Para que o Gulp funcione, é necessário ter instalado o node.js

Para verificar se existe na maquina, usar o comando abaixo:

>node –version

Como o Gulp é uma dependência do node, precisamos usar o comando abaixo na pasta do projeto para criar o arquivo .json para armazenar nossas dependências

>npm init

Comando para instalar o Gulp como dependência de desenvolvedor (O node tem dois tipos de dependência, de desenvolvedor que é usada para o desenvolvedor fazer o seu trabalho e também a dependência de aplicação onde a aplicação depende disso para rodar o projeto)

>npm install gulp@3.9.0 --save-dev

O gulp não tem um executável, portanto uma boa maneira de fazer rodar é colocando o código abaixo dentro de script no .json, onde a chave sera o comando que iremos dar no terminal e o valor é o nome do binário que sera encontrado e executado.

"scripts": {

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

"gulp": "gulp"

},

Para rodar o gulp

>Npm run gulp

Criamos um arquivo chamado gulpfile, para armazenar os scripts

Para chamar uma “instancia do gulp”, usa-se o require. O parâmetro seria o nome da pasta do gulp dentro da pasta node\_modules

O gulp tem um método chamado src para definir o caminho da pasta de onde pegaremos as imagens, outro método é o dest. Esse método serve para definor o local de armazenamento dessas imagens

var gulp = require('gulp');

gulp.src('src/img');

gulp.dest('src/img');

Modulo do gulp para minificar imagens

>npm install gulp-imagemin@2.3.0 --save-dev

O pipe é usado para ligar o retorno de cada método ao próximo método

gulp.src('src/img')

.pipe(imagemin())

.pipe(gulp.dest('src/img'));

Se executar o script direto desse forma ira apresentar erro porque o gulp precisa de uma tarefa padrão, então devemos cria-la, da seguinte forma(código completo ate agora)

var gulp = require('gulp'),

imagemin = require('gulp-imagemin')

gulp.task('build-img', function(){

gulp.src('src/img')

.pipe(imagemin())

.pipe(gulp.dest('src/img'));

})

E depois chamar no terminal dessa forma

>npm run gulp build-img

Agora sim devera funcionar, mas ainda não ira minificar as imagens pq elas estão dentro de outras pastas, para corrigir isso é só fazer a alteração abaixo

gulp.src('src/img/\*\*/\*')

Porem, dessa forma não estamos armazenando as imagens antigas e vaso algum designer quiser fazer mais alguma alteração ele teria que modificar os arquivos já melhorados e com isso a imagem com o tempo iria perder qualidade.

Para isso, devesse fazer uma cópia da pasta do projeto e fazer as devidas melhorias nessa pasta. Para isso vamos criar uma task que copia essa pasta

gulp.task('copy', function(){

gulp.src('src/\*\*/\*')

.pipe(gulp.dest('dist'));

});

Plugin para remover arquivos

>npm install [gulp-clean@0.3.1](mailto:gulp-clean@0.3.1) – save-dev

No código abaixo, escolhemos a pasta que iremos apagar e passamos no método pipe a própria função para que ela seja apagada

gulp.task('clean', function(){

gulp.src('dist')

.pipe(clean());

});

Mas seguindo essa lógica temos um problema. Antes de copiar a pasta dist precisamos garantir que a pasta dist já tenha sido apagada. Para isso dentro da função de copy colocamos a clean como dependência e sempre antes que a copy for chamada a clean sera executada ao mesmo tempo, para garantir performance, tudo é assincrono. Segue exemplo

gulp.task('copy', ['clean'], function(){

gulp.src('src/\*\*/\*')

.pipe(gulp.dest('dist'));

});

Porem nesse caso isso não pode acontecer, para resolver usamos o código abaixo onde quando terminar a execução a variável strem vai conter um valor indicando que acabou a execução

gulp.task('clean', function(){

var stream = gulp.src('dist')

.pipe(clean());

return stream;

});

