JavaScript

Level Menengah

OOP · Async/Await · Fetch API · Modules · Error Handling

RegExp · Web Storage · ES6+ · Design Patterns · Real Projects

LEVEL 2	MENENGAH	ADVANCED READY
BAB 1 OOP & Class	BAB 2 Prototype	BAB 3 Error Handling
BAB 4 Async/Await	BAB 5 Fetch API	BAB 6 ES6+ Features
BAB 7 Modules	BAB 8 RegExp	BAB 9 Web Storage

Edisi 2025 · Lanjutan dari Buku Level 1 · Disertai Proyek Nyata

Daftar Isi

BAB 1

OOP & Class

Pemrograman Berorientasi Objek di JavaScript

OOP (Object-Oriented Programming) adalah paradigma pemrograman yang mengorganisasi kode dalam bentuk 'objek' — entitas yang memiliki data (properti) dan perilaku (method). JavaScript mendukung OOP melalui Class yang diperkenalkan di ES6.

1.1 Membuat Class

Class adalah blueprint/cetakan untuk membuat objek. Gunakan keyword **class** diikuti nama class (huruf kapital di awal adalah konvensi):

class_dasar.js

```
class Hewan {
// Constructor dipanggil saat new Hewan(...)
constructor( nama, jenis ) {
this.nama = nama;
this.jenis = jenis;
this.energi = 100;
}
// Method biasa
makan( makanan ) {
this.energi += 20;
console.log( `${this.nama} makan ${makanan}. Energi: ${this.energi}` );
}
info() {
console.log( `[${this.jenis}] ${this.nama}` );
}
}
// Membuat instance (objek dari class)
const kucing = new Hewan( "Whiskers", "Kucing" );
const anjing = new Hewan( "Rex", "Anjing" );
kucing.info();
kucing.makan( "ikan" );
anjing.info();
```

Output

```
[Kucing] Whiskers
```

```
Whiskers makan ikan. Energi: 120
[Anjing] Rex
```

1.2 Inheritance (Pewarisan) dengan extends

Class bisa mewarisi properti dan method dari class lain menggunakan keyword **extends**. Gunakan **super()** untuk memanggil constructor parent:

inheritance.js

```
// Class Hewan sebagai Parent (lihat contoh sebelumnya)
class HewanPeliharaan extends Hewan {
constructor( nama, jenis, pemilik ) {
super( nama, jenis ); // panggil constructor parent
this.pemilik = pemilik;
}
// Method baru (hanya di HewanPeliharaan)
perkenalan() {
console.log(
`Halo! Saya ${this.nama}, peliharaan ${this.pemilik}.`
);
}
// Override method parent
info() {
super.info(); // panggil info() dari parent
console.log( `Pemilik: ${this.pemilik}` );
}
}
const kucingku = new HewanPeliharaan( "Luna", "Kucing", "Andi" );
kucingku.perkenalan();
kucingku.info();
kucingku.makan( "tuna" ); // method dari parent
// instanceof cek relasi
console.log( kucingku instanceof HewanPeliharaan ); // true
console.log( kucingku instanceof Hewan ); // true juga!
```

■ Output

```
Halo! Saya Luna, peliharaan Andi.
[Kucing] Luna
Pemilik: Andi
Luna makan tuna. Energi: 120
```

```
true
true
```

1.3 Getter, Setter & Static

getter_setter_static.js

```
class Lingkaran {
constructor( jariJari ) {
this._r = jariJari; // _ = konvensi private
}
// GETTER — akses seperti properti biasa
get luas() {
return Math.PI * this._r ** 2;
}
get keliling() {
return 2 * Math.PI * this._r;
}
// SETTER — validasi saat assignment
set jariJari( nilai ) {
if ( nilai <= 0 ) throw new Error( "Harus positif!" );
this._r = nilai;
}
// STATIC — dipanggil langsung dari Class
static dariDiameter( d ) {
return new Lingkaran( d / 2 );
}
}
const l = new Lingkaran( 7 );
console.log( l.luas.toFixed( 2 ); // 153.94
console.log( l.keliling.toFixed( 2 ); // 43.98
const l2 = Lingkaran.dariDiameter( 20 ); // r=10
console.log( l2.luas.toFixed( 2 ); // 314.16
```

■ Output

```
153.94
43.98
314.16
```

■■ Kapan Gunakan Class?

Gunakan Class ketika kamu perlu membuat banyak objek dengan struktur yang sama, seperti: User, Product, Animal, Shape. Untuk data sederhana, object literal {} biasa sudah cukup.

BAB 2

Prototype & This

Memahami Fondasi OOP JavaScript yang Sebenarnya

Di balik syntax Class yang rapi, JavaScript menggunakan sistem **Prototype**. Setiap objek punya referensi ke objek lain sebagai 'prototype'-nya — membentuk rantai (prototype chain). Memahami ini penting untuk debug dan performa.

