Capítulo 9

Grunt: automação de build de front-end

Mesmo com um projeto pronto para produção ainda é necessário realizar uma série de tarefas. Muitas vezes o desenvolvedor realiza essas tarefas manualmente, o que pode demandar muito tempo, além do projeto estar vulnerável a possíveis erros que o desenvolvedor possa cometer.

Para solucionar problemas como esses, foram criadas no mercado ferramentas de construção (build) de projetos como Ant, Gradle e Maven, mas há uma que nasceu voltada especialmente para programadores front-end: o **Grunt.**

O Grunt é diferente. Foi feito em JavaScript e com grande foco em automatizar tarefas de front-end. Se você, por exemplo, for seguir as boas práticas de performance para sites, já deve se preocupar em minificar CSS e JavaScript ou ainda juntar arquivos para diminuir o número de requests e até fazer CSS sprites. Ou talvez você esteja usando algum pré-processador de CSS como o LESS, SASS ou Stylus.

9.1 Um pouco sobre Node.js

O Grunt é escrito em JavaScript mas não executa no browser. Ele executa no terminal usando o **Node.js**.

O Node.js é uma ferramenta para execução de código JavaScript fora do navegador. Ele roda em servidores e no nosso terminal. Sua grande vantagem é permitir o uso da linguagem JavaScript fora do browser, facilitando nossa vida de desenvolvedores JavaScript.

O Node.js é baseado na engine V8 do Google Chrome. É o mesmo executor de JS usado dentro do browser do Google, mas disponível em outros ambientes. Para executá-lo, basta ir no terminal e rodar o comando **node**.

Dentro dele, podemos executar qualquer código JavaScript que não dependa de um browser. Por exemplo:

```
> 17 * 43
731
> var curso = "Caelum";
undefined
> console.log(curso);
Caelum
undefined
```

Mas repare que todo código que envolva o browser e o DOM não funciona. Não temos acesso a objetos como `document`, `window` ou `navigator`:

```
> alert("oi")
ReferenceError: alert is not defined
```

Executando nosso módulo de moeda

Importar um código externo JS no node é bem fácil:

```
require('./loja/js/modules/moeda.js');
```

O problema é que o node mantém os escopos dos arquivos separados, pra evitar poluição de variáveis globais. Isso quer dizer que, mesmo importando o moeda.js, a variável `formatadorMoeda` não estará disponível.

Resolvemos isso com duas mudanças: primeiro, nosso código que chama o require deve receber o módulo e salvar em uma variável:

```
var formatadorMoeda = require('./loja/js/modules/moeda.js');
```

E, depois, o próprio arquivo **moeda.js** precisa estar preparado pra devolver o módulo. No node.js, a maneira de fazer isso é atribuindo o módulo a variável `module.exports`:

```
// no final do arquivo moeda.js
module.exports = formatadorMoeda;
```

Agora podemos voltar ao Node.js, importar nosso módulo e usá-lo normalmente:

```
> var formatadorMoeda = require('./loja/js/modules/moeda.js');
undefined
> formatadorMoeda.numberParaReal(9.9);
'R$ 9,90'
> formatadorMoeda.realParaNumber('R$ 100,50');
100.5
```

Mesmo código rodando no browser e no node

Para usar o módulo de moeda no Node, precisamos colocá-lo na variável `module.exports`. Mas essa variável não existe no browser e vai fazer nosso código dar erro. Inclusive, no browser, ela nem precisa existir já que o módulo já é global.

Uma forma de deixar nosso módulo compatível tanto com browser quanto com Node é colocar a exportação do módulo em um if:

```
if (typeof module !== "undefined") {
  module.exports = formatadorMoeda;
}
```

9.2 Instalando Grunt

Para usar o Grunt é necessário ter o **Node.js** na versão 0.8.0 ou superior, o **npm** (*node package manager*) e **grunt-cli** (*command line interface*) instalados.

