POWER OF GENERICS IN TYPESCRIPT

SELVA PRASATH SELVAMANI

BOUNTEOUS

TypedCoders

Typescript Generics Exists

TypedCoders

SITUATION

TypedCoders

CREATE A FUNCTION TO RETURN FIRST NAME IN AN ARRAY

```
function getFirstStringArray(arr: string[]): string { return arr[0];
}
```

TypedCoders

CREATE A FUNCTION TO RETURN FIRST NUMBER IN AN ARRAY

```
function getFirstNumberFromAnArray(arr: number[]): number { return arr[0];
}
```

TypedCoders

CREATE A FUNCTION TO RETURN FIRST OBJECT FROM AN ARRAY

?

TypedCoders

GetFirstNumber

GetFirstString

GetFirstObject

GetFirstBoolean

TypedCoders

GENERICS THE SAVIOR

TypedCoders

WHAT IS GENERIC

GENERICS ALLOW US TO CREATE REUSABLE COMPONENTS THAT WORK WITH A VARIETY OF TYPES RATHER THAN A SINGLE ONE

TypedCoders

UP NEXT

- Generics with condition
- Multi parameters
- Typing object parameters

TypedCoders

WITH OUT GENERICS

TypedCoders

WITH GENERICS

TypedCoders

GENERICS IN FRONTEND

TypedCoders

BACKEND

TypedCoders

ADVANTAGES

- **Type safety**: Detects type errors at compile time while allowing flexibility.
- **Reusability**: One function or class works with multiple data types.
- **Type inference**: Preserves and infers types, improving IDE support (autocomplete, tooltips).
- **DRY principle**: Avoids writing multiple versions of similar code.
- **Constraints support**: Use extends to restrict types (e.g., T extends { id: number }).
- **Works with utility types**: Enables powerful patterns using Partial<T>, etc.
- **Improved maintainability**: Easier to update logic without touching type-specific code.

TypedCoders

DIS ADVANTAGES

- **Added complexity**: Generic syntax can be hard to read, especially for beginners.
- **Overengineering**: Sometimes unnecessary for simple use cases.

TypedCoders

WHAT CAN YOU DO WITH GENERIC

TypedCoders

GITHUB REPO

TypedCoders