**Observables**

- Observables são streams de dados unidirecionais, que emitem notificações sempre que ocorre uma mudança em um de seus itens, nos permitindo reconhecer essa mudança e reagir à ela.

- Essas streams se baseam em um modelo de **PubSub** (Publisher/Subscriber), onde um “publicador” publica dados e um “inscrito” fica ouvindo pelas publicações.

**RxJS (Reactive Extensions Library for JS)**

- Biblioteca para programação reativa utilizando Observables, tornando mais fácil a composição de atividades assíncronas ou códigos baseados em callback.

- É daqui que os Observables vêm.

**Subscribe()**

- Método de um observable que cria a “inscrição”. Ele precisa estar presente (implicita ou explicitamente) para o observable ser executado.

- Next, error, complete

**Unsubscribe()**

- Faz com que nos desinscrevamos de um observable.

- É Importante se desinscrever para encerrar a stream de dados, pois se não fizermos isso ela continuará ativa e consumirá recursos da máquina, deixando o site lento.

- Podemos nos desinscrever utilizando operadores RxJS no método **pipe()**, utilizando o método **unsubscribe()** de uma **Subscription**, ou utilizando o **async pipe**.

- Podemos criar uma Subscription geral, adicionar nela todos as Subscriptions criadas através do método **add()** e então no **ngOnDestroy** encerrar todas as inscrições de uma só vez.

**Pipe()**

- Método de um Observable. Através dele conseguimos construir o que chamamos de **receipts** (receitas) que servem para manipular de alguma forma o dado ou o comportamento de um Observable. As receipts são compostas por operadores RxJS.

- Como visto anteriormente, um observable só é executado se o método subscribe() se faz presente, implícita ou explícitamente. Como o pipe() só é executado se o Observable for executado, a regra é a mesma aqui.

**Alguns operadores RxJS**

- take(n: number): recebe **n** respostas do observable e então encerra a stream de dados (realiza o unsubscribe() implicitamente).

- of(p: any): retorna um observable que retorna os dados **p**. Cada dado passado por parâmetro representa uma emissão de valores.

- map(p: any[] | object): igual o método .map() dos arrays. Pode receber um array e fazer um map sobre ele, ou então um objeto e filtrar seus dados, por exemplo.

- tap(p: Function): executa a operação passada por parâmetro antes que os dados cheguem no subscribe, nos permitindo assim criar efeitos colaterais.

- skip(n: number): ignora os primeiros **n** valores emitidos pelo observable.