Instalando Node.js

Você pode baixar os instaladores do Linux, Mac e Windows na própria página do `Node.js` em http://nodejs.org/download/

Atualizando Node JS via npm

Como o Grunt necessita do Node na versão 0.8 ou superior, podemos atualizá-lo facilmente via `npm`, se necessário (cheque sua versão atual do Node com o comando `node -v`). Para isso, basta executar os seguintes comandos:

```
sudo npm config set registry http://registry.npmjs.org
sudo npm cache clean -f
sudo npm install -g n
sudo n stable
```

Instalando o Grunt

Com o `Node.js` instalado, utilize o gerenciador de pacotes `npm` na pasta do seu projeto para instalar o Grunt:

```
npm install grunt
```

Dentro da pasta do projeto, será criada a pasta **node_modules** com o **Grunt** e todas as suas dependências.

Instalando o cliente do Grunt: grunt-cli

Também é necessário instalar o cliente de linha de comando do Grunt, **o grunt-cli**:

```
npm install -g grunt-cli
```

O grunt-cli e suas dependências serão adicionados na pasta **node_modules**. Repare a opção '-g'. Ela instala o pacote globalmente, permitindo seu acesso em qualquer diretório em linha de comando.

GRUNT GLOBAL

Não é recomendada a instalação global do grunt, porque projetos diferentes podem usar diferentes versões do grunt. Já a instalação do `grunt-cli` globalmente é recomendada, para que tenhamos acesso ao grunt na linha de comando:

```
grunt
```

Caso não seja possível instalar o `grunt-cli` globalmente, você pode executar o grunt com o comando:

```
./node_modules/grunt-cli/bin/grunt
```

9.3 Preparando o projeto para usar Grunt

Para funcionar, o Grunt precisa ter no diretório raíz do projeto os arquivos **package.json** e o **Gruntfile.**

PACKAGE.JSON

É o arquivo usado pelo `npm` para guardar metadata no formato *json*. Nele ficarão informações sobre o projeto, como nome, descrição e versão:

```
// package.json
{
  "name": "mirror-fashion",
  "version": 0.0.1
}
```

Para o Grunt, usaremos a propriedade `devDependencies`. Nela, informaremos os plugins do Grunt que usaremos, inclusive a versão do Grunt utilizada:

```
{
  "name": "mirror-fashion",
  "version": 0.0.1,

  "devDependencies":  {
    "grunt": "~0.4.1",
```

```
    "grunt-cli": "~0.1.9",
    "grunt-contrib-clean": "~0.5.0",
    "grunt-contrib-copy": "~0.4.1"
  }
}
```

> **Comando npm init**
>
> Existe uma forma fácil de criar o **package.json**, através do comando `npm init`. Ele criará automaticamente o package.json intuitivamente, usando apenas a linha de comando.

Para instalar os plugins listados na propriedade `devDependencies` do arquivo *package.json*, podemos usar o comando `npm install`. No mesmo local do arquivo *package.json*, será criada uma pasta chamada *node_modules*, local onde ficarão armazenados todos os plugin baixados.

Uma forma mais fácil de instalar plugins no Grunt, é usando o comando: `npm install <modulo> --save-dev`

A opção `--save-dev` adiciona automaticamente o plugin à propriedade `devDependencies`.

Gruntfile

O **Gruntfile** é um arquivo `.js` ou `.coffee`, onde ficarão as configurações do Grunt. É nele que definiremos suas tarefas.

A função wrapper

Todo o código do Grunt ficará envolvido em uma função wrapper, que será exportada para o Node:

```
module.exports = function(grunt) {
    /* Código aqui */
};
```

Configurações de projeto e tasks

A maior parte das configurações de tarefas são passadas como objeto ao método `grunt.initConfig()`:

```
module.exports = function(grunt) {
  grunt.initConfig({
     /* Guardando o objeto package.json em pkg */
     pkg: grunt.file.readJSON('package.json')
```

```
    /* Outras tasks  aqui*/
  });
};
```

O comando `grunt.file.readJSON()` lê o arquivo *package.json* e guarda o atributo `pkg` do objeto passado ao `grunt.initConfig()`. Dessa forma, podemos acessar qualquer dado presente no *package.json*.

