
Reacting to Angular
an introduction to the best enterprise framework

In this presentation

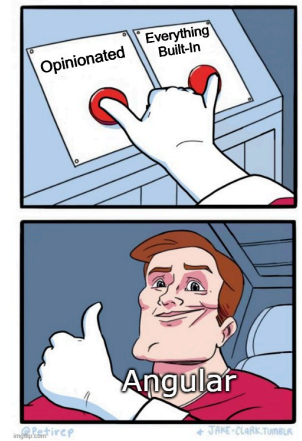
- A basic introduction to Angular
- Comparing the concept/meaning to React
- Version 17 and above
  - A lot of new features because of the Angular Renaissance
- No complicated stuff
  - No “Signal” explanation
  - No “ngZone” explanation
  - No “ChangeDetection” explanation
  - No "Dependency Injection" explanation
  - No "RxJS" explanation
  - They will be covered in the advanced course (hopefully I have time to create it)

What is Angular?

- Initial release in Sep 14, 2016 by the evil Google
  - We don't talk about version 1.x
- Enforced MVC-S (module-view-controller-service)
- Heavily used in enterprises
- Very opinionated approach
- Best in-class CLI
- Best in-class documentation
- Best in-class testing
- Best in-class upgrade guide
- Latest version: 18
- With the renaissance in version 17
- Best meta framework: AnalogJS
- Best UI library: Angular Material

You might not know about Angular!

- The reason why Typescript popular
- The unspoken father of Vue
- The inspiration of React and their fanboy
- Getting bad PR's from the transition from AngularJS to Angular 2
  - Doesn't Vue 3.x have a similar issue?
- The motivation for ESBuild, Module Federation, Dependencies Injection, and many more…
- You (normally) have ONE way of doing things correctly



Angular is FAST

- Batteries included!
  - Routing
  - HTTP Client
  - Form handling
  - ...
- CLI for scaffolding
  - `ng generate component my-component`
  - `ng generate service my-service`
  - `ng generate module my-module`
  - ...
- Better performance than both React and Vue
  - In very specific conditions

Duration in milliseconds ± 95% confidence interval (Slowdown = Duration / Fastest)

