Modern Angular

ngPune – 2024 15th Dec'24

https://github.com/Ashadeepa
https://www.linkedin.com/in/ashadeepa-debnath/
https://in.pinterest.com/ashie1709/ashieverse-creations/my-work/

Ashadeepa Debnath

ashadeepa.debnath@gmail.com

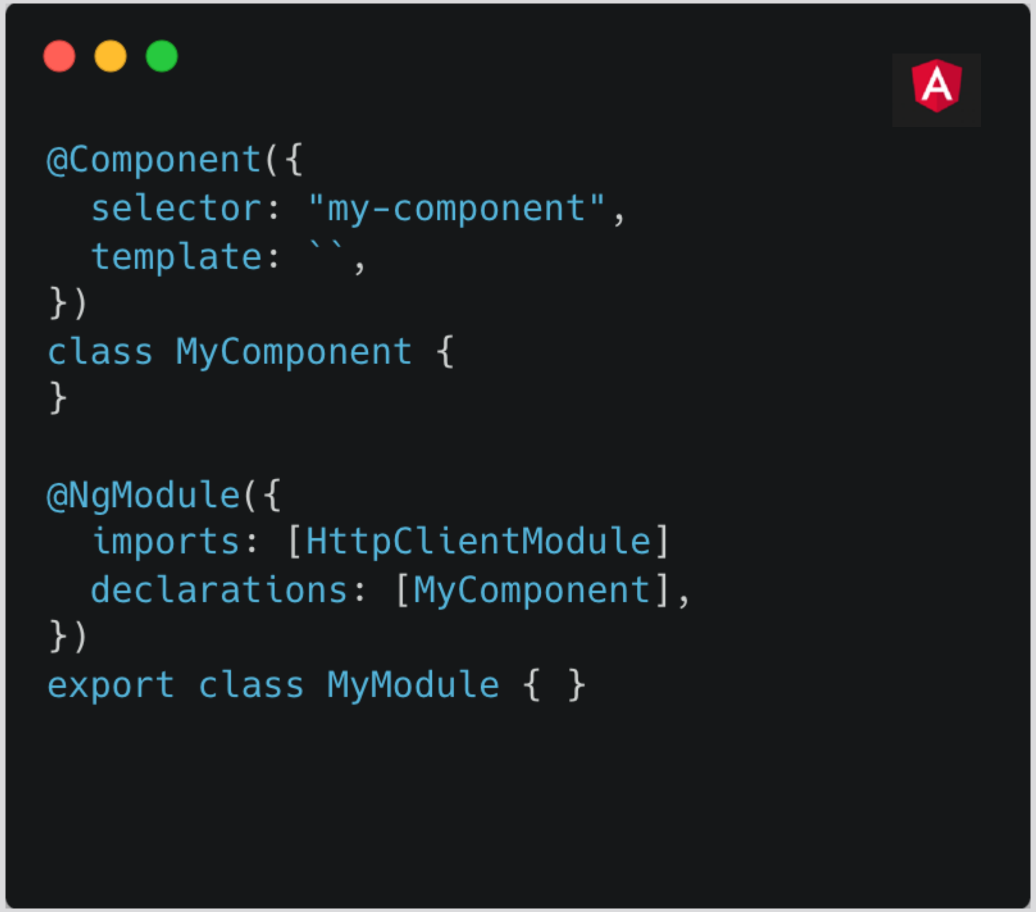
Senior Software Architect, Frontend
Nice Interactive Solutions Ltd, Pune

Angular Enthusiast, Passionate, Creative

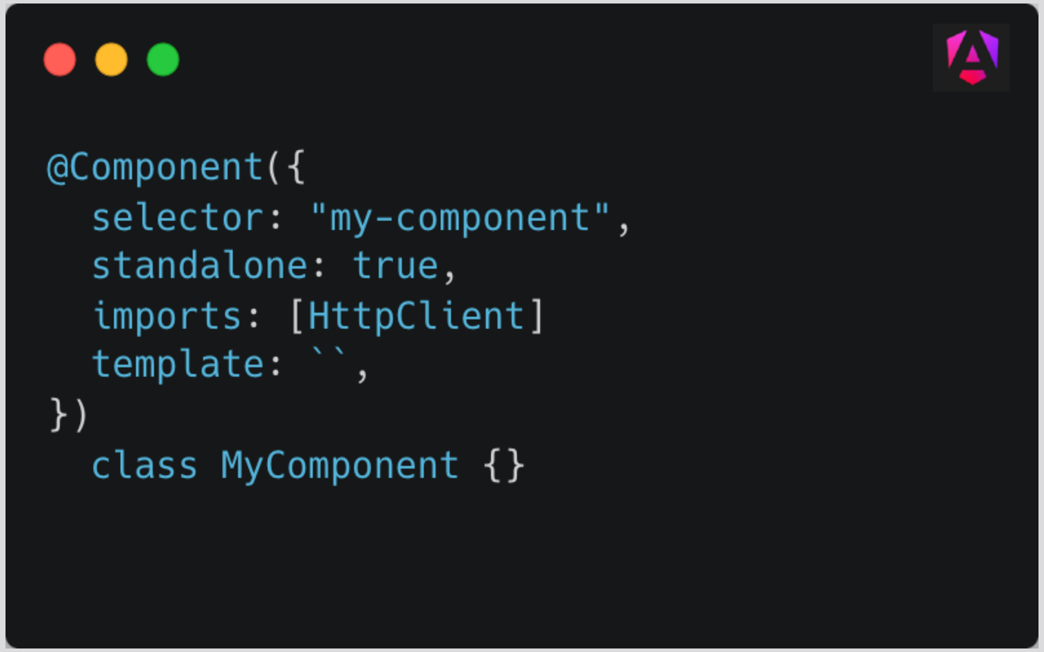
Angular Modern features 🚀

Feature	Introduced in	Purpose
Standalone Components	Angular 14+	Simplifies module management
Inject Function	Angular 14+	Simplifies dependency injection
Control Flow	Angular 17+	Better control logic in templates
Typed Forms	Angular 14+	Type-safe form handling
Inputs, Output, Queries	Angular 16+	Fine-grained reactivity

Standalone Components

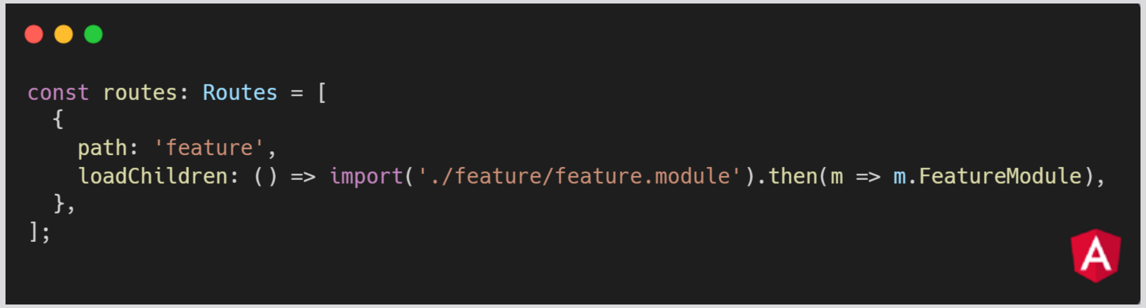


```
@Component({
  selector: "my-component",
  template: ``,
})
class MyComponent {
}

@NgModule({
  imports: [HttpClientModule]
  declarations: [MyComponent],
})
export class MyModule { }
```