Usaremos `grunt.initConfig()` para definir tarefas de plugins do `npm` facilmente, apenas passando um novo objeto, de acordo com a especificação do plugin. Precisaremos também importar o plugin em nosso Gruntfile, usando a função `grunt.loadNpmTasks('nome-do-plugin')`:

```
module.exports = function(grunt) {

  grunt.initConfig({

    /* Copia para pasta de distribuição */
    copy: {
       projeto: {
          expand: true,
          cwd: 'loja',
          src: '**',
          dest: 'dist'
        }
     }

  });

  /* Carregando o plugin  */
  grunt.loadNpmTasks('grunt-contrib-copy');
};
```

Com as tarefas definidas, podemos utilizá-las via linha de comando:

```
grunt copy:projeto
```

Repare que a task `copy` foi chamada seguinda de `:projeto`. Isso ocorre porque poderíamos ter a mesma task `copy` com diferentes alvos, por exemplo, poderíamos chamar `grunt copy:imagens` para executar a task copy que copiará apenas um diretório de imagens. Desta maneira, nosso `Gruntfile.js` ficaria assim:

```
module.exports = function(grunt) {

  grunt.initConfig({

    /* Copia para pasta de distribuição */
    copy: {
       projeto: {
          expand: true,
          cwd: 'loja',
          src: '**',
          dest: 'dist'
        },

        imagens : {
          expand: true,
          cwd: 'loja/img',
          src: '**',
          dest: 'dist/img'
        }
     }

  });

  /* Carregando o plugin  */
  grunt.loadNpmTasks('grunt-contrib-copy');
};
```

Quando uma task tem apenas um target, podemos omitir seu target.

A PROPRIEDADE EXPAND

A propriedade `expand`, quando `true`, realiza um processamento dinâmico origem-destino, habilitando algunas propriedades. As que usamos até agora foram:

- **cwd** os arquivos são relativos a esta pasta, mas sem incluí-la.
- **src** padrão que procura encontrar, relativo à **cwd**.
- **dest** prefixo de destino.
- **ext** altera a extensão com este valor para os arquivos que forem copiados para **dest**.

Definindo uma lista de tarefas

Para definirmos diversas tarefas, basta passarmos novas propriedades ao objeto do `grunt.initConfig()`. Por exemplo, se quisermos usar o plugin *clean*, para limpar diretórios, junto ao *copy*:

```
module.exports = function(grunt) {

  grunt.initConfig({

    /* Copia para pasta de distribuição */
    copy: {
       projeto: {
          expand: true,
          cwd: 'loja',
          src: '**',
          dest: 'dist'
        }
     },

     clean: ['dist']
  });

  /* Carregando o plugin  */
  grunt.loadNpmTasks('grunt-contrib-copy');
  grunt.loadNpmTasks('grunt-contrib-clean');
};
```

Se quisermos agora executar essas duas tarefas, teremos que rodar o comando grunt duas vezes: uma passando a task `clean` e outra passando a task `copy`, inclusive, teremos que lembrar de rodar primeiro a tarefa `clean` e depois `copy`.

O Grunt nos permite registrar uma nova task, que vai executar outras em uma determinada ordem:

```
grunt.registerTask('padrao', ['clean', 'copy']);
```

Ao executarmos o comando `grunt padrao`, ele rodará automaticamente as duas tasks, na ordem em que foram declaradas dentro do array no segundo parâmetro, passado à função `grunt.registerTask()`.

O primeiro parâmetro da função é o nome que daremos à nova task. É uma boa prática usar nomes semânticos para determinados grupos de tarefas, como "minifica", "concatena", "deploy", etc.

Podemos também usar o nome *'default'*, que será executado apenas chamando `grunt` sem nenhum outro parâmetro.

9.4 Exercícios: primeiros passos com Grunt

1) Na pasta **app** já temos um **package.json**, sendo assim, já podemos iniciar a instalação do grunt.

2) Abra o **terminal** e entre dentro da pasta **app** com o comando abaixo

```
cd Desktop/nomeDaSuaPasta/app
```

3) Vamos agora instalar o **grunt** em nosso projeto executando o comando abaixo no terminal. O processo demorará alguns segundos, não se preocupe:

```
npm install grunt@0.4.1 --save-dev
```

4) Instale agora o **grunt-cli**, para que possamos interagir com o grunt via terminal:

```
npm install grunt-cli@0.1.9 --save-dev
```

5) Edite o arquivo **app/Gruntfile.js** e defina a estrutura mínima do arquivo de configuração:

```
module.exports = function(grunt) {

   grunt.initConfig({

   });
}
```

