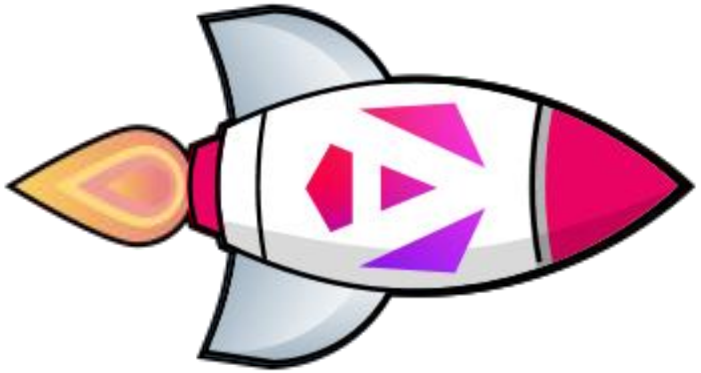
ANGULAR
ARCHITECTS

Runtime Best Practices

Alexander Thalhammer | @LX_T

Outline - Runtime Performance

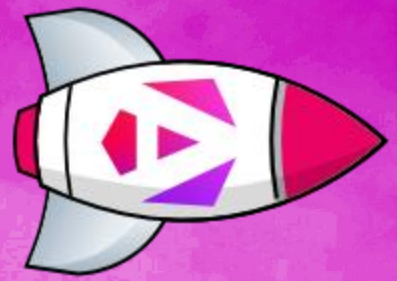


- Change Detection
- Runtime Best Practices

Runtime Best Practices

- Large @for loops
  - Using track in @for
  - Avoid large component trees
- UX improvements
  - Use spinners / preview thumbs / skeletons
  - Optimistic updates
- Bonus: RxJS Subscription Best Practices

Agenda



Handling large @for loops

ANGULAR
ARCHITECTS

Migrate to NG 17 Control Flow

- The future is here ☺
- May look a bit awkward at first sight
  - but it has (a lot) **performance** benefits and
  - on top of that it make things **easier**

- Easy migration

```
ng generate @angular/core:control-flow
```

- Make sure to add
  - @empty / @else
  - improve track @for

NG 17 Control Flow benchmark

Duration in milliseconds ± 95% confidence interval (Slowdown

Name Duration for...	vanillajs	angular-cf-nozone-v17.0.2	vue-v3.4.21	angular-cf-v17.0.2	angular-ngfor-v17.0.2	react-hooks-v18.2.0
Implementation notes	772					
Implementation link	code	code	code	code	code	code
create rows creating 1,000 rows. (5 warmup runs).	36.2 ±0.5 (1.03)	44.2 ±0.3 (1.26)	44.2 ±0.5 (1.26)	44.8 ±0.5 (1.27)	45.6 ±0.4 (1.30)	45.8 ±0.3 (1.30)
replace all rows updating all 1,000 rows. (5 warmup runs).	39.5 ±0.3 (1.03)	51.2 ±0.3 (1.33)	48.5 ±0.5 (1.26)	54.4 ±0.4 (1.41)	54.9 ±0.3 (1.43)	54.8 ±0.3 (1.42)
partial update updating every 10th row for 1,000 row. (3 warmup runs). 4 x CPU slowdown.	16.8 ±0.2 (1.07)	17.4 ±0.2 (1.11)	19.6 ±0.3 (1.25)	17.5 ±0.3 (1.11)	17.7 ±0.4 (1.13)	20.8 ±0.3 (1.32)
select row highlighting a selected row. (5 warmup runs). 4 x CPU slowdown.	2.9 ±0.2 (1.07)	3.9 ±0.2 (1.44)	4.3 ±0.2 (1.59)	3.9 ±0.1 (1.44)	3.9 ±0.1 (1.44)	5.1 ±0.2 (1.89)
swap rows swap 2 rows for table with 1,000 rows. (5 warmup runs). 4 x CPU slowdown.	18.1 ±0.2 (1.01)	19.9 ±0.4 (1.11)	20.7 ±0.3 (1.16)	20.0 ±0.3 (1.12)	170.1 ±1.2 (9.50)	166.3 ±1.2 (9.29)

https://krausest.github.io/js-framework-benchmark/current.html

Using track in @for (*ngFor)

