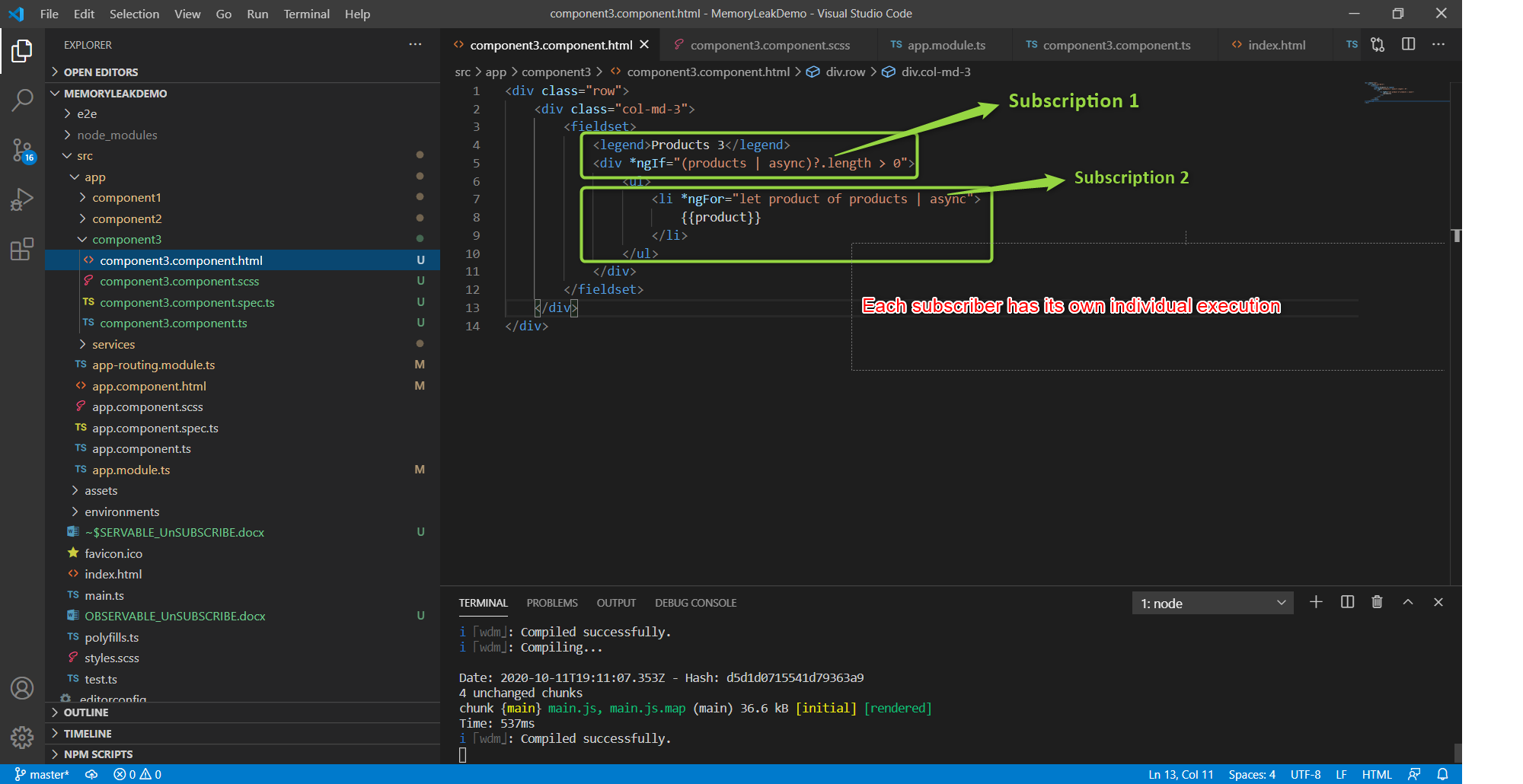
**Unsubscribe from Observables**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

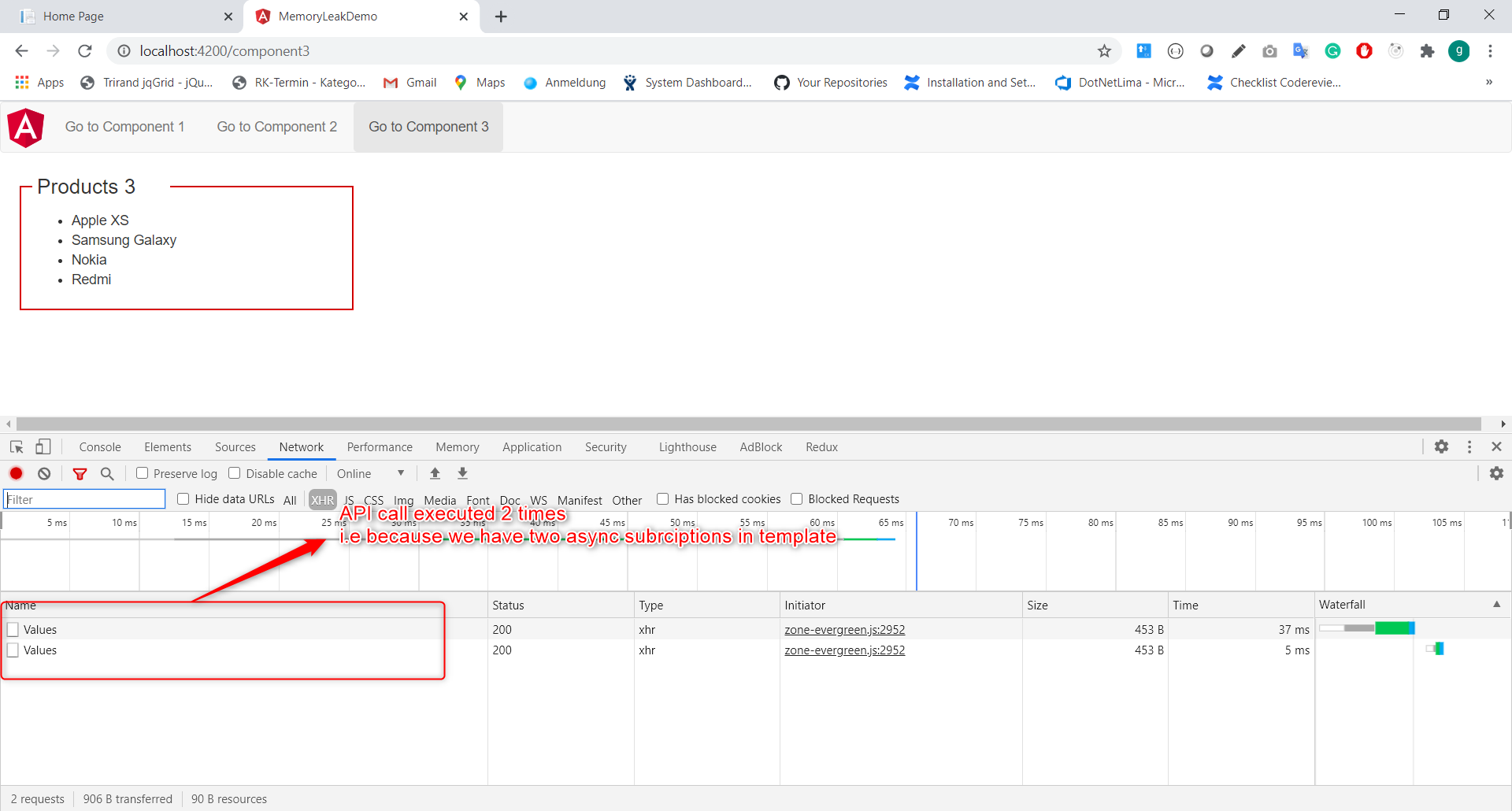
An Observable by default is unicast. Unicasting means that each subscribed observer owns an independent execution of the Observable**.**

