02

Atacando repetições invisíveis

Começando deste ponto? Você pode fazer o DOWNLOAD (https://s3.amazonaws.com/caelum-online-public/gulp/stages/04-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.

Muito bem, vamos dar uma pausa na automação de tarefas e realizar uma pequena alteração em nosso projeto. Vamos exibir a página `projeto/src/index.html` em nosso navegador. Bem, nós precisamos trocar a cor de fundo para cinza, aliás, alterar e adicionar propriedades CSS é tarefa extremamente rotineira de um programador front-end.

Vamos abrir `projeto/src/css/estilos.css`. Logo de cara, temos o seguinte seletor:

```
// projeto/src/css/estilos.css
body {
  color: #333333;
  font-family: "Lucida Sans Unicode", "Lucida Grande", Sans-serif;
}
```

Vamos adicionar um `background`:

```
// projeto/src/css/estilos.css

body {
  color: #333333;
  font-family: "Lucida Sans Unicode", "Lucida Grande", Sans-serif;
  background: grey;
}
```

Repetimos e repetimos sem percerbermos

Agora, vamos voltar para o navegador e visualizar o resultado. Ficou do jeito planejado. Agora, de surpresa, uma reflexão: quantos passos demos desde o salvamento do arquivo até sua visualização? Bem, trocamos para o navegador, recarregamos a página. Imagine o número de vezes que você precisou se preocupar em atualizar a página toda vez que alguma alteração é feita em seus arquivos HTML, CSS ou Javascript? Zilhões de vezes? E se eu te dissesse que não precisa ser assim.

Podemos automatizar esse processo realizando um carregamento ao vivo (livereloading). Toda vez que alterarmos qualquer arquivo do nosso projeto nosso navegador recarregará. Isso é fantástico, porque podemos trabalhar com tela dividida e quem usa dois monitores nem precisa trocar para a outra área de trabalho que o resultado já será exibido.

O módulo BrowserSync

Para que possamos fazer isso precisamos um módulo famoso chamado Browser-sync (http://www.browsersync.io/):

```
npm install browser-sync@2.9.8 --save-dev
```

Agora, no `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');
```

Agora, precisamos criar uma tarefa que rodará o browser-sync. Vamos chamar de `server`. Usamos esse nome porque quem servirá nossos arquivos será o `browser-sync`:

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

Essa tarefa é um tanto peculiar. A variável `browserSync` nos dá acesso a uma instância dessa poderosa ferramenta. Ela possui o método `browserSync.init` que recebe como parâmetro um objeto que representa sua configuração. Cada chave desse objeto é uma configuração em particular, é por isso que usamos a chave `server`, que configura o servidor. Para essa chave temos um objeto com a propriedade `baseDir`, o diretório que será monitorado pelo `browserSync`, em nosso caso, `src`.

Será que isso é suficiente? Vamos rodar a tarefa no terminal e verificar o resultado:

```
npm run gulp server
```

Opa! Que susto! Nosso navegador padrão do sistema operacional é aberto automaticamente e já está acessando nossa página `index.html`! Fantátisco! Como o BrowserSync é um serviço, ele roda em segundo plano travando nosso terminal. Se precisarmos rodar outras tarefas, será necessário abrir um novo terminal.

Livereloading

Então, vamos testar e colocar a cor de fundo vermelha e ver se o a página é carregada automaticamente:

```
// projeto/src/css/estilos.css
body {
  color: #333333;
  font-family: "Lucida Sans Unicode", "Lucida Grande", Sans-serif;
  background: red;
}
```

Gulp watch

Nada aconteceu! Só mostra a cor se recarregarmos manualmente nossa página. O problema é que em nenhum momento estamos solicitando que o BrowserSync recarregue, isto é, precisamos em nosso script chamar a função `BrowserSync.reload()`.

Para isso, precisamos usar uma função nativa do Gulp, a `gulp.watch`. Nela podemos dizer quais arquivos estamos monitorando e uma ação que será executada quando eles forem alterados. A ação já sabemos, é o `BrowserSync.reload`.

Dentro da task do BrowserSync vamos configurar nosso watcher:

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

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

Nosso watcher está monitorando todos os arquivos dentro de qualquer subspasta em `src`. Em seguida chamamos a função `on`. Ela recebe como primeiro parâmetro o evento que desejamos observar, em nosso caso, estamos interessados apenas nos arquivos que mudaram, por isso passamos `change`. Por fim, o útlimo parâmetro é a função que desejamos executar, nesse caso, passamos diretamente BrowserSync.reload. Como passamos a função e não a invocamos, será o watcher que a chamará toda vez que um arquivo for alterado.

Para testarmos, precisamos fechar o terminal e executar nossa tarefa novamente para que as configurações entre em vigor.

Com nosso BrowserSync no ar e devidamente atualizado, vamos colocar a cor de fundo como branco, `white`:

```
// projeto/src/css/estilos.css
body {
  color: #333333;
  font-family: "Lucida Sans Unicode", "Lucida Grande", Sans-serif;
  background: white;
}
```

Agora sim! Podemos alterar qualquer arquivo do nosso projeto que nosso navegador recarregará sozinho. Aliás, estamos usando o BrowserSync na pasta `projeto/src` e não `projeto/dist` porque queremos usar este recurso enquanto estamos na parte de desenvolvimento e não no momento de deploy.

Livereloading 1 para N

E se eu te dissesse que o BrowserSync faz mais pela gente? Tem um smartphone ou um tablet perto de você? Com o BrowserSync rodando, digite o IP da sua máquina seguido da porta 3000 nesses dispositivos (certifique-se de estar na mesma rede da sua máquina). É exibida a página do projeto, legal. Agora, experimente fazer o scroll ou clicar em algo em sua máquina, mas não tire o olho dos dispositivos. O que acontece? Eles também sofrem as mesmas ações! Se clicarmos em um botão, o botão será clicado em todos os aparelhos! Experimente agora alterar um arquivo HTML ou

CSS. O que acontece? Todos os dispositivos são alterados! Fantástico! Com o BrowserSync podemos testar nossas páginas nos mais diversos dispositivos sem a tarefa tediosa de atualizar cada um deles.

Próximo passo, exercícios!