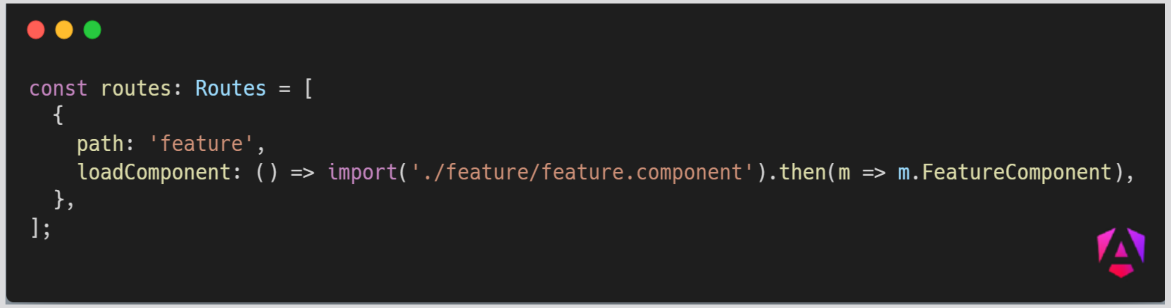


```
@Component({
  selector: "my-component",
  standalone: true,
  imports: [HttpClient]
  template: ``,
})
  class MyComponent {}
```

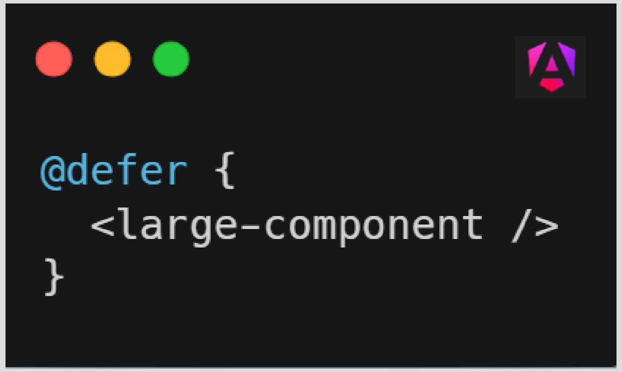
Lazy Loading via Components



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



```
const routes: Routes = [
  {
    path: 'feature',
    loadComponent: () => import('./feature/feature.component').then(m => m.FeatureComponent),
  },
];
```



```
@defer {
  <large-component />
}
```

https://angular.dev/guide/templates/defer

Standalone Components

Standard way to create components going forward

Standalone components are the default in Angular 19+

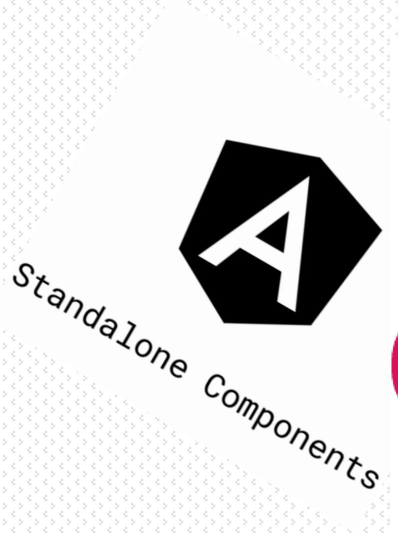
Tree- shakable

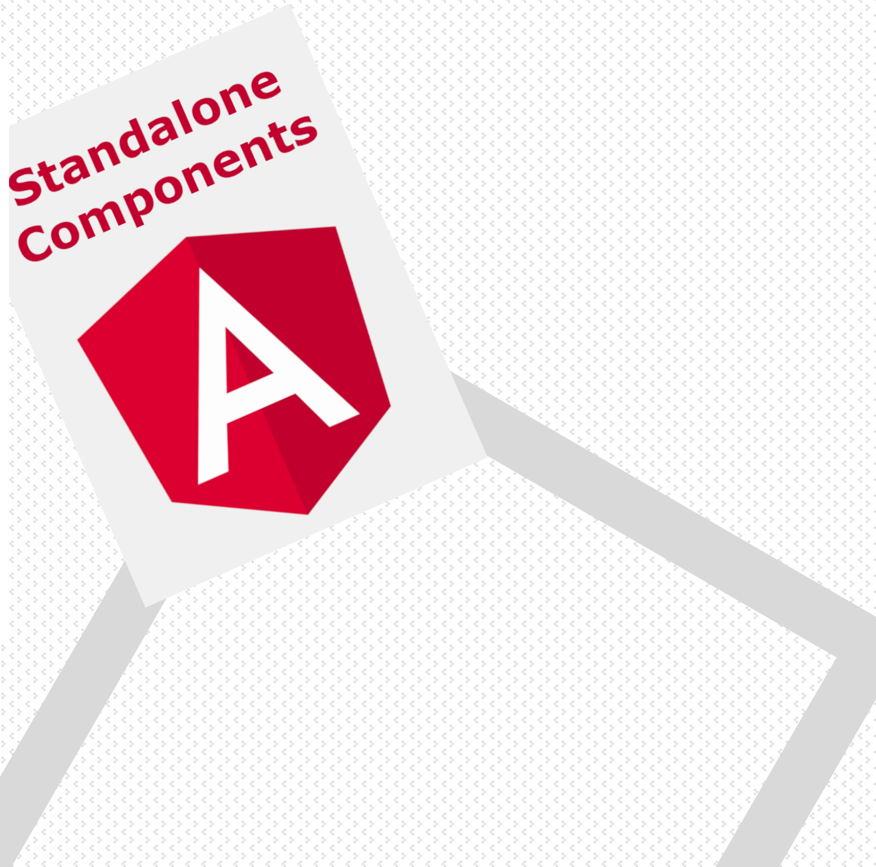
Dynamically load single components

Clear understanding of what the component depends on

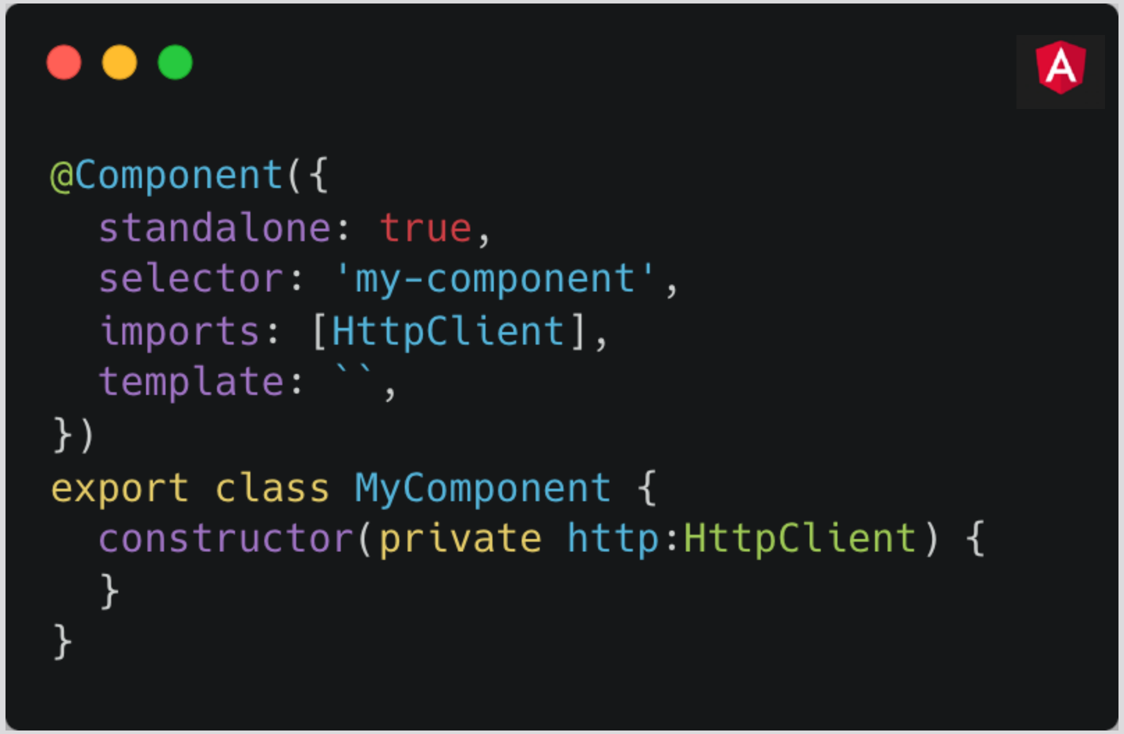
Standalone Components

```
ng generate @angular/core:standalone

ng generate @angular/core:migrate-router-to-lazy-load
```