6) Usaremos a task **copy:projeto** para que o Grunt copie todo o nosso projeto para o diretório de produção **dist**. Além disso, usaremos a task `clean` para limpar a pasta **dist** antes de realizar a cópia. Não queremos arquivos velhos nesta pasta:

```
module.exports = function(grunt) {

   grunt.initConfig({

      /* Copia os arquivos para o diretorio 'dist'*/
      copy: {
         projeto: {
            expand: true,
            cwd: 'loja',
            src: '**',
            dest: 'dist'
         }
```

```
        },

        clean: ['dist']
    });
}
```

7) As tasks `clean` e `copy` são tasks do **npm**. Logo, é necessário carregá-las em nosso Gruntfile:

```
module.exports = function(grunt) {

   grunt.initConfig({
      /* Copia os arquivos para o diretorio 'dist'*/
      copy: {
         projeto: {
           expand: true,
           cwd: 'loja',
           src: '**',
           dest: 'dist'
         }
       },

       clean: ['dist']
    });

    // registrando tasks
    grunt.registerTask('default', ['clean', 'copy']);

    // carregando das tasks
    grunt.loadNpmTasks('grunt-contrib-copy');
    grunt.loadNpmTasks('grunt-contrib-clean');
}
```

8) Ainda falta baixar os plugins. Volte para o terminal e utilize o **npm** mais uma vez para cada um deles:

```
npm install grunt-contrib-clean@0.5.0 --save-dev

npm install grunt-contrib-copy@0.4.1 --save-dev
```

9) Visualize o arquivo **app/package.json**. Ele deve estar assim:

```
{
  "name": "mirror-fashion",
  "version": "0.0.0",
  "description": "",
  "main": "index.js",
```

```
    "scripts": {
      "test": "echo \"Error: no test specified\" && exit 1"
    },
    "author": "",
    "license": "BSD",
    "devDependencies": {
      "grunt": "~0.4.1",
      "grunt-cli": "~0.1.9",
      "grunt-contrib-clean": "~0.5.0",
      "grunt-contrib-copy": "~0.4.1"
    }
}
```

10) Agora, no console do sistema, execute o grunt para que ele execute automaticamente a tarefa "default":

```
grunt
```

11) Verifique se a pasta **dist** foi criada com todos os arquivos do projeto. É esta a pasta que mais tarde será enviada para produção, mas ainda precisamos minificar nossos scripts.

9.5 Automatizando merge, minificação de scripts e CSS's

Minificação e merge de arquivos `.js` e `.css` é, com certeza, uma das tarefas mais executadas no Grunt. Os plugin's envolvidos nestas tarefas são o **grunt-contrib-concat**, o **grunt-contrib-uglify** e o **grunt-contrib-cssmin**. Como todo plugin, precisamos instalar cada um deles através do `npm` e carregar suas tasks no `Gruntfile.js` com a função `grunt.loadNpmTasks`:

No terminal:

```
npm install grunt-contrib-concat --save-dev
npm install grunt-contrib-uglify --save-dev
npm install grunt-contrib-cssmin --save-dev
```

Em `Gruntfile.js`:

```
grunt.loadNpmTasks('grunt-contrib-concat');
grunt.loadNpmTasks('grunt-contrib-uglify');
grunt.loadNpmTasks('grunt-contrib-cssmin');
```

O problem é que seremos responsáveis pela configuração das tasks de cada um deles. Para resolver isso, utilizaremos o **grunt-usemin**.

GRUNT-USEMIN

O **grunt-usemin** é um plugin que **incrivelmente facilita** a configuração das tasks envolvidas no processo de merge e minificação. Só precisamos configurar sua task para que ele gere automaticamente os parâmetros de configuração para os três módulos que vimos anteriormente.