**Problem 1**

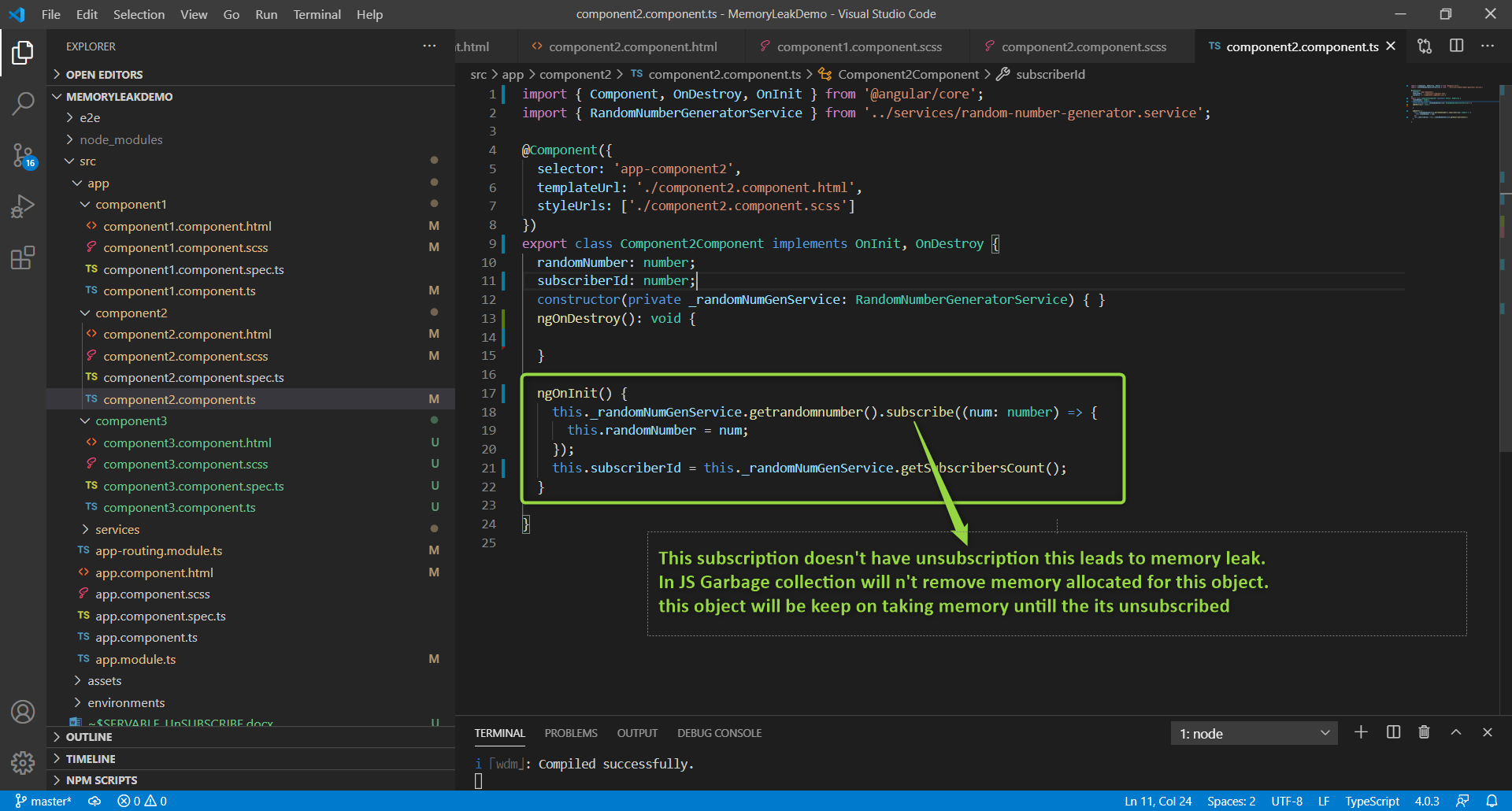
**In template used two Async subscriptions on an observable causes two times execution.**

