ChatGPT - nqt

Bài 14: Tối Ưu Hiệu Suất Ứng Dụng Angular (Performance Optimization in Angular)

Trong bài học này, chúng ta sẽ tìm hiểu cách tối ưu hiệu suất ứng dụng Angular để đảm bảo tốc độ tải nhanh, giảm độ trễ khi xử lý dữ liệu và nâng cao trải nghiệm người dùng.

---

1. Tại sao cần tối ưu hiệu suất trong Angular?

Khi ứng dụng Angular phát triển, số lượng component, service, và dữ liệu sẽ ngày càng lớn. Nếu không tối ưu, ứng dụng có thể gặp phải:

- **Render chậm**: Quá nhiều thay đổi trong UI gây tải nặng cho trình duyệt.
- **Dữ liệu lớn**: Việc truyền tải nhiều dữ liệu không cần thiết làm tăng thời gian tải.
- **Memory Leak**: Bộ nhớ bị rò rỉ do các subscription không được hủy.

---

2. Các Kỹ Thuật Tối Ưu Hiệu Suất Angular

Chúng ta sẽ tìm hiểu các cách tối ưu sau:

1. **OnPush Change Detection**
2. **TrackBy trong ngFor*
3. **Lazy Loading cho Feature Module**
4. **Unsubscribe Observables**

5. **Sử dụng Pure Pipes thay vì Methods trong Template**
6. **Debounce Time khi nhập liệu**
7. **Prefetching và Preloading Strategy trong Lazy Loading**

---

3. Chi Tiết Kỹ Thuật Tối Ưu

🔹 1. Sử dụng Change Detection Strategy: OnPush

Mặc định, Angular dùng **ChangeDetectionStrategy.Default**, nghĩa là mỗi khi có sự kiện xảy ra (click, nhập liệu, API response, v.v.), Angular sẽ re-render lại toàn bộ component.

Giải pháp: Sử dụng `ChangeDetectionStrategy.OnPush` để Angular chỉ update component khi **input thay đổi**.

📌 Ví dụ:

```
typescript
```

```
import { ChangeDetectionStrategy, Component, Input } from '@angular/core';

@Component({
  selector: 'app-user',
  template: `<p>{{ user.name }}</p>`,
  changeDetection: ChangeDetectionStrategy.OnPush // Sử dụng OnPush
})
export class UserComponent {
  @Input() user!: { name: string };
}
```

✔️ **Lợi ích**: Giảm số lần re-render, tăng hiệu suất khi có nhiều component con.

🔹 *2. Sử dụng trackBy trong ngFor*

Angular mặc định sẽ re-render toàn bộ danh sách khi có thay đổi, ngay cả khi một item không bị ảnh hưởng.

Giải pháp: Dùng `trackBy` để chỉ re-render item thực sự thay đổi.

📌 Ví dụ:

```
typescript
```

```
@Component({
  selector: 'app-user-list',
  template: `
    <ul>
      <li *ngFor="let user of users; trackBy: trackById">
        {{ user.name }}
      </li>
    </ul>
  `
})
export class UserListComponent {
  users = [
    { id: 1, name: 'Alice' },
    { id: 2, name: 'Bob' }
  ];

  trackById(index: number, user: any) {
    return user.id; // Chỉ update item nếu ID thay đổi
  }
}
```

✔️ **Lợi ích**: Tối ưu danh sách lớn, chỉ update những phần tử thực sự thay đổi.

🔹 3. Lazy Loading cho Feature Module

Thay vì tải toàn bộ ứng dụng ngay khi khởi động, ta chỉ tải module khi cần thiết.

📌 Cấu hình `app-routing.module.ts`

```
typescript
```

```
const routes: Routes = [
  { path: 'dashboard', loadChildren: () =>
import('./dashboard/dashboard.module').then(m => m.DashboardModule) }
];
```

✔️ **Lợi ích**: Giảm thời gian tải ban đầu, chỉ load module khi người dùng truy cập.

🔹 4. Unsubscribe Observables để tránh Memory Leak

Nếu không `unsubscribe`, Angular sẽ giữ lại Subscription ngay cả khi component bị hủy.

