02

Antecipando erros do futuro

Começando deste ponto? Você pode fazer o DOWNLOAD (https://s3.amazonaws.com/caelum-online-public/gulp/stages/05-capitulo.zip) completo do projeto do capítulo anterior e continuar seus estudos a partir deste capítulo. Dentro da pasta **projeto**, não esqueça de executar no terminal o comando **npm install** para baixar novamente todas as dependências.

Aprendemos a criar tarefas úteis no Gulp, mas antes de continuarmos, vamos fazer uma alteração em nosso projeto. Desativaremos nosso banner rotativo, deixando apenas a segunda imagem, aquela que é exibida por último. Para isso, vamos alterar `projeto/src/js/home.js`. Nele, há uma linha que declara uma array com dois elementos, nossos banners:

```
// código anterior omitido
var banners = ["img/destaque-home-2.png", "img/destaque-home.png"];
// código posterior omitido
```

No lugar de removermos o array, vamos remover apenas um de seus itens, o que facilitará a volta do nosso banner, caso seja necessário. Alterando a linha temos:

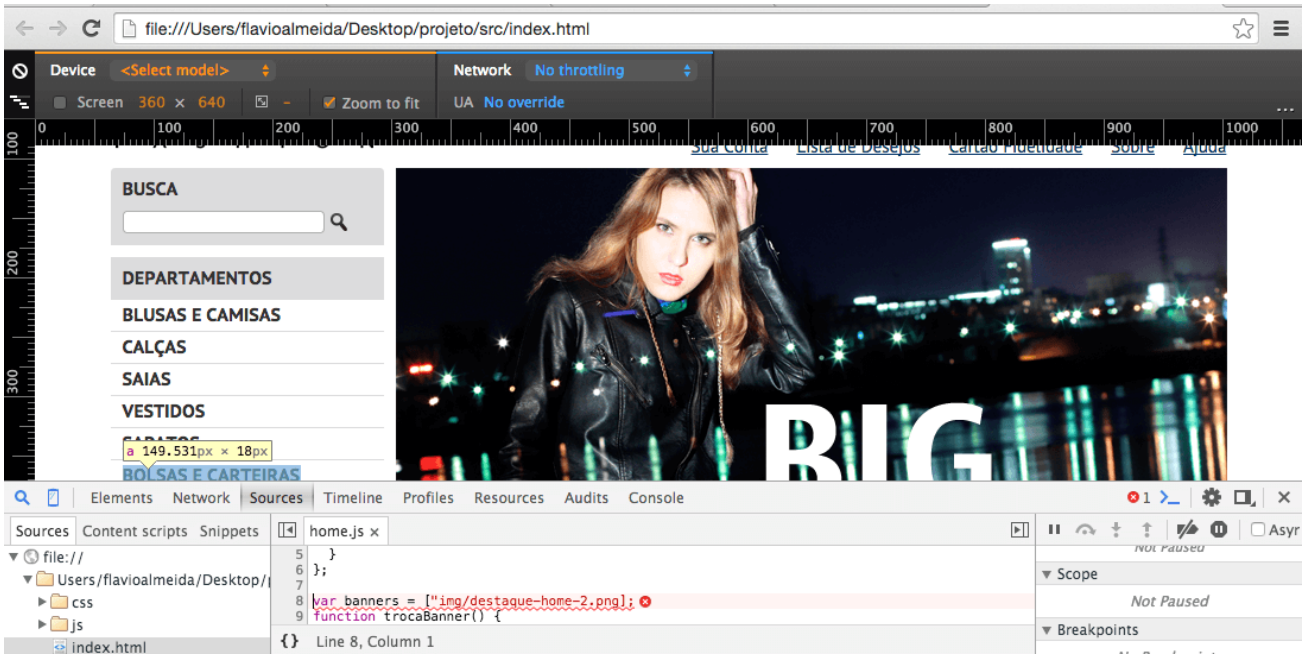
```
// código anterior omitido

// OBS: faltou as aspas, intencional aqui
var banners = ["img/destaque-home-2.png];
// código posterior omitido
```

Excelente. E agora? Será que nossa imagem do banner continuará a ser exibida? Bem, para termos certeza, vamos visualizar nossa página.

Temos um problema: não é a segunda imagem do banner que está sendo exibida, se não é, é porque nosso script falhou. Abrindo o console do navegador vemos:

O console é nosso amigo e nos diz a linha problemática, a de número 8:



Faltou fechar as aspas da string. A correção não demora e um teste demonstra que está tudo certo.

