13

As tecnologias Web e os processos

Transcrição

aula01-video04-tecnlogias-web-e-processos

Carregando nosso próprio HTML

Apesar de ser interessante poder carregar sites externos, nós temos interesse mesmo é utilizar um HTML próprio para montar o nosso `app`. Vamos começar a criar a estrutura de pastas da aplicação. Para manter a organização, vamos criar a pasta `app` que irá conter nossos arquivos estáticos. Crie sua pasta como abaixo:

```
alura-timer/
├── app
│   └── index.html
├── main.js
└── package.json
```

E no `index.html` crie uma estrutura inicial:

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <title>Alura Timer</title>
</head>
<body>
    <h1>Olá Mundo</h1>
</body>
</html>
```

Agora para que o `index.html` seja carregado com o `mainWindow` precisamos indicar através da função `loadURL`:

```
//main.js
let mainWindow = null;
app.on('ready', () => {

    mainWindow = new BrowserWindow({
            width: 800,
            height: 600
        });
    mainWindow.loadURL(`file://${__dirname}/app/index.html`);
});
```

Repare que para indicar o caminho do `index.html` estamos utilizando alguns dos recursos do Node, como o `file://` (File protocol) para apontar caminhos de arquivos e a também a variável implícita `__dirname` que nos diz qual é o diretório atual. Utilizamos as `template strings` do Javascript para facilitar a concatenção.

Rode novamente a aplicação e veja que agora conseguimos carregar o nosso arquivo.

Fechando a aplicação educadamente

Até agora temos reiniciado a aplicação de modo brusco, fechando-a pelo terminal. Vamos manter as boas práticas aqui e criar um evento para que quando todas as janelas sejam encerradas, o próprio electron se encarregue de encerrar a aplicação de modo amigável para o sistema operacional.

Vamos colocar um novo evento no objeto `app`:

```
//main.js
let mainWindow = null;
app.on('ready', () => {
    ...restante do código
});

app.on('window-all-closed', () => {
      app.quit();
 });
```

Este evento escuta pelo fechamento de todas as janelas do aplicativo, e caso isto aconteça ele encerra a aplicação através de uma função própria, retornando um código correto para o sistema operacional.

As tecnologias faltantes: CSS e JS

Como estamos utilizando uma página HTML comum, podemos importar um CSS tradicional.

Crie a pasta `css` e importe o arquivo `index.css` abaixo e verifique se a cor foi aplicada:

```
alura-timer/
├── app
│   ├── css
│   │   └── index.css
│   └── index.html
├── main.js
└── package.json
```

```
/*index.css*/
h1{
    color: blue;
}
```

```
<!-- index.html -->
<head>
    <meta charset="UTF-8">
    <title>Alura Timer</title>
    <link rel="stylesheet" href="css/index.css">
</head>
```

Agora para o Javascript, vamos criar um arquivo `renderer.js` dentro de uma pasta `/js` (já vamos entender este nome) e colocar uma pequena mensagem `Javascript Importado` lá também:

```
alura-timer/
├── app
│   ├── css
│   │   └── index.css
│   ├── js
│   │    └── renderer.js
 |    └──index.html
├── main.js
└── package.json
```

