How the change detection strategy affects performance in Angular applications with high-frequency updates

Research proposal bachelor's thesis 2020-2021

Duchatelet Thomas[1]

Abstract
Angular is a widely used web-framework and is a popular choice for developing all kinds of webapplications. Since Angular can be used to build reactive platforms, it might be suitable for realtime monitoring dashboards. Since humanity constantly tries to improve their tools, modern technologies are implemented for providing insights on big amounts of data. With the rise of the 'Internet of Things', the demand for realtime monitoring dashboards is also increasing. As IoT devices most of the time change state frequently, the default change detection strategy might not be the best solution for this case. Each strategy will be implemented and tested with high-frequency updates to investigate the effect of alternative strategies on the performance of the application. After comparing each scenario, it is expected to illustrate that performance can be maximized by implementing an alternative change detection strategy. This could lead to more responsive webapplications and more visually pleasing dashboards. This thesis will be mentored by Bonnie Brennan (Angular Nation) and will therefore be written in English.

Keywords
Web application development. Angular — Change detection — Performance

Co-promotor
Bonnie Brennan[2] (Angular Nation)

Contact: [1] thomas.duchatelet.y3284@student.hogent.be; [2] bjbrennan75@gmail.com;

Table of contents

1. Introduction

Thanks to the 'Internet of Things' (IoT),all sorts of devices can nowadays be connected to measure a wide range of events and collect loads of data using sensors. This data can be visualized on dashboards so people are given the possibility to monitor machines and processes to gather insights on the efficiency and cost of that process. For creating such a platform, Angular could be an excellent framework. One of the key features Angular developers love is the automatic change detection, a mechanism that updates the displayed web page when its data has changed. This allows to easily create real-time monitoring dashboards where the visualization adapts when the state of the IoT-device changes. Despite the beauty of Angular handling automatic updates, there is always an opposite side of the coin and the default change detection strategy is not the best for every case. According to Angular University (2020) the developer does not have to worry about change detection in 95% of the cases. Though, when developing web applications with high-frequency updates, one should consider alternative change detection strategies to maximize performance. This thesis will tackle following topics:

- How does change detection work in Angular?
- What are possible change detection strategies for Angular applications?
- What is the overall performance of the application with high-frequency updates using each strategy?
- What are the downsides of an alternative strategy for developers?
- Which strategy is preferred to maximize performance of an application with high-frequency updates?

2. State-of-the-art

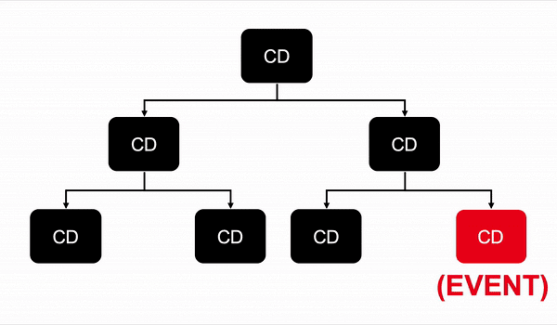
2.1 What is change detection in Angular?

Change detection in Angular loops over the models to check if the current state differs from the previous state after a trigger was fired. When a model's state has changed, the mechanism automatically updates the Document Object Model (DOM) to ensure the user interface is synchronized with the latest state of the model. (Kumar, 2020)

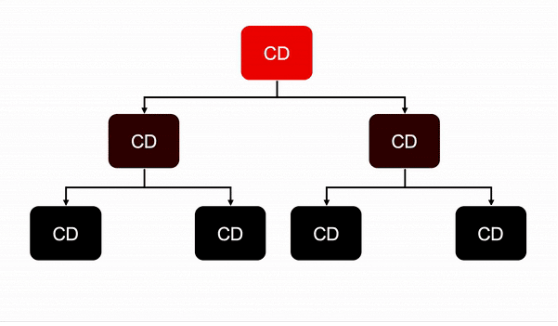
2.2 How does the default change detection strategy work in Angular?