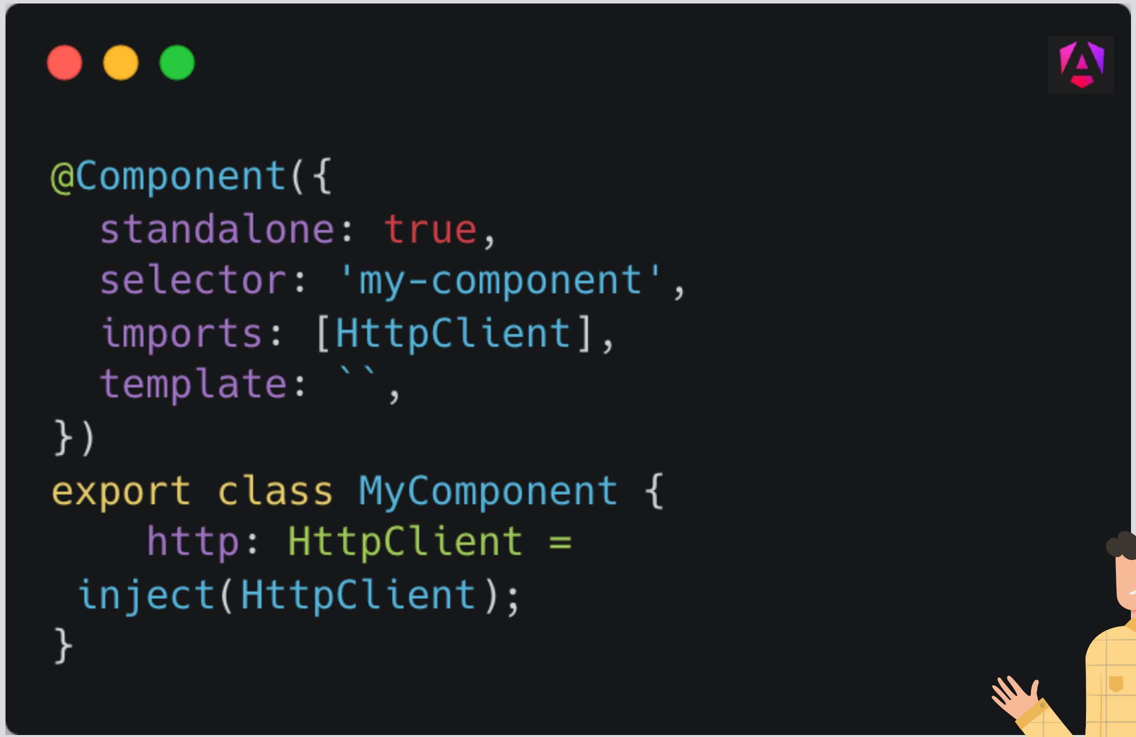




Inject Function



```
@Component({
  standalone: true,
  selector: 'my-component',
  imports: [HttpClient],
  template: ``,
})
export class MyComponent {
  constructor(private http:HttpClient) {
  }
}
```



```
@Component({
  standalone: true,
  selector: 'my-component',
  imports: [HttpClient],
  template: ``,
})
export class MyComponent {
    http: HttpClient =
 inject(HttpClient);
}
```

