Automação Front-end e Gulp

@autor: Mikael Assis

Automação Front-End

Sites úteis:

- Npm Js
- SS64 - Linha de Comando
- Documentação NPM - Comandos

Podemos usar pacotes fornecidos por empresas especializadas para facilitar nossa codificação diária, realizando tarefas então impossíveis sem o uso de tais ferramentas, como:

- Minificar arquivos Javascript, Css;
- Otimizar arquivos Javascript;
- Minificar, alterar parâmetros como cor, saturação, e crop de imagens via comando Javascript, entre outras.

NPM

NPM é um gerenciador de pacotes Node (Node Package Manager)

Para gerenciar os pacotes instalados via Node podemos iniciar o Node através do comando `npm init`, este gerará um arquivo **package.json**, que reunirá todas as informações dos pacotes instalados, bem como nome do projeto, dependências e scripts.

```
1 {
2
```

```
 "name": "gulp",
 "version": "1.0.0",
 "description": "",
 "main": "gulpfile.js",
 "dependencies": {
  "browser-sync": "^2.26.7",
  "gulp": "^4.0.2",
  "gulp-autoprefixer": "^7.0.1",
  "gulp-concat": "^2.6.1",
  "gulp-sass": "^4.1.0",
  "gulp-uglify": "^3.0.2"
 },

 "devDependencies": {
  "@babel/core": "^7.9.6",
  "@babel/preset-env": "^7.9.6",
  "gulp-babel": "^8.0.0"
 },

 "scripts": {
  "test": "echo \"Error: no test specified\" && exit 1"
 },

 "keywords": [],
 "author": "",
 "license": "ISC"

}
```

Estas informações podem ser inseridas no ato do comando `npm init`, ou podemos pular o processo inserindo o comando `npm init -y` ou `npm init -yes`.

- **Name** - *Define um nome para o projeto (Ao definir o comando `npm init -y`, o nome inserido será o do diretório atual do projeto, não podendo este diretório ter Strings separadas (nome pasta), sendo o aceito (nomePasta / nome_pasta)).*
- **Version** - *Pode ser definido uma versão para o projeto*
- **Description** - *Pode ser definido uma descrição para o projeto*
- **Dependencies** - *Armazenas as informações dos pacotes instalados, permitindo facilidade de migração do projeto para outro local. O comando `npm install`, instalará todas essas dependências nas versões descritas no mesmo*
    - **Versionamento de Pacotes (Versionamento Semântico)** - A versão de um pacote possui três notações importantes para o desenvolvimento e gerenciamento de pacotes no projeto.
        - **Major** - *Quando a detentora do pacote atualiza o plugin, inserindo mudanças que podem impedir ou dificultar no processamento correto do programa que as utiliza.*
        - **Minor** - *Quando a detentora do pacote adiciona novas funcionalidades ao pacote. Esta ação não é hábil de corromper o programa que a utiliza, uma vez que mantém a compatibilidade do mesmo.*
        - **Patch** - *Quando a detentora do pacote lança correções de falhas, mantendo a compatibilidade do pacote com a versão atual*
        - A notação corresponde à: **0.00.00** sendo **Major**.**Minor**.**Patch**
        - Os símbolos **^** e **~** correspondem à:

        - ^ - Especifica que dependências do package.json não poderão ser atualizadas caso uma **Major**, esteja disponível, atualizando somente **Minor** e **Patch**.
        - ~ - Especifica que somente **Patch** poderá ser atualizado.
    - Para informações sobre **Versionamento Semântico**, acesse Semver.org

- **devDependencies** - *São todos os programas necessários para ambiente de "dev", desenvolvimento, da aplicação. Pode ser tudo desde compressores de código, transpiladores, testes unitários, ferramentas de debug, etc. Estes não são necessários para a aplicação funcionar, mas sim para desenvolver e /ou testar.*
    - Para inserir **devDependencies** basta: `npm install pacote --save-dev`
- **Scripts** - A função Scripts possui muitos usos, porém vamos abordar o uso como um gerador de atalhos para comandos (por exemplo rodar códigos de pacotes instalados localmente)
    - O scripts pode executar qualquer comando de terminal válido. O que você faria na linha de comando, em shell, ou invocando outro script nodejs, pode ficar simplificado, com um apelido no package:

```
"scripts": {
  "atalho": "diretório/.bin/pacote/arquivo/comandos/output"
  "minclip": "node_modules/.bin/uglifyjs clipboard.js -c -o clipboard.min.js"
 },
```

```
npm run minclip
```

Executa o comando inserido em Scripts .