– Problem: Angular replaces items in @for (*ngFor) upon changes

– Identify: Easy - search for "@for (*ngFor)"

– Solution: Use the **track** function (*previously trackBy*)

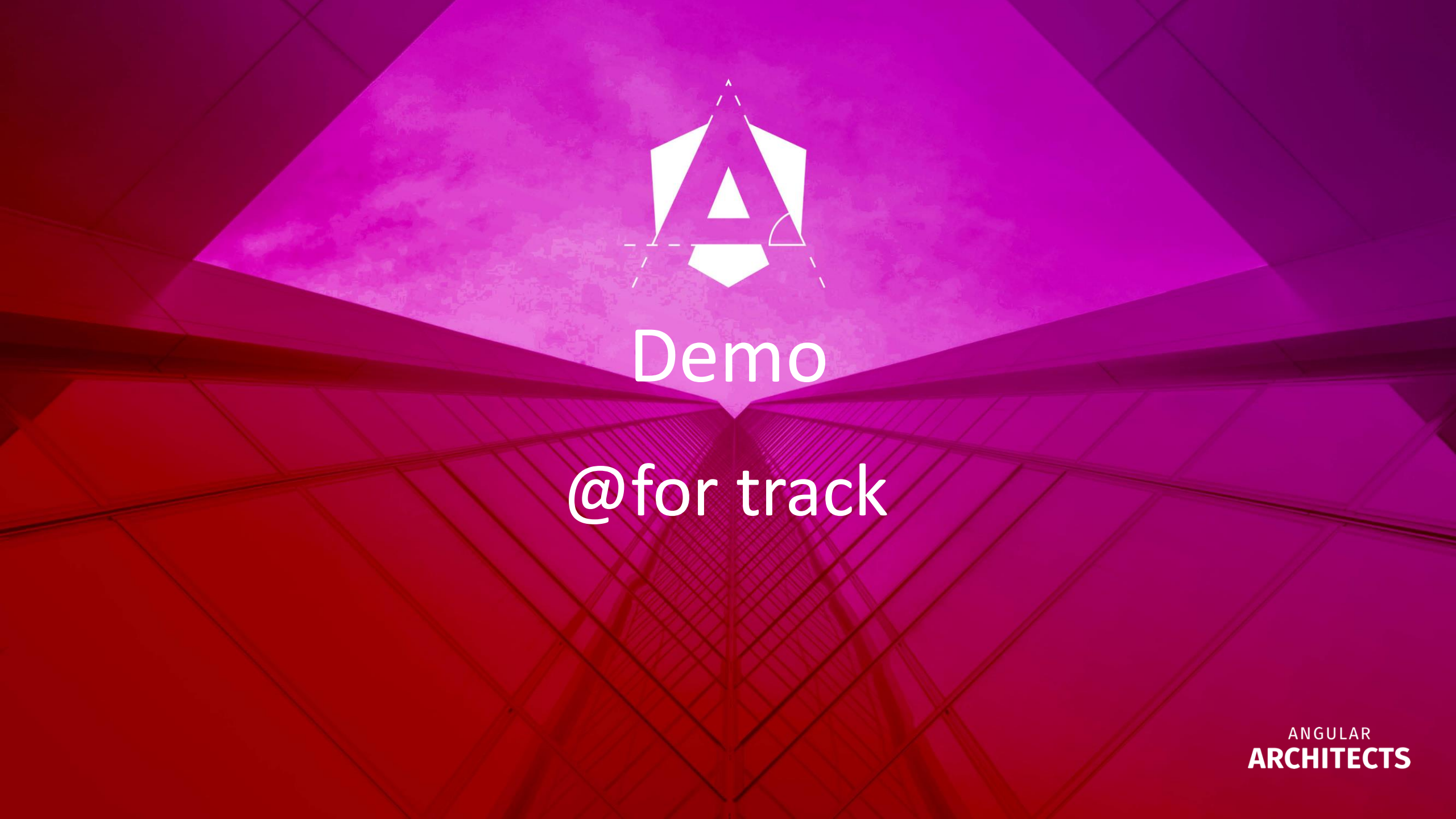
```
<li *ngFor="let dashboard of dashboards; trackBy: trackByDashboardId"

trackByDashboardId(index: number, item: Dashboard): number {
  return item.id;
}
```