****

**Results: In the current scenario we must not use multiple subscribers on same observable.**

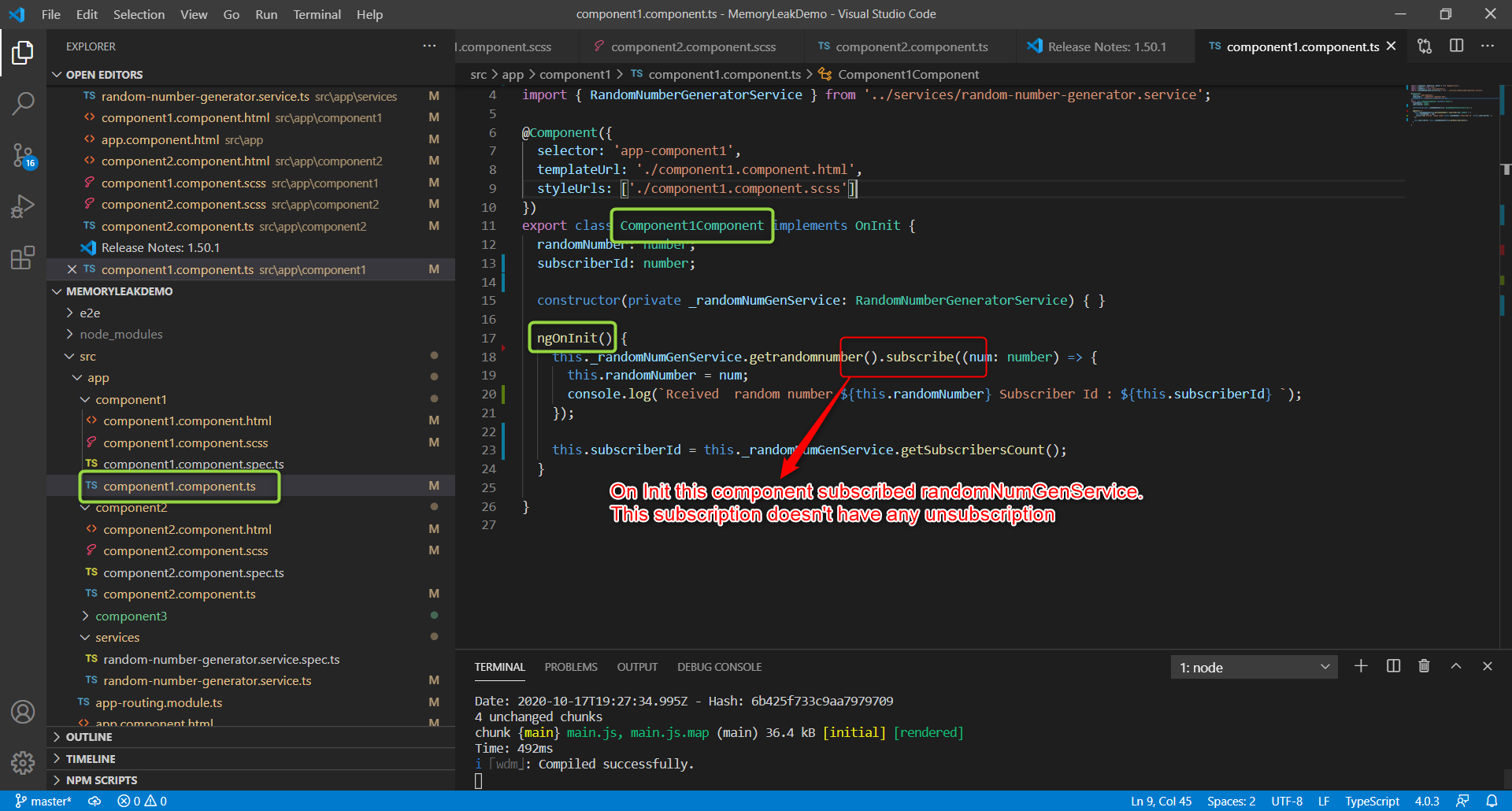
****

**Problem 2**

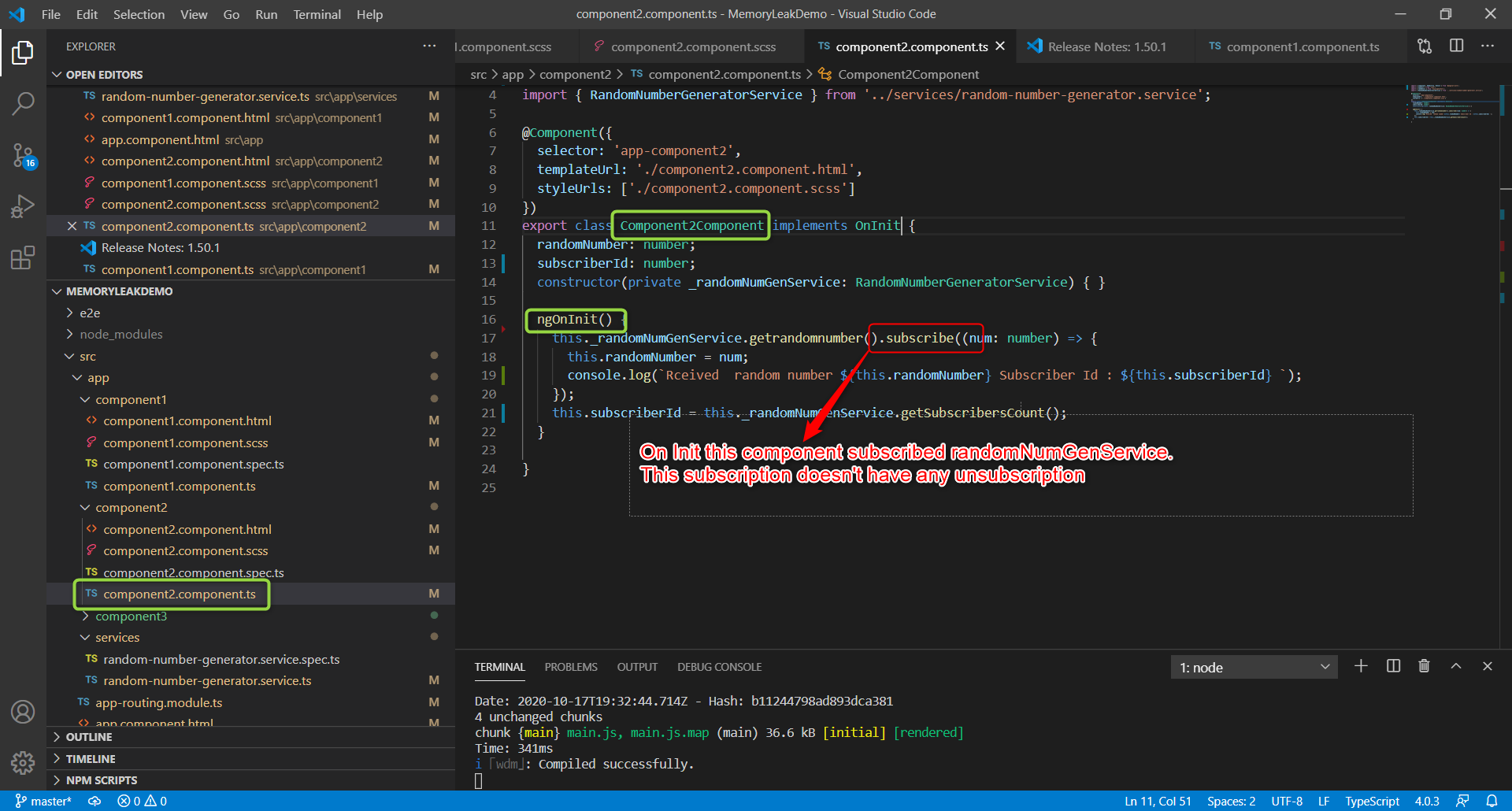
****

**I have two components, subscribing to an observable which emits a number on every 2sec.