Inject Function

More explicit

Better for inheritance

Can only use inject() during the Injection context phase

Can access services without marking them as @Injectable

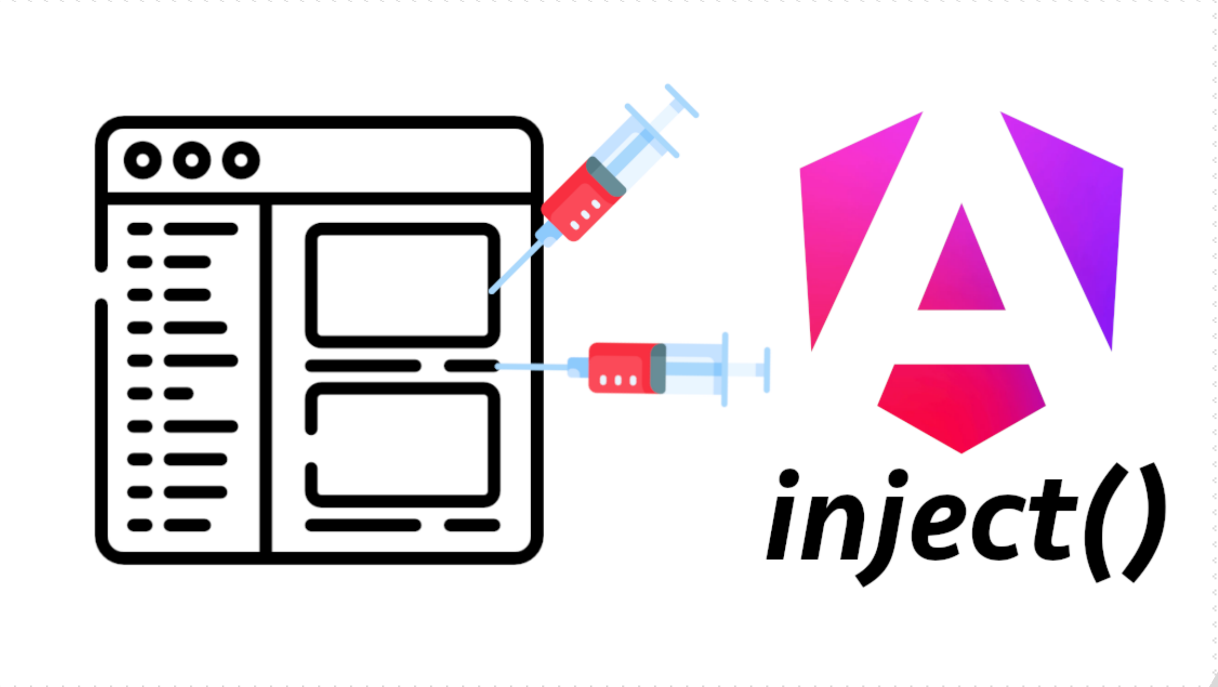
Easier to mock inside tests

https://angular.dev/guide/di/dependency-injection-context

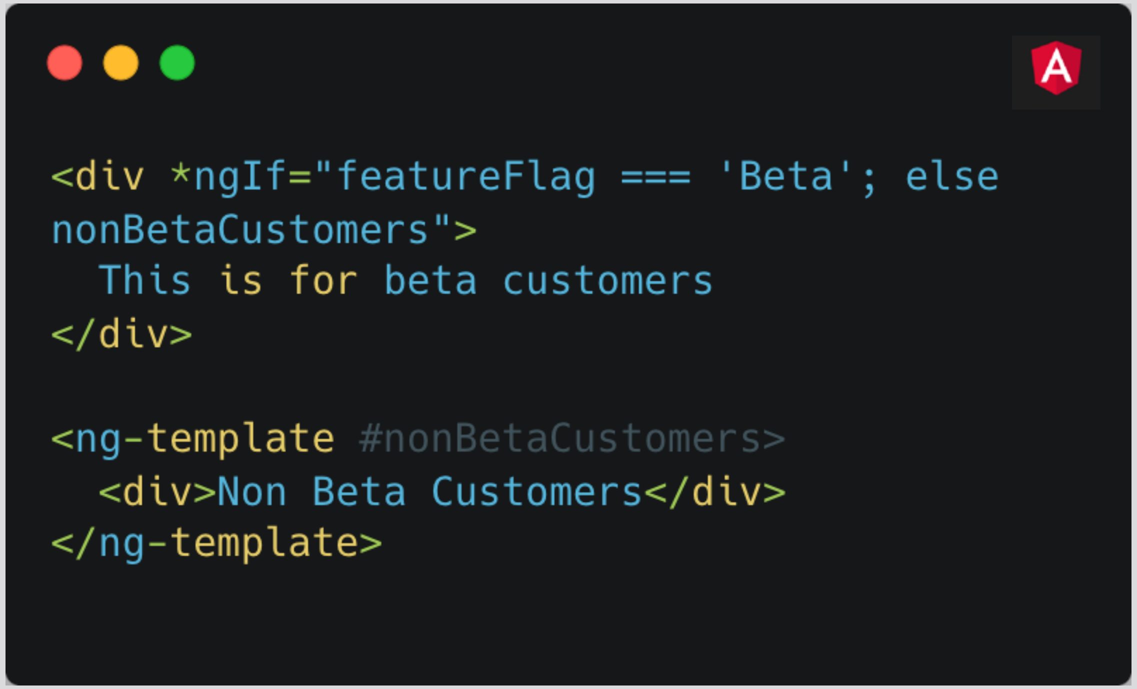
Inject Function

ng generate ngxtension:convert-di-to-inject

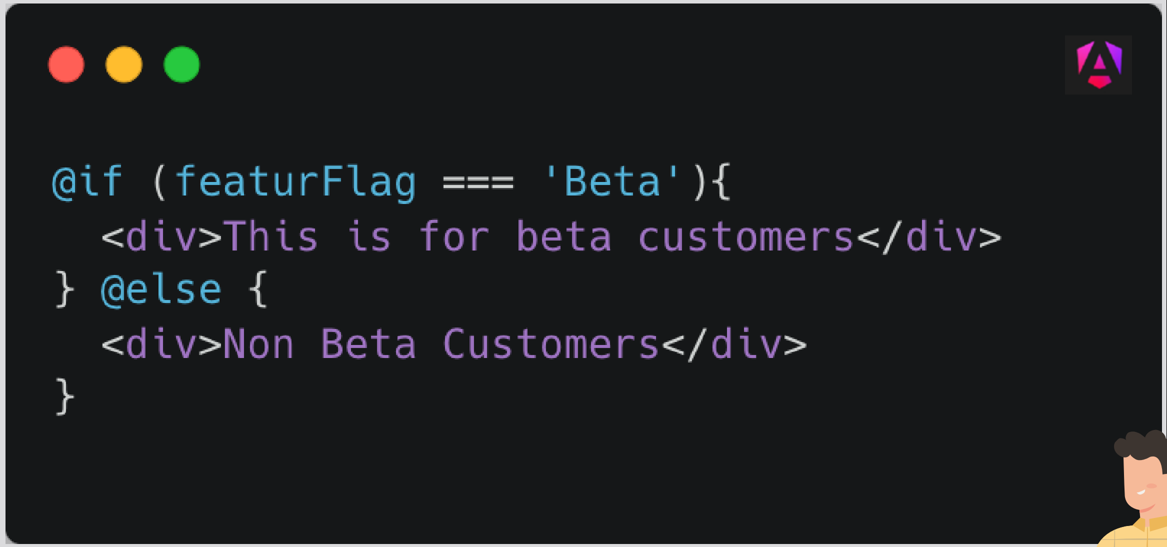
ng generate angular/core:inject-migration (Angular 18+)



Control Flow



```
<div *ngIf="featureFlag === 'Beta'; else
nonBetaCustomers">
  This is for beta customers
</div>

<ng-template #nonBetaCustomers>
  <div>Non Beta Customers</div>
</ng-template>
```



```
@if (featurFlag === 'Beta'){
  <div>This is for beta customers</div>
} @else {
  <div>Non Beta Customers</div>
}
```

