05

Template dinâmico

Transcrição

A questão agora, é que o método `update()` precisa receber o modelo, ou seja, o dado no qual a função `template()` se baseará para gerar o template.

```
class NegociacoesView {

    private _elemento: Element;

    constructor(seletor: string) {

        this._elemento = document.querySelector(seletor);
    }

    update(model: Negociacoes): void {

        this._elemento.innerHTML = this.template(model);
    }

    template(modelo: Negociacoes): string {

        return `
        <table class="table table-hover table-bordered">
            <thead>
                <tr>
                    <th>DATA</th>
                    <th>QUANTIDADE</th>
                    <th>VALOR</th>
                    <th>VOLUME</th>
                </tr>
            </thead>

            <tbody>
            </tbody>

            <tfoot>
            </tfoot>
        </table>
        `
    }
}
```

Agora, precisamos gerar uma `tr` para cada negociação encapsulada por `Negociacoes`:

```
class NegociacoesView {

    private _elemento: Element;

    constructor(seletor: string) {
```

```
        this._elemento = document.querySelector(seletor);
    }

    update(model: Negociacoes) {

        this._elemento.innerHTML = this.template(model);
    }

    template(model: Negociacoes): string {

        return `
        <table class="table table-hover table-bordered">
            <thead>
                <tr>
                    <th>DATA</th>
                    <th>QUANTIDADE</th>
                    <th>VALOR</th>
                    <th>VOLUME</th>
                </tr>
            </thead>

            <tbody>

            ${model.paraArray().map(negociacao =>
                `
                    <tr>
                        <td>${negociacao.data.getDate()}/${negociacao.data.getMonth()+1}/${nego
                        <td>${negociacao.quantidade}</td>
                        <td>${negociacao.valor}</td>
                        <td>${negociacao.volume}</td>
                    </tr>
                `).join('')}
            </tbody>

            <tfoot>
            </tfoot>
        </table>
        `
    }
}
```

Por fim, vamos chamar o método `this._negociacoesView.update(this._negociacoes)` na inicialização do nosso controller e após a inclusão de novas negociações.

```
class NegociacaoController {

    private _inputData: HTMLInputElement;
    private _inputQuantidade: HTMLInputElement;
    private _inputValor: HTMLInputElement;
    private _negociacoes = new Negociacoes();
    private _negociacoesView = new NegociacoesView('#negociacoesView');

    constructor() {
        this._inputData = <HTMLInputElement>document.querySelector('#data');
        this._inputQuantidade = <HTMLInputElement>document.querySelector('#quantidade');
```

```
        this._inputValor = <HTMLInputElement>document.querySelector('#valor');

       // atualiza a view para exibir os dados do modelo, vazio
        this._negociacoesView.update(this._negociacoes);
    }

    adiciona(event: Event) {

        event.preventDefault();

        const negociacao = new Negociacao(
            new Date(this._inputData.value.replace(/-/g, ',')),
            parseInt(this._inputQuantidade.value),
            parseFloat(this._inputValor.value)
        );

        this._negociacoes.adiciona(negociacao);

        // depois de adicionar, atualiza a view novamente para refletir os dados
        this._negociacoesView.update(this._negociacoes);
    }
}
```

Podemos adicionar quantas negociações desejamos que todas serão exibidas em nossa tabela.