Table of Contents

# Bab 10: Utility Types di TypeScript

## Penjelasan Materi

Utility Types adalah kumpulan tipe bawaan TypeScript yang memungkinkan kita untuk melakukan transformasi umum pada tipe data. Mereka membantu kita memanipulasi dan membuat tipe baru berdasarkan tipe yang sudah ada, membuat kode lebih DRY (Don’t Repeat Yourself) dan type-safe.

## Analogi yang Mudah Dipahami

Bayangkan Utility Types seperti alat-alat dalam kotak perkakas: - Partial<T> seperti pensil yang bisa menghapus sebagian gambar - Required<T> seperti stempel yang membuat semua bagian wajib diisi - Pick<T, K> seperti gunting yang memotong bagian yang diinginkan - Omit<T, K> seperti penghapus yang menghilangkan bagian tertentu - Record<K, T> seperti template yang membuat struktur seragam

## Point Penting

1. **Basic Utility Types**
   * Partial
   * Required
   * Readonly
   * Record<K,T>
2. **Selection Utility Types**
   * Pick<T,K>
   * Omit<T,K>
   * Extract<T,U>
   * Exclude<T,U>
3. **Function Utility Types**
   * ReturnType
   * Parameters
   * ConstructorParameters
   * ThisParameterType
4. **String Manipulation Types**
   * Uppercase
   * Lowercase
   * Capitalize
   * Uncapitalize
5. **Advanced Utility Types**
   * NonNullable
   * InstanceType
   * ThisType
   * Template Literal Types

## Contoh Kode dan Penjelasan

```typescript // 1. Basic Interface interface User { id: number; name: string; email: string; age?: number; address: { street: string; city: string; country: string; } }

// 2. Partial function updateUser(userId: number, updates: Partial) { // Memperbarui sebagian data user const user: User = { id: userId, name: “John”, email: “john@example.com”, address: { street: “123 Main St”, city: “New York”, country: “USA” } }; return { …user, …updates }; }

// Penggunaan: updateUser(1, { name: “Jane” }); // Valid updateUser(1, { nickname: “J” }); // Error: ‘nickname’ tidak ada di User

// 3. Required type StrictUser = Required; // Sekarang semua properti wajib, termasuk age

// 4. Pick<T,K> type UserContact = Pick<User, “email” | “address”>; // Hanya mengambil email dan address

// 5. Omit<T,K> type UserWithoutId = Omit<User, “id”>; // Menghilangkan properti id

// 6. Record<K,T> type CountryCode = “US” | “UK” | “ID”; type CountryInfo = Record<CountryCode, { name: string; currency: string; }>;

const countries: CountryInfo = { US: { name: “United States”, currency: “USD” }, UK: { name: “United Kingdom”, currency: “GBP” }, ID: { name: “Indonesia”, currency: “IDR” } };

// 7. Extract dan Exclude type StringOrNumber = string | number; type OnlyString = Extract<StringOrNumber, string>; // string type ExcludeString = Exclude<StringOrNumber, string>; // number

// 8. ReturnType dan Parameters function fetchUser(id: number): Promise { return Promise.resolve({ id, name: “John”, email: “john@example.com”, address: { street: “123 Main St”, city: “New York”, country: “USA” } }); }

type FetchUserReturn = ReturnType; // Promise type FetchUserParams = Parameters; // [number]

// 9. Template Literal Types dengan Utility Types type EventName = “click” | “focus” | “blur”; type Handler = `on${Capitalize}`;

type EventHandlers = { [K in EventName as Handler]: () => void; };

// Hasil: // { // onClick: () => void; // onFocus: () => void; // onBlur: () => void; // }

// 10. Praktik: Form Validation interface FormField { value: T; error?: string; required: boolean; validate: (value: T) => boolean; }

type FormFields = Record<string, FormField>;

interface LoginForm { username: FormField; password: FormField; rememberMe: FormField; }

const loginForm: LoginForm = { username: { value: ““, required: true, validate: (value) => value.length >= 3 }, password: { value:”“, required: true, validate: (value) => value.length >= 8 }, rememberMe: { value: false, required: false, validate: (value) => typeof value ===”boolean” } };

// 11. Advanced Example: API Response Handler type ApiResponse = { data: T; status: number; message: string; timestamp: Date; };

type ApiError = { code: number; message: string; details?: Record<string, unknown>; };

// Utility type untuk mengekstrak success response type SuccessResponse = Extract<T, { status: 200 }>;

// Utility type untuk mengekstrak error response type ErrorResponse = Extract<T, { status: 400 | 401 | 403 | 404 | 500 }>;

// Implementasi async function handleApiResponse( promise: Promise<ApiResponse> ): Promise<T | ApiError> { try { const response = await promise; if (response.status === 200) { return response.data; } throw { code: response.status, message: response.message }; } catch (error) { return { code: 500, message: “Internal Server Error”, details: { error } }; } } ```

## Cara Kerja Utility Types

1. **Type Transformation**:
   * Type inference
   * Conditional types
   * Mapped types
   * Template literal types
2. **Type Distribution**:
   * Union distribution
   * Intersection distribution
   * Conditional distribution
3. **Type Inference**:
   * Inference in generics
   * Inference in conditionals
   * Inference in mapped types

## Tips dan Trik

1. **Kombinasi Utility Types**

* // ✅ Gunakan kombinasi utility types untuk transformasi kompleks  
  type ReadonlyPartial<T> = Readonly<Partial<T>>;  
  type PickRequired<T, K extends keyof T> = Required<Pick<T, K>>;

1. **Custom Utility Types**

* // ✅ Buat utility types custom untuk kebutuhan spesifik  
  type DeepPartial<T> = {  
   [P in keyof T]?: T[P] extends object  
   ? DeepPartial<T[P]>  
   : T[P];  
  };

1. **Type Guards dengan Utility Types**

* // ✅ Gunakan utility types dalam type guards  
  function isNonNullable<T>(value: T): value is NonNullable<T> {  
   return value !== null && value !== undefined;  
  }

## Kesalahan yang Sering Dilakukan Pemula

1. **Overuse of Utility Types**

* // ❌ Buruk: Terlalu banyak transformasi  
  type ComplexType = Partial<Required<Pick<Omit<User, "id">, "name" | "email">>>;  
    
  // ✅ Baik: Lebih sederhana dan jelas  
  interface UserUpdate {  
   name?: string;  
   email?: string;  
  }

1. **Incorrect Usage of Partial**

* // ❌ Buruk: Partial tidak bekerja secara deep  
  type DeepUser = Partial<User>;  
  // address masih required!  
    
  // ✅ Baik: Gunakan DeepPartial custom  
  type DeepPartialUser = DeepPartial<User>;

1. **Misunderstanding Type Distribution**

* // ❌ Buruk: Tidak memahami distribusi union  
  type UnionType = string | number;  
  type Wrong = Partial<UnionType>; // Error  
    
  // ✅ Baik: Gunakan pada object types  
  type Correct = Partial<{ str: string; num: number }>;

### Solusi:

1. Pahami kapan menggunakan utility types
2. Buat custom utility types untuk kasus spesifik
3. Gunakan kombinasi utility types dengan bijak
4. Perhatikan type distribution
5. Dokumentasikan penggunaan utility types yang kompleks