2.1 Prototype Chain

prototype.js

```
// Semua array punya method dari Array.prototype
const arr = [1, 2, 3];
console.log( arr.hasOwnProperty( "0" ); // true (milik arr langsung)
console.log( arr.hasOwnProperty( "push" ); // false (dari Array.prototype)

// Menambah method ke prototype (hindari di production)
String.prototype.balik = function() {
return this.split( "" ).reverse().join( "" );
};
console.log( "JavaScript".balik(); // tpircSavaJ

// Cek rantai prototype
class A { }
class B extends A { }
const b = new B();
console.log( Object.getPrototypeOf( b ) === B.prototype ); // true
```

■ Output

```
true
false
tpircSavaJ
true
```

2.2 Keyword this

this mengacu pada konteks di mana fungsi dipanggil. Nilai this bisa berbeda tergantung cara fungsi dipanggil — ini adalah salah satu topik paling membingungkan di JavaScript!

this_keyword.js

```
// 1. this di dalam Object Method → objek itu sendiri
```

```
const orang = {
nama: "Budi",
sapa() { console.log( `Halo, ${this.nama}!` ); }
};
orang.sapa(); // Halo, Budi!

// 2. this di Arrow Function → mewarisi dari luar
class Timer {
constructor() { this.detik = 0; }
mulai() {
// Arrow function: this = Timer instance ■
setInterval( () => {
this.detik++; // benar!
console.log( this.detik );
} , 1000);
}
}

// 3. Explicit binding: call, apply, bind
function perkenalan( kota, negara ) {
console.log( `${this.nama} dari ${kota}, ${negara}` );
}
const user = { nama: "Siti" };

perkenalan.call( user, "Jakarta", "Indonesia" );
perkenalan.apply( user, ["Bandung", "Indonesia"] );
const fn_ = perkenalan.bind( user, "Surabaya" );
fn_( "Indonesia" );
```

■ Output

```
Halo, Budi!
Siti dari Jakarta, Indonesia
Siti dari Bandung, Indonesia
Siti dari Surabaya, Indonesia
```

2.3 Closure Lanjutan

Closure adalah fungsi yang 'mengingat' lingkup (scope) di mana ia dibuat, bahkan setelah fungsi luar selesai dieksekusi. Ini sangat berguna untuk enkapsulasi data:

closure.js

```
// Factory function dengan closure (Private State)
function buatCounter( awal = 0 ) {
let count = awal; // variabel private
```

```
  return {
    tambah( n = 1 ) { count += n; return count; }
    kurang( n = 1 ) { count -= n; return count; }
    reset() { count = awal; return count; }
    nilai() { return count; }
  };
}

const c1 = buatCounter();
const c2 = buatCounter( 100 );

console.log( c1.tambah(); // 1
console.log( c1.tambah( 5 ); // 6
console.log( c2.tambah(); // 101
console.log( c1.reset(); // 0 — c2 tidak terpengaruh
```

Output

```
1
6
101
0
```

BAB 3

Error Handling

Menangani Kesalahan dengan Elegan

Program yang baik tidak hanya bekerja saat kondisi ideal, tapi juga mampu menangani kesalahan dengan anggun. JavaScript menyediakan mekanisme try/catch/finally untuk mengelola error.

3.1 try / catch / finally

try_catch.js

```
try {
// Kode yang mungkin error
const data = JSON.parse( "ini bukan JSON valid" );
console.log( data ); // tidak sampai sini
} catch ( error ) {
console.log( "Terjadi error:", error.message );
console.log( "Tipe error:", error.name );
} finally {
// SELALU dijalankan, error atau tidak
console.log( "Blok finally selesai." );
}
// Contoh: validasi input
function bagi( a, b ) {
if ( b === 0 ) throw new RangeError( "Tidak boleh bagi nol!" );
return a / b;
}
try {
console.log( bagi( 10, 0 );
} catch ( e ) {
console.log( `[${e.name}] ${e.message}` );
}
```

Output

```
Terjadi error: Unexpected token i in JSON at position 0
Tipe error: SyntaxError
Blok finally selesai.
[RangeError] Tidak boleh bagi nol!
```