**

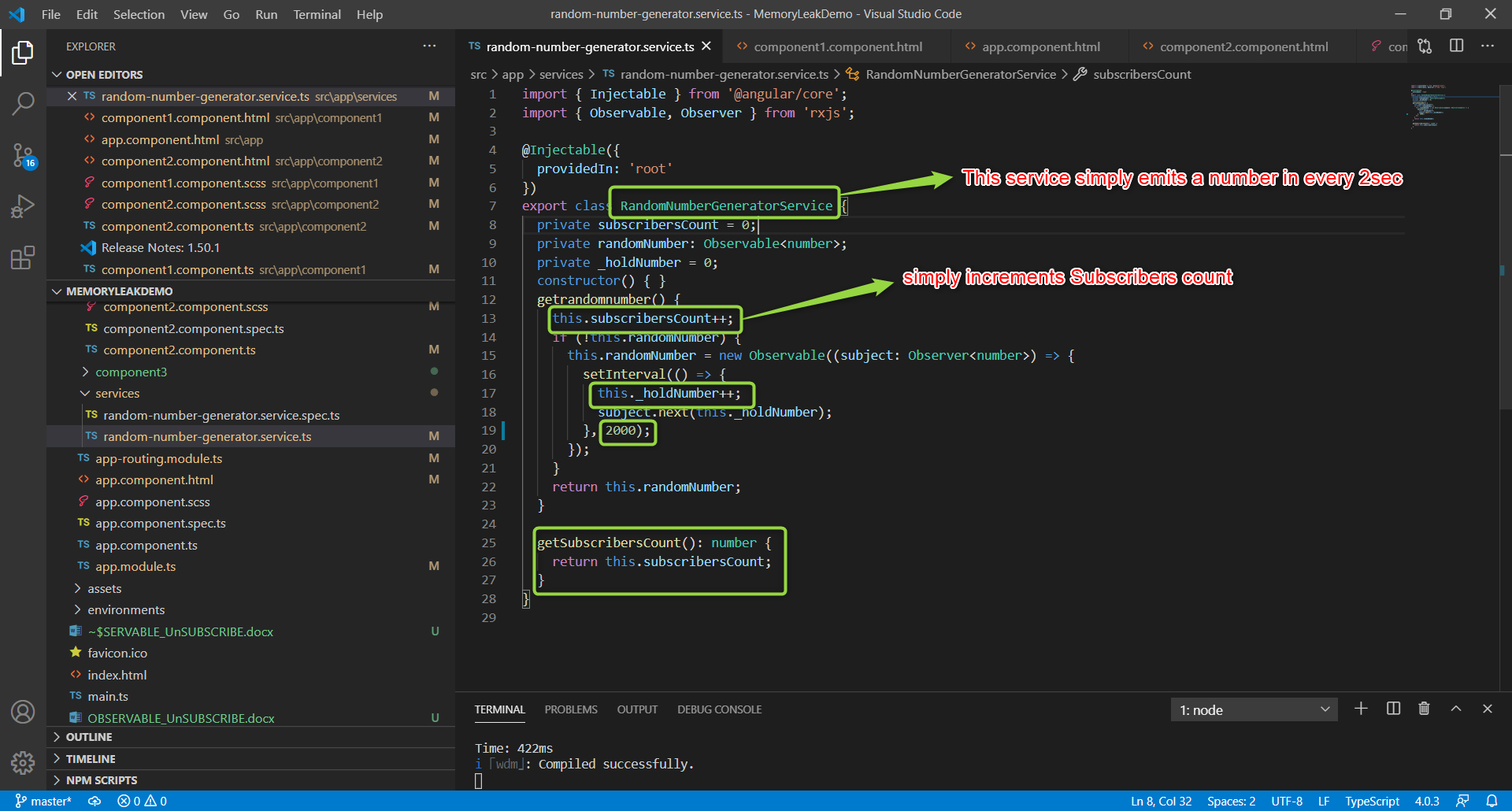
**Component1 code**

****

**Component2 code**

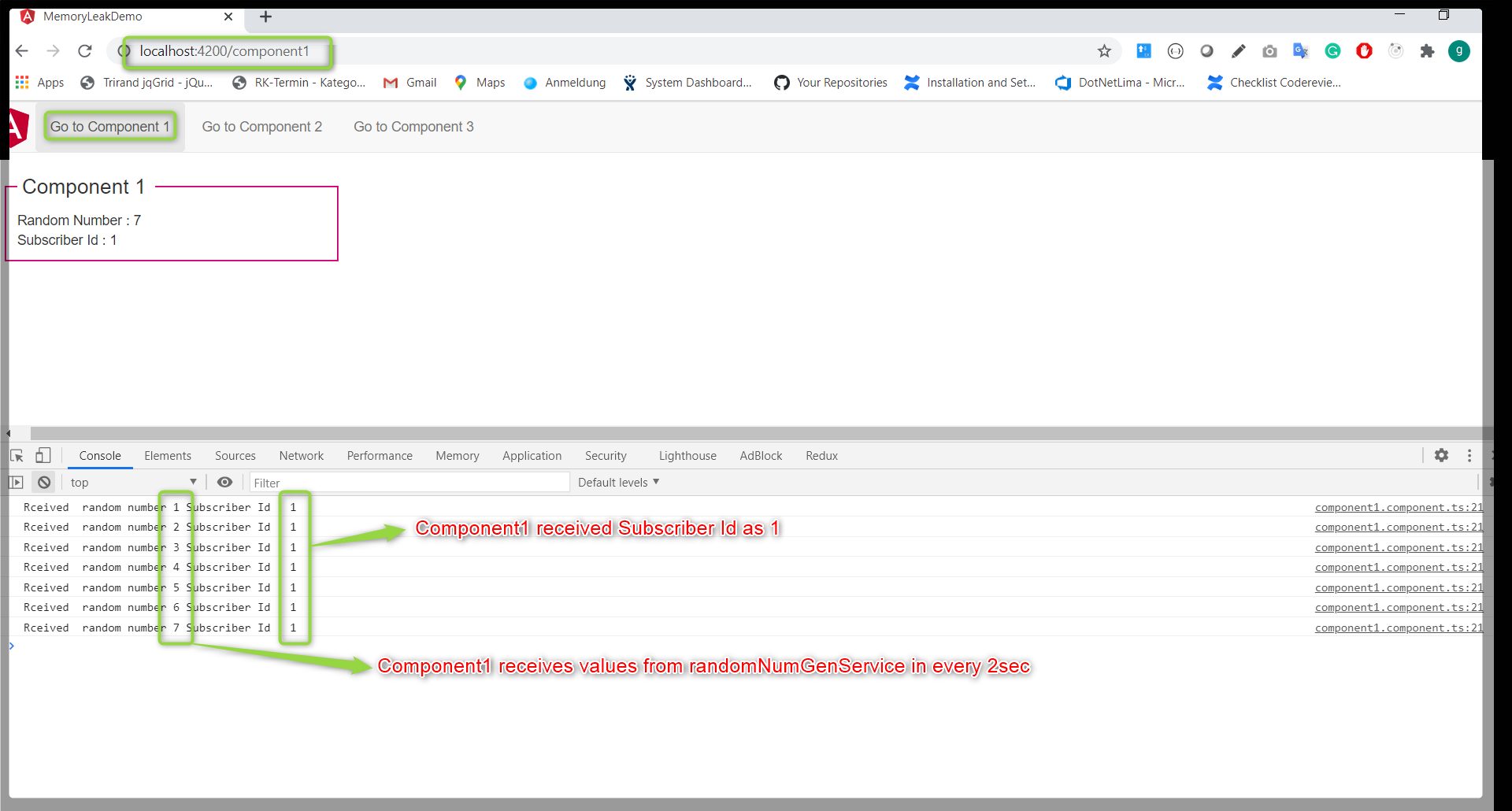
****

**Random Number Generator Service Code**

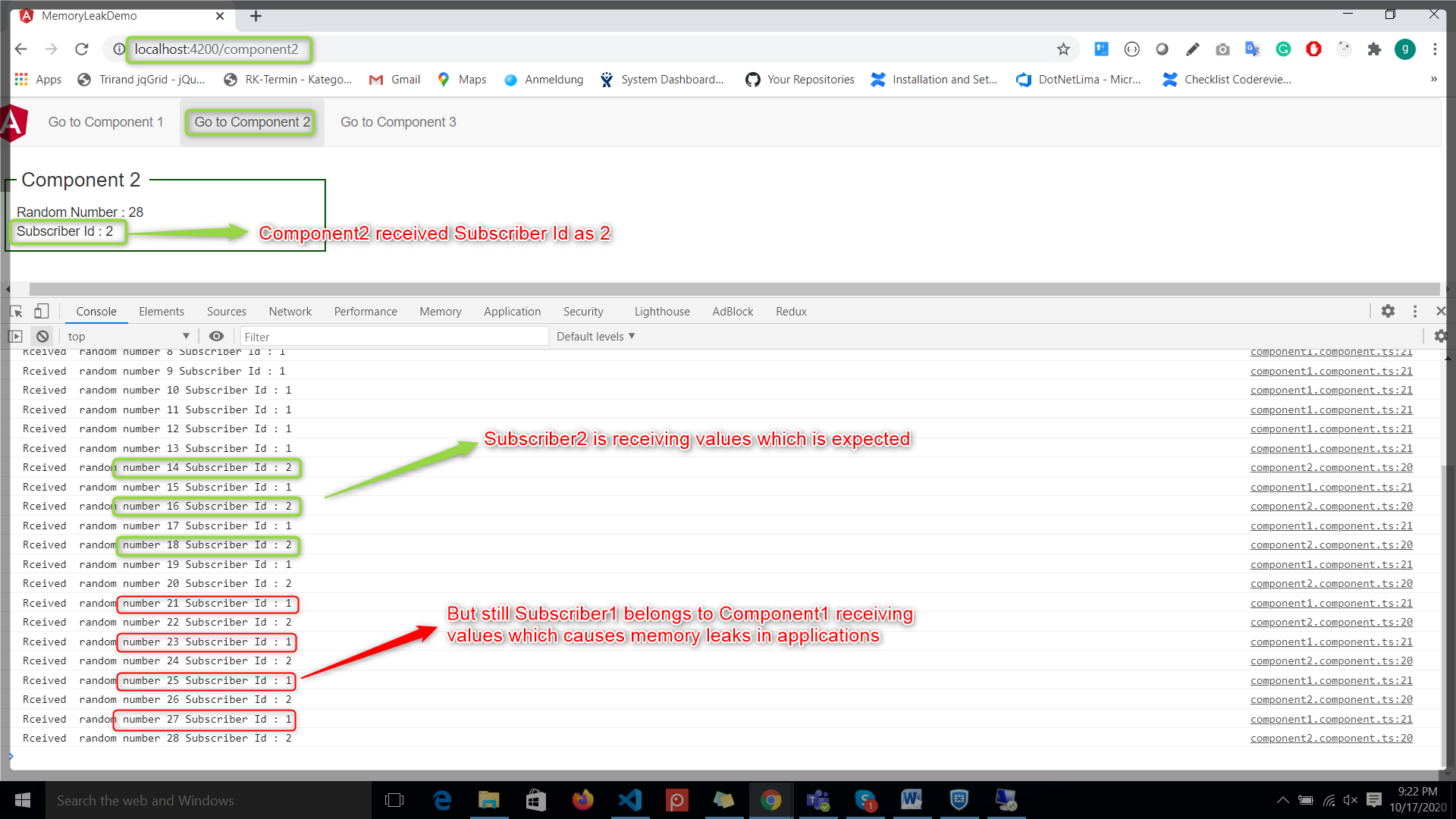
****

**Result =>**

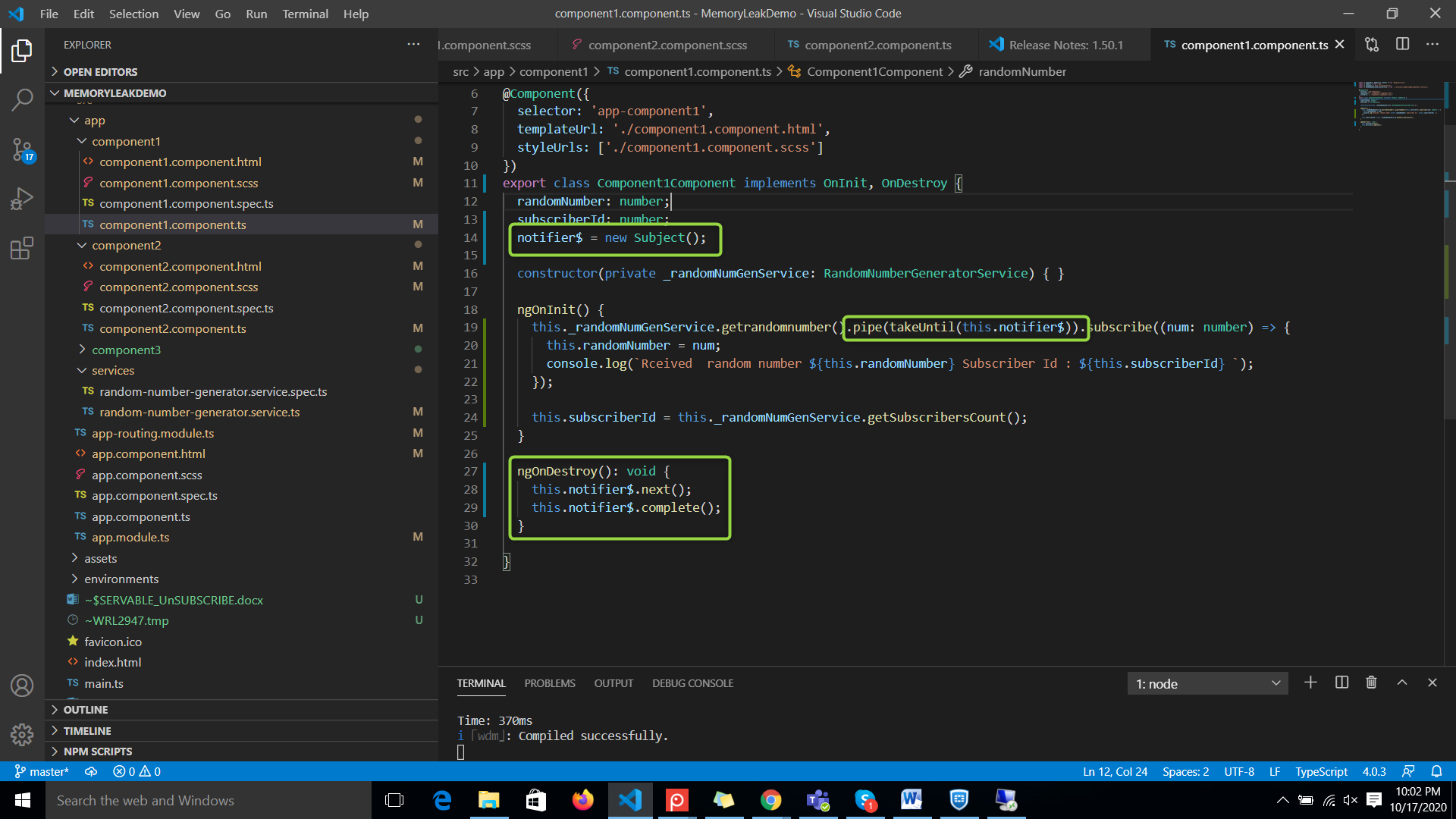