Nesse pequena tarefa, repare que precisamos abrir um navegador para testarmos nosso código por erros de sintaxe. Imagine se conseguíssemos fazer isso a cada salvamento de nossos scripts, sem a necessidade de termos que abrir o navegador toda vez. É claro que scripts que manipulam DOM precisam do navegador, mas o que estamos interessados aqui é saber se não cometemos nenhum erro de sintaxe.

Lint e detecção de erros em JavaScript

Existe um plugin do gulp chamado gulp-jshint (https://github.com/spalger/gulp-jshint), que analisa nosso arquivo JS e nos dá dicas (*hints*) de problemas ou melhorias que podem ser feitas. Na computação, existe um termo para esse procedimento que procura alguma estrutura suspeita ou não conformante com determinada sintaxe, esse procedimento se chama **lint**. Vamos instalá-lo e importá-lo em nosso `gulpfile.js` :

```
npm install gulp-jshint@1.11.2 --save-dev
```

gulpfile.js

```
var gulp = require('gulp')
  ,imagemin = require('gulp-imagemin')
  ,clean = require('gulp-clean')
  ,concat = require('gulp-concat')
  ,htmlReplace = require('gulp-html-replace')
  ,uglify = require('gulp-uglify')
  ,usemin = require('gulp-usemin')
  ,cssmin = require('gulp-cssmin')
  ,browserSync = require('browser-sync')
  ,jshint = require('gulp-jshint');
```

Lint sob demanda

Muito bem. Estamos acostumados a criar tarefas no Gulp, mas temos um caso peculiar aqui. Queremos que o `jshint` analise nosso arquivo apenas quando for salvo. Para conseguirmos isso, precisamos monitorar qualquer arquivo `js` que seja modificado para em seguida descobrirmos que arquivo é esse e processá-lo com o `jshint`.

No lugar de criarmos uma nova tarefa, vamos aproveitar e adicionar um novo `watcher` dentro da tarefa `server`. Ele monitorará arquivos `.js` e toda vez que algum deles for alterado, `jshint` o processará indicando em nosso console erros que possam existir:

```
gulp.task('server', function() {
    browserSync.init({
        server: {
            baseDir: 'src'
        }
    });

    gulp.watch('src/**/*').on('change', browserSync.reload);

    // novidade aqui
    gulp.watch('src/js/**/*.js').on('change', function(event) {
        console.log("Linting " + event.path);
        gulp.src(event.path)
            .pipe(jshint())
    });

});
```

Nosso `watcher` é bem parecido com o que já fizemos, porém na função que será executada toda vez que o arquivo for modificado, recebemos um parâmetro. Este parâmetro nos dá acesso ao arquivo que foi modificado, através de `event.path`. É o `path` do arquivo modificado que passamos para o stream de leitura do gulp. Por fim, usamos a função `.pipe` para passar esse stream ao `jshint()`.

Já podemos testar finalizando a tarefa `server` e rodando-a novamente. Vamos simplesmente abrir o arquivo problemático `projetos/src/js/home.js` e salvá-lo novamente para vermos o `jshint` em ação no console.

Onde está meu relatório de erros?

Interessante demais, nada acontece! Sabemos que nosso arquivo ainda está com problema, mas nada é exibido no terminal! O problema é a ausência de um `reporter`, isto é, alguém que saiba exibir para nós as mensagens do `jshint`. A boa notícia é que este plugin do Gulp já vem com um `reporter` padrão. Vamos habilitá-lo:

```
// código anterior comentado

gulp.task('server', function() {
    browserSync.init({
        server: {
            baseDir: 'src'
        }
    });

    gulp.watch('src/**/*').on('change', browserSync.reload);

    gulp.watch('src/js/**/*.js').on('change', function(event) {
```

```
        console.log("Linting " + event.path);
        gulp.src(event.path)
            .pipe(jshint())
            .pipe(jshint.reporter());
    });

});
```