Name Duration for...	svelte-v5.0.0-next.174	vue-v3.4.29	angular-cf-nozone-v18.0.1	vue-pinia-v3.4.29 + 2.1.7	angular-cf-v18.0.1	angular-cf-signals-v18.0.1	react-hooks-v18.2.0	qwik-v1.5.0	react-zus-tand-v18.2.0 + 4.3.6	react-re-dux-v18.2.0 + 8.0.5
Implementation notes								1139		
Implementation link	code	code	code	code	code	code	code	code	code	code
create rows creating 1,000 rows. (5 warmup runs).	37.5 ±0.2 (1.09)	43.9 ±0.3 (1.27)	46.8 ±0.2 (1.36)	46.8 ±0.3 (1.36)	48.2 ±0.3 (1.40)	47.5 ±0.3 (1.38)	45.8 ±0.3 (1.33)	60.9 ±0.6 (1.77)	50.6 ±0.2 (1.47)	53.5 ±0.4 (1.55)
replace all rows updating all 1,000 rows. (5 warmup runs).	42.9 ±0.2 (1.13)	50.3 ±0.2 (1.32)	54.0 ±0.4 (1.42)	52.8 ±0.2 (1.39)	58.8 ±0.3 (1.54)	58.0 ±0.3 (1.52)	54.8 ±0.3 (1.44)	70.7 ±0.6 (1.86)	58.9 ±0.4 (1.55)	64.0 ±0.4 (1.68)
partial update updating every 10th row for 1,000 row. (3 warmup runs). 4 x CPU slowdown.	16.9 ±0.5 (1.08)	21.0 ±0.5 (1.34)	18.4 ±0.3 (1.17)	23.7 ±0.5 (1.51)	19.7 ±0.3 (1.25)	20.1 ±0.4 (1.28)	22.1 ±0.4 (1.41)	19.2 ±10.0 (1.22)	27.2 ±0.4 (1.73)	27.1 ±0.2 (1.73)
select row highlighting a selected row. (5 warmup runs). 4 x CPU slowdown.	4.6 ±0.2 (1.77)	4.9 ±0.2 (1.88)	5.1 ±0.2 (1.96)	7.9 ±0.2 (3.04)	5.6 ±0.2 (2.15)	7.4 ±0.2 (2.85)	8.0 ±0.3 (3.08)	10.1 ±1.0 (3.88)	9.8 ±0.2 (3.77)	9.2 ±0.4 (3.54)
swap rows swap 2 rows for table with 1,000 rows. (5 warmup runs). 4 x CPU slowdown.	20.7 ±0.4 (1.09)	22.1 ±0.3 (1.16)	22.5 ±0.3 (1.18)	24.8 ±0.3 (1.31)	22.9 ±0.2 (1.21)	22.8 ±0.2 (1.20)	168.3 ±1.4 (8.86)	29.9 ±0.8 (1.57)	172.5 ±1.9 (9.08)	172.7 ±1.3 (9.09)
remove row removing one row. (5 warmup runs). 2 x CPU slowdown.	16.1 ±0.3 (1.07)	19.2 ±0.1 (1.28)	17.2 ±0.1 (1.15)	23.6 ±0.3 (1.57)	17.6 ±0.1 (1.17)	19.2 ±0.4 (1.28)	18.9 ±0.3 (1.26)	23.7 ±0.5 (1.58)	20.7 ±0.2 (1.38)	31.5 ±0.3 (2.10)
create many rows creating 10,000 rows. (5 warmup runs).	387.9 ±1.8 (1.09)	433.8 ±2.8 (1.22)	484.4 ±1.8 (1.37)	442.6 ±2.3 (1.25)	485.3 ±0.7 (1.37)	483.0 ±2.0 (1.36)	736.1 ± 26.8 (2.08)	576.4 ±2.3 (1.63)	647.2 ±1.5 (1.83)	684.6 ±2.0 (1.93)
append rows to large table appending 1,000 to a table of 1,000 rows. (5 warmup runs).	42.9 ±0.3 (1.08)	49.1 ±0.3 (1.24)	52.5 ±0.3 (1.32)	54.2 ±0.3 (1.37)	53.9 ±0.4 (1.36)	53.8 ±0.4 (1.36)	51.5 ±0.3 (1.30)	71.8 ±0.6 (1.81)	57.4 ±0.5 (1.45)	58.4 ±0.2 (1.47)
clear rows clearing a table with 1,000 rows. (5 warmup runs). 4 x CPU slowdown.	14.5 ±0.2 (1.20)	18.5 ±0.6 (1.53)	26.8 ±0.5 (2.21)	20.8 ±0.5 (1.72)	36.6 ±0.7 (3.02)	37.2 ±0.6 (3.07)	26.3 ±0.3 (2.17)	27.6 ±0.6 (2.28)	31.2 ±0.3 (2.58)	33.0 ±0.5 (2.73)
weighted geometric mean of all factors in the table	1.13	1.32	1.39	1.48	1.49	1.52	1.65	1.75	1.81	1.97
compare: Green means significantly faster, red significantly slower	compare	compare	compare	compare	compare	compare	compare	compare	compare	compare

Duration in milliseconds ± 95% confidence interval (Slowdown = Duration / Fastest)