Para instalá-lo, basta executar no terminal o comando:

```
npm install grunt-usemin --save-dev
```

O `grunt-usemin` funciona da seguinte maneira. Em nossas páginas, envolvemos os blocos de css e de script que desejamos realizar o merge e minficação utilizando um comentário especial. Por exemplo, em nossa página `pesquisa.html`:

```
<!-- build:css css/index.min.css -->
<link rel="stylesheet" href="css/reset.css">
<link rel="stylesheet" href="css/estilos.css">
<link rel="stylesheet" href="css/index.css" media="screen">
<!-- endbuild -->

<!-- build:js js/lib/pesquisa-libs.min.js -->
<script src="js/lib/jquery.js"></script>
<!-- endbuild -->

<!-- build:js js/pesquisa/pesquisa.min.js -->
<script src="js/pesquisa/filtro.js"></script>
<!-- endbuild -->
```

Repare que usamos `build:css` e `build:js` repectivamente para merge e minificação de css e JavaScript. Cada um deles recebe o caminho e o nome do arquivo resultante relativo ao arquivo HTML. Isso é importante, porque no final, além da criação dos arquivos, o HTML também será modificado para:

```
<link rel="stylesheet" href="css/index.min.css">

<script src="js/lib/pesquisa-libs.min.js"></script>
<script src="js/pesquisa/pesquisa.min.js"></script>
```

Para que esta mágica aconteça, é necessário registrar o `grunt-usemin` no Gruntfile.js, inclusive configurar duas tasks distintas. A primeira, **useminPrepare** é a que gerará um arquivo de configuração dinâmico para `grunt-contrib-concat`, `grunt-contrib-uglify`, `grunt-contrib-cssmin` e que resultará nos arquivos 'mergeados' e minficados. A segunda task, **usemin**, é a que alterará nosso

HTML para apontar para os novos arquivos. Não se preocupe, a task é tão simples quanto as que vimos anteriormente:

No Gruntfile.js:

```
module.exports = function(grunt) {

   grunt.initConfig({
      /* Copia os arquivos para o diretorio 'dist'*/
      copy: {
         projeto: {
           expand: true,
           cwd: 'loja',
           src: '**',
           dest: 'dist'
         }
     },

     clean: ['dist'],

     useminPrepare: {
       html: ['dist/**/*.html']
     },

     usemin: {
       html: ['dist/**/*.html']
     }
  });


  //registrando task para minificação

  grunt.registerTask('minifica', ['useminPrepare', 'usemin',
                                  'concat', 'uglify', 'cssmin']);

  // registrando tasks
  grunt.registerTask('default', ['clean', 'copy', 'minifica']);

  // carregando das tasks
  grunt.loadNpmTasks('grunt-contrib-copy');
  grunt.loadNpmTasks('grunt-contrib-clean');
  grunt.loadNpmTasks('grunt-contrib-concat');
  grunt.loadNpmTasks('grunt-contrib-uglify');
```

```
  grunt.loadNpmTasks('grunt-contrib-cssmin');
  grunt.loadNpmTasks('grunt-usemin');
}
```

Perceba que registramos na task 'minifica' a chamada do `useminPrepare` seguida de `usemin`. Mais importante ainda é a chamada das tasks 'concat', 'uglify' e 'cssmin', já que elas receberão as informações passadas pelo `grunt-usemin`.

9.6 Exercícios: merge e minificação de scripts/css's automáticos

1) Vamos criar agora as tasks `useminPrepare` e `usemin` para todas as páginas do nosso projeto (repare que ele executará na cópia do nosso projeto em 'dist', preservando o projeto original), inclusive vamos já carregar os módulos de que precisamos:

```
module.exports = function(grunt) {
  grunt.initConfig({

    /* Copia os arquivos para o diretorio 'dist'*/
    copy: {
       projeto: {
          expand: true,
          cwd: 'loja',
          src: '**',
          dest: 'dist'
       }
    },

    clean: ['dist'],

    useminPrepare: {
      html: ['dist/**/*.html']
    },

    usemin: {
      html: ['dist/**/*.html']
    }

  });

  grunt.registerTask('default', ['clean', 'copy']);
```

```
  grunt.loadNpmTasks('grunt-contrib-copy');
  grunt.loadNpmTasks('grunt-contrib-clean');
  grunt.loadNpmTasks('grunt-contrib-concat');
  grunt.loadNpmTasks('grunt-contrib-uglify');
  grunt.loadNpmTasks('grunt-contrib-cssmin');
  grunt.loadNpmTasks('grunt-usemin');
}
```