Using track in @for

- Automatically required

```
@for (flight of flights; track flight.id) {
   [...]
} @empty {
  No flights found.
}
```



Demo

@for track

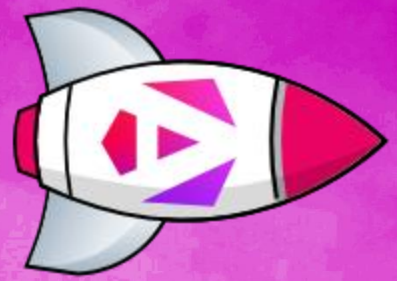
Avoid large component trees

- Problem: Too many (100+) components are loaded
- Identify: Lots of components slowing down frame rate
- Solution: On demand component rendering
  - E.g. Pagination or Angular CDKs <cdk-virtual-scrolling-component>

Demo

Virtual Scrolling

ANGULAR
ARCHITECTS



Other UX improvements

Spinners & Preview Thumbs

Twitter / Insta / …

Use spinners and preview thumbs

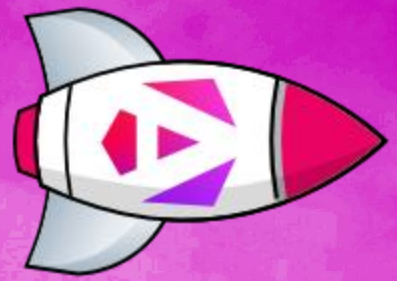
- Problem: App waits for backend before showing content
- Identify: Waiting for API data to show a view (page)
- Solution: Show view (page) immediately
  - Show spinners to indicate data is still loading
  - Even more sophisticated: show preview images (used everywhere on big platforms!)

Optimistic Updates

E.g. Like Buttons

Optimistic Updates

- Problem: App waits for backend for confirmations
- Identify: Spinner showing when clicking on save
- Solution: Confirm action immediately
  - Go back in case of an error (e.g. no network)
  - But maybe not a good idea for all user flow ☺



RxJS Subscription Best Practices

ANGULAR
ARCHITECTS

Why asynchronicity?

Asynchronous operations (API requests)

Interactive behavior (user input)

Websockets

Server Send Events (Push)

Why do we (always!) need to unsubscribe?

Avoid side effects

Avoid memory leaks

Also for HttpClient‘s get / post …

Manage your RxJS subscriptions

- Problem: Components create subscriptions without closing them
- Identify: .subscribe() without .unsubscribe() or other methods
- Solution: Unsubscribe from all Observables in your App
  - Except Angular Router Params

RxJS Subscription Management

- Explicitly with reference
  - readonly subscription = observable$.subscribe(…); // field initializer
    // subscription?.add(otherObservable$.subscribe(…)); // also possible since V6
    subscription?**.unsubscribe()**; // ngOnDestroy
- Implicitly with take until
  - ~~observable$.pipe(takeUntil(otherObservable)).subscribe(…);~~
  - observable$.pipe(**takeUntilDestroyed()**).subscribe(…);

last operator!

- Implicitly with async Pipe managed by Angular or using a Signal
  - {{ observable$ | **async** }} → **also triggers a cdr.markForCheck for OnPush ☺**
- Automatically managed by Angular
  - Router Params / ParamMap (only 1 I know where unsubscribing is not needed)

Where / when do we subscribe?

- 1 **Field initializer or constructor**
- 2 **If** @Input(s) needed → **ngOnInit** hook (needs destroyRef)
- 3 Elsewhere (needs injected destroyRef)

Demo

RxJS Subscription Management

ANGULAR
ARCHITECTS

Lab 07 Runtime Best Practices

track / Virtual Scrolling / Unsubscribing RxJS subscriptions

ANGULAR
ARCHITECTS

Runtime Best Practices

- Large @for loops
  - Using **track** in @for
  - **Avoid** large component trees
- UX improvements
  - Use spinners / preview thumbs / skeleton
  - Optimistic updates
- Bonus: **RxJS Subscription Management**

Recap

References

- Angular CDK Scrolling Comp
  - https://material.angular.io/cdk/scrolling/overview

ANGULAR
ARCHITECTS