Control Flow

Easier to read

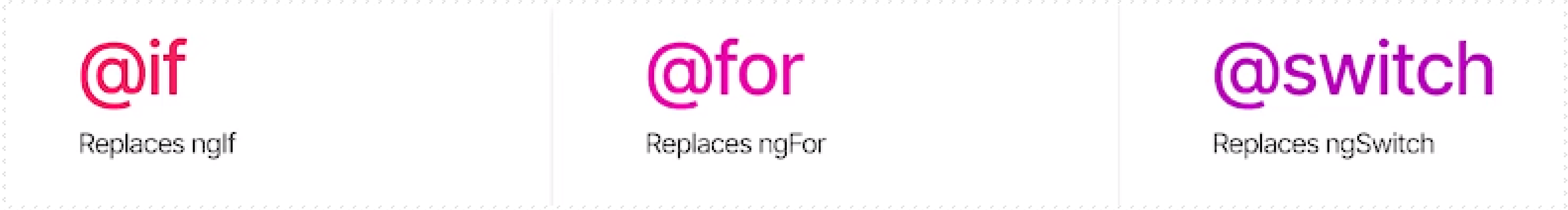
Easier to write

Performance gains

https://angular.dev/guide/di/dependency-injection-context

Control Flow

ng generate @angular/core:control-flow



@if

Replaces ngIf

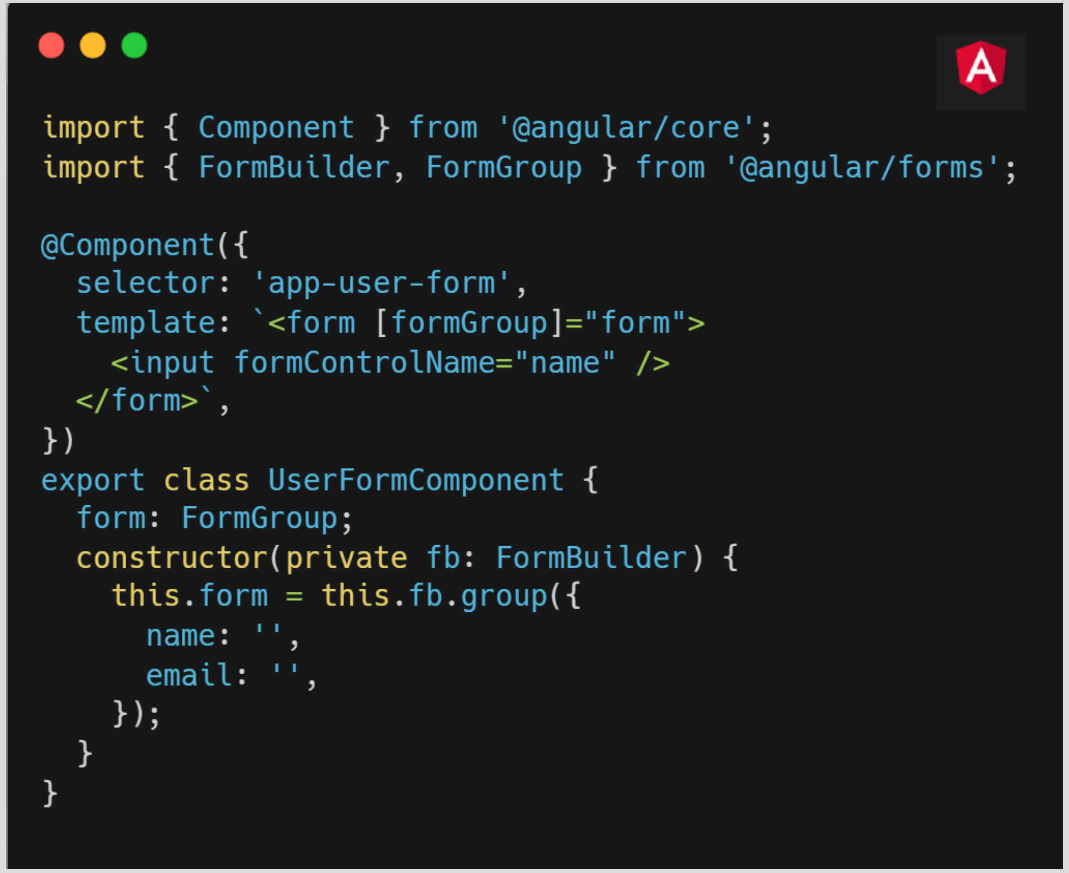
@for

Replaces ngFor

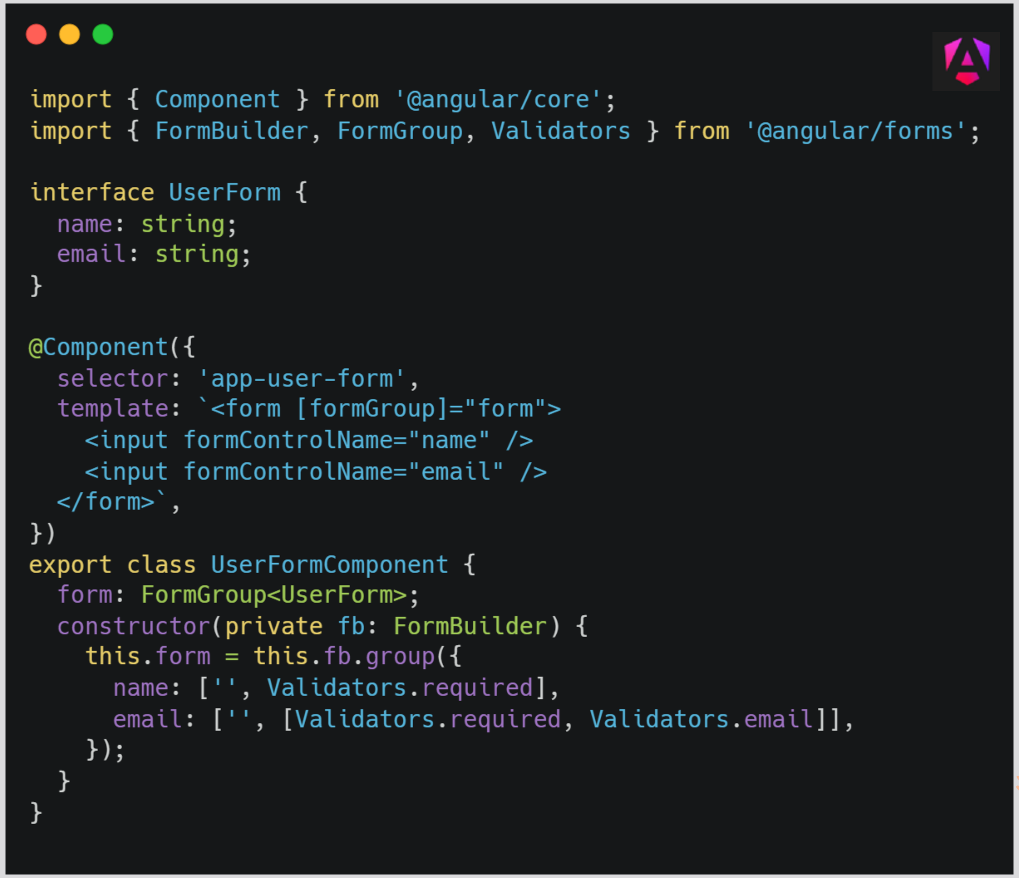
@switch

Replaces ngSwitch

Typed Forms



```
import { Component } from '@angular/core';
import { FormBuilder, FormGroup } from '@angular/forms';

@Component({
  selector: 'app-user-form',
  template: `<form [formGroup]="form">
    <input formControlName="name" />
  </form>`,
})
export class UserFormComponent {
  form: FormGroup;
  constructor(private fb: FormBuilder) {
    this.form = this.fb.group({
      name: '',
      email: '',
    });
  }
}
```

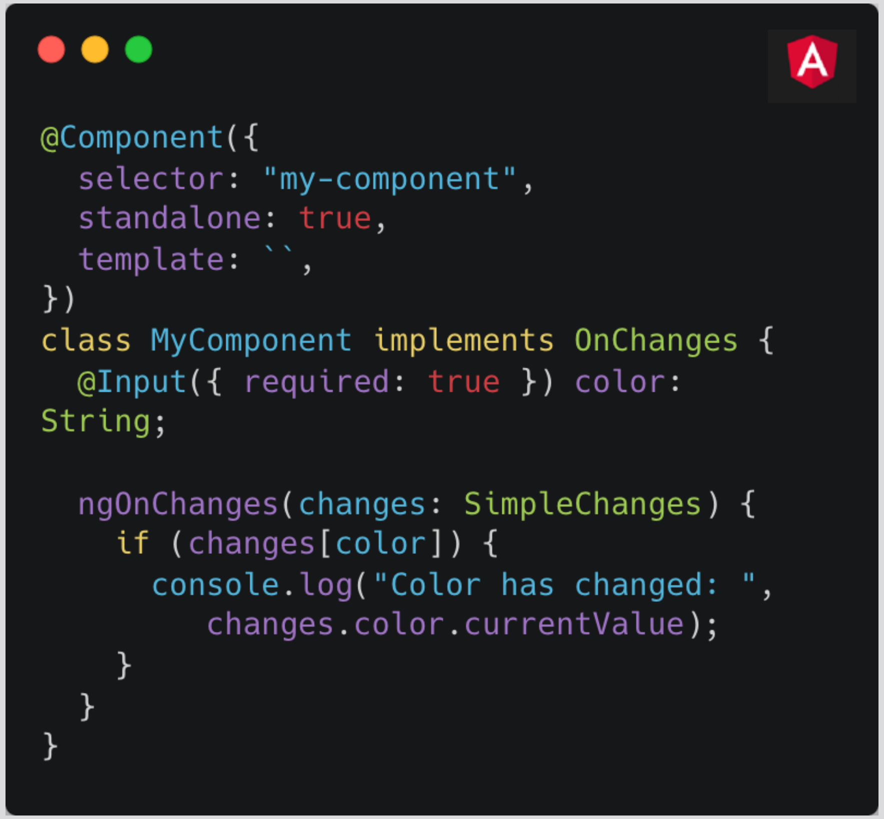


```
import { Component } from '@angular/core';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';

interface UserForm {
  name: string;
  email: string;
}

@Component({
  selector: 'app-user-form',
  template: `<form [formGroup]="form">
    <input formControlName="name" />
    <input formControlName="email" />
  </form>`,
})
export class UserFormComponent {
  form: FormGroup<UserForm>;
  constructor(private fb: FormBuilder) {
    this.form = this.fb.group({
      name: ['', Validators.required],
      email: ['', [Validators.required, Validators.email]],
    });
  }
}
```