Pacotes Locais

Um pacote local é instalado dentro do diretório atual. Pacotes instalados localmente, podem não ter seus comandos reconhecidos pela linha de comando quando acionados. Para acessar tais comandos, sem instalar tais pacotes globalmente, precisamos:

1. Acesse o diretório do projeto via **CMD**/**Bash** através dos comando **cd** (change directory).
2. Localize o diretório **node_modules**
3. Acesse o diretório .bin e localize o pacote pelo nome do mesmo

```
autor@autor ~/diretório/pastaProjeto/node_modules/.bin
$ node_modules/.bin/uglifyjs clipboard.js -c -o clipboard.min.js
```

*Obs.: O comando acima se refere à **pacote** (**uglifyjs**), nome do arquivo a ser alterado **src** (**clipboard.js**), **-c - o** (comandos: **compressed**, **output**), e **output** do arquivo (**clipboard.min.js**)*

Este comando será executado da mesma maneira, como se o pacote estivesse instalado globalmente.

Instalando pacotes Locais

```
npm install {nome do pacote} ou
npm i {nome do pacote}
```

Instala a última versão do pacote solicitado

Atualizando pacotes locais

```
npm update {nome do pacote}
```

Desinstalando pacotes locais

```
npm uninstall {nome do pacote} ou
npm remove {nome do pacote}
```

Pacotes Globais

Um pacote global é instalado para todos os usuários da máquina

Instalando pacotes Globais

```
npm install {nome do pacote} -g ou
npm i {nome do pacote} -g
```

Atualizando pacotes globais

```
npm update {nome do pacote} -g
```

Desinstalando pacotes globais

```
npm uninstall {nome do pacote} -g ou
npm remove {nome do pacote} -g
```

Pacotes de Automação

Jimp

Documentação: Jimp

Jimp é um Programa Javascrpt de Manipulação de imagens (Javascript Image Manipulation Program), capaz de cropar imagens, mudar o tom das cores, entre outros

Instalação

```
npm install jimp
```

Funções mais usadas:

- **Cover** - Muda o tamanho da imagem “crop”.
  - `imagem.cover(400, 400);`
- **Quality** - Configura a qualidade de imagens
  - `imagem.quality(60);`
- **Greyscale** - Configura a cor da imagem para tons preto e branco.
  - `imagem.greyscale();`

Exemplo de Código:

```
const jimp = require('jimp');

jimp.read('imagens/imagem1.jpg').then(function (imagem) {

  imagem.cover(400, 400).greyscale().write('imagemCinza.jpg');

}).catch(err => {

  console.error(err);

});
```

`.read()` . É responsável por ler os arquivos que serão tratados;

`.then()` - Configura um tempo mínimo para que todas as imagens passadas sejam lidas. Recebe como parâmetro uma função que recebe uma imagem;

`write()` - Salva a imagem. Recebe como parâmetro o nome final da imagem

`.catch()` - Pega os erros que poderão ser lançados caso haja algum. Recebe um err = erro, como parâmetro, e o imprime via console.

Para podermos ler várias imagens ao mesmo tempo, podemos incluí-las num array importando o pacote proprietário do Node fs/File System

Importando File System

```
const fs = require('fs');
```

Atribuindo à uma variável

```
const imagens = fs.readdirSync('./imagens/');
```

```
mikae@Mikael MINGW64 ~/Desktop/Origamid-2/Curso de Gulp/Outros (master)
$ node jimp.js

mikae@Mikael MINGW64 ~/Desktop/Origamid-2/Curso de Gulp/Outros (master)
$ node jimp.js
[ 'imagem1.jpg', 'imagem2.jpg', 'imagem3.jpg', 'imagem4.jpg' ]
```

Através da chamada do arquivo Jimp.js, ao inserirmos a variável em um `console.log()`, recebemos o retorno acima, sendo este um vetor de imagens.

Função otimizada com Array

```
imagens.forEach(function (arquivo) {
  jimp.read('imagens/' + arquivo).then(function (imagem) {
    imagem.cover(400, 400)
    .greyscale()
    .write('imagens/otimizadas/' + arquivo);

  }).catch(err => {
    console.error(err);
  });
});
```

SVGO

Documentação: SVGO

O pacote SVGO é capaz de minimizar e otimizar arquivos svg, transformando os mesmo em Path, excluindo comentários desnecessários e diminuindo seus tamanhos.

