Table of Contents

# Bab 12: Testing di TypeScript

## Penjelasan Materi

Testing di TypeScript adalah proses penting untuk memastikan kualitas dan keandalan kode. TypeScript menyediakan fitur type-safety yang membantu dalam testing, dan dapat diintegrasikan dengan berbagai framework testing populer seperti Jest, Mocha, dan Jasmine. Testing membantu kita menemukan bug lebih awal, memastikan kode berfungsi sesuai harapan, dan memudahkan refactoring.

## Analogi yang Mudah Dipahami

Bayangkan testing seperti proses quality control di pabrik: - Unit Testing seperti memeriksa setiap komponen secara individual - Integration Testing seperti memastikan komponen bekerja sama dengan baik - End-to-End Testing seperti menguji produk akhir secara keseluruhan - Mock Objects seperti prototype untuk pengujian - Test Coverage seperti checklist pemeriksaan kualitas

## Point Penting

1. **Unit Testing**
   * Test setup
   * Test runners
   * Assertions
   * Test isolation
2. **Integration Testing**
   * Component integration
   * API testing
   * Database testing
   * Service testing
3. **End-to-End Testing**
   * Browser automation
   * User flow testing
   * Performance testing
   * Cross-browser testing
4. **Test Doubles**
   * Mocks
   * Stubs
   * Spies
   * Fakes
5. **Testing Best Practices**
   * Test organization
   * Test coverage
   * Test maintenance
   * Continuous Integration