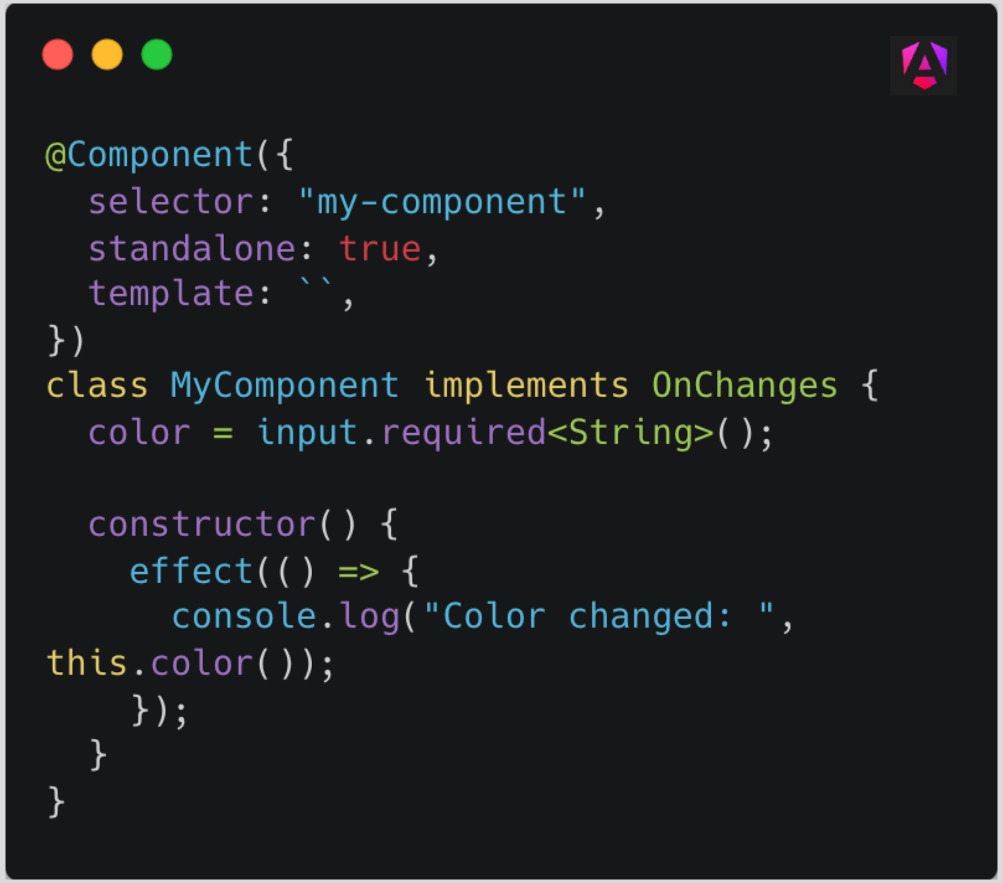
Typed Forms

- Type Safety
- Value Type Enforcement
- No More Type Assertions
- Nested Form Groups and Arrays
- Backward-compatible with existing Angular forms
- Form Structure Validation
- Eliminates the need for repetitive typecasting or *as* assertions.

Signal Inputs



```
@Component({
  selector: "my-component",
  standalone: true,
  template: ``,
})
class MyComponent implements OnChanges {
  @Input({ required: true }) color:
String;

  ngOnChanges(changes: SimpleChanges) {
    if (changes[color]) {
      console.log("Color has changed: ",
        changes.color.currentValue);
    }
  }
}
```



```
@Component({
  selector: "my-component",
  standalone: true,
  template: ``,
})
class MyComponent implements OnChanges {
  color = input.required<String>();

  constructor() {
    effect(() => {
      console.log("Color changed: ",
this.color());
    });
  }
}
```

Signal Inputs

Base on Signals

Gets away from annotations

Easier to detect changes

Read Only Signal

Can transform input value

Signal Inputs

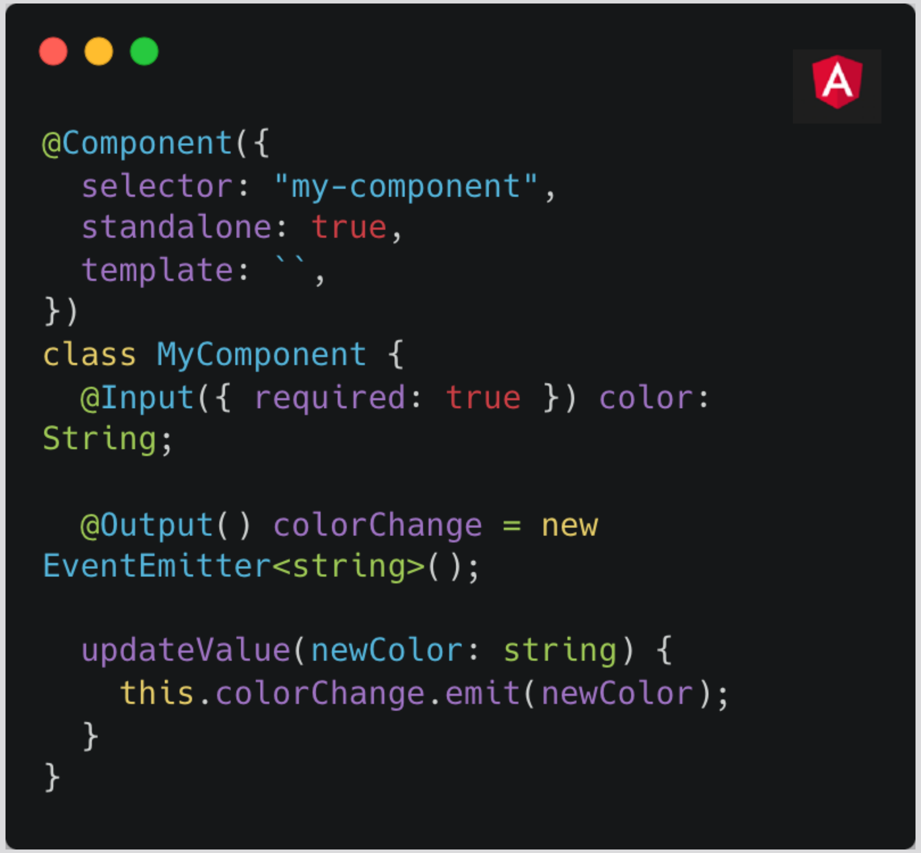
```
ng generate ngxtension:convert-signal-inputs
```

```
ng generate @angular/core:signal-input-migration (Angular 19+)
```