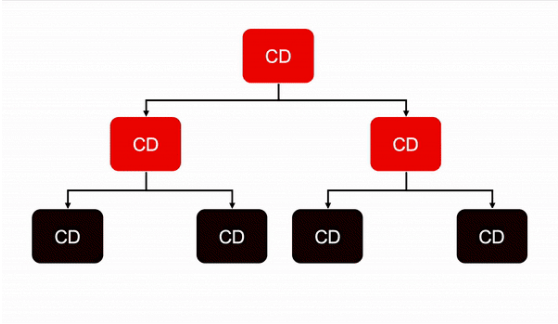
By default, Angular uses Zone.js to trigger change detection. Zone.js does not detect changes itself, instead the responsibility of Zone.js is notifying Angular when to run change detection. (Inatomi, 2020) The Angular change detection mechanism then loops over the component tree (starting from the top) to check for changes. This strategy can influence performance in applications with lots of components as every component gets checked every time an event (like a button click, AJAX request, timer and so on) has triggered change detection. If it is known for sure some components will not change at certain events, other strategies might be interesting regarding the application's performance. Therefore this strategy is called *dirty checking*. (Hoffmann, 2019) The images below demonstrate the mechanism:

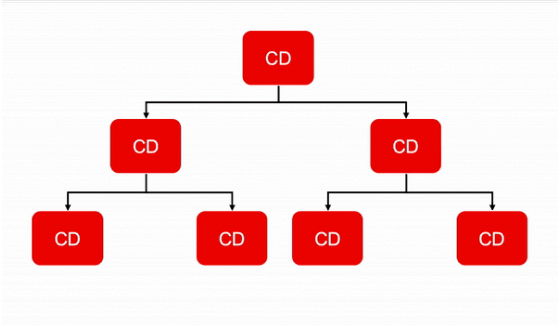
1. An event triggers the change detection



2. Change detection checks the component tree top to bottom







(Hoffmann, 2019)

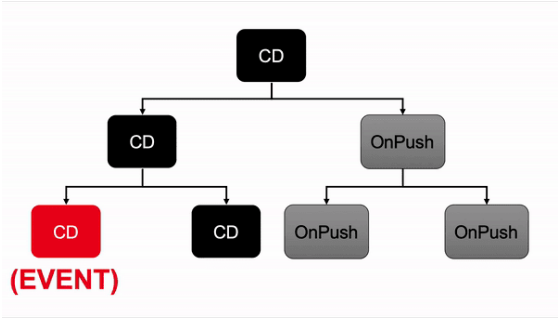
2.3 What are the alternatives to the default change detection strategy?

There are two major strategies developers can use:

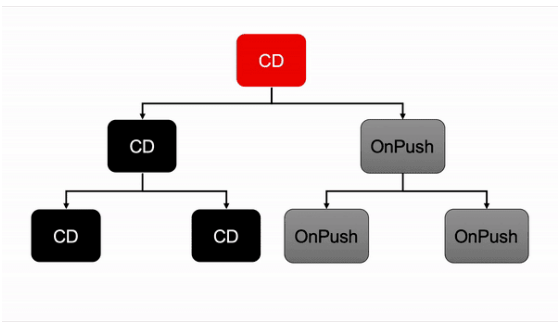
- Default
- OnPush

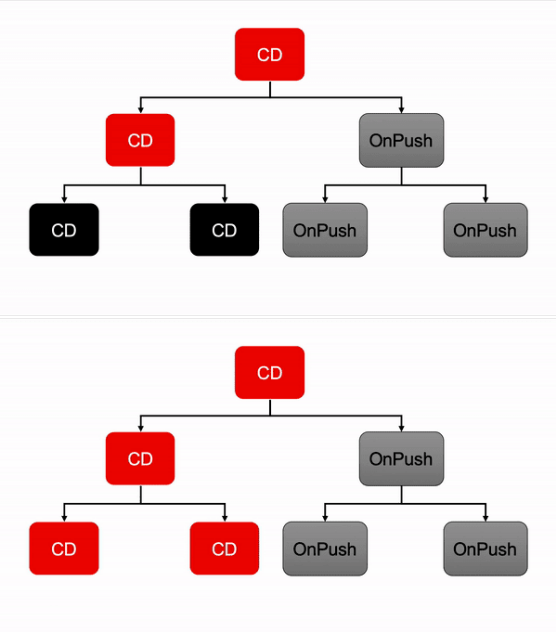
With the *OnPush* strategy it is possible to skip components when checking the component tree for changes. This can save time when dealing with lots of immutable components. The images below illustrate the *OnPush* mechanism.

