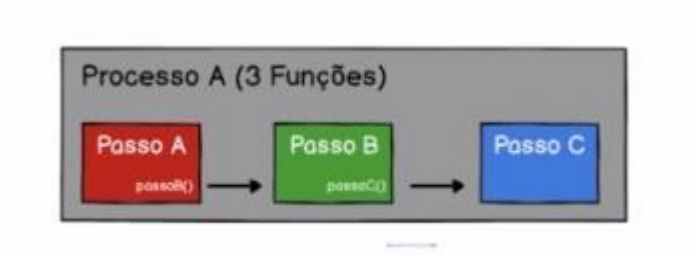
Não seria legal aplicar o Middleware em processos que exigem que os dados passem por toda uma camada de funções para serem processados, como na verificação de autenticidade por exemplo, nenhum processo da verificação poderia ser pulado, sendo assim o padrão de projeto middleware seria desnecessário.

Diferenças entre o Middleware e outros processos convencionais:

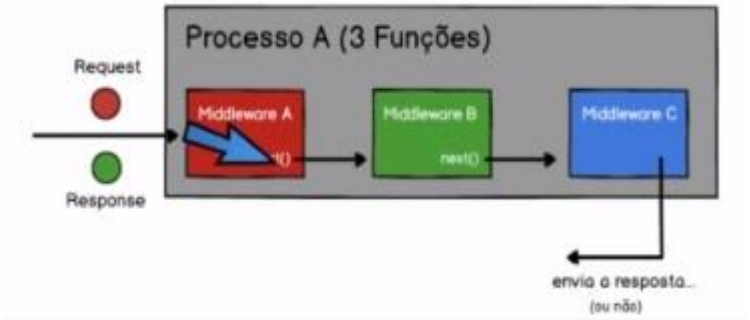
Note na imagem abaixo temos um padrão de projeto procedural, onde um procedimento é seguido após o outro, ás vezes todos os procedimentos estão amarrados dentro de uma única função...



Já na imagem abaixo temos um padrão de projeto funcional, onde uma função obrigatoriamente chama uma determinada função específica para ser executada logo após ela, esse tipo de padrão de projeto não possuí a liberdade do padrão middleware...



Por fim, na imagem abaixo, temos um padrão Middleware, onde uma aplicação recebe os dados, e de acordo com os dados, ela vai executar uma função ou outra. Para ela não importa que função vai começar, ou qual vai terminar, ou se alguma não será executada, pois serão os dados que dirão para onde a aplicação deverá ir...



Veja um exemplo prático do uso de Middleware...

```
//MIDDLEWARE:

//Abaixo temos um exemplo simples da implementação de um Middleware Pattern:
//Perceba abaixo que temos uma função para cada passo, á cada passo um valor string é implementado num objeto chamado "ctx", além disso, os passos também receberão uma função "next" que será chamada ao final de cada passo...
const passo1 = (ctx, next) => {
    ctx.valor1 = 'mid1'
    next()
}

const passo2 = (ctx, next) => {
    ctx.valor2 = 'mid2'
    next()
}

const passo3 = ctx => ctx.valor3 = 'mid3' //Somente o passo 3 não chamará uma próxima função...

//Abaixo temos a função "Handle" que dá início ao processo do middleware...
const execMiddle = (ctx, ...handles) => { //Perceba que a função recebe um objeto, e um conjunto de funções que serão as nossas handles, as funções serão executadas na ordem em que nós passamos as handles...
    const execPasso = indice => { //Dentro da função principal, temos uma função que recursiva que recebe um valor índice que será responsável por capturar os valores das funções passados ao rest "...handles"
        handles && indice < handles.length && handles[indice](ctx, () => execPasso(indice + 1)) //Essa função compara se handles é verdadeiro, pois se não houver nenhuma função passada, não tem por que executar a handle principal, também compara se o valor índice passado na chamada da função execPasso() não ultrapassa a quantidades de funções que existem no array gerado pelo rest "...handles" e por fim executa a função equivalente ao valor índice e recebe como next a recursividade da função execPasso, que vai executar a próxima função indice de passada no rest de "...handles"
    }
    execPasso(0)
}

//Percebaque podemos executar as funções na ordem que quisermos...
//Executando as handles em ordem de passo...
const ctx1 = {}
execMiddle(ctx1, passo1, passo2, passo3)
console.log(ctx1)
```

```
//Executando as handles começando pelo passo 2...
const ctx2 = {}
execMiddle(ctx2, passo2, passo1, passo3)
console.log(ctx2)

//Executando as handles iniciando pelo passo 3, um passo que não irá chamar nenhum next()...
const ctx3 = {}

execMiddle(ctx3, passo3, passo2, passo1)
console.log(ctx3)
```

RESULTADO NO CONSOLE...

```
[Running] node "c:\Users\Almoxarifado\Documents\javascript\arquivos_das_aulas\143-Middleware_Pattern.js"
{ valor1: 'mid1', valor2: 'mid2', valor3: 'mid3' }
{ valor2: 'mid2', valor1: 'mid1', valor3: 'mid3' }
{ valor3: 'mid3' }

[Done] exited with code=0 in 0.115 seconds
```