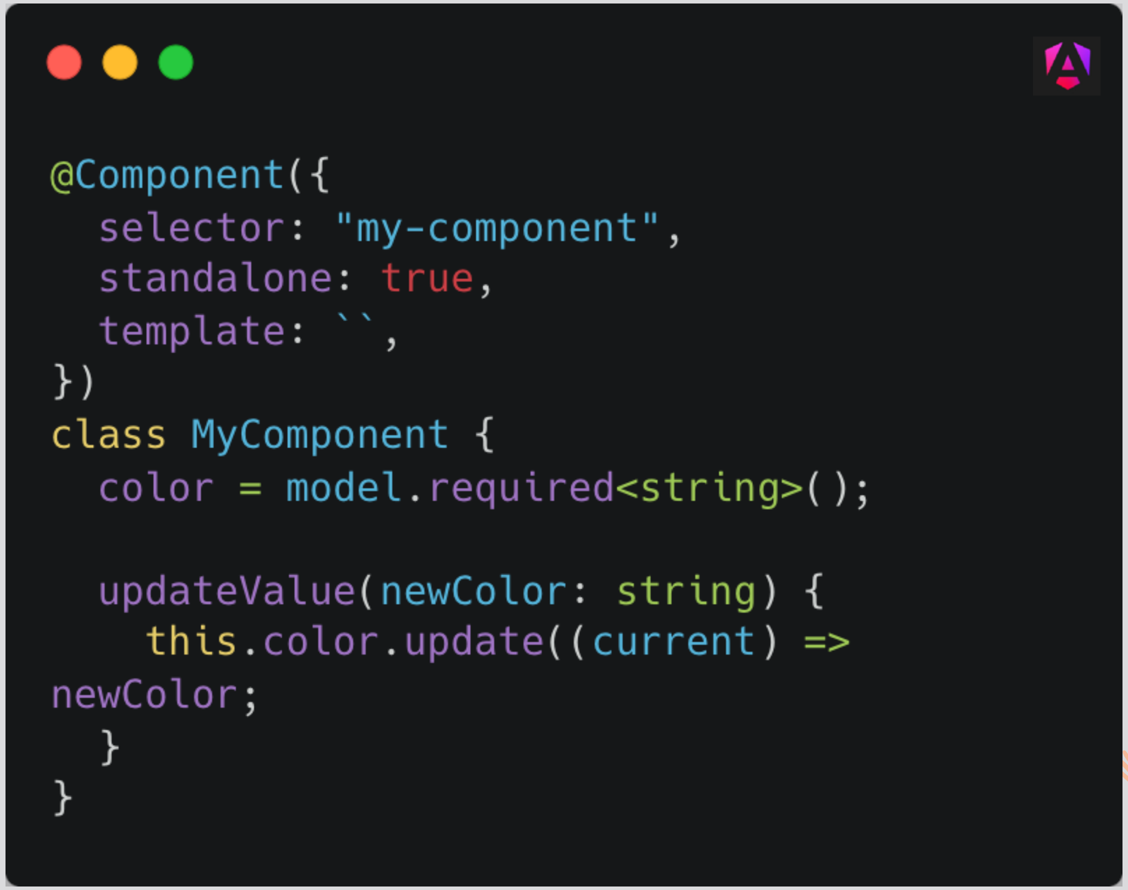
Angular

Signals

Signal Model Inputs



```
@Component({
  selector: "my-component",
  standalone: true,
  template: ``,
})
class MyComponent {
  @Input({ required: true }) color:
String;

  @Output() colorChange = new
EventEmitter<string>();

  updateValue(newColor: string) {
    this.colorChange.emit(newColor);
  }
}
```



```
@Component({
  selector: "my-component",
  standalone: true,
  template: ``,
})
class MyComponent {
  color = model.required<string>();

  updateValue(newColor: string) {
    this.color.update((current) =>
newColor;
  }
}
```

Signal Model Inputs

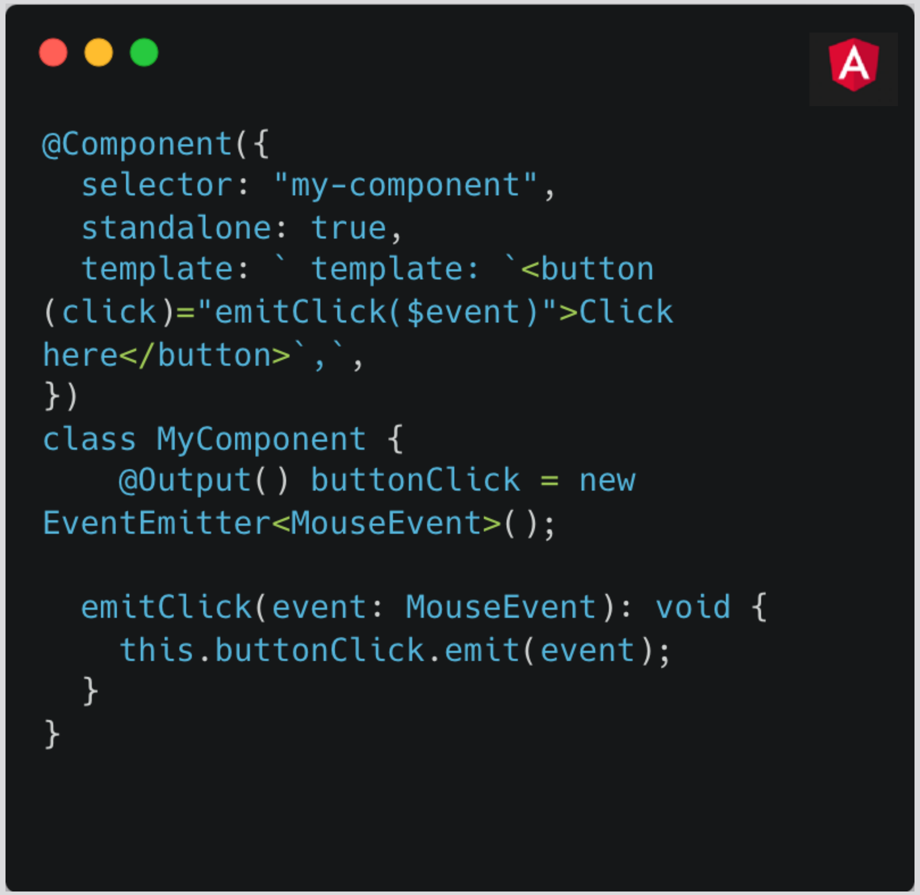
Base on Signals

Writeable Signal

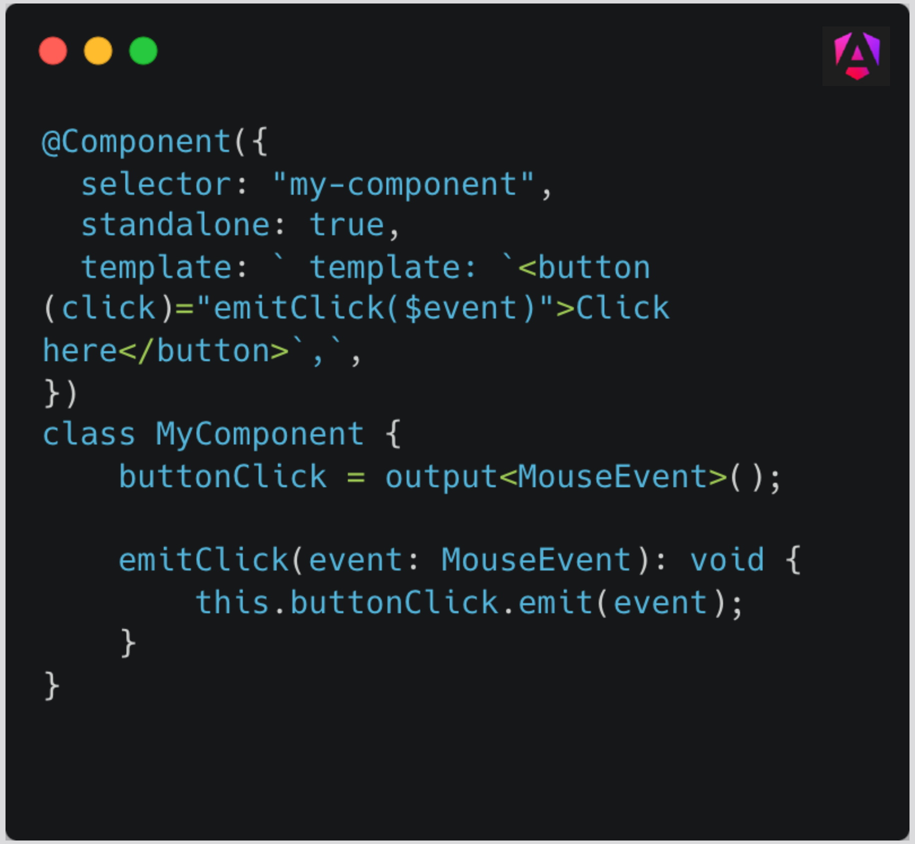
Two-way binding Signal

Cannot transform input value

Outputs



```
@Component({
  selector: "my-component",
  standalone: true,
  template: ` template: `<button
(click)="emitClick($event)">Click
here</button>`,`,
})
class MyComponent {
    @Output() buttonClick = new
EventEmitter<MouseEvent>();

  emitClick(event: MouseEvent): void {
    this.buttonClick.emit(event);
  }
}
```



```
@Component({
  selector: "my-component",
  standalone: true,
  template: ` template: `<button
(click)="emitClick($event)">Click
here</button>`,`,
})
class MyComponent {
    buttonClick = output<MouseEvent>();

    emitClick(event: MouseEvent): void {
        this.buttonClick.emit(event);
    }
}
```

