Engenharia de software II – Trabalho: código limpo

Membros: Nathan Bandeira, Nathan barbosa, João otávio

Introdução :

Decidimos por analisar o trabalho final da cadeira de programação orientada a objetos, A Livraria.

o projeto em si contem o arquivo Main e as classes:

***Emprestimo**
Dedicado a gerenciar os empréstimos dado aos membros

***Livro**

Dedicado ao CRUD dos livros.

***Membro**

Dedicado ao CRUD dos membros

***Pessoa**

Recurso para a criação dos ‘’Membros’’

Analise Geral :

As propostas de analise pelo grupo foram dos seguintes arquivos.

***Membros**

Código original :

Caso 1:

```
let membros = [];
if (fs.existsSync("./data/membros.json")) {
  const data = fs.readFileSync("./data/membros.json", "utf-8");
  membros = JSON.parse(data);
}
```

No código original, várias operações acabam repetindo a verificação fs.existsSync("./data/membros.json") para saber se o arquivo existe antes de continuar. Pode ser centralizado em uma função de verificação para evitar duplicação de código.

Proposta/melhoria:

```
private LeArquivo(): any[] {
  if (fs.existsSync("./data/membros.json")) {
    const data = fs.readFileSync("./data/membros.json", "utf-8");
    return JSON.parse(data);
  }
  return [];
}
```

Utilizando o DRY: Reutiliza a lógica de leitura em um único local assim reduzindo duplicação.

Utilizando SRP: Isola a responsabilidade de leitura do arquivo em uma função separada, deixando o código mais claro.

Caso 2:

```
let membro = membros.find((membro: any) => membro.cpf === cpf);
```

As operações de encontrar um membro (find) e de atualizar ou deletar estão espalhadas por diferentes métodos, com lógica semelhante para cada um deles. Isso pode ser refatorado para criar um método genérico que busca um membro com base no CPF e trate os em diferentes cenários.

Proposta/Melhoria

```
private encontrarMembroPorCpf(cpf: string): any {
  const membros = this.readFile();
  return membros.find((membro: any) => membro.cpf === cpf);
}
```

Utilizando SRP: Centraliza a lógica de busca em um único método

Clean code: Explicita o que o método faz.

PONTO DE DESTAQUE:

```
public adicionar(): void {
  const membroData = {
    nome: this._nome,
    endereco: this._endereco,
    cpf: this._cpf,
    telefone: this._telefone,
  };
```

O método adicionar está bem estruturando utilizando de maneira certa a Leitura e a escrita de arquivos separando a lógica de manipulação de dados.

[Justificativa]

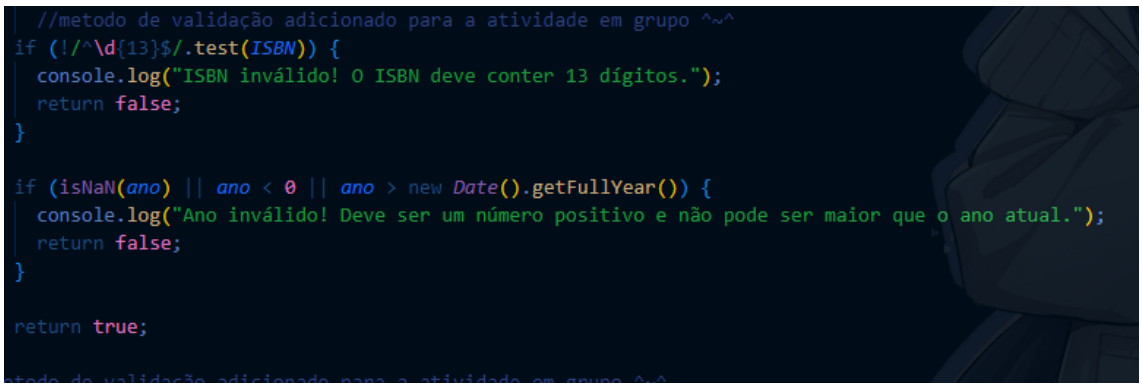
Código explícito: Criação de um objeto membroData para encapsular as informações antes de adicioná-las ao array de membros, o que torna o código mais fácil de entender.

Princípio da Simplicidade: A função faz uma coisa apenas, o que segue o princípio de funções pequenas e com responsabilidades bem definidas.

Caso 3: Validação de dados não desenvolvida.

Atualmente, não há validação dos dados inseridos. Por exemplo, o campo ISBN aceita qualquer valor digitado, assim como o ano de publicação pode ser um número além do esperado.

Proposta/Melhoria



```
//metodo de validação adicionado para a atividade em grupo ^~^
if (!/^\d{13}$/.test(ISBN)) {
  console.log("ISBN inválido! O ISBN deve conter 13 dígitos.");
  return false;
}

if (isNaN(ano) || ano < 0 || ano > new Date().getFullYear()) {
  console.log("Ano inválido! Deve ser um número positivo e não pode ser maior que o ano atual.");
  return false;
}

return true;
```

Adicionar validações para garantir que o ISBN siga um formato específico e que o ano de publicação seja válido (por exemplo, um número positivo e dentro de um intervalo aceitável). Além disso, valores em branco não devem ser aceitos

Conclusão:

O Projeto atende a muitas boas práticas, como modularidade e separação clara das funcionalidades, mas há áreas que poderiam ser melhoradas em termos de reutilização e simplificação de código. As principais melhorias sugeridas focam em centralizar a lógica repetida em métodos reutilizáveis, melhorando assim a manutenibilidade e a clareza do código.

CheckList

[V] Princípio DRY (Don't Repeat Yourself): Eliminar repetições desnecessárias de lógica, como a verificação da existência do arquivo.

[V] Princípio SRP (Princípio da Responsabilidade Única): Cada função deve ter uma única responsabilidade, evitando misturar várias lógicas.

[V] Código explícito: O uso de nomes descritivos e criação de variáveis intermediárias para tornar o código claro.

[V] Manutenibilidade: Simplificação da lógica para facilitar futuras alterações e adições ao código.