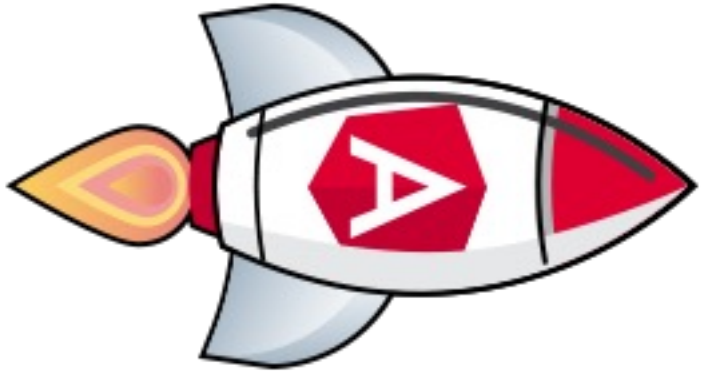
ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Runtime Performance
Further Runtime

Alex Thalhammer

Outline 03 - Runtime Performance

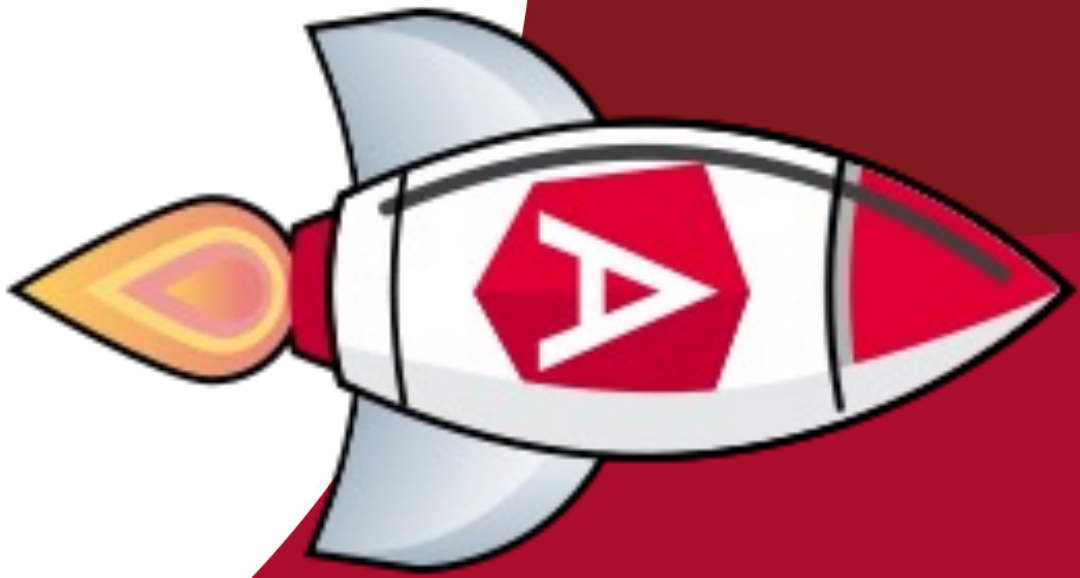


- Change Detection
- Further Runtime

Outline

- Use trackBy in *ngFor if possible
- Avoid large component trees
- Use Spinners and preview thumbs
- Optimistic updates
- Bonus: RxJS Subscription Management

Handling large ngFor loops



Using trackBy in ngFor

- Problem: *Angular replaces all items in ***ngFor** upon changes*
- Identify: Easy - search for "*ngFor"
- Solution: Use the trackBy function