Name Duration for...	vue-v3.4.29	angular-cf-v18.0.1	react-hooks-v18.2.0
Implementation notes			
Implementation link	code	code	code
create rows creating 1,000 rows. (5 warmup runs).	43.9 ± 0.3 (1.27) 0.000%	48.2 ± 0.3 (1.40) 100.000%	45.8 ± 0.3 (1.33) 0.000%
replace all rows updating all 1,000 rows. (5 warmup runs).	50.3 ± 0.2 (1.32) 0.000%	58.8 ± 0.3 (1.54) 100.000%	54.8 ± 0.3 (1.44) 0.000%
partial update updating every 10th row for 1,000 row. (3 warmup runs). 4 x CPU slowdown.	21.0 ± 0.5 (1.34) 0.061%	19.7 ± 0.3 (1.25) 100.000%	22.1 ± 0.4 (1.41) 0.000%
select row highlighting a selected row. (5 warmup runs). 4 x CPU slowdown.	4.9 ± 0.2 (1.88) 0.001%	5.6 ± 0.2 (2.15) 100.000%	8.0 ± 0.3 (3.08) 0.000%
swap rows swap 2 rows for table with 1,000 rows. (5 warmup runs). 4 x CPU slowdown.	22.1 ± 0.3 (1.16) 0.008%	22.9 ± 0.2 (1.21) 100.000%	168.3 ± 1.4 (8.86) 0.000%
remove row removing one row. (5 warmup runs). 2 x CPU slowdown.	19.2 ± 0.1 (1.28) 0.000%	17.6 ± 0.1 (1.17) 100.000%	18.9 ± 0.3 (1.26) 0.000%
create many rows creating 10,000 rows. (5 warmup runs).	433.8 ± 2.8 (1.22) 0.000%	485.3 ± 0.7 (1.37) 100.000%	736.1 ± 26.8 (2.08) 0.000%
append rows to large table appending 1,000 to a table of 1,000 rows. (5 warmup runs).	49.1 ± 0.3 (1.24) 0.000%	53.9 ± 0.4 (1.36) 100.000%	51.5 ± 0.3 (1.30) 0.000%
clear rows clearing a table with 1,000 rows. (5 warmup runs). 4 x CPU slowdown.	18.5 ± 0.6 (1.53) 0.000%	36.6 ± 0.7 (3.02) 100.000%	26.3 ± 0.3 (2.17) 0.000%
weighted geometric mean of all factors in the table	1.32	1.49	1.65
compare: Green means significantly faster, red significantly slower	compare	stop compare	compare

Duration in milliseconds ± 95% confidence interval (Slowdown = Duration / Fastest)