Outputs

Not Signal Based

Consistency with input API

EventEmitters are no longer based on RxJS under the hood

Performance

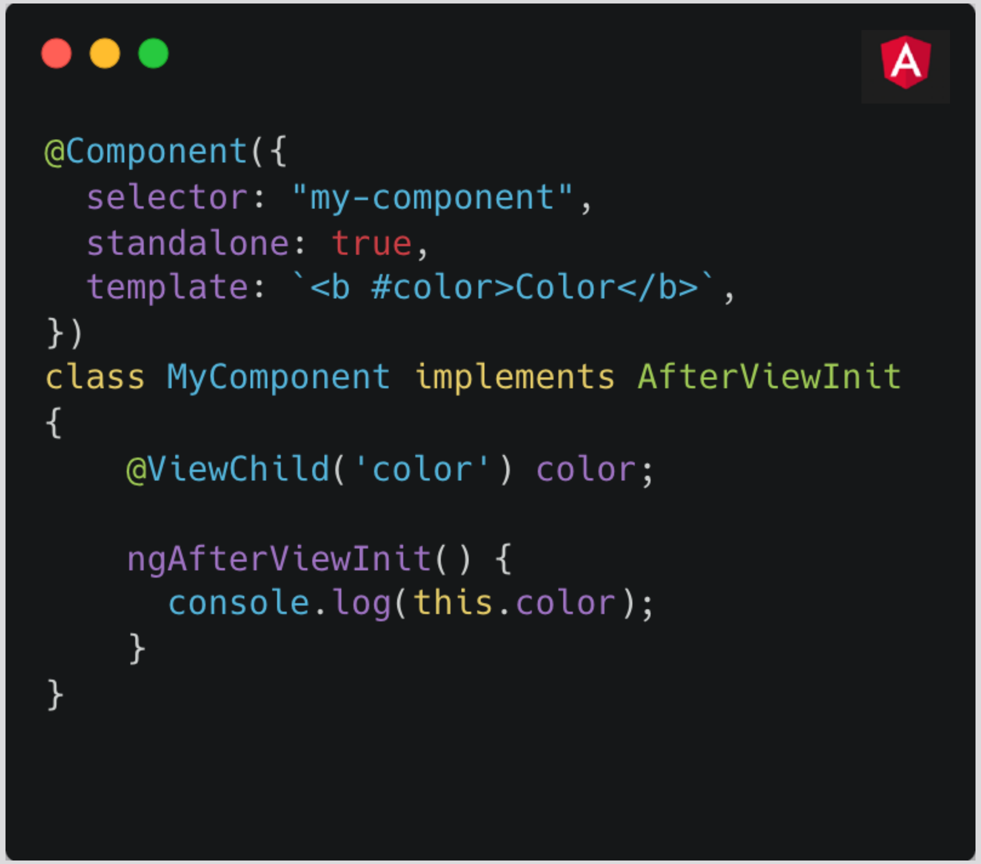
for RXJs - subscribe(), outputFromObservable(), outputToObservable() if needed

Outputs

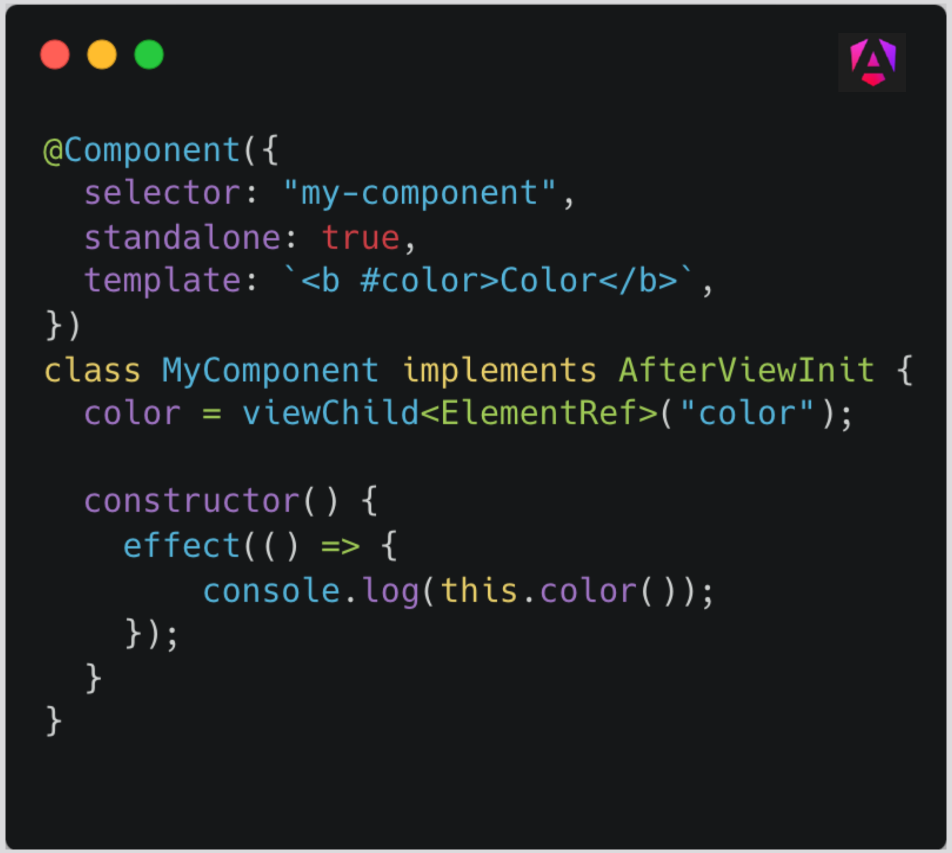
```
ng generate ngxtension:convert-outputs
```

```
ng generate @angular/core:output-migration  (Angular 19+)
```

Signal Queries



```
@Component({
  selector: "my-component",
  standalone: true,
  template: `<b #color>Color</b>`,
})
class MyComponent implements AfterViewInit
{
    @ViewChild('color') color;

    ngAfterViewInit() {
      console.log(this.color);
    }
}
```



```
@Component({
  selector: "my-component",
  standalone: true,
  template: `<b #color>Color</b>`,
})
class MyComponent implements AfterViewInit {
  color = viewChild<ElementRef>("color");

  constructor() {
    effect(() => {
        console.log(this.color());
    });
  }
}
```

Signal Queries

viewChild, viewChildren

contentChild, contentChildre n

No life cycle hooks needed

Signal Queries

```
ng generate ngxtension:convert-queries
```

```
ng generate @angular/core:signal-queries-migration
```
(Angular 19+)

https://blog.angular-university.io/angular-signal-components/

Feature	Introduced in	Purpose
Functional Router Guards	Angular 15+	Cleaner routing
SSR Hydration	Angular 16+	Faster server-side rendering
Functional Router Guards	Angular 15+	Cleaner routing
Angular CLI Enhancements	Angular 14+	Easier debugging and build optimization
Improved Performance with Ivy and Ahead-of-Time Compilation	Angular 14+	Faster initial load times and reduced JavaScript bundle size

Conclusion

Key Takeaways

- Typed Forms improve code quality and developer productivity by leveraging TypeScript features.
- Signals optimize state management performance with fine-grained reactivity.
- Standalone Components simplify the architecture and make modular development easier.
- Injection Functions provide more flexible and dynamic dependency injection.

Conclusion

Generic Best Practices for Angular Development

- Keep Angular updated to take advantage of the latest features.
- Structure Your Application Modularly
- Keep Components Small and Reusable
- Optimize Performance
- Follow Angular Style Guide
- Implement Proper Error Handling
- Ensure Proper Testing Coverage
- Use Angular Material for Consistent UI
- Keep Security in Mind.

Modulare Archittume Typed Forms
Compoent Russabion Businss Logic
Modulare Architecture Typed Forms
Rertusabity Component Oxtizament
Compencteed Business Magic Perfonce-based
Service-Based Business Mobisation
Service-Based Monaigmment
Security Secuity Trestings
Sucrity Testing Security Practices

THANK YOU

https://github.com/Ashadeepa
https://www.linkedin.com/in/ashadeepa-debnath/