Giải pháp: Dùng `takeUntil` hoặc `async pipe`.

📌 Cách 1: Dùng `takeUntil`

```
typescript
```

```
import { Component, OnDestroy } from '@angular/core';
import { Subject } from 'rxjs';
import { takeUntil } from 'rxjs/operators';

@Component({
  selector: 'app-example',
  template: `<p>{{ data }}</p>`
})
export class ExampleComponent implements OnDestroy {
  private unsubscribe$ = new Subject<void>();

  data: any;

  constructor(private apiService: ApiService) {
    this.apiService.getData()
      .pipe(takeUntil(this.unsubscribe$))
      .subscribe(res => this.data = res);
  }

  ngOnDestroy() {
    this.unsubscribe$.next();
    this.unsubscribe$.complete();
  }
}
```

📌 Cách 2: Dùng `async pipe`

```
html
```

```
<p>{{ apiService.getData() | async }}</p>
```

✔️ **Lợi ích**: Tránh memory leak khi component bị hủy.

🔹 5. Sử dụng Pure Pipes thay vì Methods trong Template

Nếu dùng method trong template, Angular sẽ gọi lại mỗi lần có change detection.

📌 Ví dụ xấu (tránh dùng method trong template)

```
html
```

```
<p>{{ calculateTotal() }}</p>
```

```
typescript
```

```
calculateTotal() {
  return this.items.reduce((acc, item) => acc + item.price, 0);
}
```

🔴 **Vấn đề**: Hàm `calculateTotal()` được gọi lại nhiều lần không cần thiết.

✔️ **Giải pháp**: Dùng **Pure Pipe**

```
typescript
```

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'totalPrice',
  pure: true
})
export class TotalPricePipe implements PipeTransform {
  transform(items: any[]): number {
    return items.reduce((acc, item) => acc + item.price, 0);
  }
}
```

```
html
```

```
<p>{{ items | totalPrice }}</p>
```

✔️ **Lợi ích**: Tăng hiệu suất vì **Pure Pipe** chỉ chạy khi dữ liệu thay đổi.

🔹 6. Debounce Time khi nhập liệu

Nếu nhập liệu và gọi API liên tục, hệ thống có thể bị chậm.

Giải pháp: Dùng `debounceTime()` để giảm số lần gọi API.

📌 Ví dụ:

```
typescript
```

```
import { Component } from '@angular/core';
import { FormControl } from '@angular/forms';
import { debounceTime } from 'rxjs/operators';

@Component({
  selector: 'app-search',
  template: `<input [formControl]="searchControl" placeholder="Search">`
})
export class SearchComponent {
  searchControl = new FormControl('');

  constructor() {
    this.searchControl.valueChanges.pipe(debounceTime(300))
      .subscribe(value => {
        console.log('Gọi API với:', value);
      });
  }
}
```

✔️ **Lợi ích**: Giảm số lần gọi API khi nhập liệu.

🔹 7. Prefetching và Preloading Strategy

Có thể **preload** module trước khi người dùng cần, giúp tăng tốc trải nghiệm.

📌 Cấu hình `app-routing.module.ts`

```
typescript
```

```
import { PreloadAllModules, RouterModule, Routes } from '@angular/router';

const routes: Routes = [
  { path: 'dashboard', loadChildren: () =>
import('./dashboard/dashboard.module').then(m => m.DashboardModule) }
];

@NgModule({
  imports: [RouterModule.forRoot(routes, { preloadingStrategy:
PreloadAllModules })],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

✔️ **Lợi ích**: Load module trước khi người dùng truy cập.

---

4. Tổng Kết

- **OnPush Change Detection**: Giảm số lần render.
- **trackBy trong ngFor*: Tối ưu danh sách lớn.
- **Lazy Loading**: Chỉ tải module khi cần.
- **Unsubscribe Observables**: Tránh rò rỉ bộ nhớ.
- **Pure Pipes**: Tránh gọi lại method không cần thiết.
- **Debounce Time**: Giảm số lần gọi API khi nhập liệu.
- **Preloading Strategy**: Tải trước module quan trọng.

✔️ Với những kỹ thuật này, bạn có thể tối ưu hiệu suất Angular hiệu quả hơn! 🚀