2) Como estamos usando tasks do *npm*, é necessário baixá-las. **No terminal**, use os comandos:

```
npm install grunt-contrib-concat@0.3.0 --save-dev
npm install grunt-contrib-uglify@0.2.2 --save-dev
npm install grunt-contrib-cssmin@0.6.2 --save-dev
npm install grunt-usemin@2.0.0 --save-dev
```

3) É um incômodo ter que rodar cada task do grunt individualmente. Podemos registrar uma série de tasks, para fazer o que queremos automaticamente:

```
module.exports = function(grunt) {
  grunt.initConfig({

    /* Copia os arquivos para o diretorio 'dist'*/
    copy: {
       projeto: {
          expand: true,
          cwd: 'loja',
          src: '**',
          dest: 'dist'
       }
    },

    clean: ['dist'],

    useminPrepare: {
      html: ['dist/**/*.html']
    },

    usemin: {
      html: ['dist/**/*.html']
    }

  });
```

```
  //registrando task para minificação
  grunt.registerTask('minifica', ['useminPrepare', 'usemin',
                                  'concat', 'uglify', 'cssmin']);

  grunt.registerTask('default', ['clean', 'copy']);

  grunt.loadNpmTasks('grunt-contrib-copy');
  grunt.loadNpmTasks('grunt-contrib-clean');
  grunt.loadNpmTasks('grunt-contrib-concat');
  grunt.loadNpmTasks('grunt-contrib-uglify');
  grunt.loadNpmTasks('grunt-contrib-cssmin');
  grunt.loadNpmTasks('grunt-usemin');
}
```

4) Queremos que a task 'minifica' rode através da task 'default'. Para isso, adicione a chamada de 'minifica' como última na task 'default':

```
grunt.registerTask('default', ['clean', 'copy', 'minifica']);
```

5) Em **pesquisa.html**, temos que indicar quais css's e scripts serão concatenados e minificados. Para isso, vamos começar envolvendo os ccs's com comentários do *useminPrepare* usando **build:css**:

```
<!-- build:css css/index.min.css -->
<link rel="stylesheet" href="css/reset.css">
<link rel="stylesheet" href="css/estilos.css">
<link rel="stylesheet" href="css/index.css" media="screen">
<!-- endbuild -->
```

6) Faça a mesma coisa agora para os script's da página, só que deste vez, usando **build:js**:

```
<!-- build:js js/lib/pesquisa-libs.min.js -->
<script src="js/lib/jquery.js"></script>
<!-- endbuild -->

<!-- build:js js/pesquisa/pesquisa.min.js -->
<script src="js/pesquisa/filter.js"></script>
<!-- endbuild -->
```

7) Rode o agora o comando `grunt` no terminal para executar a task 'default' e verifique se na pasta 'dist' os arquivos mergeados e minificados, inclusive verifique se o arquivo `pesquisa.html` foi alterado para apontar para os novos arquivos:

```
grunt
```

9.7 Para saber mais: uglify e cssmin de outros arquivos

Com o **usemin**, note que a minificação do JS e do CSS só ocorre nos arquivos concatenados. Mas podemos configurar o uglify e o cssmin pra executar em todos os arquivos do projeto bem facilmente.

Basta adicionar uma configuração que indica que todos os .js devem ser uglifyzados e todos .css devem ser css minificados:

```
uglify: {
  main: {
    expand: true,
    cwd: 'dist/',
    src: ['**/*.js'],
    dest: 'dist/'
  }
},
cssmin: {
  main: {
    expand: true,
    cwd: 'dist/',
    src: ['**/*.css'],
    dest: 'dist/'
  }
}
```

9.8 Minificação de imagens

Toda vez que os designers criavam uma nova imagem para o site, a equipe front-end da Mirror Fashion precisava otimizá-la antes de colocá-la em produção. Como a produção de conteúdo do site é incessante, a equipe constantemente tinha que intervir no processo.

