



06

## Tratamento de erros e notificação do usuário

### Transcrição

Hoje, quando acessamos nossa aplicação sem que nossa API esteja rodando o que acontece? Vemos uma página com apenas o menu com conteúdo em branco. Se abrirmos o console do Chrome vemos informações mais detalhadas.

Deixar a página em branco pode suscitar muitas dúvidas no usuário a respeito do que está acontecendo. É por isso que precisamos exibir uma mensagem para que ele saiba que não foi possível buscar as imagens e que ele pode tentar mais tarde.

Aprendemos que toda promise possui no método `then` dois callbacks, funções que são chamadas em um tempo futuro. O primeiro é o callback de sucesso e o segundo de fracasso. Sendo assim, podemos alterar o componente `Home` desta forma:

```
// alurapic/src/components/home/Home.vue
// código anterior omitido
created() {

  this.service = new FotoService(this.$resource);

  this.service.lista()
    .then(
      fotos => this.fotos = fotos,
      err => {
        console.log(err); // Logando o erro que veio do
```

*server para o desenvolvedor*

```
    this.mensagem = 'Não foi possível obter as fotos.  
Tenta mais tarde.';  
  
  });  
}
```

*// código posterior omitido*

COPIAR CÓDIGO

Se pararmos nosso servidor com a nossa API e recarregarmos nossa aplicação, vemos a mensagem de erro sendo exibida. Funciona, mas a podemos aplicar uma solução ainda melhor.

Veja que em todos os lugares que o método `lista` de `FotoService` for chamado, teremos que logar a mensagem de erro que veio do servidor, o que é uma boa prática para em seguida definir a mensagem que desejamos exibir para o usuário. Essa mensagem é importante, porque não faz sentido mostrar o erro que veio do servidor para a tela dele, é uma informação muito técnica que só geraria mais confusão.

Então, podemos isolar o processo de logar a mensagem de erro do servidor e a mensagem de erro de alto nível, isto é, aquela feita para o usuário no próprio serviço. A vantagem dessa abordagem é que todos os lugares que chamarem o método `lista` não precisaram logar o erro e nem definir sua própria mensagem de erro.

Alterando `FotoService` :

*// alurapic/src/domain/foto/FotoService.js*

```
export default class FotoService {
```

*// código anterior omitido*

```
lista() {  
  
    return this._resource  
        .query()  
        .then(  
            res => res.json(),  
            err => {  
                console.log(err);  
                throw new Error('Não foi possível obter  
as fotos. Tenta mais tarde');  
            }  
        )  
    }  
  
    // código posterior omitido  
}
```

[COPIAR CÓDIGO](#)

Veja que na função `then` que realizamos a conversão dos dados para json, no callback de erro, logamos o erro e lançamos um exceção com nossa mensagem de erro. Quem chamar o método `lista`, além de passar uma função para obter o resultado da operação, passar um callback de erro, terá acesso a mensagem de erro definida pelo serviço. Veja que em nenhum momento quem usa o método `lista` precisará logar o erro e definir a mensagem.

Alterando o componente `Home` :

```
// alurapic/src/components/home/Home.vue  
// código anterior omitido  
  
created() {  
  
    this.service = new FotoService(this.$resource);  
  
    this.service.lista()  
        .then(fotos => this.fotos = fotos, err =>
```

```
this.mensagem = err.message);  
}
```

*// código posterior omitido*

[COPIAR CÓDIGO](#)

Muito mais enxuto e o valor de `this.mensagem` será `err.message`, o texto do objeto de `Error` lançado pelo serviço.

Podemos usar essa estratégia em vários lugares da nossa aplicação, mas por brevidade vamos aplicar em mais um local apenas, na exclusão. Alterando `FotoService`:

```
export default class FotoService {
```

*// código anterior omitido*

```
  apaga(id) {  
  
    return this._resource  
      .delete({ id })  
      .then(null, err => {  
        console.log(err);  
        throw new Error('Não foi possível remover a  
foto. Tente mais tarde');  
      });  
  }
```

*// código posterior omitido*

```
}
```

[COPIAR CÓDIGO](#)

Como a operação de deleção não nos devolve nenhum dado, passamos `null` para `then` e em seguida passamos o callback de erro. Nele usamos a mesma estratégia que utilizamos em `lista`.

Por fim, vamos alterar em `Home` a parte do código que solicita a nossa API a exclusão de uma foto:

*// código anterior omitido*

```
methods: {  
  
  remove(foto) {  
  
    this.service  
      .apaga(foto._id)  
      .then(  
        () => {  
          let indice = this.fotos.indexOf(foto);  
          this.fotos.splice(indice, 1);  
          this.mensagem = 'Foto removida com sucesso'  
        },  
        err => this.mensagem = err.message  
      )  
  }  
  
},
```

*// código posterior omitido*

COPIAR CÓDIGO

Por fim, em nosso `FotoService` é possível escrutinarmos a resposta de erro vinda do servidor, por exemplo, para sabermos o status da requisição entre outras coisas que podem ser levadas em consideração para lançarmos uma mensagem de erro mais apropriada com o erro vindo do servidor varia. Para saber as possíveis propriedades que o objeto de resposta de erro contém basta verificar sua saída no console.

