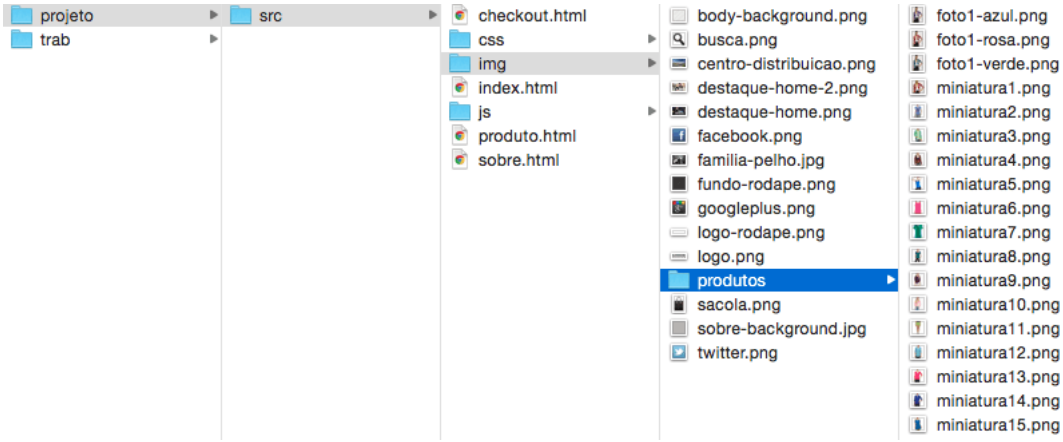
03

O peso das imagens

O projeto está pronto. Está pronto mesmo?

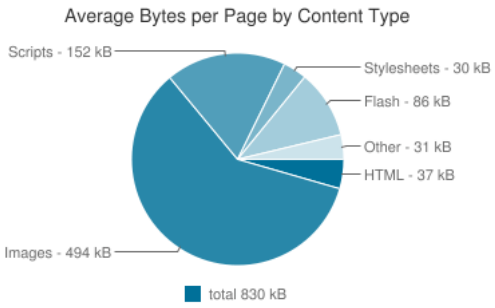
Temos aqui um projeto front-end pronto para entrar em produção:



Temos arquivos HTML, CSS, JS e imagens, nada fora do comum. Então, já podemos colocar o projeto no ar? Vejamos.

O peso das imagens

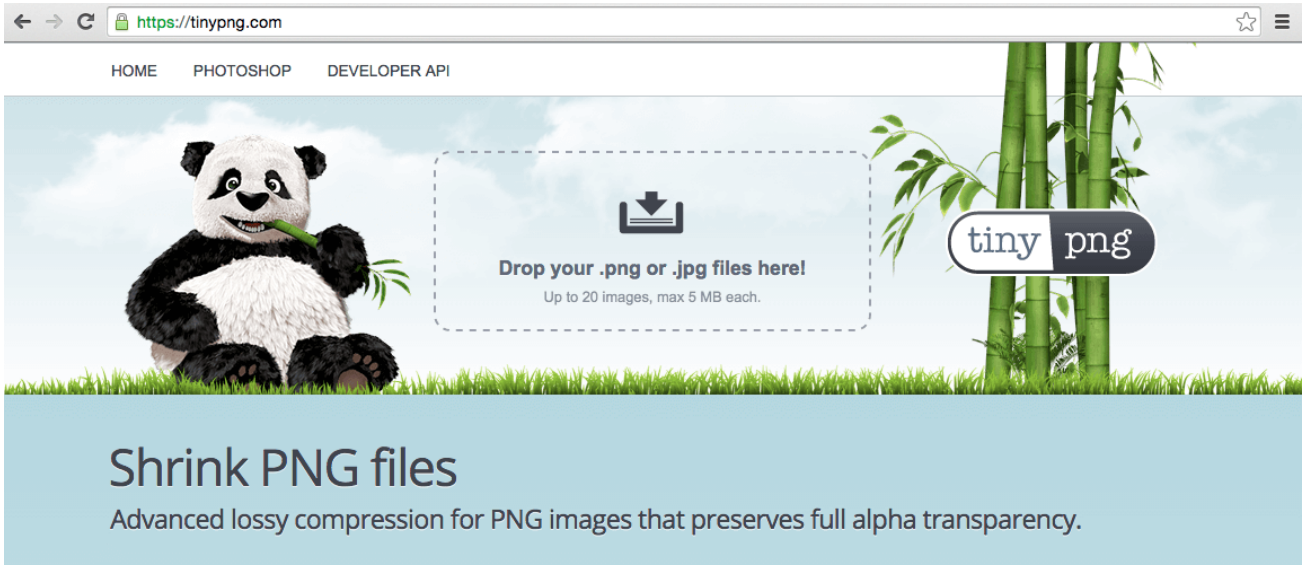
Passeando pelo projeto, vemos que ele possui algumas imagens. Vamos dar uma olhada nos tamanhos delas? Nem são tão grandes assim, mas se você já visitou o O **HTTPArchive.org** ele possui gráficos com informações históricas coletadas mensalmente sobre os 17 mil sites mais acessados da Internet mundial. Um desses gráficos mostra que mais de 60% do peso de uma página é por causa das imagens:



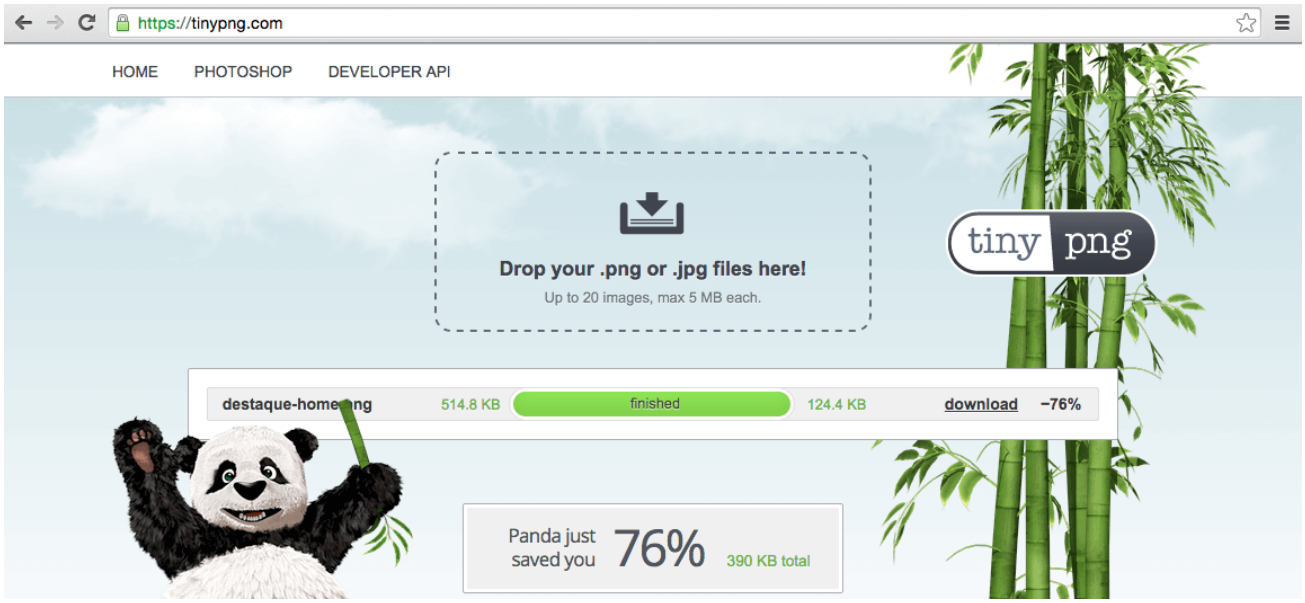
Será que podemos tornar ainda menor nossas imagens otimizando-as?

Otimizações manuais

Qual ferramenta usar? Que tal o TinyPNG (https://tinypng.com)? É uma ferramenta online a gratuita que otimiza arquivos `png` e `jpg`:



Agora é só clicar em **Drop your .png or .jpeg files here!** que abrirá o diálogo de seleção de arquivos. Vamos escolher uma das imagens, por exemplo, `projeto\src\img\destaque-home.png` . Assim que você escolher o arquivo, o upload iniciará. No final, será exibido no seu navegador uma estatística para indicar quanto a nossa imagem diminuiu, inclusive um link `download` para que possamos baixar o arquivo:



O site otimiza nossa imagem oferecendo a opção de salvá-la em nossa máquina, excelente. Vamos substituir o arquivo original pelo compactado e concluímos nossa tarefa. Será?