Name Duration for...	vue-v3.4.29	angular-cf-nozone-v18.0.1	vue-pinia-v3.4.29 + 2.1.7	angular-cf-v18.0.1	angular-cf-signals-v18.0.1	react-hooks-v18.2.0	react-zus-tand-v18.2.0 + 4.3.6
Implementation notes							
Implementation link	code	code	code	code	code	code	code
create rows creating 1,000 rows. (5 warmup runs).	43.9 ± 0.3 (1.27) 0.000%	46.8 ± 0.2 (1.36) 100.000%	46.8 ± 0.3 (1.36) 87.177%	48.2 ± 0.3 (1.40) 0.000%	47.5 ± 0.3 (1.38) 0.121%	45.8 ± 0.3 (1.33) 0.029%	50.6 ± 0.2 (1.47) 0.000%
replace all rows updating all 1,000 rows. (5 warmup runs).	50.3 ± 0.2 (1.32) 0.000%	54.0 ± 0.4 (1.42) 100.000%	52.8 ± 0.2 (1.39) 0.000%	58.8 ± 0.3 (1.54) 0.000%	58.0 ± 0.3 (1.52) 0.000%	54.8 ± 0.3 (1.44) 63.134%	58.9 ± 0.4 (1.55) 0.000%
partial update updating every 10th row for 1,000 row. (3 warmup runs). 4 x CPU slowdown.	21.0 ± 0.5 (1.34) 0.000%	18.4 ± 0.3 (1.17) 100.000%	23.7 ± 0.5 (1.51) 0.000%	19.7 ± 0.3 (1.25) 0.002%	20.1 ± 0.4 (1.28) 0.000%	22.1 ± 0.4 (1.41) 0.000%	27.2 ± 0.4 (1.73) 0.000%
select row highlighting a selected row. (5 warmup runs). 4 x CPU slowdown.	4.9 ± 0.2 (1.88) 32.166%	5.1 ± 0.2 (1.96) 100.000%	7.9 ± 0.2 (3.04) 0.000%	5.6 ± 0.2 (2.15) 0.004%	7.4 ± 0.2 (2.85) 0.000%	8.0 ± 0.3 (3.08) 0.000%	9.8 ± 0.2 (3.77) 0.000%
swap rows swap 2 rows for table with 1,000 rows. (5 warmup runs). 4 x CPU slowdown.	22.1 ± 0.3 (1.16) 2.743%	22.5 ± 0.3 (1.18) 100.000%	24.8 ± 0.3 (1.31) 0.000%	22.9 ± 0.2 (1.21) 6.227%	22.8 ± 0.2 (1.20) 11.413%	168.3 ± 1.4 (8.86) 0.000%	172.5 ± 1.9 (9.08) 0.000%
remove row removing one row. (5 warmup runs). 2 x CPU slowdown.	19.2 ± 0.1 (1.28) 0.000%	17.2 ± 0.1 (1.15) 100.000%	23.6 ± 0.3 (1.57) 0.000%	17.6 ± 0.1 (1.17) 0.058%	19.2 ± 0.4 (1.28) 0.000%	18.9 ± 0.3 (1.26) 0.000%	20.7 ± 0.2 (1.38) 0.000%
create many rows creating 10,000 rows. (5 warmup runs).	433.8 ± 2.8 (1.22) 0.000%	484.4 ± 1.8 (1.37) 100.000%	442.6 ± 2.3 (1.25) 0.000%	485.3 ± 0.7 (1.37) 17.958%	483.0 ± 2.0 (1.36) 95.729%	736.1 ± 26.8 (2.08) 0.000%	647.2 ± 1.5 (1.83) 0.000%
append rows to large table appending 1,000 to a table of 1,000 rows. (5 warmup runs).	49.1 ± 0.3 (1.24) 0.000%	52.5 ± 0.3 (1.32) 100.000%	54.2 ± 0.3 (1.37) 0.000%	53.9 ± 0.4 (1.36) 0.000%	53.8 ± 0.4 (1.36) 0.001%	51.5 ± 0.3 (1.30) 0.008%	57.4 ± 0.5 (1.45) 0.000%
clear rows clearing a table with 1,000 rows. (5 warmup runs). 4 x CPU slowdown.	18.5 ± 0.6 (1.53) 0.000%	26.8 ± 0.5 (2.21) 100.000%	20.8 ± 0.5 (1.72) 0.000%	36.6 ± 0.7 (3.02) 0.000%	37.2 ± 0.6 (3.07) 0.000%	26.3 ± 0.3 (2.17) 1.109%	31.2 ± 0.3 (2.58) 0.000%
weighted geometric mean of all factors in the table	1.32	1.39	1.48	1.49	1.52	1.65	1.81
compare: Green means significantly faster, red significantly slower	compare	stop compare	compare	slower! compare	slower! compare	compare	slower! compare

Angular Installation

- Node.js v18.9 or later
- Install the Angular CLI globally: `npm install -g @angular/cli`
- Create a new Angular project: `ng new my-app`
- Start the development server: `ng serve`
- Build the project: `ng build`
  - Multiple targets:
    - `browser` - Webpack
    - `browser-esbuild` - ESBuild
    - `application` - ESBuild with Node Server (eg: Desktop apps)
    - `ng-packagr` - Library packaging

Angular Installation

- Run tests: `ng test`
  - Default using Karma and Jasmine (zero-config)
- Run end-to-end tests: (`ng e2e`)https://angular.dev/tools/cli/end-to-end
  - CLI step-by-step setup
- Best part - Scaffolded code using (`ng generate <schematic> [options]`)https://angular.dev/cli/generate

