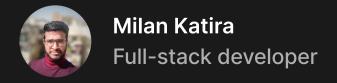


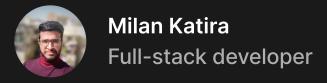
TYPESCRIPT GENERIC TYPES







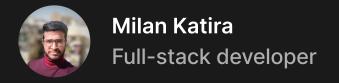
Typescript Generics



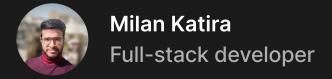
Swipe →

Generics in TypeScript allow you to write reusable code that can work with different data types. This makes your code more flexible and less repetitive, because you don't have to write separate functions or classes for each data type that you want to work with.

Generics are denoted using angle brackets
< > and can be used to define a type
parameter



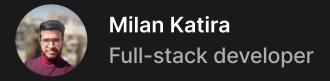
```
function identity<T>(arg: T): T {
  return arg;
// Usage
let result = identity("hello");
console.log(result); // "hello"
result = identity(42);
console.log(result); // 42
```



Swipe →

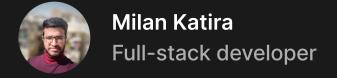


With classes and interfaces





```
interface Container<T> {
  add(item: T): void;
  get(index: number): T;
  size(): number;
}
```



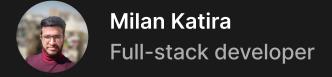
```
class StringContainer implements Container<string> {
  private items: string[] = [];
  add(item: string): void {
    this.items.push(item);
  get(index: number): string {
    return this.items[index];
  }
  size(): number {
    return this.items.length;
}
```



Milan Katira Full-stack developer

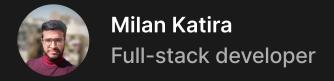
Swipe →

```
const container = new StringContainer();
container.add("Hello");
container.add("World");
console.log(container.get(0)); // "Hello"
console.log(container.get(1)); // "World"
console.log(container.size()); // 2
```



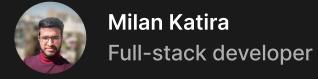


generic to enforce constraints





```
function sum<T extends number>(numbers: T[]): number {
  let result = 0;
  for (let number of numbers) {
    result += number;
  }
  return result;
}
const numbers = [1, 2, 3, 4, 5];
const result = sum(numbers); // result will be 15
const strings = ["hello", "world"];
const result = sum(strings); // Type error: Argument
of type 'string[]' is not assignable to parameter of
type 'number[]'.
```



THANKS FOR READING 800

I hope you enjoyed and find this post useful.