1. An event triggers the change detection.



2. Change detection checks the component tree top to bottom, but skips the parts where *OnPush* is used.





(Hoffmann, 2019)

Thanks to the custom strategy implemented by the developer, Angular will not update the components where *OnPush* is defined unless:

- The reference of an *@Input* property changes.
- An event handler is triggered in the component or one of its children.
- An observable that uses the *async* pipe in the template provides a new value.
- The change detection is triggered by the developer.

This strategy, where change detection is manually triggered, might be cumbersome and error prone as the developer should import and call the change detection explicitly every time an object's properties might have changed. Since the release of Angular 9 coming with the brand new Ivy compiler, Zone.js can be excluded from the project and a typescript decorator can handle a manually triggered change detection strategy. Buomprisco (2019) uses the Θ*cmp* property to get the component definitions processed by Angular at runtime. A component definition is a data structure with many metadata properties used by the Ivy runtime, for example the lifecycle hooks. (Brink Nielsen, 2019) He then used said properties to override the *onInit* and *onDestroy* methods of the decorated component. In the *onInit* method, his decorator checks for each property if it is an object or an observable. If it is an object, it converts to a proxy and uses Angular Ivy's Θ*markdirty* method to mark the component in its handler function. When the property is an observable, it subscribes to the observable and stores the subscription in a list on the component.

3. Methodology

The first section of the thesis will contain an in-depth analysis of change detection in Angular and a visualization of the change detection process. In the second phase, an IoT device (or simulation) will be programmed to send its sensor data to a *signalR hub*. Afterwards three basic web applications will be developed to visualize live-data of the IoT device's state in the browser. In the first application the default strategy will be used so it relies on Zone.js for triggering change detection. In the second application the *OnPush* strategy will be implemented where Zone.js is manually called to trigger change detection. For the third strategy Zone will be excluded from the project and Ivy's Θ*markdirty* method will be used to trigger change detection. The developer's experience of each strategy can be measured by keeping track of the time it takes to build each application. At the last phase, the performance of each strategy can be measured and compared when the IoT device fires data to the *signalR hub* with high frequency. When the frequency of updates increases, the differences in performance between the scenarios should be distinguishable to the eye. But since numbers tell the tale, performance will be measured using following metrics:

- Size of chunks that have to be downloaded
- Monetary cost to view the page
- Processor usage
- Time it takes to see a first paint
- Time it takes to see a first meaningful paint
- Google's speed index

(Arsenault, 2017)

4. Expected results

The size of chunks that have to be downloaded should provide a lower number at the third method since there is no need for Zone.js. This could affect the monetary cost to view the page and processor usage by causing them to give lower thus better results. The time it takes to see a first meaningful paint and the speed index are interesting numbers, since these are the ones that a user notices the most.

5. Expected conclusions

When fewer requests are made, all three scenarios should provide a viable solution with little to no remarkable differences. As the amount of updates increases and the application is pushed to its limits, the manual strategies (2 and 3) should return better responsiveness as they will not evaluate each component every time an update is fired. The first scenario should be the easiest to develop as Zone.js worries about change detection. Once the decorator used in the third scenario is implemented correctly, the manual triggering strategy should also be easy to use for future expansion of the project.

References

Angular University. (2020, december 18). *Angular Change Detection - How Does It Really Work?* https://blog.angular-university.io/how-does-angular-2-change-detection-really-work/

Arsenault, C. (2017, november 9). *Measuring Web Performance - Analyzing What Matters Most*. https://www.keycdn.com/blog/measuring-web-performance

Brink Nielsen, L. G. (2019, december 26). *Component features with Angular Ivy*. https://indepth.dev/posts/1100/component-features-with-angular-ivy

Buomprisco, G. (2019, november 18). *Quantum Angular: Maximizing Performance by Removing Zone*. https://blog.bitsrc.io/quantum-angular-maximizing-performance-by-removing-zone-e0eefe85b8d8

Hoffmann, M. (2019, november 18). *The Last Guide For Angular Change Detection You'll Ever Need*. https://www.mokkapps.de/blog/the-last-guide-for-angular-change-detection-you-will-ever-need/

Inatomi, S. (2020, maart 18). *Angular: Test Reactiveness with OnPush strategy*. https://blog.lacolaco.net/2020/03/angular-app-reactiveness-en/

Kumar, D. (2020, juli 31). *Change Detection in Angular*. https://debugmode.net/2020/07/31/change-detection-in-angular/