What is change detection

https://angular.io/guide/change-detection

Change detection is the process through which Angular checks to see whether your application state has changed, and if any DOM needs to be updated.

Angular runs its change detection mechanism periodically so that changes to the data model are reflected in an app's view.

At a high level, Angular walks your components from top to bottom, looking for changes.

On every change detection cycle, Angular synchronously:

- **Evaluates all template expressions in all components**, unless specified otherwise, based on that each component's detection strategy

- **Executes** the `ngDoCheck`, `ngAfterContentChecked`, `ngAfterViewChecked`, and `ngOnChanges` **lifecycle hooks**.

A single slow computation within a template or a lifecycle hook can slow down the entire change detection process because Angular runs the computations sequentially.

Optimizing slow computations

Optimizing the underlying algorithm

Don't use inpure heavy computation methods in template

```
<div {{ inpureHeavyComputation(data) }}>
```

Use pure pipes for caching (impure pipes are called on each detection cycle)

```
<div {{  data | customPurePipe }}>
```

https://angular.io/guide/pipes

Minimize computations in lifecycle hooks methods

```
@Input() value: number;
let oldValue: number = 0;
ngDoCheck() {
   If (oldValue !== value) {
      // add functionality here
      oldValue = value;
   }
}
```

Use memoization for caching pure methods

```
@ memoize
pureheavyComputation(data) {
...
}
```

https://www.npmjs.com/package/memoizee

https://medium.com/angular-in-depth/how-to-improve-angular-performance-by-just-adding-just-8-characters-877bde708ddd

npm i memoizee

```
import * as memoizee from 'memoizee';


// create the @memoize custom decorator
export function memoize() {
    return function(target, key, descriptor) {
        const oldFunction = descriptor.value;
        const newFunction = memoizee(oldFunction);
        descriptor.value= function () {
            return newFunction.apply(this, arguments);
        };
    };
};
```

Use Zone.js and ngZones for further optimization

https://angular.io/guide/change-detection-zone-pollution

https://angular.io/guide/zone

- **Zone.js is an execution context that persist across async tasks and a signaling mechanism, that Angular uses to detect when an application state might have changed**. It captures asynchronous operations like `setTimeout`, network requests, and event listeners. Angular schedules change detection based on signals from Zone.js

- A Zone is an execution context that persists across async tasks

https://www.w3schools.com/js/js_this.asp

https://www.youtube.com/watch?v=3IqtmUscE_U&t=150s&ab_channel=ng-conf

- There are cases in which scheduled tasks or microtasks don't make any changes in the data model, which makes running change detection unnecessary. Common examples are:
  - `requestAnimationFrame`, `setTimeout` or `setInterval`
  - Task or microtask scheduling by third-party libraries

```
constructor(private ngZone: NgZone) {}
ngOnInit() {
    this.ngZone.runOutsideAngular(() => {
        // this code will not trigger change detection
    });
}
```

Skipping component subtrees

- **JavaScript, by default, uses mutable data structures** that you can reference from multiple different components. **Angular runs change detection over your entire component tree** to make sure that the most up-to-date state of your data structures is reflected in the DOM.
- Change detection is sufficiently fast for most applications. However, **when an application has an especially large component tree, running change detection across the whole application can cause performance issues**. You can address this by configuring change detection to only run on a subset of the component tree.
- If you are confident that a part of the application is not affected by a state change, you can use OnPush to skip change detection in an entire component subtree.

Using OnPush

- **OnPush change detection instructs Angular to run change detection for a component subtree only** when:
- The root component of the subtree receives new inputs as the result of a template binding. Angular compares the current and past value of the input with ==
- **Angular handles an event** *(e.g. using event binding, output binding, or* `@HostListener`*)* in the subtree's root component or any of its children **whether they are using OnPush change detection or not**.
- You can set the change detection strategy of a component to `OnPush` in the `@Component` decorator

Demo App

- https://github.com/marius-oprea/onPush