3.2 Custom Error Class

Buat custom error untuk membedakan jenis kesalahan dalam aplikasimu:

custom_error.js

```
// Custom Error dengan extend Error
class ValidationError extends Error {
constructor( pesan, field ) {
super( pesan );
this.name = "ValidationError";
this.field = field;
}
}
class DatabaseError extends Error {
constructor( pesan, kode ) {
super( pesan );
this.name = "DatabaseError";
this.kode = kode;
}
}
function daftarUser( user ) {
if (! user.email)
throw new ValidationError( "Email wajib diisi", "email" );
if ( user.email.includes( "@" ) === false)
throw new ValidationError( "Format email salah", "email" );
console.log( "User terdaftar:", user.email );
}
try {
daftarUser( { email: "budimail.com" }) ;
} catch ( e ) {
if ( e instanceof ValidationError ) {
console.log( `Validasi gagal [${e.field}]: ${e.message}` );
} else {
throw e; // lempar ulang jika bukan ValidationError
}
}
```

■ Output

```
Validasi gagal [email]: Format email salah
```

Tipe Error Bawaan	Kapan Terjadi

SyntaxError	Kode tidak valid secara sintaks (JSON.parse salah)
ReferenceError	Variabel yang belum didefinisikan diakses
TypeError	Operasi pada tipe data yang salah
RangeError	Nilai di luar rentang yang diizinkan
URIError	encodeURI/decodeURI dengan karakter tidak valid

BAB 4

Asynchronous JavaScript

Callback, Promise, dan async/await

JavaScript berjalan secara **single-thread** — hanya bisa menjalankan satu hal pada satu waktu. Untuk operasi yang memakan waktu (request ke server, baca file), JavaScript menggunakan mekanisme **asynchronous** agar program tidak 'membeku'.

■■ Event Loop Singkat

JavaScript menggunakan Event Loop: kode sync dijalankan dulu, lalu callback/promise yang sudah selesai dijalankan satu per satu dari queue. Ini yang membuat JS bisa 'seolah multitasking' meski single-thread.

4.1 Callback & Callback Hell

callback.js

```
// Callback — fungsi yang dikirim sebagai argumen
function ambilData( id, callback ) {
setTimeout( () => {
const data = { id: id, nama: "Budi" };
callback( null, data ); // pola (error, hasil)
} , 1000 );
}

// Callback hell — bertingkat-tingkat (HINDARI!)
ambilData( 1, ( err1, user ) => {
ambilData( user.id, ( err2, orders ) => {
ambilData( orders.id, ( err3, detail ) => {
// semakin dalam → semakin susah dibaca ■
});
});
});
```

4.2 Promise — Solusi Callback Hell

Promise adalah objek yang merepresentasikan hasil operasi async di masa depan. Promise punya 3 state: **pending**, **fulfilled**, atau **rejected**.

promise.js

```
// Membuat Promise
```

```
function ambilUser( id ) {
return new Promise( ( resolve, reject ) => {
setTimeout( () => {
if ( id > 0 ) {
resolve({ id, nama: "Siti" }); // berhasil
} else {
reject( new Error( "ID tidak valid" )); // gagal
}
} , 500 );
});
}
// Menggunakan Promise dengan .then() .catch() .finally()
ambilUser( 1 )
.then( user => console.log( `Dapat user: ${user.nama}` ))
.catch( err => console.log( `Error: ${err.message}` ))
.finally() => console.log( "Selesai" ));
// Promise Chaining — solusi callback hell
ambilUser( 1 )
.then( user => ambilUser( user.id )) // chain!
.then( data => console.log( data ))
.catch( err => console.log( err ));
```

4.3 async / await — Cara Terbaik

async/await membuat kode async terlihat seperti kode sync biasa — jauh lebih mudah dibaca dan di-debug:

async_await.js

```
// Fungsi async selalu mengembalikan Promise
async function getDataUser( id ) {
try {
// await = tunggu Promise selesai
const user = await ambilUser( id );
console.log( `User: ${user.nama}` );
// Mudah ditulis berurutan seperti sync
const detail = await ambilUser( user.id );
console.log( `Detail: ${detail.nama}` );
return detail;
} catch ( error ) {
console.log( "Gagal:", error.message );
```

```
}
}
getDataUser( 1 );
// Promise.all — jalankan PARALEL (lebih cepat!)
async function ambilBanyak() {
const [u1, u2, u3] = await Promise.all([
ambilUser( 1 ),
ambilUser( 2 ),
ambilUser( 3 ),
]);
console.log( u1.nama, u2.nama, u3.nama );
}
```

■ Output

```
User: Siti
Detail: Siti
Siti Siti Siti
```

■ async/await vs Promise.then()

Untuk satu operasi: keduanya setara. Untuk banyak operasi berurutan: gunakan async/await (lebih bersih). Untuk operasi paralel: tetap gunakan Promise.all() di dalam async/await.

BAB 5

Fetch API & HTTP

Komunikasi dengan Server dan REST API

Fetch API adalah cara modern untuk melakukan HTTP request di JavaScript — menggantikan XMLHttpRequest yang lama. Dengan Fetch, kamu bisa mengambil data dari server, mengirim data, dan berkomunikasi dengan REST API.

