Angular Performance Optimization

Everything you need to know

Abdelfattah Ragab

Angular Performance Optimization

Everything you need to know

Abdelfattah Ragab

Introduction

Welcome to the book Angular Performance Optimization.

In this book, I will show you all the ways you can optimize the performance of your Angular application. You may already be familiar with concepts like change detection and lazy loading, but there are plenty of other strategies and techniques you can use to significantly improve your app's performance. This book explains these advanced optimization methods and provides you with practical insights and actionable tips.

Whether you are a beginner or an experienced developer, this book is designed to give you the knowledge and tools you need to effectively optimize your Angular applications.

Let us get started.

Why Optimize Angular Applications?

Optimizing Angular applications is crucial for enhancing performance, improving user experience, and ensuring efficient resource management. Key strategies include change detection, lazy loading, and other performance techniques.

How can I measure the Angular Application Performance?

Measuring the performance of Angular applications is essential for identifying bottlenecks and ensuring a smooth user experience. There are several tools and techniques available to help you assess and optimize performance.

Key Performance Metrics

1. **Time to Interactive (TTI):** This metric measures how long it takes for the application to become fully interactive, meaning users can interact with it without delays.
2. **Total Page Load Time:** This measures the overall time taken for the entire page to load, including all resources and dependencies.

Tools for Performance Measurement

1. **Google Chrome DevTools:**
    - The **Performance tab** allows you to record and analyze the performance of your Angular application. You can start profiling while interacting with your app to identify slow areas, particularly during change detection cycles
    - **Lighthouse:** This tool provides automated audits for performance, accessibility, and SEO. It generates a detailed report with actionable insights to help optimize your application.
2. **Angular DevTools:**
   Angular DevTools extends the capabilities of Chrome DevTools with specific debugging and profiling tools tailored specifically for Angular developers
3. **WebPageTest:**
   This tool allows you to test your application's speed from various locations and browsers. It provides insights into loading speed and can help detect performance issues specific to Angular applications.
4. **Protractor:**
   While primarily a testing tool for Angular applications, Protractor can simulate user interactions and help assess performance during end-to-end tests.

Best Practices for Performance Measurement

Profile Regularly: Regular profiling during development can help catch performance issues early. Use the Chrome DevTools to record interactions and analyze the performance data.
Focus on User Interactions: When profiling, interact with the parts of the application that are known to be slow. This targeted approach can yield more relevant insights.
Benchmarking: Establish baseline performance metrics to compare against as you make changes to the application. This helps in understanding the impact of optimizations.

Use Angular Server-Side Rendering (SSR)

Server-Side Rendering (SSR) with Angular Universal is a powerful technique that can significantly improve the performance of your Angular applications. By rendering pages on the server, you can improve initial load times, improve SEO, and provide a better user experience.
Benefits of SSR:

- **Faster Initial Load:** Since the server sends pre-rendered HTML, users can see content faster.

- **Improved SEO:** Search engines can crawl and index your content better if it is rendered on the server, which leads to better visibility in search results.
- **Enhanced User Experience:** Users can interact with the application faster because they do not have to wait for the JavaScript to load and execute before they see the content.

Utilize TrackBy with ngFor

When rendering lists with `ngFor`, Angular can use the trackBy function to recognise which elements have been changed, added or removed. This prevents the entire list from being re-rendered if only a few elements change, which improves performance.
Angular 18 introduces the `@for` syntax, which provides a more intuitive method for traversing elements. This syntax is similar to traditional JavaScript loops.
The @for now forces developers to use a tracking function.
typescript

```
trackById(index: number, item: any):
number {
    return item.id; // Return the unique
id of the item
  }
```

template

```
<ul>
```

```
  @for (let item of items; track
trackById) {
    <li>{{ item.name }}</li>
  }
</ul>
```

Change Detection Optimization

Angular uses a change detection mechanism to update the view when the model changes. Angular uses the Default change detection strategy, which checks all components in the component tree. Angular looks for changes to data-bound values in a change detection process that runs after every DOM event: every keystroke, mouse move, timer tick, and server response.
To optimize this:
Use OnPush Change Detection: This strategy tells Angular to check for changes only when input properties change or when an event occurs, reducing unnecessary checks.
With the OnPush strategy, Angular only checks the component when:

- An `@Input()` property changes.
- An event occurs within the component.
- An observable linked to the component emits a new value.

This means that if a component's inputs remain the same, Angular skips checking that component, reducing the overall number of checks performed during change detection.

Use Observables and Async Pipe

Take advantage of Angular's reactive programming model by using observables. The asynchronous pipe automatically subscribes to observables and takes care of unsubscribing, reducing memory leaks and improving performance.

To use the `async` pipe, simply bind your observable to the template. Here is an example:

```
import { Component, OnInit } from
'@angular/core';
import { Observable } from 'rxjs';
import { DataService } from
'./data.service';

@Component({
  selector: 'app-data',
  template: `
    <div *ngIf="data$ | async as data;
else loading">
      <ul>
        <li *ngFor="let item of data">{{
item.name }}</li>
```

```
      </ul>
    </div>
    <ng-template
#loading>Loading...</ng-template>
  `
})
export class DataComponent implements
OnInit {
  data$: Observable<any[]>;

  constructor(private dataService:
DataService) {}

  ngOnInit() {
    this.data$ =
this.dataService.getData(); // Returns
an observable
  }
}
```

Use Observables and RxJS Operators to Optimize Data Handling

Optimizing data processing in Angular applications using observables and RxJS operators can significantly improve performance by reducing the number of emissions and managing asynchronous data more efficiently.