- Special case when the counter reaches 0 is not properly handled.

This is correct, as mentioned in the code notes, no exceptions are being treated at this point. Simply because no expected behavior has been specified.

This is a good example of the kind of questions to be raised at a practical time, like for example an overview / design meeting, or standup.

Code Notes:

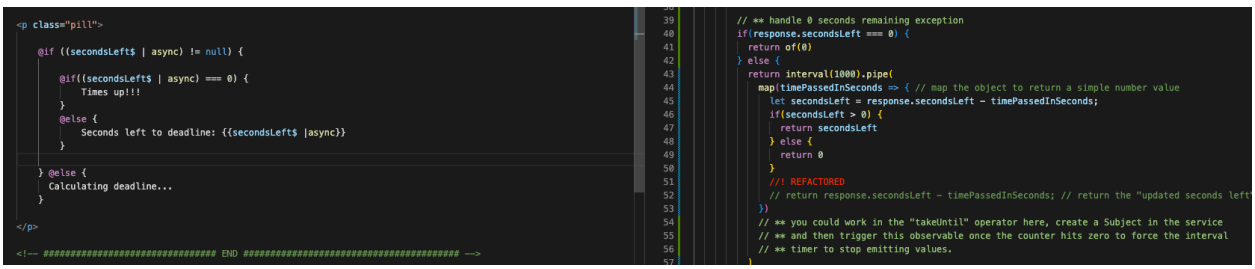
```
getSecondsLeftToDeadline(): Observable<number> {

  // ** What the mock API server return statement would look like in it's basic form
  // ** and by "basic" I refer to it not tending to error handling, for example, or
  // ** dealing with exceptions etc.
```

Because there are many ways to implement some version of logic to show that the time has passed, consider this as just one way to do it. Stop showing the countdown, and show a relevant message.

Off course there are many more complex action one can trigger at this point, and every one of those options may well effect the final code and solution.

For this refactor, you will now notice handling the “zero seconds remaining” condition in the service, and HTML files:

1. at API returning 0 seconds remaining
2. when the counter reaches zero.
3. Additional if clause in the HTML file to manage values of null, 0, or greater than 0, with a message that indicates when the time is up.



• OnPush change detection is missing.

On push change detection is not missing. It is simply not required as per the question / specifications. This also steps into the realm of comprehension, perception, and possibly "splitting hairs" a little.

So, to highlight my perspective:

The question reads… "Write **an Angular component** (and any other Angular classes/functions if needed) **that will retrieve data from backend** and will display "Seconds left to deadline: X" countdown timer (X should be updated every second)."

The **bolded** and underlined text implies a single component that retrieves the data from the backend (nest practices, by using a service).

Making change detection, and the last sentence aimed at it's consideration for performance optimized code, redundant: "Please take your time to write performance optimized code." Although, this now changes focus toward how performance in delivering the data to the view (i.e. focussing on delivering the correct observable logic and reactive design). In this case, focusing on using the asynchronous pipe, for example.

Change detection only comes into play when passing reactive data between parent and child components. Implying the requirement for multiple components:

1. the parent component, activating and retrieving the data from the API.
2. And then the parent component then passes the data on the child component (that uses an input decorator for the data, and implements the change detection strategy "OnPush" to manage change detection efficiently).

So in short, both solutions are theoretically correct. And one could debate this further by going into detail about when which solution makes more sense.

This is simply an explanation to argue why change detection is not present, and when it would be.

If the question called for multiple components, or parent child data strategies, it certainly would be included in the solution.

And if you want me to present a version of it, don't hesitate to ask (it would probably have been faster to implement than it took me to create this feedback page!).

- I would expect standalone components in such modern code.

An astute observation. Perhaps a debatable / opinion based topic in itself, depending on personal perception, preference, or even architectural, project, or team coding standards:

“Stand alone components” are not taking over Angular Modules. In fact, it is in no way “better” either, but simply a different way of approaching the same concept. I.e. Managing imports better.

So it stilll comes down to the individual skill and understanding of imports and how to manage them. And both have their own drawbacks and advantages.

In general, Angular is made up of a few complex concepts. Modules, and Observables being two of those.

The Angular team is actively trying to make the learning curve shorter for new developers to join the community.

Hence, signals and stand alone components.

Yes there are many perks to signals (above say, primitive values), and stand alone components skips learning about modules and make it faster to get to a point where someone can deliver code… But they are not meant to replace modules (or observables in the case of signals), rather, they simply present a less complex way to manage imports (in small projects), and present reactive data (with the long term goal of a “Zoneless” Angular, with very little concern for change detection strategies… Eventually).

I have no problem with the perceived complexity modules bring. In fact, I fully embrace the clear manageability it provides when writing and managing the code and the application. And prefer importing something once, in stead of every time a component needs them.

In fact, I feel modules especially make sense in “large” complex applications.

That being said, I’m not agains stand alone components. For example, here are three reasons I would choose stand alone components above modules:

1. company / project standards
2. trying to accommodate junior / new generation developers (ties in with company and project standards - it’s been stipulated by management as a requirement)
3. very small application (Maybe. Because if you know how modules work, and you know how to use the CLI, does it really take longer to use modules? I’d argue it takes me less time.)