O problema dos processos manuais

Apesar de funcional, temos um problema: imagine fazer isso com todas as imagens do projeto? Temos mais de 40! Ainda há outro problema: cada imagem otimizada deverá ser salva na pasta correta em nosso projeto. Além disso, essa é apenas a tarefa de compactação de imagens, e se tivéssemos outras?

Workflow e sua automação

Se você é um desenvolvedor front-end já deve ter se preocupado não apenas em otimizar imagens, mas em *minificar* e *concatenar* seus scripts, inclusive já deve ter experimentado o uso de algum pré-processador como LESS (http://lesscss.org/).

A qualidade e quantidade das tarefas variam de acordo com as necessidades do seu projeto e essas tarefas acabam gerando um **fluxo (workflow)** identificável e que muitas vezes é documentado para ser usado por toda a equipe.

O problema é que tudo que é feito pelo ser humano está sujeito a erro. Por mais que tenhamos um manual nos dizendo o que fazer, nada nos impede de pularmos um dos passos, o que pode afetar diretamente o resultado final.

Para solucionar problemas como esse, foram criadas no mercado ferramentas de construção (build) de projetos como Ant (http://ant.apache.org/), Gradle (http://www.gradle.org/) e Maven (http://maven.apache.org/), mas há aquelas que são voltadas especialmente para programadores front-end como **Grunt (http://gruntjs.com/)** e o **Gulp**. Neste treinamento aprenderemos o **Gulp**.

Bem-vindo ao Gulp

O Gulp (http://gulpjs.com/) é uma *ferramenta de build* totalmente feita em JavaScript tornando-a atrativa no mundo front-end, especialmente para nosso projeto. Porém, para que funcione, precisamos do Node.js (http://nodejs.org/) instalado em nosso máquina.

IMPORTANTE: antes de continuar, faça o primeiro exercício do capítulo (https://cursos.alura.com.br/course/gulp/section/1/task/3). Nele você terá o link do projeto que usaremos neste treinamento, inclusive você terá as instruções de como instalar o Node.js nas plataformas Windows/Linux/Mac e sugestões de editores de texto. É fundamental que você faça esse exercício antes de continuar.

Fez o primeiro exercício do capítulo baixando o projeto e instalando a infraestrutura necessária do treinamento, certo? Muito bem, podemos continuar.

A primeira coisa é termos o Gulp instalado para que possamos criar nossos scripts de automação. Sua instalação é feita através do terminal e dentro da pasta **projeto** (não esqueça de mudar para essa pasta através do terminal).

O Gulp é um módulo do Node.js e todos os seus módulos são instalados através do npm (https://www.npmjs.com/). O npm acessa um repositório público na web com vários projetos que podem ser consumidos pela nossa aplicação, inclusive o Gulp. O primeiro passo antes de baixarmos o Gulp é criarmos o arquivo **package.json** que manterá uma lista de todos os módulos que formos instalando em nossa aplicação.

O arquivo `package.json` possui uma estrutura peculiar e por esta razão há um assistente para sua criação. No terminal, dentro da pasta **projeto** vamos executar o comando:

```
npm init
```

Um assistente fará perguntas como o nome do projeto, sua versão entre outras. Você pode responder atentamente cada uma das perguntas feitas, porém você pode dar ENTER para todas elas que um padrão será adotado. Por exemplo, o nome do projeto será o nome da pasta na qual o comando foi executado. O resultado final será o arquivo `package.json` com a seguinte estrutura:

```
{
  "name": "projeto",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
```

```
  },
  "author": "",
  "license": "ISC"
}
```

Excelente, agora vamos baixar o Gulp através do comando `npm install`. Este comando recebe como parâmetro o nome do módulo, em nosso caso, `gulp` seguido de `@` e sua versão. Caso tivéssemos omitido a versão, a mais atual seria baixada. É importante usarmos a mesma versão deste treinamento para evitarmos qualquer problema de compatibilidade. Por fim, ainda queremos que o módulo fique listado em `package.json` através de `--save-dev`:

```
npm install gulp@3.9.0 --save-dev
```

Quando o comando terminar, veremos que a pasta **node_modules** foi criada em nosso projeto. Dentro dela, existe a pasta `gulp` que curiosamente tem o nome do mesmo módulo que instalamos, mas isso não é por acaso. A pasta criada tem sempre o nome do módulo que instalamos através do npm. E como ficou nosso `package.json`, vamos visualiza-lo?

```
{
  "name": "projeto",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "gulp": "^3.9.0"
  }
}
```

Confirmamos que o Gulp e sua versão estão presentes em nosso arquivo.

