

Simplificando ainda mais nosso código com promises

Durante esta seção criamos a classe `NegociacaoService`. Esta classe centraliza operações que realizamos com nosso back-end, mais notadamente aquelas que buscam negociações. Ela também serve para encapsular o uso de outra classe que criamos, a `HttpService`. Esta última, encapsula a complexidade de se realizar requisições Ajax devolvendo uma promise para determinadas operações.

Vejamos o código de `HttpService` mais uma vez:

```
class HttpService {

  get(url) {

    return new Promise((resolve, reject) => {

      let xhr = new XMLHttpRequest();

      xhr.open('GET', url);

      xhr.onreadystatechange = () => {

        if(xhr.readyState == 4) {

          if(xhr.status == 200) {

            resolve(JSON.parse(xhr.responseText));
          } else {

            reject(xhr.responseText);
          }
        }
      };

      xhr.send();

    });
  }

  post(url, dado) {

    return new Promise((resolve, reject) => {

      let xhr = new XMLHttpRequest();
      xhr.open("POST", url, true);
      xhr.setRequestHeader("Content-type", "application/json");
      xhr.onreadystatechange = () => {

        if (xhr.readyState == 4) {

          if (xhr.status == 200) {
```

```

        resolve(JSON.parse(xhr.responseText));
    } else {

        reject(xhr.responseText);
    }
}
});
xhr.send(JSON.stringify(dado)); // usando JSON.stringify para converter objeto
});
}
}

```

Todos os métodos `get` e `post` retornam uma promise. Até ai tudo bem, nenhuma novidade.

Agora, vejamos o código da classe `NegociacaoService` que usa `HttpService` :

```

class NegociacaoService {

    constructor() {

        this._http = new HttpService();
    }

    obterNegociacoesDaSemana() {

        return new Promise((resolve, reject) => {

            this._http
                .get('negociacoes/semana')
                .then(negociacoes => {
                    console.log(negociacoes);
                    resolve(negociacoes.map(objeto => new Negociacao(new Date(objeto.data), obje
                ))
                .catch(erro => {
                    console.log(erro);
                    reject('Não foi possível obter as negociações da semana');
                });
        });
    }

    obterNegociacoesDaSemanaAnterior() {

        return new Promise((resolve, reject) => {

            this._http
                .get('negociacoes/anterior')
                .then(negociacoes => {
                    console.log(negociacoes);
                    resolve(negociacoes.map(objeto => new Negociacao(new Date(objeto.data), obje
                ))
                .catch(erro => {
                    console.log(erro);
                    reject('Não foi possível obter as negociações da semana anterior');
                });
        });
    }
}

```

```

    });
  });
}

obterNegociacoesDaSemanaRetrasada() {

  return new Promise((resolve, reject) => {

    this._http
      .get('negociacoes/retrasada')
      .then(negociacoes => {
        console.log(negociacoes);
        resolve(negociacoes.map(objeto => new Negociacao(new Date(objeto.data), obje
      ))
      .catch(erro => {
        console.log(erro);
        reject('Não foi possível obter as negociações da semana retrasada');
      });
  });
}
}

```

Veja que seus métodos também devolvem uma promise. Mas espere um pouco! Se a classe `NegociacaoService` já recebe o resultado de `HttpService` que é uma promise, porque no lugar de criarmos uma nova promise, não fazemos com que os métodos de `NegociacaoService` retorne a promise de `HttpService`? Sim, isso é possível!

O primeiro passo, é removermos o `new Promise((resolve, reject) => {})` de todos os métodos de `NegociacaoService`:

```

class NegociacaoService {

  constructor() {

    this._http = new HttpService();
  }

  obterNegociacoesDaSemana() {

    this._http
      .get('negociacoes/semana')
      .then(negociacoes => {
        console.log(negociacoes);
        resolve(negociacoes.map(objeto => new Negociacao(new Date(objeto.data), obje
      ))
      .catch(erro => {
        console.log(erro);
        reject('Não foi possível obter as negociações da semana');
      });
  }

  obterNegociacoesDaSemanaAnterior() {

```

```

        this._http
            .get('negociacoes/anterior')
            .then(negociacoes => {
                console.log(negociacoes);
                resolve(negociacoes.map(objeto => new Negociacao(new Date(objeto.data), obje
            })
            .catch(erro => {
                console.log(erro);
                reject('Não foi possível obter as negociações da semana anterior');
            });
    }

    obterNegociacoesDaSemanaRetrasada() {

```

```

        this._http
            .get('negociacoes/retrasada')
            .then(negociacoes => {
                console.log(negociacoes);
                resolve(negociacoes.map(objeto => new Negociacao(new Date(objeto.data), obje
            })
            .catch(erro => {
                console.log(erro);
                reject('Não foi possível obter as negociações da semana retrasada');
            });
    }
}