React Installation

- Doesn't provide official CLI
- Recommended to use metaframework instead (`Next.js`, `Remix`, `Expo`, ...)
- Community template CLI
  - `create-react-app` for purely React
  - `create-next-app` / `create-t3-app` for Next.js
  - `create-remix` for Remix
- Everything else is created by hand

Angular installation

- Official CLI: `ng new my-app`
- Purely SPA recommendation.
- Offer SSR with Angular Universal or with metaframework like `AnalogJS`
- Community packages can utilize Angular CLI to trigger post-script commands
  - `ng add @angular/material` to add Angular Material
  - `ng add @ngrx/store` to add NgRx
- Scaffolded project, modules, components, services, ... very easy

Template Language

Both Angular and React has their own way of writing templates.

Angular

- HTML-based template language
- Adding Angular-specific syntax (like `*ngFor`, `*ngIf`, or new control flow syntax `@if { }`, etc.)
- ✨NEW✨ Now you can also write HTML in your Angular TypeScript files!

React

- JSX-based template language
- Mostly JavaScript syntax for handling control flow (eg: `[].map(() => <Component />)`, `true ? <Component /> : null`, etc.)

React

```tsx
// src/app.tsx
import Welcome from './welcome';

export function App() {
  return (
    <Welcome />
  );
}

 export default App;
```

Angular

```ts
// src/app.component.ts
import { Component } from '@angular/core';
import { RouterModule } from '@angular/router';
import { WelcomeComponent } from './welcome.component';

@Component({
  selector: 'app-root',
  standalone: true,
  imports: [WelcomeComponent, RouterModule],
  template: `
    <app-welcome />
  `,
})
export class AppComponent {}
```

Angular Single-File Component

```ts
// src/app.component.ts
@Component({
  selector: 'app-root',
  standalone: true,
  imports: [WelcomeComponent, RouterModule],
  template: `
    <app-welcome />
  `,
})
 export class AppComponent {}
```

Angular Traditional Component

```ts
// src/app.component.ts
@Component({
  standalone: true,
  imports: [WelcomeComponent, RouterModule],
  selector: 'app-root',
  templateUrl: './app.component.html',
})
export class AppComponent {}
```

```css
/* src/app.component.css */
```

```html
<!-- src/app.component.html -->
<app-welcome />
```

React

```tsx
// src/app.tsx
export function App() {
  return (
    <div>
      Hello World
    </div>
  );
}
```

Angular

```ts
// src/app.component.ts
@Component({
  standalone: true,
  selector: 'app-root',
  template: `
    <div>
      Hello World
    </div>
  `,
})
export class AppComponent {}
```

React

```tsx
// src/sizer.tsx
export const Sizer = ({ size, onSizeChange }) => {
  return (
    <input value={size} onChange={onSizeChange} />
  );
};

// src/app.tsx
export const App = () => {
  const [size, setSize] = useState(50);

  const onSizeChange = (event) => {
    setSize(event.target.value);
  };

  return (
    <Sizer size={size} onSizeChange={onSizeChange} />
  );
}
```

Angular

```ts
// src/sizer.component.ts
@Component({
  standalone: true,
  selector: 'app-sizer',
  template: `
    <input [value]="size" (input)="onResize($event)" />
  `,
})
export class SizerComponent {
  @Input() size: number;
  @Output() sizeChange = new EventEmitter<number>();

  onResize(event: Event) {
    this.sizeChange.emit(
      (event.target as HTMLInputElement).value
    );
  }
}

// src/app.component.ts
@Component({
  ...,
  selector: 'app-root',
  template: `
    <app-sizer [size]="size"
```

React

```tsx
// src/app.tsx
export function App() {
  return (
    <input />
  );
}
```

Angular

```ts
// src/app.component.ts
@Component({
  standalone: true,
  selector: 'app-root',
  template: `
    <input />
  `,
})
export class AppComponent {}
```