**User currently viewing Component1**

****

**And User navigates to Component2**

****

**Using takeUntil which unscribes when component is destroyed resolves the memory leaks problems.**

****

**Best practices on unsubscribing from observables.**

1. **Using Async Pipe (In Templates)**
2. **takeUntil (notifier) RxJS operator**
3. **https://www.npmjs.com/package/@ngneat/until-destroy**

**The takeUntil() solution is great but unfortunately it comes also with a couple of disadvantages**

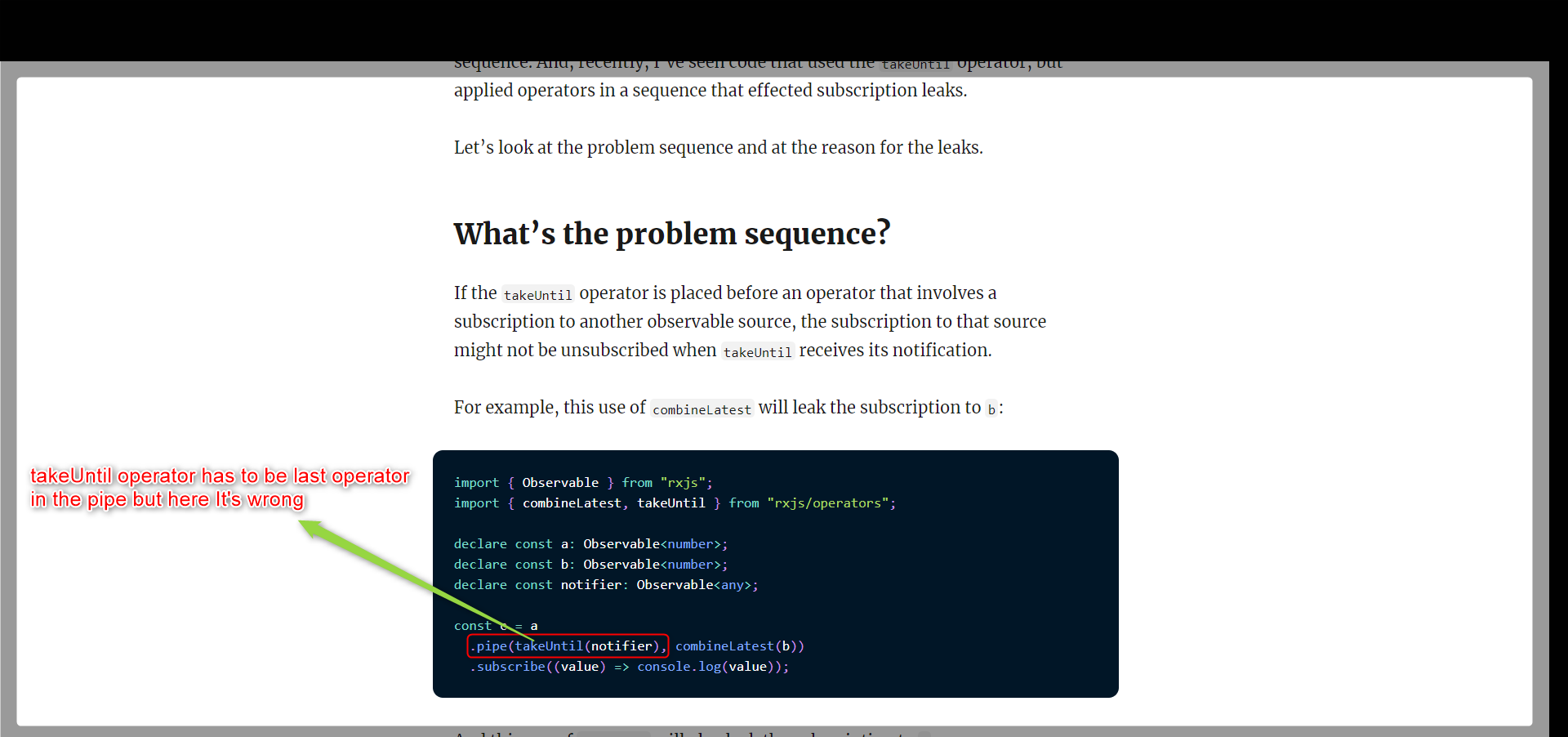
**1. Forgetting to implement the OnDestroy interface itself**