```

O próximo passo é usarmos a instrução `return` na frente da chamada de `this._http.get`. Como o método retorna uma promise, o que estamos fazendo é retornar esta promise nos métodos de `NegociacaoService`:

```

class NegociacaoService {

    constructor() {

        this._http = new HttpService();
    }

    obterNegociacoesDaSemana() {

        return this._http
            .get('negociacoes/semana')
            .then(negociacoes => {
                console.log(negociacoes);
                resolve(negociacoes.map(objeto => new Negociacao(new Date(objeto.data), obje

```

```

    })
    .catch(erro => {
      console.log(erro);
      reject('Não foi possível obter as negociações da semana');
    });
  }

  obterNegociacoesDaSemanaAnterior() {

    return this._http
      .get('negociacoes/anterior')
      .then(negociacoes => {
        console.log(negociacoes);
        resolve(negociacoes.map(objeto => new Negociacao(new Date(objeto.data), obje
      ))
      .catch(erro => {
        console.log(erro);
        reject('Não foi possível obter as negociações da semana anterior');
      });
  }

  obterNegociacoesDaSemanaRetrasada() {

    return this._http
      .get('negociacoes/retrasada')
      .then(negociacoes => {
        console.log(negociacoes);
        resolve(negociacoes.map(objeto => new Negociacao(new Date(objeto.data), obje
      ))
      .catch(erro => {
        console.log(erro);
        reject('Não foi possível obter as negociações da semana retrasada');
      });
  }
}

```

Como não estamos mais retornando `return new Promise((resolve, reject) => {})`, não temos mais as funções `resolve` e `reject` para passarmos o resultado e o erro, caso exista. E agora?

A ideia é a seguinte, se uma função `then` possui um retorno, este retorno é acessível para quem encadear uma nova chamada à função `then`. Sendo assim, onde há `resolve` trocaremos por um `return`. Mas cuidado, não esqueça de remover também os `()` do `resolve`!

```

class NegociacaoService {

  constructor() {

```

```
this._http = new HttpService();
}

obterNegociacoesDaSemana() {

    return this._http
        .get('negociacoes/semana')
        .then(negociacoes => {

            console.log(negociacoes);

            return negociacoes.map(objeto => new Negociacao(new Date(objeto.data), obje
        ))
        .catch(erro => {
            console.log(erro);
            reject('Não foi possível obter as negociações da semana');
        });

}

obterNegociacoesDaSemanaAnterior() {

    return this._http
        .get('negociacoes/anterior')
        .then(negociacoes => {

            console.log(negociacoes);

            return negociacoes.map(objeto => new Negociacao(new Date(objeto.data), obje
        ))
        .catch(erro => {
            console.log(erro);
            reject('Não foi possível obter as negociações da semana anterior');
        });

}

obterNegociacoesDaSemanaRetrasada() {

    return this._http
        .get('negociacoes/retrasada')
        .then(negociacoes => {

            console.log(negociacoes);

            return negociacoes.map(objeto => new Negociacao(new Date(objeto.data), obje
        ))
        .catch(erro => {
            console.log(erro);
```

```
        reject('Não foi possível obter as negociações da semana retrasada');
    });
}
}
```

Do jeito que está, podemos fazer algo como:

```
let service = new NegociacaoService();
service
    .obterNegociacoesDaSemana()
    .then(negociacoes => /* faz alguma coisa com as negociações */)
```

Veja que é exatamente a maneira que já utilizávamos antes. O que mudou foi apenas a criação de uma promise extra na definição dos métodos.

Mas ainda não acabou! E se um erro acontecer? No lugar de usarmos `reject`, lançamos uma exceção em seu lugar:

```
class NegociacaoService {

    constructor() {

        this._http = new HttpService();
    }

    obterNegociacoesDaSemana() {

        return this._http
            .get('negociacoes/semana')
            .then(negociacoes => {

                console.log(negociacoes);

                return negociacoes.map(objeto => new Negociacao(new Date(objeto.data), objeto.qi
            ))
            .catch(erro => {
                console.log(erro);
                throw new Error('Não foi possível obter as negociações da semana');
            });
    }

    obterNegociacoesDaSemanaAnterior() {

        return this._http
            .get('negociacoes/anterior')
            .then(negociacoes => {

                console.log(negociacoes);

                return negociacoes.map(objeto => new Negociacao(new Date(objeto.data), objeto.qi
            ))
            .catch(erro => {
```

```
        console.log(erro);
        throw new Error('Não foi possível obter as negociações da semana anterior');
    });
}

obterNegociacoesDaSemanaRetrasada() {

    return this._http
        .get('negociacoes/retrasada')
        .then(negociacoes => {

            console.log(negociacoes);

            return negociacoes.map(objeto => new Negociacao(new Date(objeto.data), objeto.qi
        ))
        .catch(erro => {
            console.log(erro);
            throw new Error('Não foi possível obter as negociações da semana retrasada');
        });

    }
}
```

Veja que, com essa alteração, poupamos algumas linhas de código e tornamos o código da classe `NegociacaoService` mais legível. É claro, isso só funciona porque `HttpService` devolve uma promise. Se não devolvesse, `NegociacaoService` precisaria retornar uma promise, como havíamos feito.