Veja que passamos o stream do `jshint` para o stream `jshint.reporter()`. Agora, vamos salvar novamente o arquivo e verificar se algo é exibido no terminal, mas claro, precisamos parar nossa tarefa `server` e executá-la novamente, porque alteramos nosso `gulpfile.js`:

```
Linting projeto/src/js/home.js
/Users/flavioalmeida/Desktop/projeto/src/js/home.js: line 8, col 42, Unclosed string.
/Users/flavioalmeida/Desktop/projeto/src/js/home.js: line 9, col 25, Unclosed string.
/Users/flavioalmeida/Desktop/projeto/src/js/home.js: line 10, col 62, Unclosed string.
/Users/flavioalmeida/Desktop/projeto/src/js/home.js: line 11, col 23, Unclosed string.
/Users/flavioalmeida/Desktop/projeto/src/js/home.js: line 12, col 2, Unclosed string.
/Users/flavioalmeida/Desktop/projeto/src/js/home.js: line 13, col 32, Unclosed string.
/Users/flavioalmeida/Desktop/projeto/src/js/home.js: line 14, col 1, Unclosed string.
/Users/flavioalmeida/Desktop/projeto/src/js/home.js: line 15, col 42, Unclosed string.
/Users/flavioalmeida/Desktop/projeto/src/js/home.js: line 16, col 1, Unclosed string.
/Users/flavioalmeida/Desktop/projeto/src/js/home.js: line 17, col 39, Unclosed string.
/Users/flavioalmeida/Desktop/projeto/src/js/home.js: line 18, col 53, Unclosed string.
/Users/flavioalmeida/Desktop/projeto/src/js/home.js: line 19, col 4, Unclosed string.
/Users/flavioalmeida/Desktop/projeto/src/js/home.js: line 8, col 16, Unclosed string.
/Users/flavioalmeida/Desktop/projeto/src/js/home.js: line 8, col 15, Unmatched '['.
/Users/flavioalmeida/Desktop/projeto/src/js/home.js: line 19, col undefined, Unrecoverable synta

15 errors
```

Uau! 15 erros irrecuperáveis! É tanto erro, porque o local onde ferimos a sintaxe do JavaScript acaba impactando em todas as linhas de código que a sucedem. Nesse log verborrágico, a primeira informação indica qual a linha que ocorreu o problema e ainda da informações sobre o erro.

Um relatório ainda melhor

Nosso `jshint` funciona, mas a apresentação da mensagem de erro é um tanto estranha. Por que ele repete várias vezes o caminho do arquivo? Oras, bastava ele exibir uma única vez e depois detalhar os problemas. O `reporter` padrão do `jshint` deixa a desejar, mas nem tudo está perdido. Podemos baixar módulos de reportagem e substuir o `reporter` padrão. Um muito famoso é o `jshint-stylish`. Vamos baixá-lo através do npm:

```
npm install jshint-stylish@2.0.1 --save-dev
```

Como qualquer módulo, precisa ser importado em nosso `gulpfile.js`:

gulpfile.js

```
var gulp = require('gulp')
  ,imagemin = require('gulp-imagemin')
```

```
    ,clean = require('gulp-clean')
    ,concat = require('gulp-concat')
    ,htmlReplace = require('gulp-html-replace')
    ,uglify = require('gulp-uglify')
    ,usemin = require('gulp-usemin')
    ,cssmin = require('gulp-cssmin')
    ,browserSync = require('browser-sync')
    ,jshint = require('gulp-jshint')
    ,jshintStylish = require('jshint-stylish');
```

Agora, é só substituir o `reporter` padrão do `jshint` pelo `jshintStylish`:

```
// código anterior omitido
gulp.watch('src/js/**/*.js').on('change', function(event) {
    console.log("Linting " + event.path);
    gulp.src(event.path)
        .pipe(jshint())
        .pipe(jshint.reporter(jshintStylish));
});
// código posterior omitido
```

Agora, vamos reiniciar nosso `BrowserSync` e verificar o resultado para ver se muda muito:

```
/Users/flavioalmeida/Desktop/projeto/src/js/home.js
  line 8   col 42         Unclosed string.
  line 9   col 25         Unclosed string.
  line 10  col 62         Unclosed string.
  line 11  col 23         Unclosed string.
  line 12  col 2          Unclosed string.
  line 13  col 32         Unclosed string.
  line 14  col 1          Unclosed string.
  line 15  col 42         Unclosed string.
  line 16  col 1          Unclosed string.
  line 17  col 39         Unclosed string.
  line 18  col 53         Unclosed string.
  line 19  col 4          Unclosed string.
  line 8   col 16         Unclosed string.
  line 8   col 15         Unmatched '['.
  line 19  col undefined  Unrecoverable syntax error. (100% scanned).

  ✖  3 errors
  ⚠  12 warnings
```

Que mudança! Um relatório mais limpo! Vamos corrigir nosso `home.js` e ver qual mensagem ele exibe para arquivos corretos.

Quando o arquivo não possui erros de sintaxe ou algo que precise de atenção (*warning*) nada é exibido no console, a não ser o caminho do arquivo que estamos imprimindo sempre.

Nem o CSS escapa!