Outra vantagem de salvarmos nossas dependências em `package.json` é que você pode compartilhar seu projeto sem a pasta `node_modules`, que deve ser baixada novamente através do comando `npm install`, sem qualquer parâmetro. O comando lerá do `package.json` todos os módulos de que sua aplicação depende baixando-os todos de uma só vez, levando em consideração cada versão utilizada.

E agora? Já podemos executar o Gulp? Sim, mas como ele foi instalado dentro da pasta `node_modules` seremos obrigados a escrever toda vez `./node_modules/gulp/bin/gulp.js`. Podemos resolver alterando nosso arquivo `package.json`.

Editando `package.json`, temos a seguinte chave:

```
// código anterior omitido
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
// código posterior omitido
```

Vamos adicionar a chave `"gulp" : "gulp"`:

```
// código anterior omitido
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "gulp": "gulp"
  },
// código posterior omitido
```

Agora, se quisermos executar o Gulp através do terminal basta executarmos o comando:

```
npm run gulp
```

Excelente, nosso Gulp é executado, mas recebemos um erro:

```
No gulpfile found
```

Esse erro acontece, porque toda vez que o Gulp é executado ele procura o arquivo **gulpfile.js**.

O arquivo gulpfile.js e novas tarefas

Vamos criar o arquivo `gulpfile.js` dentro da pasta **projeto** e rodar o Gulp logo em seguida. Será que funciona?

```
Task 'default' is not in your gulpfile
Please check the documentation for proper gulpfile formatting
```

Recebemos outra mensagem de erro, desta vez com o texto **"Task 'default' is not in your gulpfile"**. Faz todo sentido, `task` em português significa tarefa e não configuramos tarefa alguma em nosso arquivo `gulpfile.js` . Qual será nossa primeira tarefa? Ainda tem duvidas? A minificação automática de imagens!

Criando nossa primeira tarefa: minificação de imagens

O primeiro passo, antes de qualquer tarefa, é termos acesso ao Gulp que baixamos através do npm em nosso `gulfile.js` . No mundo Node.js quando queremos carregar um módulo em nosso script usamos a função **require** que recebe como parâmetro o nome do módulo. Então, vamos carregar o módulo do Gulp recebendo o retorno da função `require` em uma variável com mesmo nome, mas poderia ser qualquer outro:

```
// gulpfile.js

var gulp = require('gulp');
```

É através da variável `gulp` que interagimos com um objeto que representa o módulo Gulp. Este objeto possui uma série de métodos auxiliares. Qual deles usaremos? Com certeza aquele que lê a pasta com as imagens do nosso projeto, os arquivos origem da nossa tarefa. É por isso que existe o método **gulp.src** que recebe como parâmetro o nome de um arquivo ou de uma pasta. Alterando nosso script:

```
// gulpfile.js
var gulp = require('gulp');
```

```
gulp.src('src/img');
```

O que esse comando faz é gerar um fluxo de leitura para a origem `projeto/src/img` e todos os seus arquivos. Muito bem, estamos lendo dessa pasta, mas qual será a pasta destino? A mesma, pois queremos ler as imagens, otimizá-las e gravá-las em seu local de origem. Definimos o destino através do método **gulp.dest** que cria um fluxo de escrita para a mesma pasta.

```
// gulpfile.js
var gulp = require('gulp');

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

Temos um fluxo de leitura e nosso fluxo de escrita: leremos as imagens e gravaremos no mesmo lugar, mas espere um pouco: onde fica a minificação das imagens? Teria que ser realizada entre o fluxo de leitura e o de escrita, certo? Certíssimo, mas o Gulp não vem com esse recurso por padrão. Para essa tarefa especial usaremos o plugin do Gulp chamado **gulp-imagemin (https://github.com/sindresorhus/gulp-imagemin)**.

Todo plugin do Gulp é também um módulo do Node.js e deve ser instalado através do npm. Vamos lá? Em nosso terminal:

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

Como já vimos, a instalação desse módulo criará a pasta `node_modules/gulp-imagemin` e como usamos `--save-dev` ele será adicionado como dependência em nosso arquivo `package.json` .

Agora, em nosso `gulpfile.js` , vamos importar o módulo através da função `require` e para nossa comodidade armazená-lo em uma variável chamada `imagemin` :

