03

Entendendo a validação à fundo

Nosso código de validação é um pouco complexo. Se algo não ficou claro para você, recomendo que leia a explicação abaixo, caso contrário, fique à vontade para pular para a próxima atividade!

Para começar, vamos partir do componente **Formulario**.

Formulário.js

Vimos no capítulo anterior que validar um campo específico é relativamente mais fácil do que um formulário inteiro.

Nosso objetivo é fazer com que nosso código seja genérico o suficiente a ponto de conseguir validar todos os campos do meu formulário atual e também de outros formulários que podem vir a existir na minha aplicação. Em outras palavras, estamos desenvolvendo uma solução para validação e não uma validação para aquele formulário em específico.

Para validarmos de forma genérica, precisamos de algumas informações básicas, por isso adicionamos algumas outras chaves em cada regra nossa que é criada no **constructor**

```
this.validador = new FormValidator([
            {
              campo: 'nome',
              metodo: 'isEmpty',
              validoQuando: false,
              mensagem: 'Entre com um nome'
            },
            {
              campo: 'livro',
              metodo: 'isEmpty',
              validoQuando: false,
              mensagem: 'Entre com um livro'
            },
            {
                campo: 'preco',
                metodo: 'isInt',
                args: [{min: 0, max: 99999}],
                validoQuando: true,
                mensagem: 'Entre com um valor numérico'
              }
          ]);
```

Perceba como a condição de comparação muda de acordo com o campo, enquanto a definição de válido para os campos **nome** e **livro** é diferente da definição de válido para o campo **preco**.

Enquanto **livro** e **nome** são válidos quando o retorno é **false**, **preco** é válido para retorno **true**. Por isso criamos a chave **validoQuando**.

Precisamos também de uma mensagem específica para cada campo, para que consigamos emitir uma mensagem específica de cada campo.

No **submitFormulario**, condicionamos a submissão à validade dos campos e capturamos os campos inválidos e emitimos a mensagem específica.

FormValidator.js

Agora no **FormValidator**, onde está nossa lógica de validação.

Nessa classe, temos dois métodos: O **valido** e o **valida**.

Enquanto o **valido** cria um grande objeto dizendo que o formulário é válido, o método **valida** verifica cada campo de acordo com as regras definidas e **modifica** esse objeto criado pelo **valido** de maneira a tornar esse formulário inválido caso seja necessário.

Dando uma olhada no método válido:

```
valido() {
    //criação do objeto
        const validacao = {};

    //populando o objeto de acordo com a quantidade de campos
    //criando a chave isInvalid e atribuindo false para cada
    //**TODOS OS CAMPOS COMEÇAM VÁLIDOS!**
        this.validacoes.map(regra => (
            validacao[regra.campo] = { isInvalid: false, message: '' }
        ));

    //retorna um grande objeto com uma chave externa isValid
    //junto com todos os outros campos.
        return { isValid: true, ...validacao };

    }

}
```

Para ficar mais claro, a estrutura do objeto retornado é:

```
{
isValid: true,
nome: { isInvalid: false, message: '' },
livro: { isInvalid: false, message: '' },
preco: { isInvalid: false, message: ''}
}
```

Agora com esse objeto em mãos, podemos começar a verificar os valores de entrada do usuário e modificá-lo caso necessário.

Com isso, vamos agora para o método **valida**

O objetivo do método **valida** é verificar a entrada do usuário utilizando o método que foi passado e comparando com o valor de **validoQuando**

Vamos ver esse método:

```
valida(state) {

        //itera pelo array de regras de validação e constrói
        //um objeto validacao e retorna-o


        //começa assumindo que está tudo válido, recebe o
    //objeto do método valido.
        let validacao = this.valido();

        this.validacoes.forEach(regra => {

            //Se o campo não tiver sido marcado
            //anteriormente como invalido por uma regra.
            if (!validacao[regra.campo].isInvalid) {
                //Determina o valor do campo, o método a ser invocado
                //e os argumentos opcionais pela definição da regra
                const campoValor = state[regra.campo.toString()];
                const args = regra.args || [];
                //if ternário para estar preparado caso
                //alguém passe o método direto sem ser string
                const metodoValidacao = typeof regra.metodo === 'string' ?
                    validador[regra.metodo] : regra.metodo;

                //invoca o método específico da regra
                if (metodoValidacao(campoValor, ...args, state) !== regra.validoQuando) {

                    //modifica o objeto no campo específico
                    validacao[regra.campo] = {
                        isInvalid: true,
                        message: regra.mensagem
                    };
                    validacao.isValid = false;

                }
            }


        });
        return validacao;
    }
```

Repare que quando recuperamos o método a ser utilizado, estamos nos blindando caso alguém decida passar o método diretamente e não apenas o nome em formato de string.

Chamamos o método específicado e comparamos com o valor de **validoQuando**, se for diferente, modificamos nosso objeto, alterando a chave **isInvalid** para **true** e passando a menssagem que recebemos nas regras.

Feito isso, alteramos a chave **isValid** para **false**, agora que nosso formulário se tornou inválido e então retornamos todo o objeto.

Apesar de parecer um pouco completo, se pensarmos passo à passo as coisas ficam mais fáceis.

Se algo ainda não ficou claro para você, não hesite em me procurar no fórum!