Excelente, será que podemos fazer a mesma coisa para arquivos `css` ? Claro que sim. Usaremos o módulo [gulp-csslint (https://github.com/lazd/gulp-csslint)](https://github.com/lazd/gulp-csslint). Vamos baixá-lo pelo npm e importá-lo em nosso `gulpfile.js` :

```
npm install gulp-csslint@0.2.0 --save-dev
```

gulpfile.js:

```
var gulp = require('gulp')
  ,imagemin = require('gulp-imagemin')
  ,clean = require('gulp-clean')
  ,concat = require('gulp-concat')
  ,htmlReplace = require('gulp-html-replace')
  ,uglify = require('gulp-uglify')
  ,usemin = require('gulp-usemin')
  ,cssmin = require('gulp-cssmin')
  ,browserSync = require('browser-sync')
  ,jshint = require('gulp-jshint')
  ,jshintStylish = require('jshint-stylish')
  ,csslint = require('gulp-csslint');
```

Também adicionaremos um `watcher` na tarefa `server` . A estrutura é praticamente idêntica ao que já fizemos com o `jshint` :

```
gulp.task('server', function() {
    browserSync.init({
        server: {
            baseDir: 'src'
        }
    });

    gulp.watch('src/**/*').on('change', browserSync.reload);

    gulp.watch('src/js/**/*.js').on('change', function(event) {
        console.log("Linting " + event.path);
        gulp.src(event.path)
            .pipe(jshint())
            .pipe(jshint.reporter(jshintStylish));
    });

    // novidade para lint de css
    gulp.watch('src/css/**/*.css').on('change', function(event) {
        console.log("Linting " + event.path);
        gulp.src(event.path)
            .pipe(csslint())
            .pipe(csslint.reporter());
    });

});
```

Veja que o `csslint` também utiliza um reporter, aliás muito decente. Vamos testar e ver o resultado.

Vamos reiniciar nossa task `server` e rodá-la novamente. Agora, vamos experimentar alterar `sobre.css`, mas não vamos forçar nenhum erro, vamos ver no que vai dar:

```
csslint: There are 8 problems in /Users/flavioalmeida/Desktop/projeto/src/css/sobre.css.

sobre.css
1: warning at line 30, col 1
Heading (h1) has already been defined.
h1 {

sobre.css
2: warning at line 33, col 1
Heading (h2) has already been defined.
h2 {

sobre.css
3: warning at line 36, col 1
Heading (h3) has already been defined.
h3 {

sobre.css
4: warning at line 59, col 1
Don't use IDs in selectors.
#centro-distribuicao {

sobre.css
5: warning at line 64, col 1
Don't use IDs in selectors.
#rodape {

sobre.css
6: warning at line 69, col 1
Don't use IDs in selectors.
#rodape img {

sobre.css
7: warning at line 75, col 1
Don't use IDs in selectors.
#familia-pelho {

sobre.css
8: warning
You have 2 h1s, 2 h2s, 2 h3s defined in this stylesheet.
```

Opa! Como assim? Tantos erros? Olhando bem, não há erro algum, apenas avisos (*warning*) indicando que nosso arquivo poderia estar melhor. Por exemplo, nosso lint tem inteligência para saber quando repetimos a declaração de seletores. Temos o `h1` duplicado. Para resolvermos isso, podemos agrupar todas as propriedades de `h1` dentro de um mesmo `h1`. O mesmo problema acontece com `h3`.

Outro ponto interessante é que ele diz para não usarmos seletores de ID. Usar este tipo de seletor não está errado (tanto é verdade que não recebemos um erro), mas é uma boa prática evitá-los para não cairmos em problema de especificidade, aquele quando queremos sobrescrever uma propriedade CSS por outra. Seletor de ID tem alta especificidade, bem, é claro que esse é um assunto interessante, mas nosso foco aqui é o Gulp, certo? Vamos dar um

tapa em nosso arquivo `sobre.css` agrupando propriedades e trocando os seletores de ID por seletores de classe e, claro, precisaremos alterar em `projeto/src/sobre.html` , para que use esses seletores e não mais os de ID.

Depois das alterações, quando salvarmos nosso arquivo, nenhuma mensagem é exibida! Muito intessante essa ferramenta. Agora, vamos forçar um erro clássico, vamos esquecer um ponto e vírgula no meio de uma propriedade:

sobre.css

```
// removido o ponto e vírgula de uma proprieade que não é a última
figure {
  background: #F2EDED;
  border: 1px solid #ccc
  text-align: center;
}
```

Assim que salvarmos nosso arquivo, recebemos a mensagem:

```
1: warning at line 11, col 3
Expected end of value but found 'text-align'.
  border: 1px solid #ccc

sobre.css
2: error at line 12, col 13
Expected RBRACE at line 12, col 13.
  text-align: center;
```

Os dois erros foram consequência da ausência do ponto e vírgula. Vamos consertar e curtir mais essa ferramenta.

Agora que você curtiu todo esse ferramental é hora dos exercícios.