Instalação

```
npm install -g svgo
```

Para usar este pacote não é necessário criar um arquivo Javascript, basta pela linha de comando:

`svgo caminho dos arquivos(src)  -o "output" diretório final(dest)`

```
mikae@Mikael MINGW64 ~/Desktop/Origamid-2/Curso de Gulp/outros (master)
$ svgo svg/*.svg -o svg/otimizados

basic_world.svg:
Done in 42 ms!
1.378 KiB - 58.4% = 0.573 KiB
```

Todos os arquivos nas pasta foram otimizados

Gulp

Sites úteis:

- Gulp Js

Introdução

Gulp é uma ferramenta de automação de tarefas em Javascript, capaz de automatizar tarefas como compilação e otimização de arquivos Sass, minificação de documentos Javascript entre outros.

Documentação Gulp - Get Started

Para que as funcionalidades do pacote Gulp possam ser utilizadas, precisamos seguir os seguintes passos;

Precisamos verificar se os seguintes programas estão instalados na máquina, através dos comandos via Bash/CMD:

```
node --version
npm --version
npx --version
```

Os comandos acima serão responsáveis por mostrar ao usuário a versão atual dos programas Node, e do gerenciador de pacotes NPM e NPX. Caso a mensagem retornada indique que tais comando não foram encontrados os pacotes em questão devem ser instalados.

Instalação Gulp Package

Para efetuar a instalação do Pacote Gulp, via linha de comando:

```
        npm install --global gulp-cli
```

O pacote acima será instalado globalmente, mas para que o pacote gulp funcione corretamente uma versão do mesmo deve ser instalada dentro do diretório do projeto que se trabalhará, através do comando:

```
        npm install gulp
```

Para verificar a versão do Gulp instalada, via CMD/Bash :

```
        gulp --version
```

O comando Gulp, na linha de comando, demonstra se o pacote foi instalado com sucesso, porém informará que configurações iniciais devem ser feitas para que o mesmo funcione corretamente:

```
autor@autor ~/diretório
$ gulp
[14:39:02] No gulp file found
```

Obs.: Se nenhum arquivo gulp foi criado anteriormente, a mensagem acima será mostrada, para resolver crie um arquivo `gulpfile.js` na pasta do projeto

Após criado, rode o comando novamente.

```
autor@autor ~/diretório
$ gulp
[16:43:56] Using gulpfile ~\Desktop\Origamid-2\Curso de Gulp\teste\gulpfile.js
[16:43:56] Task never defined: default
[16:43:56] To list available tasks, try running: gulp --tasks
```

Obs.: A mensagem acima informa que o arquivo criado não contém nenhuma tarefa que possa ser executada.

Importando Gulp-Package

Para criarmos uma nova tarefa, precisamos incluir a importação do pacote para dentro do arquivos `gulpfile.js`, abaixo podemos notar que essa importação pode ser feita de duas maneiras:

EC6+

```
import { src, dest, task, watch as _watch, parallel } from 'gulp';
```

Para pacotes importados na versão ECMASCRIPT6+ podemos usar como variável/pipe o nome do próprio pacote importado.

EC6-

```
const nomedaVariavel= require('pacoteInstaladoAnteriormente');
const gulp = require('gulp');
```

Criando uma tarefa

```
function nomeFuncao() {
    return gulp.src('diretorio/')
        .pipe(execução)
        .pipe(gulp.dest('diretorio/'));
};
```

Nota: Toda função criada em Gulp com a funcionalidade de alterar um arquivo final, deve ter seu código escrito em um return, para que o código retornado seja brevemente executado.

Busca um/vários arquivo(s)/source de determinado diretório

```
gulp.src('diretorio')
```

Quando se precisa especificar vários arquivos de determinada extensão, podemos:

```
gulp.src('css/scss/*.scss')
```

Ou

```
gulp.src('css/scss/**/*.scss')
```

Busca todos os arquivos da mesma extensão dentro de um diretório anterior/posterior

---

Definindo Pipes/Canos

Um pipe, como o próprio nome especifica “cano”, redireciona o código através de funções pré-estabelecidas pelo pacote, para que este seja alterado conforme desejado:

```
.pipe(sass({ outputStyle: "compressed" }))
```

Vários pipes podem ser conectados uns aos outros através de pontos finais "."

A variável que recebe a importação do pacote, no início do arquivo `Gulpfile.js` deve conter seu nome dentro da pipe, referindo qual pacote deve ser utilizado.

