04

Iniciando nosso Timer

Transcrição

Vamos implementar o **timer.js** como se ele fosse um módulo do Node, assim conseguiremos fazer uma requisição dele em qualquer lugar do nosso código. Já podemos fazer isso no **renderer.js**:

```
const timer = require('./timer');
```

Para criar um módulo, a primeira coisa que devemos fazer é utilizar o `module.exports`, no **timer.js**:

```
// timer.js

module.exports = {

}
```

Esse objeto terá diversas funções, como uma função para iniciar o *timer* e outra para pará-lo:

```
// timer.js

module.exports = {
    iniciar() {

    }, parar() {

    }
}
```

Desse modo, conseguiremos manipular o objeto `timer`, utilizar suas funções, etc. Por exemplo:

```
const timer = require('./timer');

timer.iniciar();
timer.parar();
```

Assim, separamos as responsabilidades, deixando o **timer.js** com toda a lógica e no arquivo principal, nós só requisitamos isso. Essa será a lógica de parte do código do nosso projeto, criaremos alguns módulos, cada um responsável por algo individual, e chamando-os através de "*requires*".

Iniciando a implementação do timer

Agora podemos pensar na lógica do nosso *timer*, da inicialização do tempo. O *timer* é um cronômetro, que cresce de 1 em 1 segundo, logo precisamos de um controlador de segundos, e quando clicamos no botão de *play*, o tempo inicializará, sendo trocado no HTML da página.

Se queremos que o *timer* manipule o elemento HTML, precisamos ter acesso ao mesmo, logo vamos recebê-lo na função `iniciar`:

```
// timer.js

module.exports = {
    iniciar(el) {

    }, parar() {

    }
}
```

Assim, quem quiser chamar a função `iniciar`, deve passar para a mesma o elemento que contém o tempo a ser manipulado. Já podemos fazer isso no **renderer.js**. Vamos selecionar o elemento que contém o tempo, que é um `span` com a classe `tempo` e passá-lo para a função `iniciar`:

```
// renderer.js

const { ipcRenderer } = require('electron');
// importando o timer.js
const timer = require('./timer');

let linkSobre = document.querySelector('#link-sobre');
let botaoPlay = document.querySelector('.botao-play');
// selecionando o span
let tempo = document.querySelector('.tempo');

linkSobre.addEventListener('click' , function(){
    ipcRenderer.send('abrir-janela-sobre');
});

let imgs = ['img/play-button.svg', 'img/stop-button.svg'];
botaoPlay.addEventListener('click' ,function () {
    // chamando a função iniciar, passando o elemento com o tempo
    timer.iniciar(tempo);
    imgs = imgs.reverse();
    botaoPlay.src = imgs[0];
});
```

Agora que temos acesso ao tempo, no **timer.js** podemos manipulá-lo, já que a cada 1 segundo, devemos acrescentar o tempo no *timer*. Para calcular esse 1 segundo, podemos utilizar a função `setInterval`:

```
// timer.js

module.exports = {
    iniciar(el) {
        setInterval(function() {

        }, 1000);
    }, parar() {
```

```
    }
}
```

Agora, precisamos alterar o conteúdo de texto do elemento, o seu `textContent`, para o seu conteúdo atual mais 1 segundo. Mas quando acessamos o seu conteúdo de texto, como o próprio nome já diz, ele é um texto! Para alterá-lo, nós temos que *parsear* esse texto, extrair os segundos, somar 1, e se chegar a 60 segundos, temos que zerar os segundos e acrescentar 1 minuto. Logo, há toda uma lógica para lidar com esse tempo, mas claro que não temos que fazer essa lógica na mão pois já existem diversas bibliotecas do JavaScript que fazem esse controle de período de tempo, como data e hora, para nós.

Trabalhando com tempo

Então, ao invés de adicionarmos o tempo na mão, vamos utilizar uma biblioteca que é especialista nisso, deixando que ela gerencie o tempo para nós. Vamos passar uma string para ela, pedir para que some um segundo, e nos retorne, para que possamos trabalhar com ela. Essa biblioteca é a **Moment.js** (https://momentjs.com/), especialista em trabalhar com o tempo.

Para utilizá-la, primeiro devemos instalá-la. Para isso, no terminal, vamos entrar **dentro da pasta do nosso projeto**, e executar o seguinte comando:

```
npm install moment@2.17.1 --save
```

Com a biblioteca instalada, podemos importá-la no nosso projeto:

```
// timer.js

const moment = require('moment');

module.exports = {
    iniciar(el) {
        setInterval(function() {

        }, 1000);
    }, parar() {

    }
}
```

Agora, para utilizá-la, como possuímos uma duração de tempo, podemos utilizar a função `duration`, claro, de `moment`, passando o tempo para ela, que é representado pelo conteúdo de texto do elemento, logo `el.textContent`. Além disso, vamos guardar o retorno dessa função em uma variável:

```
// timer.js

const moment = require('moment');

module.exports = {
    iniciar(el) {
        let tempo = moment.duration(el.textContent);
        setInterval(function() {
```

```
        }, 1000);
    }, parar() {

    }
}
```