## Contoh Kode dan Penjelasan

```typescript // 1. Basic Unit Test dengan Jest // calculator.ts export class Calculator { add(a: number, b: number): number { return a + b; }

subtract(a: number, b: number): number {  
 return a - b;  
}  
  
multiply(a: number, b: number): number {  
 return a \* b;  
}  
  
divide(a: number, b: number): number {  
 if (b === 0) {  
 throw new Error("Division by zero");  
 }  
 return a / b;  
}

}

// calculator.test.ts import { Calculator } from ‘./calculator’;

describe(‘Calculator’, () => { let calculator: Calculator;

beforeEach(() => {  
 calculator = new Calculator();  
});  
  
describe('add', () => {  
 it('should add two numbers correctly', () => {  
 expect(calculator.add(2, 3)).toBe(5);  
 expect(calculator.add(-1, 1)).toBe(0);  
 expect(calculator.add(0, 0)).toBe(0);  
 });  
});  
  
describe('divide', () => {  
 it('should divide two numbers correctly', () => {  
 expect(calculator.divide(6, 2)).toBe(3);  
 });  
  
 it('should throw error when dividing by zero', () => {  
 expect(() => calculator.divide(1, 0)).toThrow('Division by zero');  
 });  
});

});

// 2. Integration Test Example // userService.ts interface User { id: number; name: string; email: string; }

class UserService { private api: ApiClient; private cache: Map<number, User>;

constructor(api: ApiClient) {  
 this.api = api;  
 this.cache = new Map();  
}  
  
async getUser(id: number): Promise<User> {  
 if (this.cache.has(id)) {  
 return this.cache.get(id)!;  
 }  
  
 const user = await this.api.fetchUser(id);  
 this.cache.set(id, user);  
 return user;  
}

}

// userService.test.ts describe(‘UserService Integration’, () => { let userService: UserService; let apiClient: jest.Mocked;

beforeEach(() => {  
 apiClient = {  
 fetchUser: jest.fn()  
 };  
 userService = new UserService(apiClient);  
});  
  
it('should fetch and cache user data', async () => {  
 const mockUser = { id: 1, name: 'John', email: 'john@example.com' };  
 apiClient.fetchUser.mockResolvedValue(mockUser);  
  
 // First call - should fetch from API  
 const result1 = await userService.getUser(1);  
 expect(result1).toEqual(mockUser);  
 expect(apiClient.fetchUser).toHaveBeenCalledTimes(1);  
  
 // Second call - should return from cache  
 const result2 = await userService.getUser(1);  
 expect(result2).toEqual(mockUser);  
 expect(apiClient.fetchUser).toHaveBeenCalledTimes(1);  
});

});

// 3. E2E Test Example dengan Cypress // cypress/integration/login.spec.ts describe(‘Login Flow’, () => { beforeEach(() => { cy.visit(‘/login’); });

it('should login successfully with valid credentials', () => {  
 cy.get('[data-testid="username"]').type('testuser');  
 cy.get('[data-testid="password"]').type('password123');  
 cy.get('[data-testid="login-button"]').click();  
  
 cy.url().should('include', '/dashboard');  
 cy.get('[data-testid="welcome-message"]')  
 .should('contain', 'Welcome, Test User');  
});  
  
it('should show error with invalid credentials', () => {  
 cy.get('[data-testid="username"]').type('wronguser');  
 cy.get('[data-testid="password"]').type('wrongpass');  
 cy.get('[data-testid="login-button"]').click();  
  
 cy.get('[data-testid="error-message"]')  
 .should('be.visible')  
 .and('contain', 'Invalid credentials');  
});

});

// 4. Testing Async Code // asyncService.ts class AsyncService { async fetchData(): Promise<string[]> { const response = await fetch(‘https://api.example.com/data’); return response.json(); }

async processData(data: string[]): Promise<number> {  
 return new Promise(resolve => {  
 setTimeout(() => {  
 resolve(data.length);  
 }, 1000);  
 });  
}

}

// asyncService.test.ts describe(‘AsyncService’, () => { let service: AsyncService;

beforeEach(() => {  
 service = new AsyncService();  
 global.fetch = jest.fn();  
});  
  
it('should fetch and process data', async () => {  
 const mockData = ['item1', 'item2', 'item3'];  
 (global.fetch as jest.Mock).mockResolvedValue({  
 json: () => Promise.resolve(mockData)  
 });  
  
 const data = await service.fetchData();  
 expect(data).toEqual(mockData);  
  
 const result = await service.processData(data);  
 expect(result).toBe(3);  
});

});

// 5. Testing with Custom Matchers // matchers/toBeWithinRange.ts expect.extend({ toBeWithinRange(received: number, floor: number, ceiling: number) { const pass = received >= floor && received <= ceiling; if (pass) { return { message: () => `expected ${received} not to be within range ${floor} - ${ceiling}`, pass: true }; } else { return { message: () => `expected ${received} to be within range ${floor} - ${ceiling}`, pass: false }; } } });

// Using custom matcher describe(‘Custom Matcher Example’, () => { it(‘should be within range’, () => { expect(100).toBeWithinRange(90, 110); expect(101).toBeWithinRange(100, 102); }); });

// 6. Snapshot Testing // component.tsx interface ButtonProps { label: string; onClick: () => void; disabled?: boolean; }

const Button: React.FC = ({ label, onClick, disabled }) => ( <button onClick={onClick} disabled={disabled} className={`btn ${disabled ? ‘btn-disabled’ : ‘btn-primary’}`} > {label} );

// component.test.tsx describe(‘Button’, () => { it(‘should match snapshot’, () => { const onClick = jest.fn(); const { container } = render( ); expect(container).toMatchSnapshot(); });

it('should match disabled snapshot', () => {  
 const onClick = jest.fn();  
 const { container } = render(  
 <Button label="Click me" onClick={onClick} disabled />  
 );  
 expect(container).toMatchSnapshot();  
});

}); ```

## Cara Kerja Testing

1. **Test Execution Flow**:
   * Test discovery
   * Test setup
   * Test execution
   * Test teardown
2. **Assertion Mechanism**:
   * Value comparison
   * Type checking
   * Exception handling
   * Async resolution
3. **Mocking System**:
   * Function replacement
   * Module mocking
   * Timer mocking
   * Network request mocking

## Tips dan Trik

1. **Effective Test Organization**

* // ✅ Gunakan describe blocks untuk grouping  
  describe('UserService', () => {  
   describe('authentication', () => {  
   it('should login user', () => {  
   // test  
   });  
    
   it('should logout user', () => {  
   // test  
   });  
   });  
    
   describe('profile', () => {  
   it('should update profile', () => {  
   // test  
   });  
   });  
  });

1. **Test Data Factory**

* // ✅ Buat factory untuk test data  
  function createUser(override: Partial<User> = {}): User {  
   return {  
   id: 1,  
   name: 'John Doe',  
   email: 'john@example.com',  
   ...override  
   };  
  }  
    
  // Penggunaan  
  const user = createUser({ name: 'Jane' });

1. **Custom Test Utilities**

* // ✅ Buat utility functions untuk testing  
  function setupTestDatabase() {  
   // setup logic  
  }  
    
  function cleanupTestDatabase() {  
   // cleanup logic  
  }  
    
  beforeEach(setupTestDatabase);  
  afterEach(cleanupTestDatabase);

## Kesalahan yang Sering Dilakukan Pemula

1. **Tidak Mengisolasi Test**

* // ❌ Buruk: Test saling mempengaruhi  
  let counter = 0;  
    
  describe('Counter', () => {  
   it('should increment', () => {  
   counter++;  
   expect(counter).toBe(1);  
   });  
    
   it('should be zero', () => {  
   expect(counter).toBe(0); // Fails!  
   });  
  });  
    
  // ✅ Baik: Setiap test terisolasi  
  describe('Counter', () => {  
   let counter: number;  
    
   beforeEach(() => {  
   counter = 0;  
   });  
    
   it('should increment', () => {  
   counter++;  
   expect(counter).toBe(1);  
   });  
    
   it('should be zero', () => {  
   expect(counter).toBe(0);  
   });  
  });

1. **Terlalu Banyak Mock**

* // ❌ Buruk: Over-mocking  
  it('should process user', async () => {  
   const dbMock = jest.fn();  
   const cacheMock = jest.fn();  
   const loggerMock = jest.fn();  
   const validatorMock = jest.fn();  
   // Too many mocks!  
    
   // ✅ Baik: Gunakan test doubles dengan bijak  
   it('should process user', async () => {  
   const userService = new UserService(testDb);  
   const result = await userService.processUser(1);  
   expect(result).toBeDefined();  
   });  
  });

1. **Tidak Testing Edge Cases**

* // ❌ Buruk: Hanya test happy path  
  it('should divide numbers', () => {  
   expect(calculator.divide(4, 2)).toBe(2);  
  });  
    
  // ✅ Baik: Test edge cases  
  describe('divide', () => {  
   it('should divide numbers', () => {  
   expect(calculator.divide(4, 2)).toBe(2);  
   });  
    
   it('should handle zero division', () => {  
   expect(() => calculator.divide(4, 0)).toThrow();  
   });  
    
   it('should handle negative numbers', () => {  
   expect(calculator.divide(-4, 2)).toBe(-2);  
   });  
  });

### Solusi:

1. Isolasi setiap test case
2. Gunakan test doubles dengan bijak
3. Test edge cases dan error cases
4. Maintain test suite
5. Gunakan continuous integration