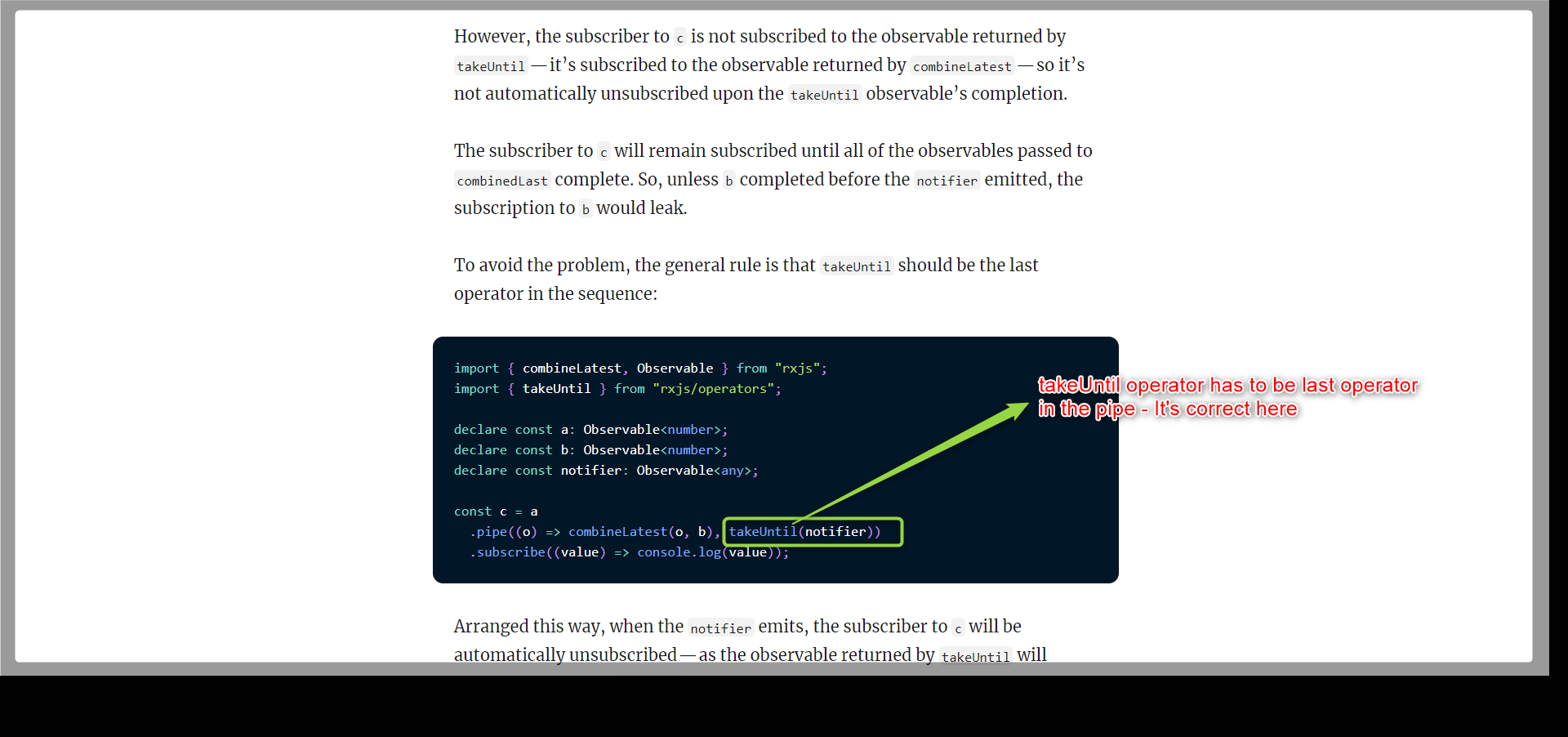
**2. Forgetting to call .next() and .complete() methods in the ngOnDestroy implementation (leaving it empty)**

**A possible solution would be to implement (or find if it exists) a custom tslint rule which will check for missing (or empty) ngOnDestroy() methods in every component which can also be problematic because not every component uses subscriptions…**

**TS rule is already available => https://github.com/cartant/rxjs-tslint-rules#rules**

**We should not forget about the fact that the takeUntil operator has to be last operator in the pipe (usually) to prevent situation when subsequent operator return additional observables which can prevent clean up.**