Para que o arquivo alterado seja salvo em determinado diretório, podemos incluir a função `dest('diretório')`, do Gulp para tal:

```
.pipe(gulp.dest('diretorio/'));
```

Gulp Tasks

Uma task/tarefa executa a ação da função associada, através da linha de comando, quando chamada. Para definirmos uma task, podemos:

EC6+

```
exports.nomeDaTask= nomeDaFuncao;
```

Em versões acima do ECMASCRIPT 6, devemos definir o nome da Task com o mesmo nome definido para a função, senão erros na linha de comando podem ser acionados.

EC6-

```
gulp.task('nomeDaTask', 'nomeDaFuncao');
gulp.task('browserSync', browser);
```

Já para versões abaixo do ECMASCRIPT 6, podemos definir um nome diferente (String) para a task, ao qual poderemos usar para chamar a Task na linha de comando

Tarefa Default

Uma tarefa Default é a tarefa Padrão do Gulp, quando acionada via Bash/CMD, pode executar várias tarefas, especificadas em sua chamada no arquivos *Gulpfile.js*.

Esta possui dois parâmetros *"series"* e *"parallel"* que podem ser usadas para determinar o fluxo que as tarefas serão aplicadas, sendo série, tarefas são executadas uma após a outra, e parallel, tarefas executadas ao mesmo tempo "paralelamente".

```
exports.default = gulp.parallel(watch, browser, compilaSass, gulpJs);
```

Gulp Sass

Documentação: Gulp-Sass

O pacote Gulp-Sass é responsável por transformar um ou vários arquivos scss, em css. Através da função podemos minificar os arquivos finais.

Instalação:

```
npm install gulp-sass
```

Exemplo de Código:

```
function compilaSass() {
    return gulp.src('css/scss/*.scss')
        .pipe(sass({ outputStyle: "compressed" }))
        .pipe(gulp.dest('css/'))
};
```

O atributo {outputStyle: 'compressed'} indica que o arquivo final será minificado.

Task:

```
exports.compilaSass, compilaSass;
```

Gulp Watch

A função watch do gulp é responsável por "escutar", determinados diretórios ou arquivos, e na menor possibilidade de mudança, são executadas determinadas tarefas em funções. Uma watch pode conter dentro de `gulp.watch('')` outras tasks, que podem ser executadas em série, ou paralelamente, quando houverem mudanças em tais arquivos .

Ex.: `gulp.watch('css/scss/*.scss', gulp.series('tarefa1', 'tarefa2'))`

```
function watch() {
    gulp.watch('css/scss/*.scss', compilaSass);
    gulp.watch('js/assets_js/*.js', gulpJs);
    gulp.watch('*.html').on('change', browserSync.reload);
};
```

Todos os arquivos em questão serão “ouvidos” pela função Watch que ao menor sinal de mudança; compilará os arquivos Scss para Css, concatenará arquivos Js em um único arquivo, e chamará a função BrowserSync quando arquivos html foram alterados, recarregando a página index em seguida com as informações atualizadas.

BrowserSync

Documentação: BrowserSync

O pacote *BrowserSync*, cria uma conexão via servidor disponibilizando os arquivos passados em função para que sejam renderizados automaticamente via Browser. Os links disponibilizados podem ser acessados por todos os dispositivos conectados na mesma rede de internet (celulares, computadores).

Instalação:

```
$ npm install browser-sync --save-dev
```

Importação

Ao importar o pacote precisamos passar o método `.create()` ao final

```
const browserSync = require('browser-sync').create();
```

Iniciação

Antes de utilizada, o pacote deve ser inicializado:

```
function browser() {
    browserSync.init({
        server: {
            baseDir: "./"
        }
    });
};
```

Nota: O atributo `baseDir`, é passado quando os arquivos que serão renderizados estão em um determinado diretório. Porém, através do atributo `Proxy`, podemos passar um endereço de servidor (Xampp, Mamp)

```
gulp.task('browser-sync', function() {
    browserSync.init({
        proxy: "endereçoServidor"
    });
});
```

*Este também possibilita através da função **Stream**, que arquivos css sejam renderizados sem que a página toda seja carregada, e o atributo **reload** permite que arquivos (html, php, etc..), também sejam renderizados.*

Para atualizar o `Html` automaticamente não podemos inserir mais de um elemento na função Watch, como neste exemplo,

```
function watch() {
    gulp.watch('css/scss/*.scss', compilaSass);
    gulp.watch('*.html', browserSync);
};
```

pois a função Watch não permite esse argumento para o *BrowserSync*, sendo necessário o uso da função `.on('change')`, que fará uma ação quando o elemento em watch for alterado;

```
function watch() {
    gulp.watch('css/scss/*.scss', compilaSass);
    gulp.watch('*.html').on('change', browserSync.reload);
};
```

O método **reload** (atributo da função *BrowserSync*, é responsável por recarregar a página, quando alterada).

