01

Enviando dados para outro processo

Transcrição

Agora que o nosso *timer* já está funcionando, incluindo os botões de *play* e *stop*. Mas ao fechá-lo e abri-lo novamente, o tempo volta para o instante 00:00:00, ou seja, o tempo que passamos estudando em um curso específico não é guardado, pois o *timer* não está persistindo essa informação em nenhum lugar.

E é exatamente isso que faremos neste capítulo, veremos um dos modos de fazer persistência de dados com Electron e com o que o Node nos disponibiliza. Então, salvaremos o tempo corrido no nosso sistema, e quando a aplicação for reaberta, o tempo estará lá salvo, assim poderemos ver quanto tempo já estudamos determinado curso e dar sequência aos estudos.

Enviando os dados do tempo e do curso para outro processo

Queremos salvar o tempo no momento em que ele for parado, e claro, devemos associar esse tempo ao seu curso, pois o tempo de estudo para cada curso será diferente, logo deve haver essa separação, de quanto estudamos e em cada curso.

Para salvar o tempo no momento em que ele for parado, devemos mexer na função `parar()`, do módulo **timer.js**. Dentro dessa função, podemos implementar a lógica de salvar o tempo, mas estaríamos colocando muita responsabilidade dentro do módulo, já que ele só deve controlar o *timer*, e não realizar persistência de dados.

Então, o que vamos fazer é, quando o tempo for parado, nós vamos enviar um evento para o processo principal (**main.js**) e dentro dele salvaremos os dados. Já sabemos o que devemos fazer quando queremos nos comunicarmos com o processo principal, importamos o `ipcRenderer` e utilizamos a sua função `send`, passando o evento. Chamaremos o evento de `curso-parado`:

```
// timer.js

const { ipcRenderer } = require('electron');

// restante do código omitido

module.exports = {
    // função iniciar omitida
    parar() {
        clearInterval(timer);
        ipcRenderer.send('curso-parado');
    }
    // função segundosParaTempo omitida
}
```

Só que além do evento, queremos enviar mais duas coisas: qual o curso que está sendo parado, e o tempo estudado:

```
// timer.js

const { ipcRenderer } = require('electron');

// restante do código omitido
```

```
module.exports = {
    // função iniciar omitida
    parar() {
        clearInterval(timer);
        ipcRenderer.send('curso-parado', curso, tempoEstudado);
    }
    // função segundosParaTempo omitida
}
```

Mas ainda não temos acesso a esses dados, logo, precisamos disponibilizar dentro da função `parar` o curso que está sendo estudado e o seu tempo. O curso está disponível no HTML, na página **index.html**, e possui a classe `curso`. Logo, no **renderer.js**, podemos selecioná-lo e passá-lo por parâmetro para a função `parar`:

```
// renderer.js

let curso = document.querySelector('.curso');

// restante do código comentado

botaoPlay.addEventListener('click', function() {
    if(play) {
        // passando o conteúdo de texto do span para a função parar
        timer.parar(curso.textContent);
        play = false;
    } else {
        timer.iniciar(tempo);
        play = true;
    }
    imgs = imgs.reverse();
    botaoPlay.src = imgs[0];
});
```

Agora, na função `parar`, nós recebemos o `curso` como argumento:

```
// timer.js

const { ipcRenderer } = require('electron');

// restante do código omitido

module.exports = {
    // função iniciar omitida
    parar(curso) {
        clearInterval(timer);
        ipcRenderer.send('curso-parado', curso, tempoEstudado);
    }
    // função segundosParaTempo omitida
}
```

Falta agora o tempo estudado. Nós já temos acesso aos segundos do tempo estudado, através da variável `segundos`, então basta formatá-lo, utilizando a função `segundosParaTempo` do módulo, salvando o resultado na variável `tempoEstudado`:

```
// timer.js

const { ipcRenderer } = require('electron');

// restante do código omitido

module.exports = {
    // função iniciar omitida
    parar(curso) {
        clearInterval(timer);
        let tempoEstudado = this.segundosParaTempo(segundos);
        ipcRenderer.send('curso-parado', curso, tempoEstudado);
    }
    // função segundosParaTempo omitida
}
```

Escutando o evento no processo principal

Resta agora escutar esse evento no processo principal, no **main.js**:

```
ipcMain.on('curso-parado', () => {

});
```

No segundo parâmetro, podemos receber as variáveis `curso` e `tempoEstudado`, além do `event`, que vem sempre por padrão:

```
ipcMain.on('curso-parado', (event, curso, tempoEstudado) => {

});
```

Para verificar se estamos escutando os parâmetros corretos, vamos realizar uma impressão com *template string*:

```
ipcMain.on('curso-parado', (event, curso, tempoEstudado) => {
    console.log(`O curso ${curso} foi estudado por ${tempoEstudado}`);
});
```

Ao testar aplicação, vemos que conseguimos receber os dados com sucesso, restando realizar a sua persistência, que veremos a partir do próximo vídeo!