09

Trabalhando com Enuns

Transcrição

Vamos alterar nosso código para não permitir que negociações feitas no sábado ou no domingo sejam cadastradas.

```
// app-src/controllers/NegociacaoController.ts

  adiciona(event: Event) {

      event.preventDefault();

      let data = new Date(this._inputData.val().replace(/-/g, ','));

      if(data.getDay() == 0 || data.getDay() == 6) {

          this._mensagemView.update('Somente negociações em dias úteis, por favor!');
          return
      }

      const negociacao = new Negociacao(
          data,
          parseInt(this._inputQuantidade.val()),
          parseFloat(this._inputValor.val())
      );

      this._negociacoes.adiciona(negociacao);

      this._negociacoesView.update(this._negociacoes);
      this._mensagemView.update('Negociação adicionada com sucesso!');
  }
```

No entanto, ver em nosso código algo como `==0` e `==6` não nos diz muita coisa. Poderíamos até ter criado uma variável como `sabado` ou `domingo`, contudo o TypeScript possui um recurso mais elegante para o problema que estamos vendo, as enumerations.

Dentro do mesmo arquivo que definimos nosso controller, vamos criar a enum `DiaDaSemana`:

```
enum DiaDaSemana {
    Domingo,
    Segunda,
    Terca,
    Quarta,
    Quinta,
    Sexta,
    Sabado,
}
```

Por padrão, o valor de cada um começa de 0 e vai até 6, pois `Sabado` é o sétimo elemento.

Usando em nosso código temos:

```
    adiciona(event: Event) {

        event.preventDefault();

        let data = new Date(this._inputData.val().replace(/-/g, ','));

        if(data.getDay() == DiaDaSemana.Sabado || data.getDay() == DiaDaSemana.Domingo) {

            this._mensagemView.update('Somente negociações em dias úteis, por favor!');
            return
        }

        const negociacao = new Negociacao(
            data,
            parseInt(this._inputQuantidade.val()),
            parseFloat(this._inputValor.val())
        );

        this._negociacoes.adiciona(negociacao);

        this._negociacoesView.update(this._negociacoes);
        this._mensagemView.update('Negociação adicionada com sucesso!');
    }
```

Podemos deixar o nosso código ainda mais claro, criando um método privado em nosso controller, chamado `ehDiaUtil`, para nos retornar `true` ou `false` se a data na negociação é um dia útil:

```
import { NegociacoesView, MensagemView } from '../views/index';
import { Negociacao, Negociacoes } from '../models/index';

export class NegociacaoController {

    private _inputData: JQuery;
    private _inputQuantidade: JQuery;
    private _inputValor: JQuery;
    private _negociacoes = new Negociacoes();
    private _negociacoesView = new NegociacoesView('#negociacoesView');
    private _mensagemView = new MensagemView('#mensagemView');

    constructor() {
        this._inputData = $('#data');
        this._inputQuantidade = $('#quantidade');
        this._inputValor = $('#valor');
        this._negociacoesView.update(this._negociacoes);
    }

    adiciona(event: Event) {

        event.preventDefault();

        let data = new Date(this._inputData.val().replace(/-/g, ','));

        if(!this._ehDiaUtil(data)) {

            this._mensagemView.update('Somente negociações em dias úteis, por favor!');
```

```
            return
        }

        const negociacao = new Negociacao(
            data,
            parseInt(this._inputQuantidade.val()),
            parseFloat(this._inputValor.val())
        );

        this._negociacoes.adiciona(negociacao);

        this._negociacoesView.update(this._negociacoes);
        this._mensagemView.update('Negociação adicionada com sucesso!');
    }

    private _ehDiaUtil(data: Date) {

        return data.getDay() != DiaDaSemana.Sabado && data.getDay() != DiaDaSemana.Domingo;
    }
}

enum DiaDaSemana {

    Domingo,
    Segunda,
    Terca,
    Quarta,
    Quinta,
    Sexta,
    Sabado
}
```

Excelente, melhoramos um pouco mais a legibilidade do nosso código.