Com o tempo em mãos, podemos convertê-lo para segundos, através da sua função `asSeconds()`. Isso é bem útil para nós, já que assim podemos somar 1 segundo sem problemas. Vamos aproveitar para criar uma variável e inicializá-la dentro do módulo com os segundos:

```
// timer.js

const moment = require('moment');
let segundos;

module.exports = {
    iniciar(el) {
        let tempo = moment.duration(el.textContent);
        segundos = tempo.asSeconds();
        setInterval(function() {

        }, 1000);
    }, parar() {

    }
}
```

Assim, dentro do `setInterval`, conseguimos somar 1 segundo, utilizando a variável `segundos`:

```
// timer.js

const moment = require('moment');
let segundos;

module.exports = {
    iniciar(el) {
        let tempo = moment.duration(el.textContent);
        segundos = tempo.asSeconds();
        setInterval(function() {
            segundos++;
        }, 1000);
    }, parar() {

    }
}
```

Só que não podemos colocar esses segundos dentro do elemento HTML, já que não queremos o tempo em segundos, e sim no formato inicial que está na página. Logo, precisamos formatar esse tempo.

Formatando os segundos

Vamos criar uma nova função, com a responsabilidade de formatar, de converter os segundos para o tempo formatado, ela se chamará **`segundosParaTempo`**. Se ela precisa dos segundos para formatá-lo, vamos passá-lo por parâmetro para a função:

```
// timer.js

const moment = require('moment');
let segundos;

module.exports = {
    iniciar(el) {
        let tempo = moment.duration(el.textContent);
        segundos = tempo.asSeconds();
        setInterval(function() {
            segundos++;
        }, 1000);
    }, parar() {

    }, segundosParaTempo(segundos) {

    }
}
```

Infelizmente não há uma função que formate diretamente o tempo, algo como uma função `tempo.format()`, mas dá para contornarmos isso e formatarmos o tempo do jeito que queremos.

O que nós podemos fazer é retornar um novo objeto **`moment`**, que por enquanto é uma data vazia vazio:

```
// timer.js
// mostrando somente a função segundosParaTempo

segundosParaTempo(segundos) {
    return moment();
}
```

Se é um objeto vazio de uma data, podemos colocar qualquer data que quisermos dentro dele. Podemos dizer que o momento começa no início de um dia, ou seja, algo como 00:00:00. Fazemos isso chamando a função **`startOf`**, passando o valor **`'day'`** para ela:

```
// timer.js
// representando somente a função segundosParaTempo

segundosParaTempo(segundos) {
    return moment().startOf('day');
}
```

Mas não estamos interessados no início de um dia e sim no momento que os segundos representam. Para isso, podemos atribuir os segundos à data, através da função **`seconds`**:

```
// timer.js
// representando somente a função segundosParaTempo
```

```
segundosParaTempo(segundos) {
    return moment().startOf('day').seconds(segundos);
}
```

Por fim, agora podemos utilizar a função `format`, especificando o formato em que o objeto deve estar. Fazemos isso chamando a função `format`, passando para ela a máscara `"HH:mm:ss"`:

```
// timer.js
// representando somente a função segundosParaTempo

segundosParaTempo(segundos) {
    return moment().startOf('day').seconds(segundos).format("HH:mm:ss");
}
```

Implementada a função, basta chamá-la na hora de atualizar o conteúdo de texto do elemento:

```
// timer.js

const moment = require('moment');
let segundos;

module.exports = {
    iniciar(el) {
        let tempo = moment.duration(el.textContent);
        segundos = tempo.asSeconds();
        setInterval(function() {
            segundos++;
            // colocando o novo tempo no elemento
            el.textContent = segundosParaTempo(segundos);
        }, 1000);
    }, parar() {

    }, segundosParaTempo(segundos) {
        return moment().startOf('day').seconds(segundos).format("HH:mm:ss");
    }
}
```

Mas a função `segundosParaTempo` é uma função do módulo, logo, dentro da função interna do `setInterval`, ela não consegue ser acessada, já que o escopo da função interna não enxerga para fora da função `setInterval`. Então, se utilizarmos o `this`, também não irá funcionar.

Para manter o mesmo escopo, como estamos utilizando ES6, podemos utilizar uma *arrow function*, para que o escopo do lado de fora da função `setInterval` seja o mesmo do lado de dentro, assim poderemos utilizar o `this`:

```
// timer.js

const moment = require('moment');
let segundos;

module.exports = {
    iniciar(el) {
```

```
            let tempo = moment.duration(el.textContent);
            segundos = tempo.asSeconds();
            setInterval(() => {
                segundos++;
                // colocando o novo tempo no elemento, agora utilizando o this
                el.textContent = this.segundosParaTempo(segundos);
            }, 1000);
        }, parar() {

        }, segundosParaTempo(segundos) {
            return moment().startOf('day').seconds(segundos).format("HH:mm:ss");
        }
    }
```

Agora, ao testar a nossa aplicação e clicar no botão de *play*, o *timer* funciona corretamente! Mas atualmente só conseguimos iniciar o *timer*, ao tentar pará-lo, além do *timer* não parar, o tempo fica acelerado.

Veremos como corrigir isso no próximo capítulo :)