04

Compilando arquivos em tempo real

Transcrição

Nossa estratégia deixa um pouco a desejar, porque sempre que alterarmos um arquivo teremos que ir no Terminal e o rodar o comando `build`.

```
npm run build
```

A ação deve ser realizada quando fazemos o deploy, para garantir que está tudo compilado. Mas será que o desenvolvedor vai querer ter esta responsabilidade a todo momento? Por isso, o Babel vem como *watcher*, um observador de arquivos que automaticamente fará o processo de transcompilação quando um arquivo for alterado. Para habilitá-lo, vamos no arquivo `package.json` e adicionaremos o `watch`:

```
{
  "name": "client",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts"": {
      "test": "echo \"Error: no test specified\" && exit 1",
      "build": "babel js/app-es6 -d js/app --source-maps",
      "watch": "babel js/app-es6 -d js/app --source-maps --watch"
    },
    "author": "",
    "license": "ISC",
    "devDependencies": {
      "babel-cli": "^6.10.1",
      "babel-preset-es2015": "^6.9.0"
  }
}
```

No Terminal, vamos executar o `watch`:

```
npm run watch
```

Ele irá compilar os arquivos e o Terminal ficará monitorando a modificação de todos eles. O processo de compilação correrá bem e ao recarregarmos a página de cadastro, tudo funcionará corretamente.

Para quem nunca havia trabalhado com o transpiler, mostramos que é possível trabalhar facilmente com Babel, também vimos como é o processo de transcompilação. Mesmo se usarmos recursos ainda mais avançados do JavaScript, o Babel conseguirá transcompilar para um código compatível para o ES5. Desta forma, conseguimos trabalhar com o que há de mais moderno, sem nos preocuparmos com compatibilidade.

Porém, se você está trabalhando com um navegador que não suporta Promise, terá que utilizar um polyfill para o mesmo. Neste caso, o Babel não resolverá. Temos ainda alguns truques, mas que ficarão para outro curso.