Template-driven Forms

- Suitable for simple forms
- Uses two-way data binding
- Not very scalable

React

- Most control flow is done with JavaScript syntax
- Annoying restrictions:
  - No `if` statements - replaced with ternary operators or logical `&&` / `||` operators
  - No `for` loops - replaced with `map`
  - No `switch` statements - replaced multiple ternary operators
  - Have to return a single element - wrap in a `<Fragment>` if needed
  - Can't use `class` - use `className` instead
  - Can't use `style` - use `style` object instead

Angular

- Most control flow is done with Angular-specific syntax
- Annoying restrictions:
  - Have to remember new syntax (`*ngIf` , `*ngFor` , etc., or `@if` , `@for` , … with new control flow syntax)
  - `<ng-template>` , `<ng-container>` , `<ng-content>` , and other Angular-specific elements
  - `[<name>]` and `(<name>)` syntax for binding and events (`[(ngModel)]` for two-way binding)

React

```tsx
// src/app.tsx
export function App() {
  const isGoodbye = true;

  return (
    <h1>Hello World</h1>
  );
}
```

Angular

```ts
// src/app.component.ts
@Component({
  selector: 'app-root',
  template: `
    <h1>Hello World</h1>
  `,
})
export class AppComponent {
  isGoodbye = true;
 }
```

React

```tsx
// src/app.tsx
export function App() {
  const arrNames = ['Alice', 'Bob', 'Charlie'];

  return (
    <div>
      <h1>Hello World</h1>
    </div>
  );
}
```

Angular

```tsx
// src/app.component.ts
@Component({
  selector: 'app-root',
  template: `
    <div>
      <h1>Hello</h1>
    </div>
  `,
})
export class AppComponent {
  arrNames = ['Alice', 'Bob', 'Charlie'];
}
```

More stuff on Angular's `@for` directive.

Angular Routing

```
// app.component.ts
@Component({
  selector: 'app-root',
  imports: [RouterOutlet],
  template: `
    <div>
      <h1>App Component</h1>
      <router-outlet></router-outlet>
    </div>
  `,
})
export class AppComponent {}

// app.routes.ts
export const routes: RouterConfig = [];

// app.config.ts
import { routes } from './app.routes';
export const appConfig: ApplicationConfig = {
  providers: [provideRouter(routes)]
};
```

Angular Routing

```
// app.routes.ts
export const routes: RouterConfig = [
  { path: '', component: HomeComponent },
  { path: 'about', component: AboutComponent },
];
```

Angular Routing

There are more features that cannot be covered in this short introduction.

- Lazy Loading
- Route Guards
  - CanLoad - decide if the module should be loaded (used for lazy loading)
  - CanMatch - decide if the route should be matched, happens before the activation check
  - CanActivate - for the route itself / parent level navigation
  - CanActivateChild - for child routes / siblings navigation
  - CanDeactivate - for leaving the current route
  - Resolve - pre-fetch data before activating the route
  - *Of course, you can chain those guards together for reusable.*


ONE DOES NOT SIMPLY
FORM HANDLING
imgflip.com

Handling Form in React

- Doesn’t provide any built-in form handling
- Install multiple libraries to handle forms
  - `react-hook-form` (hot stuff nowaday)
  - `yup` / `zod` for validation

Handling Form in React

```
export const MyForm = () => {
  return (
    <form>
      <div>
        <label>Email</label>
        <input type="email" id="email" />
      </div>
      <div>
        <label>Password</label>
        <input type="password" id="password" />
      </div>
      <button type="submit">Submit</button>
    </form>
  );
};
```

Handling Form in Angular - Template-Driven Form

```
@Component({
  ...,
  template: `
    <form>
      <div>
        <label>Email</label>
        <input type="email" id="email" />
      </div>
      <div>
        <label>Password</label>
        <input type="password" id="password" />
      </div>
      <button type="submit">Submit</button>
    </form>
  `,
})
export class LoginComponent {}
```