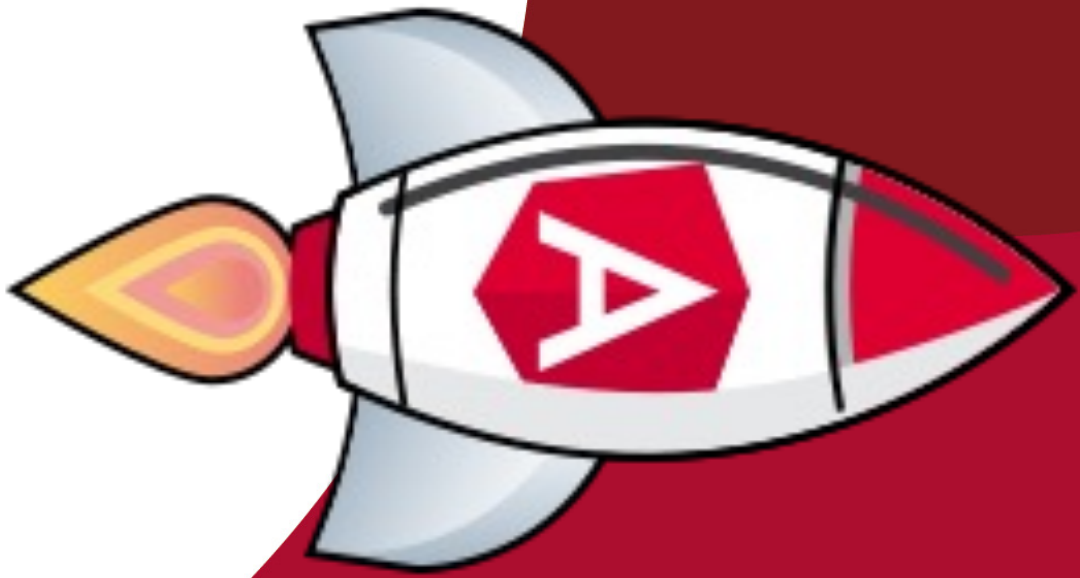
```
<li *ngFor="let dashboard of dashboards; trackBy: trackByDashboardId"

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

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>

Other UX improvements



Spinners & Preview Thumbs

Twitter / Insta / …

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

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

ANGULAR ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE ARCHITECT

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)

Why asynchronicity?

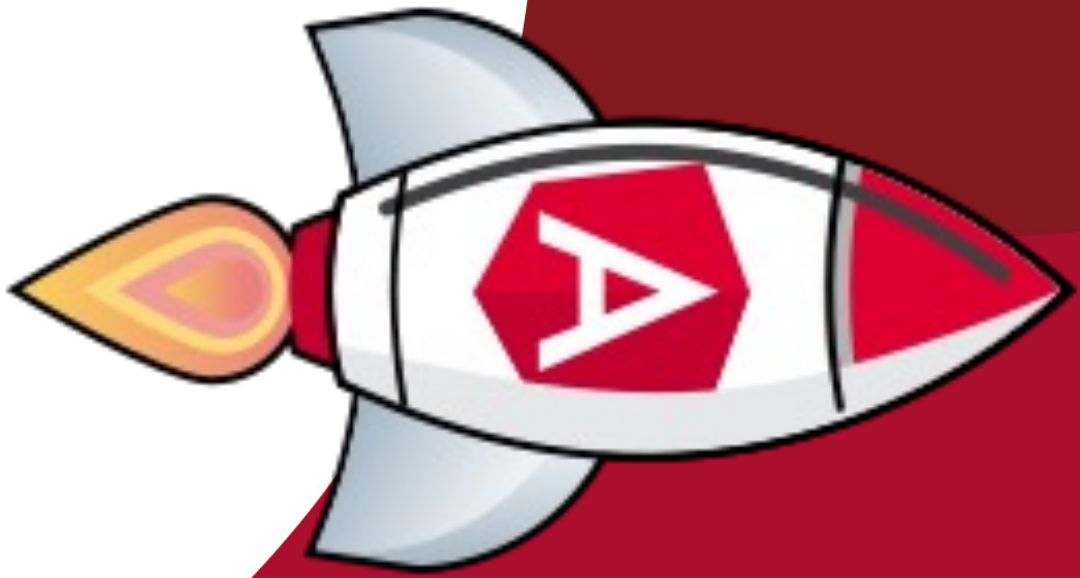
Asynchronous operations (API requests)

Interactive behavior (user input)

Websockets

Server Send Events (Push)

RxJS Subscription Best Practices



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

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

Why do we (always!) need to unsubscribe?

Avoid side effects

Avoid memory leaks

Also for HttpClient‘s get / post …

RxJS Subscription Management

- Explicitly

  ```
  let subscription = observable$.subscribe(…);
  // subscription.add(otherObservable$.subscribe(…)); // also possible since V6
  subscription?.unsubscribe();
  ```

- Implicitly
  - ~~observable$.pipe(takeUntil(otherObservable)).subscribe(…);~~
  - observable$.pipe(**takeUntilDestroyed()**).subscribe(…);

  } **last operator!**

- Implicitly with async Pipe in Angular

  {{ observable$ | **async** }} → **also triggers a cdr.markForCheck for OnPush**

- Automatic by Angular
  - Angular Router Params (the only 1 I know where unsubscribing is not needed)

Where do we subscribe?

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

DEMO – Unsubscribing

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

Lab

Further Runtime Performance

ANGULAR ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE ARCHITECT

Recap

- **Use trackBy in *ngFor if possible**
- **Avoid large component trees**
- Use Spinners and preview thumbs
- Optimistic updates
- Bonus: **RxJS Subscription Management**

References

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