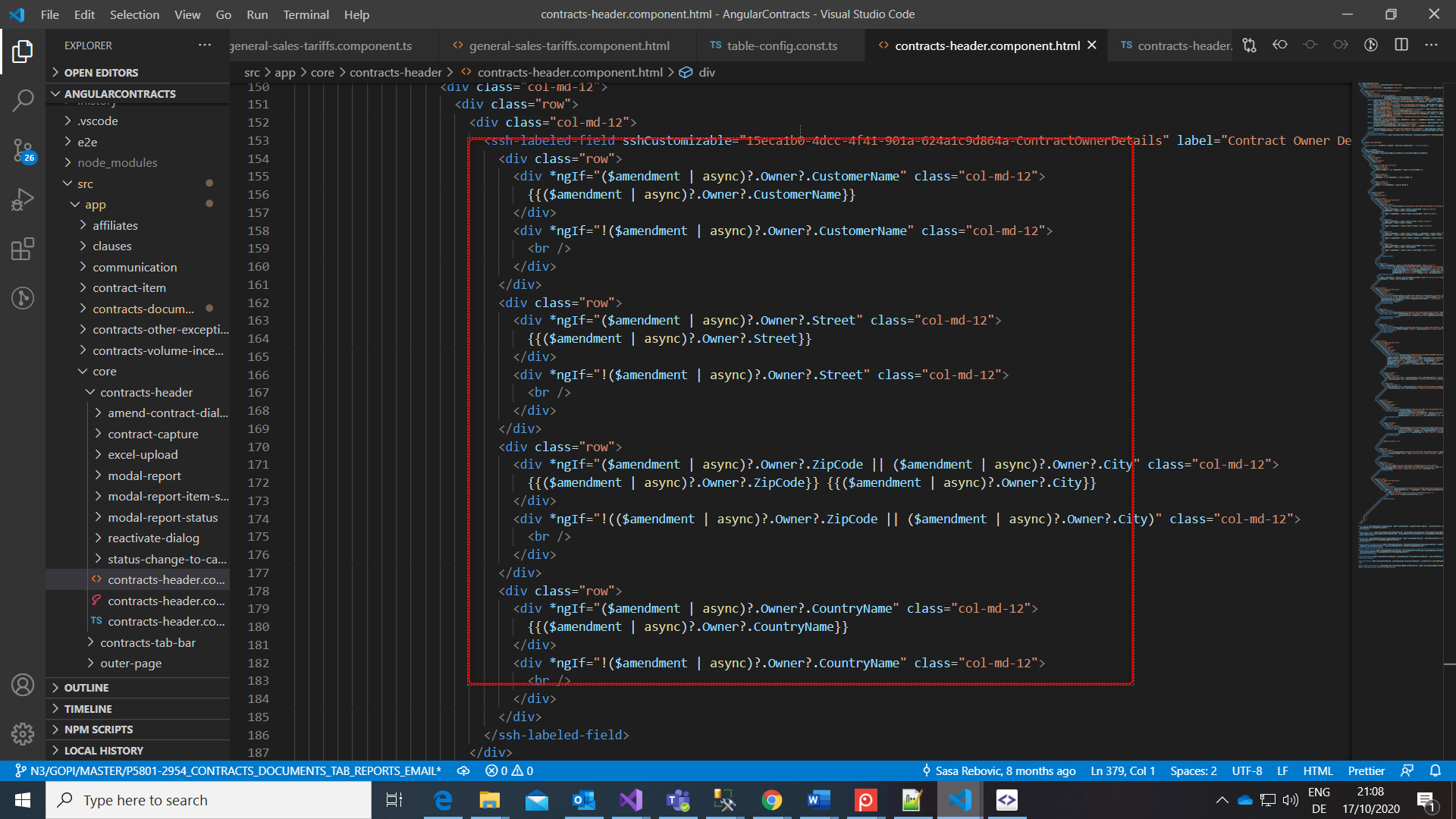
****

****

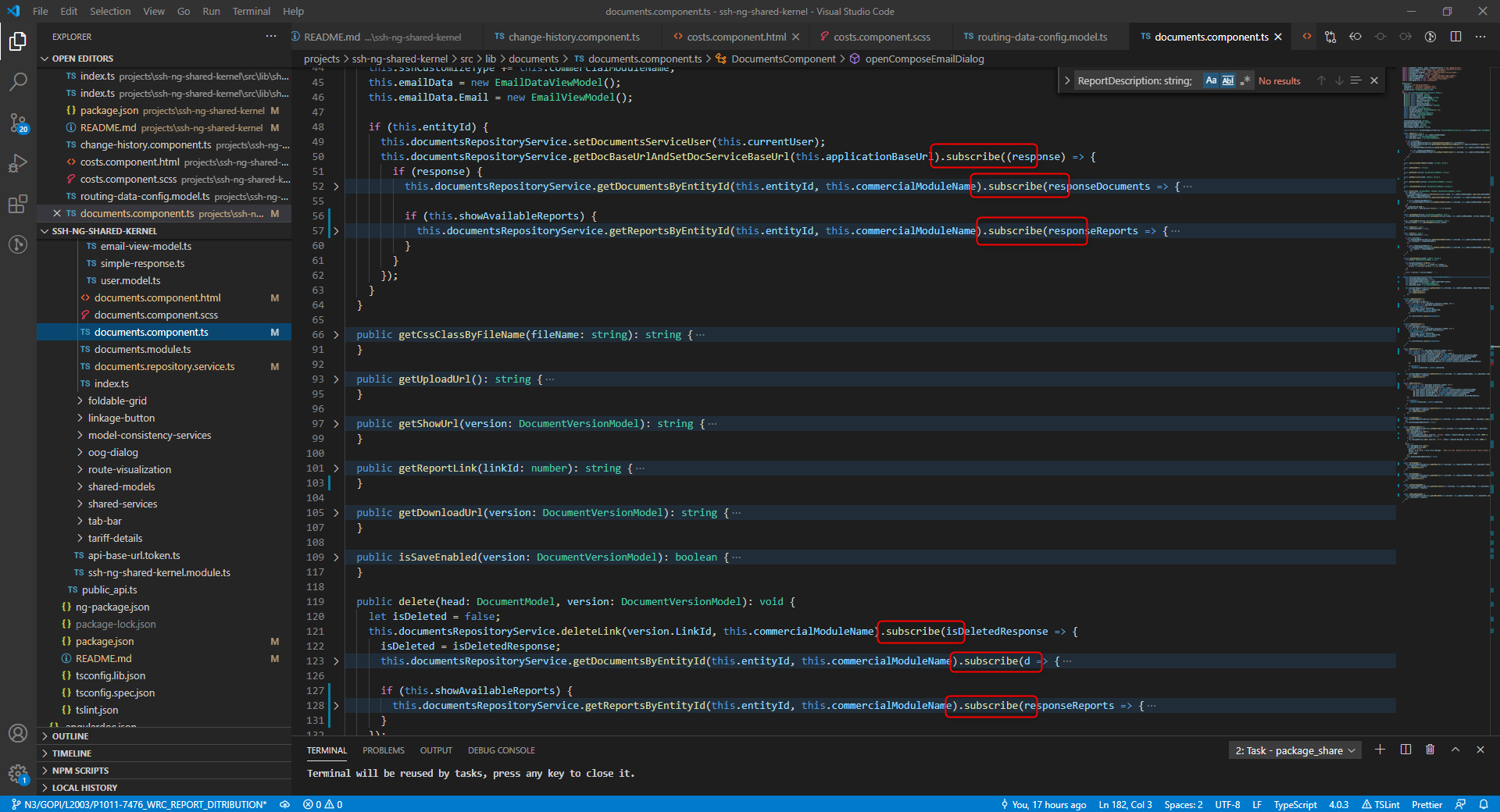
**Reference article =>** [**https://ncjamieson.com/avoiding-takeuntil-leaks/**](https://ncjamieson.com/avoiding-takeuntil-leaks/)

**OBSERVATIONS**

**Multiple subscriptions on same observable.**

****

**Subscriptions without unsubscribing.**

****