Também podemos alterar a função Watch para "escutar" outras extensões do diretório, passando para o parâmetro `.watch` um Array das extensões desejadas:

```
function watch() {
    gulp.watch('css/scss/*.scss', compilaSass);
    gulp.watch(['*.html', '*.php']).on('change', browserSync.reload);
};
```

Javascript Concat

Documentação Concat

O pacote *Concat*, concatena pequenos pedaços de arquivos Javascript, e retona em um único arquivo todos esses arquivos mergidos.

Instalação:

```
npm install gulp-concat
```

Forma Correta

```
function gulpJs() {
    return gulp.src('js/assets_js/*.js')
        .pipe(concat('main.js'))
        .pipe(gulp.dest('js/'));
}
```

Obs.: A função Concat, recebe como primeiro parâmetro o output do arquivo final `main.js`

Forma Incorreta

```
function gulpJs() {
    return gulp.src('js/*.js')
        .pipe(concat('main.js'))
        .pipe(gulp.dest('js/'));
}
```

Note que o código acima está redirecionando o arquivo final `Main.js` *para a mesma pasta onde está buscando os arquivos que serão concatenados. Esta ação pode gerar um loop infinito na linha de comando, visto que toda vez que o arquivo* `main.js` *for gerado, a função watch perceberá uma mudança e concatenará os arquivos novamente. Para corrigir, redirecione os arquivos para um diretório diferente, ou sinalize à função que este arquivo deve ser ignorado inserindo os parâmetros em vetor: `gulp.watch(['js/*.js', '!js/main.js']);`**

Ordem de Concatenação

O pacote *Concat* concatena os arquivos por ordem alfabética, porém se uma outra ordem for desejada, insira os arquivos e seus respectivos diretórios em um array, definindo a ordem desejada:

```
function gulpsJs() {
  return gulp.src(['./diretorio/arq3.js', './diretorio/arq1.js',
'./diretorio/arq2.js'])
    .pipe(concat('main.js'))
    .pipe(gulp.dest('./diretorio/'));
});
```

Babel

Documentação Babel

Compila o código *Javascript* moderno para versões de código *Javascript* que sejam suportados por browsers não atualizados

Instalação

```
npm install gulp-babel @babel/core @babel/preset-env
```

Podemos usar o babel em uma função existente, que já trate arquivos ***Javascript*** com outras funções encadeadas:

```
function gulpJs() {
    return gulp.src('js/assets_js/*.js')
        .pipe(concat('main.js'))
        .pipe(babel({
            presets: ['@babel/env']
        }))
        .pipe(gulp.dest('js/'));
}
```

*Obs.: Note que o pacote Babel foi inserido após o pacote **Concat**, visto que sua inserção acima traria erros para o código, podendo fazer com que códigos sejam inseridos de forma duplicada, devido a concatenação trazer blocos de arquivos **Javascript**.*

Uglify

Documentação Uglify

O pacote Uglify minimiza arquivos Javascript.

Instalação

```
npm install gulp-uglify
```

*Nota: Este pacote minifica arquivos de versões anteriores do Javascript (abaixo do Ecmascript 6), portanto para arquivos igual ou acima desta versão, usar o Pacote **Gulp-uglify-es**. No exemplo abaixo, estamos utilizando o Babel para converter arquivos **Ecma 6** para versões abaixo, portanto funcionará normalmente.*

Exemplo de Código:

```
function gulpJs() {
    return gulp.src('js/assets_js/*.js')
        .pipe(concat('main.js'))
        .pipe(babel({
            presets: ['@babel/env']
        }))
        .pipe(uglify())
        .pipe(gulp.dest('js/'))
        .pipe(browserSync.stream());
}
```

Autoprefixer

Documentação Autoprefixer

O pacote *Autoprefixer*, é responsável por incluir nos atributos css, prefixos (webkit,moz), quando determinado atributo não é aceito por determinados navegadores

Instalação

```
$ npm install gulp-autoprefixer
```

Exemplo de Código:

```
function compilaSass() {
    return gulp.src('css/scss/*.scss')
        .pipe(sass({ outputStyle: "compressed" }))
        .pipe(autoprefixer({ cascade: false }))
        .pipe(gulp.dest('css/'))
        .pipe(browserSync.stream());
}
```