01

# jQuery, tem espaço?

### Transcrição

*Começando deste ponto? Você pode fazer o download (https://s3.amazonaws.com/caelum-online-public/typescript/05-alurabank.zip) completo do projeto do capítulo anterior e continuar seus estudos a partir deste capítulo.*

Nossa aplicação continua funcionando e evoluindo. Realizamos manipulação de DOM em alguns pontos da nossa aplicação. Apesar da nossa manipulação não gastar muitas linhas de código, podemos incorporar o jQuery em nosso projeto. Essa biblioteca de mais de 10 anos conquistou o coletivo imaginários dos desenvolvedores front-end e ainda é utilizada em vários projetos, possuindo um ecossistema de plugins gigantesco. Com ela, podemos nos blindar com problemas de compatibilidade de muitos smartphones Android que usam versões antigas do Webkit, que foi substituído pelo Blink nas novas versões do Chrome. Além disso, podemos utilizar a sintaxe elegante que o jQuery oferece.

Já temos o script do jQuery dentro da pasta `alurabank/app/lib`. Vamos importá-lo em `app/index.html` como primeiro `script`:

```
<!-- app/index.html -->
<!-- código anterior omitido -->
    <script src="lib/jquery.min.js"></script>
    <script src="js/models/Negociacao.js"></script>
    <script src="js/controllers/NegociacaoController.js"></script>
    <script src="js/models/Negociacoes.js"></script>
    <script src="js/views/View.js"></script>
    <script src="js/views/NegociacoesView.js"></script>
    <script src="js/views/MensagemView.js"></script>
    <script src="js/app.js"></script>
<!-- código anterior -->
```

Agora que já temos o jQuery importado em nosso projeto, vamos começar alterando a classe `app/ts/views/View.ts`. Em vez de utilizarmos `document.querySelector`, vamos utilizar o famoso aliás para o jQuery, o `$`:

```
abstract class View<T> {

    private _elemento: Element;

    constructor(seletor: string) {

       // erro de compilação
        this._elemento = $(seletor);
    }

    update(model: T) {

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

```
    abstract template(model: T): string;
}
```

Em nossa primeira alteração, recebemos um erro de compilação:

```
Cannot find name '$'.
```

O jQuery vive no escopo global e através do JavaScript podemos acessá-lo sem delongas. Contudo, com TypeScript, o compilador não entende que `$` vive no escopo global e espera encontrar a declaração da variável `$` em nosso código. Faz todo sentido, pois o compilador pode nos avisar caso estejamos acessando uma variável que não foi declarada ou que escrevermos seu nome errado.

Podemos fazer com que o compilador faça "vista grossa" para a variável `$`. Para isso, no início do nosso arquivo, vamos colocar a instrução `declare var $:any`:

```
declare var $: any;

abstract class View<T> {

    private _elemento: Element;

    constructor(seletor: string) {

        this._elemento = $(seletor);
    }

    update(model: T) {

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

    abstract template(model: T): string;
}
```

Com essa manobra, silenciamos o compilador. Agora, para ele, `$` é um objeto do tipo `any`. Por fim, no método `update`, vamos trocar de `this._elemento.innerHTML` para `this._elemento.html`, o método do jQuery Object responsável em atualizar o elemento do DOM com base numa string:

```
declare var $: any;

abstract class View<T> {

    private _elemento: Element;

    constructor(seletor: string) {

        this._elemento = $(seletor);
    }

    update(model: T) {

        // erro de compilação
```

```
        // Property 'html' does not exist on type 'Element'
        this._elemento.html(this.template(model));
    }

    abstract template(model: T): string;
}
```

Ganhamos outro erro de compilação. Faz sentido, porque o tipo `Element` não possui o método `html`. E agora? Vamos mudar a propriedade `_elemento` do tipo `Element` para `any`. Isso é possível, pois estamos definindo o tipo `any` explicitamente:

```
declare var $: any;

abstract class View<T> {

    protected _elemento: any;

    constructor(seletor: string) {

        this._elemento = $(seletor);
    }

    update(model: T) {

        this._elemento.html(this.template(model));
    }

    abstract template(model: T): string;
}
```

Perfeito, nosso código compila, inclusive podemos testar nossa aplicação e tudo funcionará, mas essa solução deixa muito à desejar. Primeiro, trabalhando com o tipo `any` não temos qualquer verificação do compilador do TypeScript e nem autocomplete. Estamos abdicando de um recurso poderoso da linguagem. Por exemplo, podemos escrever errado o nome do método `htm` e o TypeScript não saberá nos dizer antecipadamente que o método não existe. E agora?

A boa notícia é que o TypeScript possui um recurso interessantíssimo, que pode trazer de volta sua análise sintática, checagem de tipos e autocomplete não só com jQuery, mas com várias bibliotecas famosas do mercado.