Angular: Type vs Interface

Understand the Difference
Write Better Code

Keyur Jivani
.NET core | Angular | SQL | AWS

Angular

Introduction

- TypeScript gives us two powerful tools for type-checking: type and interface
- Both are used to define the shape of data, yet they have key differences
- Knowing when to use one over the other helps in creating clean, scalable Angular codebases

Keyur Jivani
.NET core | Angular | SQL | AWS

Angular

What is an Interface?

- Used to define the structure of an object
- Supports **declaration merging**
- Extendable using extends
- Preferred when modeling the shape of class-like objects

```
1 interface User {
2   id: number;
3   name: string;
4 }
```

Keyur Jivani
.NET core | Angular | SQL | AWS

Angular

What is a Type?

- **Type** is a type alias — used to define union, intersection, primitives, tuples, etc.
- More flexible and composable
- Can represent complex constructs

```
type User = {
  id: number;
  name: string;
};
type APIResponse = User | Error;
```

Keyur Jivani
.NET core | Angular | SQL | AWS

Angular

Key Differences

Feature	Interface	Type
Declaration merging	✅ Yes	❌ No
Extending multiple	✅ Yes	✅ Yes
Unions & Intersections	❌ No	✅ Yes
Classes implementation	✅ Yes	✅ Yes
Better for React Props	✅ Yes	✅ Yes

Keyur Jivani
.NET core | Angular | SQL | AWS

Angular

When to Use What?

- Use **interface** when:
  - You expect multiple declarations/extensions
  - Working with class-based code (OOP)

- Use **type** when:
  - You need to use union or intersection types
  - Creating utility types or complex type expressions

Keyur Jivani

.NET core | Angular | SQL | AWS

Angular

Real-World Angular Use Case

- Defining a model:

```typescript
interface Product {
  id: number;
  name: string;
  price: number;
}
```

- Defining a union type for a state:

```typescript
type ProductState = 'available' | 'out-of-stock' | 'discontinued';
```

Keyur Jivani
.NET core | Angular | SQL | AWS

Angular

Advanced Techniques

- Mapped types:

```
1 type ReadOnly<T> = {
2   readonly [K in keyof T]: T[K];
3 };
```

- Combining with utility types:

```
1 type PartialProduct = Partial<Product>;
```

Keyur Jivani
.NET core | Angular | SQL | AWS

Angular

Common Mistakes

- Using interface where type is better for unions
- Overusing type and losing clarity
- Forgetting that interface supports declaration merging

Keyur Jivani
.NET core | Angular | SQL | AWS

Angular

Performance & Tooling

- Both compile to zero in JS — no runtime cost
- Interfaces offer better readability and tooling in some IDEs
- Types offer more flexibility in functional patterns

Keyur Jivani
.NET core | Angular | SQL | AWS

Angular

Best Practices

- ✅ Use interface for data models and class contracts
- ✅ Use type for unions, intersections, and utility types
- ✅ Keep types simple and expressive
- ✅ Avoid unnecessary abstraction

Keyur Jivani
.NET core | Angular | SQL | AWS

Angular

Summary – Cheat Sheet

✅ interface: Best for extending and declaring shapes

✅ type: Best for flexibility and unions

🧠 Think of interface as a structure, type as a toolkit

✨ Use what communicates intention clearly to others (and your future self)

Keyur Jivani
.NET core | Angular | SQL | AWS

Angular

Thank You

Questions? Thoughts? Experiences to share?

Let’s continue the conversation.

Keyur Jivani
.NET core | Angular | SQL | AWS

Angular