Cada navegador tem um numero de requisições que podem ser feitas de uma única vez, se uma pagina web passar esse numero o próximo arquivo fica na fila ate surgir espaço para mais uma requisição. Para evitar esse tipo de coisa e deixar a experiência do usuário melhor, podemos juntar vários arquivos em um único para o servidor fazer reduzir a apenas uma requisição. Para isso iremos usar mais um plugin

>npm install [gulp-concat@2.6.0](mailto:gulp-concat@2.6.0) –save-dev

Para poder concatenar os arquivos a função seria como o exemplo abaixo

gulp.task('build-js', function(){

gulp.src('dist/js/\*\*/\*.js')

.pipe(concat('all.js'))

.pipe(gulp.dest('dist/js'));

});

Porem o html continuaria a ser chamado pelos nomes antifos, para isso podemos usar o plugin gulp-html-replace

>npm install [gulp-html-replace@1.5.4](mailto:gulp-html-replace@1.5.4) –save-dev

A função para usar é como no exemplo abaixo. Nos parâmetros da função htmlReplace passamos uma chave -> valor.

gulp.task('build-html', function(){

gulp.src('dist/\*\*/\*.html')

.pipe(htmlReplace({

js: 'js/all.js'

}))

.pipe(gulp.dest('dist'));

});

A chave seria um nome dado para uma marcação no html para o plugin saber onde estão os nomes dos arquivos que estão sendo chamados e o valor seria o nome do arquivo e com o caminho do novo arquivo gerado

<!-- build:js -->

<script src="js/jquery.js"></script>

<script src="js/home.js"></script>

<!-- endbuild -->

Da forma que esta nosso script ainda não temos muito de performance, então vamos juntar todas as funções para chamarmos de uma vez só tudo o que fizemos ate agora

gulp.task('default', ['copy'], function(){

gulp.start('build-img', 'build-js', 'build-html');

});

Como agora temos a tarefa default, só precisar chamar no console o seguinte comando

>npm run gulp

Do jeito que foi feito o build-js, quando executarmos pode ser que o site apresente erro pq ele junta todos os arquivos sem uma ordem definida. Para resolver isso precisamos definir uma ordem de carregamento do novo arquivo. Segue modelo abaixo

gulp.task('build-js', function(){

gulp.src(['dist/js/jquery.js',

'dist/js/home.js',

'dist/js/produto.js'])

.pipe(concat('all.js'))

.pipe(gulp.dest('dist/js'));

});

Para minificar os arquivos e reduzirmos ainda mais o tamanho do projeto usamos o plugin uglify

>npm install gulp-uglify@1.4.1 –save-dev

Para podemos usar, no método de concatenar os js adicionamos um pipe a mais para que ele antes de gravar já minifique os arquivos

gulp.task('build-js', function(){

gulp.src(['dist/js/jquery.js',

'dist/js/home.js',

'dist/js/produto.js'])

.pipe(concat('all.js'))

.pipe(uglify())

.pipe(gulp.dest('dist/js'));

});

Deixar os arquivos em um só é muito pratico, porem os arquivos começam a ficar grande demais e isso tbm não é bom para o projeto. Para isso vamos usar o plugin usemin

>npm install [gulp-usemin@0.3.14](mailto:gulp-usemin@0.3.14) –save-dev

Com esse novo plugin nem precisaremos das tarefas build-js e build-html, então podemos apaga-la e criar uma nova função para substitui-la.

Nessa função como argumento da função usemin passamos todos os plug-ins que serão usados ao encontrar a marcação js ou qualquer outra

gulp.task('usemin', function(){

gulp.src('dist/\*\*/\*.html')

.pipe(usemin({

js: [uglify],

css: [cssmin]

})

.pipe(gulp.dest('dist')));

});

E no html mudamos a marcação tbm

<!-- build:js js/index.min.js-->

<script src="js/jquery.js"></script>

<script src="js/home.js"></script>

<!-- endbuild -->

<!-- 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/mobile.css">

<!-- endbuild-->

Para podemos usar o cssmin

>npm install gulp-cssmin@0.1.7 --save-dev

Para automatizar o recarregamento do navegador ao salvar um arquivo, instalar o plugin abaixo:

>npm install [browser-sync@2.9.8](mailto:browser-sync@2.9.8) –save-dev

Função para criar uma task de servidor onde não sera mais necessário o reload da pagina ao salvar um arquivo. Basedir seria a pasta raiz do projeto

Já o método watch é próprio do gulp. É ele que fica de olho se a pasta teve alteração, e caso tenha com a função de chande executamos o reload do plugin browser sync

gulp.task('server', function(){

browserSync.init({

server: {

baseDir: 'src'

},

})

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

});

Para acessarmos nosso projeto através de um servidor web eu faço: [http://localhost:3000](http://localhost:3000/)(a porta é do seu servidor, pode ser 8080, etc.), veja que a porta é a mesma do BrowserSync e isso nos causará problemas. Podemos resolver isso facilmente habilitando o proxy do BrowserSync. Editando nosso gulpfile.js:

gulp.task('server', function() {

browserSync.init({

server: {

proxy: "localhost:3000"

}

});

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

});

Veja que não usamos mais a propriedade baseDir.

Quando rodarmos a tarefa server no terminal, o navegador será aberto automaticamente, porém a porta será acrescida de 1. Teremos: http://localhost:3001. Agora podemos usar o BrowserSync em conjunto com nosso servidor favorito. É claro, tanto o BrowserSync quando o servidor precisam estar rodando.

Para receber um aviso de que esta tudo funcionando e o código não está quebrado, podemos usar a biblioteca abaixo

>npm install [gulp-jshint@1.11.2](mailto:gulp-jshint@1.11.2) –save-dev

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

Esse bloco de código colocamos dentro da função server. Onde o event possui varias propriedades do arquivo que sofreu modificação, no método path por exemplo pegamos o caminho desses arquivos, o método repórter sera para ser retornado os erros visualmente. Se deixamos sem o jshint ira reconhecer os erros mas não iria mostrar no console os mesmos.(Esse parâmetro dentro do repórter é opcional, ele representa um outro plugin para deixar o retorno mais bonitinho)

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

gulp.src(event.path)

.pipe(jshint())

.pipe(jshint.reporter(jshintStylish));

});

Plugin para retornos de erros de css

>npm install [gulp-csslint@0.2.0](mailto:gulp-csslint@0.2.0) –save-dev

Para usar é basicamento o mesmo funcioanemento do js, dentro do server

Quando estamos desenvolvendo não ficamos de olho para que o código funcione em todos os navegadores, para evitar esse problema podemos usar um plugin que automaticamente coloca os prefix no código

>npm install [gulp-autoprefixier@3.0.2](mailto:gulp-autoprefixier@3.0.2) –save-dev

Para usar, uma boa maneira é chamar na função usemin antes de minificar os arquivos css

css: [autoprefixer ,cssmin]

Para definir a quantidade de versões passadas o plugin precisa preparar o código na raiz do projeto criamos um arquivo sem extensão com a quantidade, como exemplo abaixo

last 40 versions

site para referencia do que pode ser usado em quais navegadores

<http://www.caniuse.com>

Automatizar compilação dearquivos less

>npm install [gulp-less@3.0.3](mailto:gulp-less@3.0.3) –save-dev

Definir variáveis e função em less

@cor-especial: #dcdcdc;

.texto-sombreado(@top, @left, @blur, @color){

text-shadow: @top @left @blur @color;

}

Chamar uma função

.texto-sombreado(1px, 1px, 2px, #FFF);

Chamar uma variável

background-color: @cor-especial;

Obsevavel para pasta de arquivos less para serem interpretados automaticamente e irem pra pasta de css, a função error serve para que quando o less retornar um erro o server não exiba um erro e pare tudo

gulp.watch('src/less/\*\*/\*.less').on('change', function(event) {

var stream = gulp.src(event.path)

.pipe(less().on('error', function(erro) {

console.log('LESS, erro compilação: ' + erro.filename);

console.log(erro.message);

}))

.pipe(gulp.dest('src/css'));

});