Handling Form in Angular - Reactive Form

```
// import ReactiveFormsModule at the root module
// (if you plan to use it in multiple places)

@NgModule({
  ...,
  imports: [
    ...,
    // FormsModule, // template-driven
    ReactiveFormsModule,
  ],
...
})
export class AppModule {}
```

Handling Form in Angular - Form Group

```
@Component({
  ...,
  template: `
    <form>
      <input type="email" id="email" />
      <input type="password" id="password" />
      <button type="submit">Login</button>
    </form>
  `,
})
export class LoginComponent {}
```

Handling Form in Angular - Nested Form (Object)

```
@Component({
  ...,
  template: `
    <form [formGroup]="profileForm" (ngSubmit)="onSubmit()">
      <input type="text" id="name" formControlName="name" />
      <textarea id="address" formControlName="address"></textarea>
      <button type="submit">Update</button>
    </form>
  `,
})
export class LoginComponent {
  profileForm = new FormGroup({
    name: new FormControl(''),
    address: new FormControl(''),
  });

  onSubmit() {
    const payload = this.profileForm.value;
  }
}
```

Handling Form in Angular - Nested Form (Array)

```
@Component({
  ...,
  template: `
    <form [formGroup]="profileForm" (ngSubmit)="onSubmit()">
      <input type="text" id="name" formControlName="name" />
      <!-- If you don't want to have real DOM container -->
      <ng-container formGroupName="address">
        <input type="text" id="street" formControlName="street" />
        <input type="text" id="city" formControlName="city" />
        <input type="text" id="state" formControlName="state" />
        <input type="text" id="zip" formControlName="zip" />
      </ng-container>
      <button type="submit">Update</button>
    </form>
  `,
})
export class LoginComponent {
  profileForm = new FormGroup({
    name: new FormControl(''),
    address: new FormControl({
      street: new FormControl(''),
      city: new FormControl(''),
      state: new FormControl(''),
      zip: new FormControl(''),
    }),
```

Angular Form Builder 

```
export class LoginComponent {
  profileForm = new FormGroup({
    name: new FormControl(''),
    addresses: new FormControl(
      new Array().fill(new FormGroup({
        street: new FormControl(''),
        city: new FormControl(''),
        state: new FormControl(''),
        zip: new FormControl(''),
      }), 3),
    ),
  });
}
```

Angular Component 👉 Angular Template

```
export class LoginComponent {
  fb = inject(FormBuilder);

  profileForm = this.fb.group({
    name: new FormControl(''),
    addresses: this.fb.array(...),
  });

  get addresses() {
    return this.profileForm.get('addresses') as FormA
  }

  createAddressForm() {
    return this.fb.group({
      street: new FormControl(''),
      city: new FormControl(''),
      state: new FormControl(''),
      zip: new FormControl(''),
    });
  }

  addAddress() {...}

  removeAddress(at: number) {...}
}
```

```
@Component({
  template: `
    <form [formGroup]="profileForm" (ngSubmit)="onSubmit()">
      <input type="text" id="name" formControlName="name" />
      <ng-container formGroupName="address">
        <input type="text" id="street" formControlName="street" />
        <input type="text" id="city" formControlName="city" />
        <input type="text" id="state" formControlName="state" />
        <input type="text" id="zip" formControlName="zip" />
      </ng-container>
      <button type="submit">Update</button>
    </form>
  `,
})
export class LoginComponent {...}
```

Angular is a great framework

- Router features (lazy loading, route guards)
- HTTP Client and Interceptors
- In-depth Form Builder
- Dependency Injection (You are using it in React without knowing it)
- RxJS (Observables)
- Signal! (the future of Angular)
- Testing (Jest, Jasmine, Karma)
- SSR and metaframework!
- … and many more

Angular - An introduction

Thank you and see you in the next sessions!