

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:

```

Mac-mini-de-Caelum-3:client flavio$ npm run build

> client@1.0.0 build /Users/flavio/Desktop/aluraframe/client
> babel js/app-es6 -d js/app --source-maps

js/app-es6/boot.js -> js/app/boot.js
js/app-es6/controllers/NegociacaoController.js -> js/app/controllers/NegociacaoController.js
js/app-es6/dao/NegociacaoDao.js -> js/app/dao/NegociacaoDao.js
js/app-es6/helpers/Bind.js -> js/app/helpers/Bind.js
js/app-es6/helpers/DateHelper.js -> js/app/helpers/DateHelper.js
js/app-es6/lib/datex.js -> js/app/lib/datex.js
js/app-es6/models/ListaNegociacoes.js -> js/app/models/ListaNegociacoes.js
js/app-es6/models/Mensagem.js -> js/app/models/Mensagem.js
js/app-es6/models/Negociacao.js -> js/app/models/Negociacao.js
js/app-es6/polyfill/fetch.js -> js/app/polyfill/fetch.js
js/app-es6/services/ConnectionFactory.js -> js/app/services/ConnectionFactory.js
js/app-es6/services/HttpService.js -> js/app/services/HttpService.js
js/app-es6/services/NegociacaoService.js -> js/app/services/NegociacaoService.js
js/app-es6/services/ProxyFactory.js -> js/app/services/ProxyFactory.js
js/app-es6/views/MensagemView.js -> js/app/views/MensagemView.js
js/app-es6/views/NegociacoesView.js -> js/app/views/NegociacoesView.js
js/app-es6/views/View.js -> js/app/views/View.js
Mac-mini-de-Caelum-3:client flavio$

```

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");
    }
  }

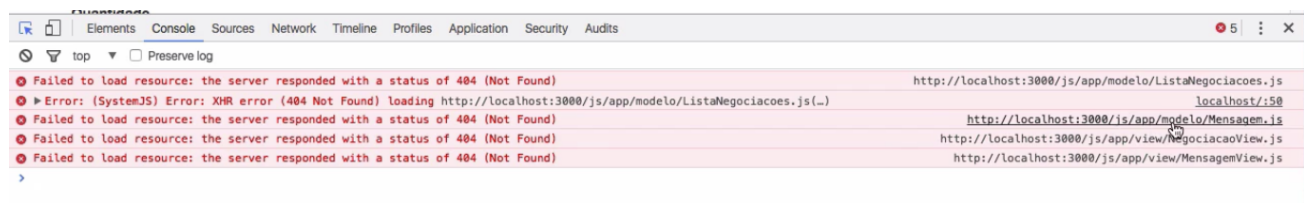
  //...

```

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`:

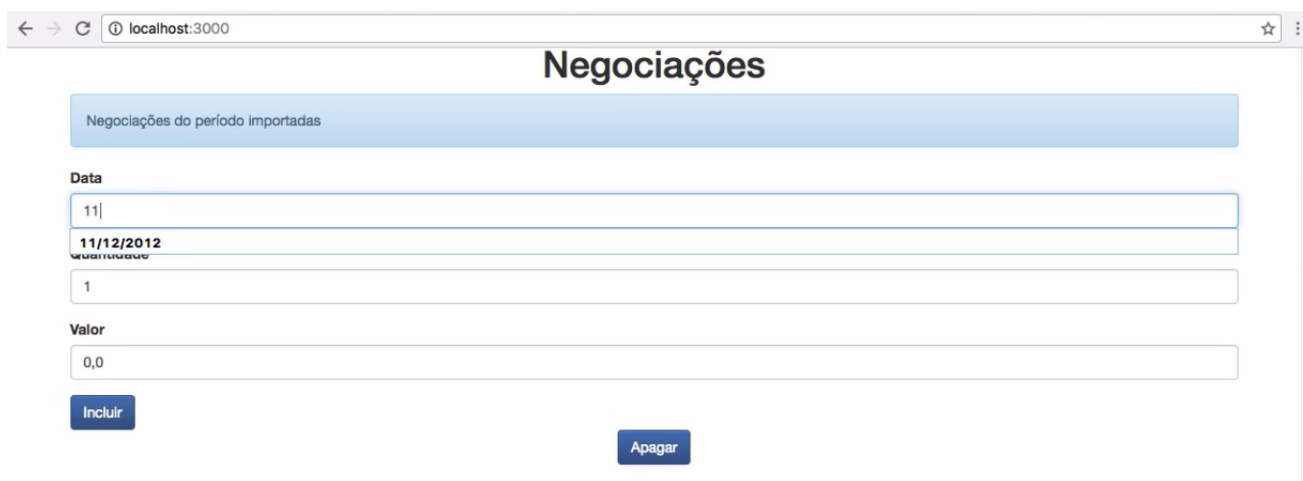
```
import {View} from './View';
import {DateHelper} from '../helpers/DateHelper';

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.



The screenshot shows a web browser window with the address bar displaying 'localhost:3000'. The page title is 'Negociações'. Below the title is a light blue header bar with the text 'Negociações do período importadas'. The main content area contains a form with two sections: 'Data' and 'Valor'. The 'Data' section has a date input field showing '11/12/2012' and a quantity input field showing '1'. The 'Valor' section has a value input field showing '0,0'. At the bottom of the form are two buttons: 'Incluir' and 'Apagar'.

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