06

Babel e transcompilação de módulos

Transcrição

O arquivo `boot.js` instanciará a controller, sendo responsável por associar os métodos do `NegociacaoController` com os eventos da View. Porém, o nosso código ainda não vai funcionar. Isto porque estamos usando como *loader* o `system.js` , por isso, os módulos que o Babel transcompila realizarão a ação usando a sintaxe do mesmo arquivo para auxiliar a importação.

O *transpiler* é importante neste processo porque ele mudará o código dos módulos para adequá-los ao *loader*. De volta ao Terminal, na pasta `client` , instalaremos o novo módulo - um plugin do Babel.

```
npm install babel-plugin-transform-es2015-modules-systemjs@6.9.0 --save-dev
```

O plugin transforma o código do ES2015 para usar o SystemJS. É fundamental instalá-lo no Babel para que tudo funcione corretamente.

Talvez, você tenha achado "burocrático" usar o módulo do ES6. Eu concordo...

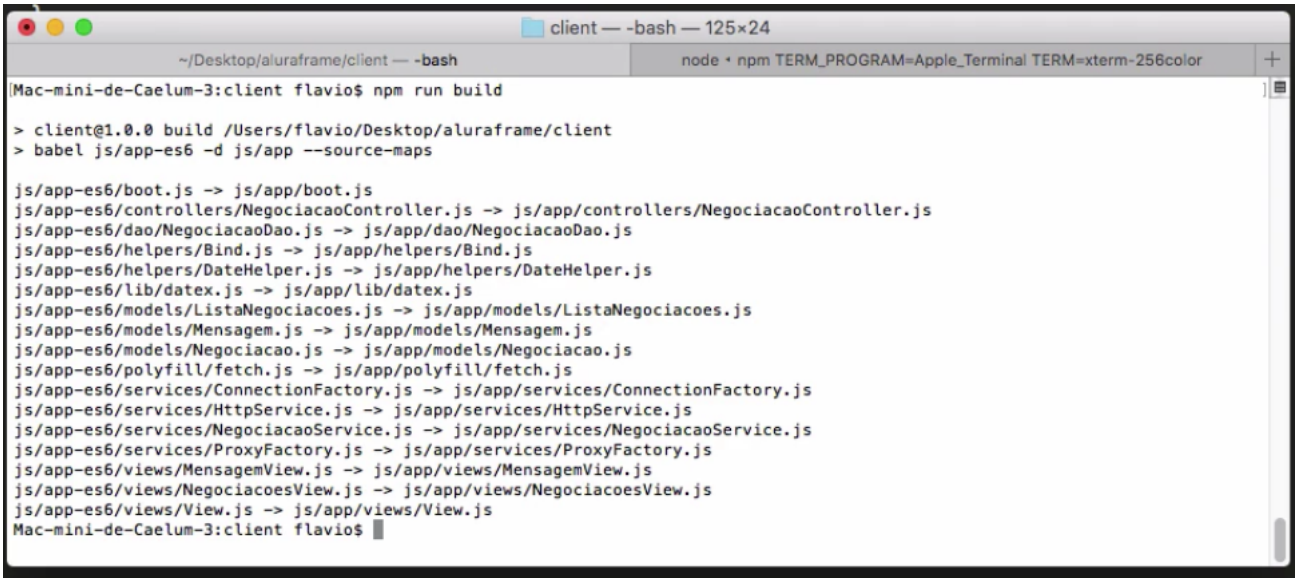
Após fazermos a gravação do plugin, vamos configurar para que o Babel utilize o recurso recém instalado, no arquivo `.babelrc` .

```
{
   "presets" : ["es2015"],
   "plugins" : ["transform-es2015-modules-systemjs"]
}
```

Sem a configuração, o Babel não compilará os módulos adequadamente para a importação no `System.js` . Em seguida, no Terminal, vamos fazer a compilação com o comando `build` :

```
npm run build
```

O processo de compilação funcionará corretamente:



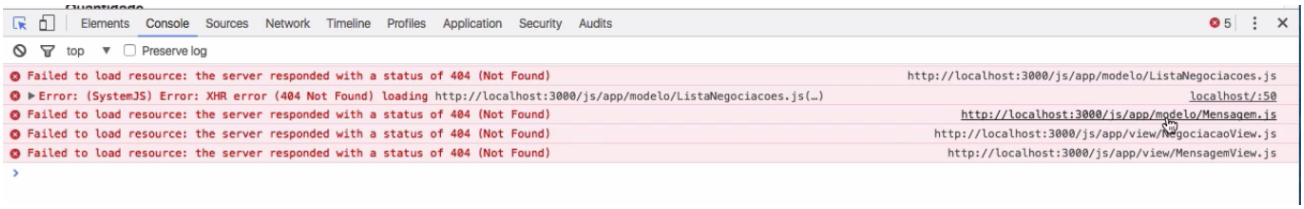
Se analisarmos os arquivos gerados pelo *transpiler*, localizados na pasta `app`, veremos que aparecerá uma sintaxe nova. Por exemplo, no arquivo `NegociacaoDao`, encontraremos o `System.register` aparecendo no seguinte trecho:

```
System.register([], function(_export, _context) {
    "use strict";

    var _createClass, NegociacaoDao;

    function _classCallCheck(instance, Constructor) {
        if (!(instance instanceof Constructor)) {
            throw new TypeError("Cannot call a class as a function");
        }
    }
//...
```

Conseguimos importar o loader. Depois, no Terminal, deixaremos o `watch` ligado.

```
npm run watch
```

Ao recarregarmos a página, ela não funcionará. No Console, veremos diversas mensagens de erro.



Não foi encontrado o `ListaNegociacoes`, `Mensagem`, além de outros arquivos. Teremos que fazer algumas correções nos caminhos das importações do `NegociacaoController.js`. Onde escrevemos `modelo`, substituiremos por `models`. Também modificaremos de `view` para `views` e `NegociacaoView` para `NegociacoesView`:

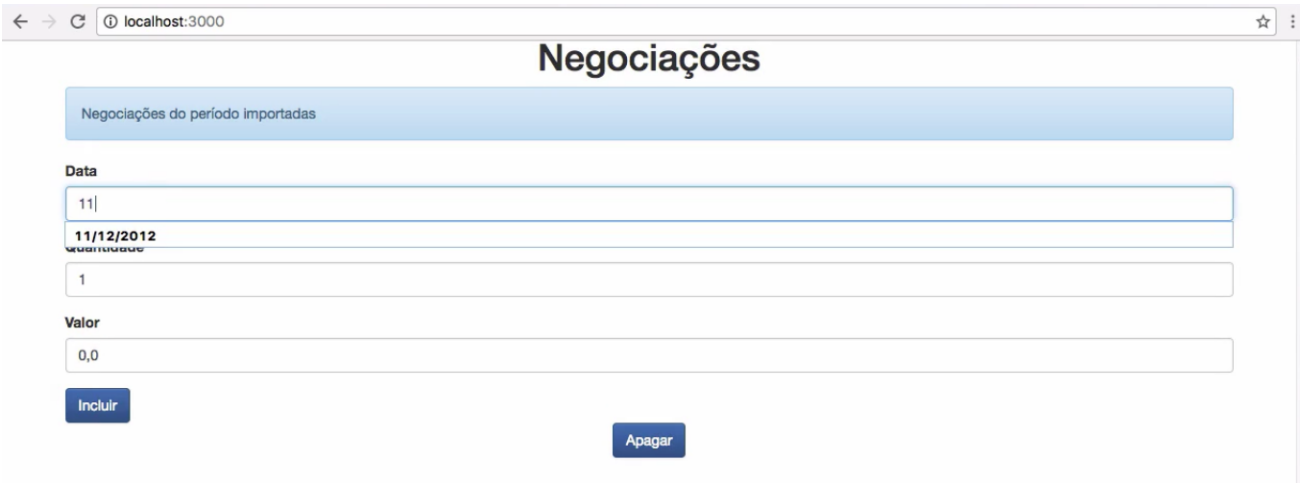
```
import {ListaNegociacoes} from '../models/ListaNegociacoes';
import {Mensagem} from '../models/Mensagem';
import {NegociacoesView} from '../views/NegociacoesView';
import {MensagemView} from '../views/MensagemView';
```

```
import {NegociacaoService} from '../services/NegociacaoService';
import {DateHelper} from '../helpers/DateHelper';
import {Bind} from '../helpers/Bind';
import {Negociacao} from '../models/Negociacao';

export class NegociacaoController {

    constructor() {}

//...
```

Observe que faltou também adicionar o `export` no `NegociacaoController` para que este possa ser importado pelo `boot.js`. Depois, em `NegociacoesView.js`, importaremos o `DateHelper`:

```
import {View} from './View';
import {DateHelper} from '../helpers/DateHelper';

export class NegociacoesView extends View {

  constructor(elemento) {

      super(elemento);
  }
//...
```

Apesar do erro, não nós que importamos o scripts. Foi o System.JS quem realizou as importações. Com as alterações, no Console já não veremos nenhuma mensagem de erro.



Se cadastrarmos uma nova negociação, ela será importada corretamente. Agora a nossa aplicação inteira está usando o sistema de módulos do ES2015.