```
//renderer.js
console.log("Javascript carregado!");
```

E como todo bom Javascript, podemos importá-lo em nosso arquivo HTML. Agora temos uma pequena novidade, que já que estamos utilizando o NodeJs por debaixo dos panos, podemos utilizar os seus recursos em nosso Javascript.

Então, ao importar um novo arquivo, podemos fazer do modo tradicional, através da tag `<script>` :

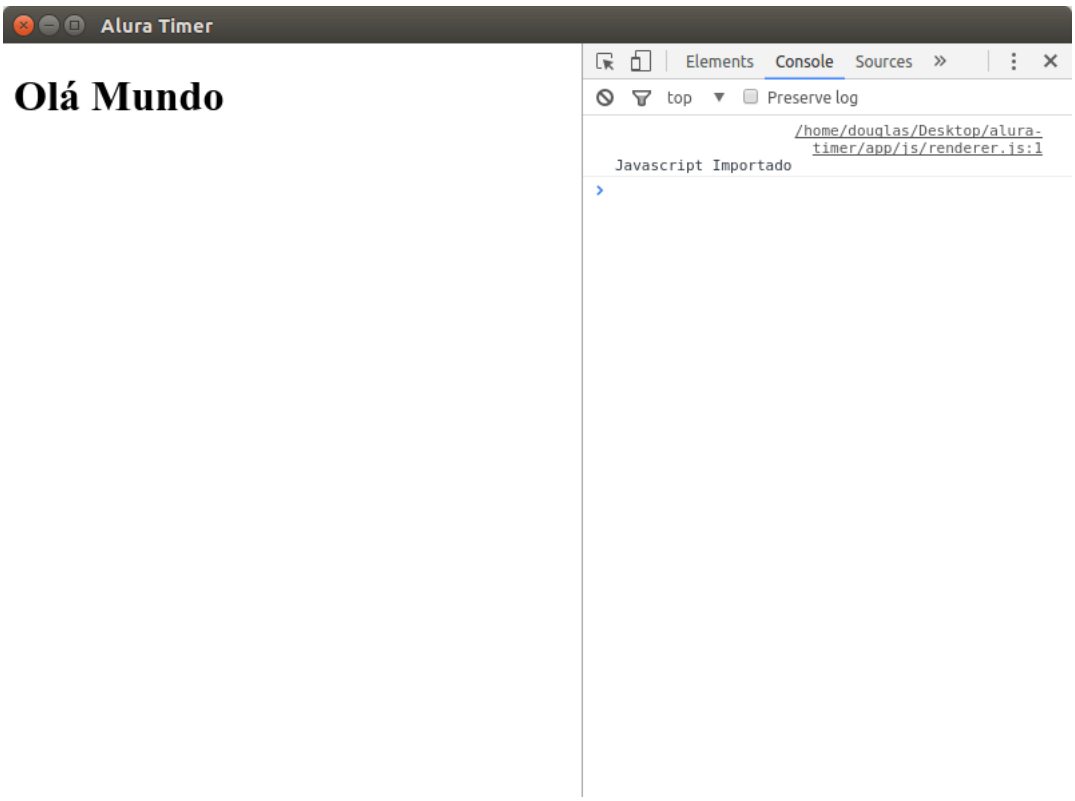
```
<!-- index.html -->
<body>

    <script src="js/renderer.js"></script>
</body>
```

Ou utilizando o sistemas de módulos do Node, com o `require` :

```
<!-- index.html -->
<body>
    <h1>Olá Mundo</h1>
    <script>require('./js/renderer.js')</script>
</body>
```

Vamos utilizar este segundo modo neste curso. Rode novamente seu app através do comando `npm start` . Note que dessa vez não foi exibido no terminal na mensagem de nosso `console.log` , pois afinal isto foi um Javascript importado no navegador, logo a mensagem deve ser exibida no console do navegador. Como estamos utilizando o Chromium, podemos abrir seu DevTools através do atalho `CTRL/CMD+SHIFT+I` :



Veja que é diferente entre um `console.log()` num arquivo importado pelo navegador e um `console.log()` em nosso `main.js`. Esta diferença acontece pelo modo como o Electron trabalha com suas janelas, vamos entender isto melhor.

Os processos no Electron

As aplicações feitas em Electrons funcionam em cima de processos, e toda a aplicação começa no processo principal, o nosso *Main Process*. O *Main Process* é o processo de entrada do Electron, aonde a nossa aplicação inicia, representada pelo nosso `main.js`. Este é um processo mesmo, do Sistema Operacional, que pode até ser visualizado em seu monitor de recursos.

As janelas criadas, são os processos de visualização, conhecidos como os *Renderer Process*, que são outros processos independentes do *Main Process*, que tem foco de ser a View para o usuário.

O Electron cria estes dois diferentes tipos de processos pois cada um deles é especialista em certas coisas:

O **Main Process** é único na aplicação, existe um único que roda na aplicação inteira, não tem uma interface visual e é o controlador principal da aplicação. Ele é quem gera os novos **Renderer Process**.

Os **Renderer Process** são os que os usuários veem e interagem, podendo ser mais de um. Normalmente cada Renderer Process está associado a uma janela visual, mas é possível criar render process para tarefas de alto processamento que não são visuais, para que ocorram paralelamente ao processo principal e não travem a aplicação.

Além disto, cada processo tem acesso a API's específicas do Electron, que serão as API's que vamos explorar nos próximos capítulos.