```
// gulpfile.js
var gulp = require('gulp')
  ,imagemin = require('gulp-imagemin');

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

Tudo certo, agora precisamos saber como conectar o fluxo de leitura ao `imagemin` que precisa saber quais arquivos considerar e este último ao fluxo de escrita para gravar os arquivos resultantes da minificação. Na construção civil, usamos tubos que podem ser conectados de diversas formas para levar o fluxo de água de um local para o outro. Imagine o `imagemin` como o aquecedor entre as ligações, que opera sobre o fluxo. A água que passar por ele estará transformada, isto é, quente.

A analogia com tubos não foi por acaso, porque é através do método `.pipe` (tubo) que ligamos fluxos, seja ele de leitura ou escrita. Ele recebe como parâmetro outro fluxo que desejamos nos conectar. Vamos ligar o fluxo de leitura ao `imagemin` e este último ao fluxo de escrita:

```
// gulpfile.js
var gulp = require('gulp')
  ,imagemin = require('gulp-imagemin');

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

Excelente, agora temos a entrada (leitura) que será processado pelo `imagemin` e seu resultado associado a uma saída (escrita). Será que isso é suficiente? Vamos testar:

```
npm run gulp
```

Não foi dessa vez, recebemos a seguinte mensagem de erro no console:

```
Task 'default' is not in your gulpfile
```

Configuramos uma tarefa, mas não é nossa tarefa padrão, muito menos tem um nome. No Gulp, precisamos executar nosso código através de tarefas e toda tarefa possui um nome. É por isso que usamos a função `gulp.task` para criar tarefas. Nossa tarefa se chamará `build-img` :

```
// gulpfile.js
var gulp = require('gulp')
  ,imagemin = require('gulp-imagemin');

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

 // linha quebrada para ajudar na leitura
  gulp.src('src/img')
    .pipe(imagemin())
    .pipe(gulp.dest('src/img'));
});
```

O método `gulp.task` recebe como primeiro parâmetro o nome da nossa tarefa e como segundo uma função que será executada assim que a tarefa foi chamada através do Gulp no terminal. Para chamá-la, usamos `npm run gulp NomeDaTarefa` . Vamos testar:

```
npm run gulp build-img
```

Nenhum erro no terminal, mas também nenhuma imagem foi minificada. Como sabemos? Olhando o log:

```
Starting 'build-img'...
Finished 'build-img' after 11 ms
gulp-imagemin: Minified 0 images  <--- nenhuma img foi minificada
```

O problema é que nosso fluxo de leitura está considerando a pasta `src/img` e não seus arquivos. Podemos resolver isso usando o seguinte curinga `src/img/**/*` . Usamos o padrão `**` para indicarmos que queremos varrer todas as pastas dentro de `src/img` e usamos `*` para indicar que queremos todos os arquivos dessa pasta. Isso é o que chamamos de **glob pattern**. Vamos alterar nosso script e testar mais uma vez, mas antes vamos verificar o tamanho da imagem

`src/img/destaque-home.png` que é 515KB. Agora que sabemos seu tamanho original, podemos comparar com o resultado final. Alterando:

```
// gulpfile.js
var gulp = require('gulp')
  ,imagemin = require('gulp-imagemin');

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

  gulp.src('src/img/**/*')
    .pipe(imagemin())
    .pipe(gulp.dest('src/img'));
});
```

Agora, executando nossa tarefa:

```
npm run gulp build-img
```

Recebemos a mensagem no console:

```
Starting 'build-img'...
Finished 'build-img' after 10 ms
gulp-imagemin: Minified 32 images (saved 181.09 kB - 8.7%)
```

Percebam que temos ainda uma estatística indicando a quantidade em kb e em percentual do total minificado de imagens. Chique, não? Com um único comando no terminal podemos processar, 100, 200 ou mais imagens de uma só vez!

Olhando nosso arquivo `src/img/destaque-home.png` ele agora tem 451kb, antes era 515kb.

Chegou a hora de praticar agora o que você aprendeu.