O Grunt também permite automatizar uma tarefa como essa através do **grunt-contrib-imagemin**. Sua instalação é como qualquer módulo:

```
npm install grunt-contrib-imagemin@0.3.0 --save-dev
```

Como qualquer módulo, precisa ser carregado no `Gruntfile.js`:

```
grunt.loadNpmTasks('grunt-contrib-imagemin');
```

Por fim, resta apenas configurar a task também no `Gruntfile.js`:

```
imagemin: {
   projeto: {
     expand: true,
     cwd: 'loja/img',
     src: '**/*.{png,jpg,gif}',
     dest: 'dist/img'
   }
}
```

A task acima criará as imagens otimizadas na pasta de distribuição do projeto, sendo assim, sua task deve ser registrada após a cópia dos arquivos:

```
grunt.registerTask('minifica', ['useminPrepare', 'usemin', 'concat',
                                'uglify', 'cssmin', 'imagemin']);
```

9.9 Exercícios - Minificação de imagens automática

A equipe já tem toda infraestrutura do Grunt para automatizar uma série de tarefas, inclusive a minificação de imagens. Vamos implementá-la.

1) Instale o modulo **grunt-contrib-imagemin** através do npm:

```
npm install grunt-contrib-imagemin@0.3.0 --save-dev
```

2) Edite **Gruntfile.js** e carregue o módulo:

```
grunt.loadNpmTasks('grunt-contrib-imagemin');
```

3) Adicione a task do imagemin:

```
imagemin: {
   projeto: {
     expand: true,
     cwd: 'dist/img',
     src: '**/*.{png,jpg,gif}',
     dest: 'dist/img'
   }
}
```

4) Por fim, adicione a task `imagemin` como última target a ser executada pela task 'minifica':

```
grunt.registerTask('minifica', ['useminPrepare', 'usemin', 'concat',
                                'uglify', 'cssmin', 'imagemin']);
```

5) Agora, teste o resultado. Rode o Grunt e verifique no terminal a saída. Nela, você terá um resumo do tamanho final de cada arquivo. Lembre-se que as imagens originais continuam intactas, apenas as que estão na pasta de distribuição foram alteradas.

```
grunt
```

9.10 Watch e LiveReload com grunt

O grunt abre milhões de possibilidades para integrarmos plugins e outros recursos do node.js. Um deles é rodar um servidor HTTP local pra facilitar o desenvolvimento, assim não precisamos mais abrir os arquivos como file://

grunt-contrib-connect

Conseguimos isso com o **grunt-contrib-connect** que roda um servidor Node.js na nossa aplicação. Assim podemos acessar apenas http://localhost:8000/loja/pesquisa.html por exemplo.

grunt-contrib-watch

Para monitorar alterações de arquivos utilizaremos **grunt-contrib-watch**:

```
npm install grunt-contrib-watch@0.5.3 --save-dev
```

Mas podemos ir além. O plugin **grunt-contrib-watch** se integra com o connect e provê um mecanismo bastante útil no desenvolvimento chamado de **LiveReload**. A ideia é recarregar o browser automaticamente quando houver mudanças nos arquivos. Chega de reload na mão!

Se você usar 2 monitores então, é muito prático deixar o browser aberto em um deles e o editor em outro. Conforme edita, o browser é atualizado sozinho. Melhor ainda ao desenvolver com vários browsers e dispositivos móveis: todos os navegadores são atualizados automaticamente de uma só vez!

9.11 Exercícios - livereload

1) Primeiro, instale os plugins necessários, como fizemos antes no terminal:

```
npm install grunt-contrib-watch@0.5.3 --save-dev
npm install grunt-contrib-connect@0.5.0 --save-dev
```

2) Abra seu **Gruntfile.js** e adicione a configuração dos plugins no final:

```
grunt.loadNpmTasks('grunt-contrib-watch');
grunt.loadNpmTasks('grunt-contrib-connect');
```

Você pode até criar uma task nova **run** pra executar o projeto mais facilmente:

```
grunt.registerTask('run', ['connect', 'watch']);
```

3) Por fim, faça a configuração dos plugins no grunt init:

```
watch: {
  options: {
    livereload: true
  },
  todos: {
    files: ['loja/**/*']
  }
},

connect: {
  server: {
    options: {
      livereload: true
    }
  }
}
```

4) Execute **grunt run** no terminal e abra a página http://localhost:8000/loja/pesquisa.html

Repare que o comando do grunt não termina, ele continua executando. Isso porque ele está ativamente observando seu diretório pra disparar o reload automático.

Teste isso. Altere um CSS ou JavaScript e note a página sendo recarregada imediatamente com suas alterações.

9.12 Para saber mais: LESS, compilação em tempo real

A equipe da Mirror Fashion decidiu adotar LESS em seus projetos. LESS é uma linguagem baseada em CSS (mesma ideia, sintaxe familiar) com recursos que fazem falta no CSS em algumas situações. É também chamado de pré-processador pois, na verdade, é usado para gerar um arquivo CSS no final.

O problema é que toda vez que um arquivo `.less` for alterado, teremos que executar manualmente um pré-processador de nossa escolha, o que torna inviável a adotação do LESS por questões práticas. Mas nem tudo esta perdido, o grunt pode automatizar esta tarefa para nós.

O primeiro passo é escolher um pré-processador.

GRUNT-CONTRIB-LESS

Já existe um módulo para o grunt responsável para compilação de arquivos `.less`. Sua instalação, como qualquer outro módulo, é através do gerenciador de dependências do Node.js:

```
npm install grunt-contrib-less@0.7.0 --save-dev
```

Ele não é suficiente, precisamos ainda de alguém que fique "vigiando" nossos arquivos `.less` e, toda vez que algum deles for alterado, este "alguém" executará a task que compilará o arquivo. Já vimos como o `grunt-contrib-watch` pode nos ajudar.

CONFIGURANDO AS TASKS EM GRUNTFILE.JS

Depois de instaladas nossas dependências, precisamos registrar as tasks no `Gruntfile.js`: a task `less% e alterar a task` watch

```
less: {
    compilar: {
        expand: true,
        cwd: 'loja/less/',
        src: '**/*.less',
        dest: 'loja/css',
        ext: '.css'
    }
},

watch: {
  options: {
    livereload: true
  },

  todos: {
    files: ['loja/**/*']
  },

  less: {
    files: 'loja/less/*.less',
    options: {
      event: ['all']
    },
    tasks : 'less'
  }
}
```

Não precisamos registrar nossa task, já que task 'watch' já esta sendo chamada pela task 'run'.

Para terminar nossas configurações, precisamos a task `grunt-contrib-less`:

```
grunt.loadNpmTasks('grunt-contrib-less');
```

Executando grunt watch

Para que o grunt monitore seus arquivos `.less` basta executar no terminal:

```
grunt run
```

Experimente criar em **app/loja/less** o arquivo **teste.less** com a seguinte estrutura:

```
@color : white;

div {
  color: @color;
}
```

Se o `grunt watch` estiver rodando em background, qualquer alteração em `teste.less` gerará automaticamente o arquivo **app/loja/css/teste.css**. Experimente alterar a cor:

```
@color : green;

div {
  color: @color;
}
```

Depois experimente alterar para outra cor e veja se o arquivo **teste.css** foi atualizado.

9.13 Exercício opcional: compilando LESS com grunt watch

Vamos fazer com o que o grunt compile nossos arquivos `.less` automaticamente através do `grunt watch`.

1) No terminal, instale a dependência abaixo:

```
npm install grunt-contrib-less@0.7.0 --save-dev
```

2) Edite **app/Gruntfile.js** e adicione a task 'less' e altere a task 'watch':

```
less: {
  compilar: {
    expand: true,
    cwd: 'loja/less/',
    src: '**/*.less',
    dest: 'loja/css',
    ext: '.css'
  }
},

watch: {
  options: {
    livereload: true
  },

  todos: {
    files: ['loja/**/*']
  },

  less: {
    files: 'loja/less/*.less',
    tasks : 'less'
  }
}
```

3) Não precisamos registrar nossa task, já que task 'watch' já esta sendo chamada pela task 'run'.

4) Para terminar nossas configurações, precisamos carregar a task 'grunt-contrib-less':

```
grunt.loadNpmTasks('grunt-contrib-less');
```

5) Crie a pasta **app/loja/less** e dentro dela crie o arquivo **teste.less** com a seguinte estrutura:

```
@color : white;

div{
  color: @color;
}
```

6) No terminal, na pasta **app**, execute o comando:

```
grunt run
```

7) Agora, com `grunt run` rodando em background, altere o CSS, por exemplo, usando a cor `green`:

```
@color : green;

div{
  color: @color;
}
```

8) Verique na pasta **app/loja/css** se o arquivo **teste.css** foi criado. Experimente realizar outras modificações em **teste.less** e verifique a mudança em **teste.css**.