5.1 GET Request — Mengambil Data

fetch_get.js

```
// URL publik untuk latihan
const API_URL = "https://jsonplaceholder.typicode.com";

async function ambilPosts() {
try {
// fetch() mengembalikan Promise
const response = await fetch( `${API_URL}/posts` );

// Cek status HTTP
if (! response.ok ) {
throw new Error( `HTTP Error: ${response.status}` );
}

// Parse JSON dari body response
const posts = await response.json();

console.log( `Total: ${posts.length} posts` );
console.log( posts.slice( 0, 2 );
return posts;
} catch ( error ) {
console.error( "Gagal ambil data:", error.message );
}
}

ambilPosts();
```

Output

```
Total: 100 posts
[{userId:1, id:1, title:"sunt aut facere...", body:"..."},
{userId:1, id:2, title:"qui est esse", body:"..."}]
```

5.2 POST Request — Mengirim Data

fetch_post.js

```
async function buatPost( postBaru ) {
const response = await fetch( `${API_URL}/posts`, {
method: "POST",
headers: {
"Content-Type": "application/json",
"Authorization": "Bearer TOKEN_ANDA",
},
body: JSON.stringify( postBaru ),
});
const hasil = await response.json();
console.log( "Post dibuat:", hasil );
return hasil;
}
buatPost({
title: "Belajar Fetch API",
body: "Fetch adalah cara modern HTTP request",
userId: 1
});
```

Output

```
Post dibuat: { id: 101, title: "Belajar Fetch API", body: "Fetch adalah...", userId: 1
}
```

5.3 Semua HTTP Method

Method	Fungsi	Contoh Penggunaan
GET	Mengambil data	Tampilkan daftar produk
POST	Membuat data baru	Daftar akun, buat postingan
PUT	Update data (semua)	Edit profil lengkap
PATCH	Update sebagian	Ubah hanya nama/email
DELETE	Hapus data	Hapus komentar, hapus akun

fetch_methods.js

```
// UPDATE dengan PUT
async function updatePost( id, data ) {
```

```
const res = await fetch( `${API_URL}/posts/${id}`, {
method: "PUT",
headers: { "Content-Type": "application/json" },
body: JSON.stringify( data )
});
return res.json();
}
// DELETE
async function hapusPost( id ) {
const res = await fetch( `${API_URL}/posts/${id}`,
{ method: "DELETE" } ) );
console.log( res.status === 200 ? "Terhapus" : "Gagal" );
}
```

BAB 6

Fitur ES6+ Modern

Sintaks Modern yang Wajib Dikuasai

ES6 (2015) dan versi berikutnya membawa banyak fitur powerful yang membuat kode lebih ringkas, aman, dan ekspresif. Di bab ini kita akan mempelajari fitur-fitur yang paling sering digunakan di kode modern.

6.1 Optional Chaining (?.) & Nullish Coalescing (??)

optional_chaining.js

```
const user = {
nama: "Rina",
alamat: {
kota: "Surabaya",
// tidak ada provinsi
},
// tidak ada telepon
};

// Tanpa optional chaining — TypeError jika null!
// console.log(user.telepon.nomor); // ■ ERROR

// DENGAN optional chaining ?. — aman!
console.log( user.telepon?.nomor ); // undefined (bukan error)
console.log( user.alamat?.kota ); // Surabaya
console.log( user.alamat?.provinsi ); // undefined

// Nullish Coalescing ?? — nilai default jika null/undefined
const telepon = user.telepon ?? "Tidak ada telepon";
console.log( telepon ); // Tidak ada telepon

// ?? vs || : bedanya?
console.log( 0 || "default" ); // default (0 = falsy di ||)
console.log( 0 ?? "default" ); // 0 (?? hanya null/undefined)
```

■ Output

```
undefined
Surabaya
undefined
Tidak ada telepon
default
```

```
0
```

6.2 Destructuring Lanjutan

destructuring_adv.js

```
// Destructuring dengan nilai default
const { nama = "Anonymous", umur = 0, kota = "Jakarta" } = { nama: "Dewi" };
console.log( nama, umur, kota ); // Dewi 0 Jakarta

// Nested destructuring
const profil = { data: { nama: "Andi", usia: 25 }, skills: ["JS", "Python"] };
const { data: { nama: namaPengguna, usia }, skills: [skill1] } = profil;
console.log( namaPengguna, usia, skill1 ); // Andi 25 JS

// Destructuring di parameter fungsi
function tampilUser({ nama, email, role = "user" }) {
console.log( `${nama} | ${email} | ${role}` );
}
tampilUser({ nama: "Rudi", email: "rudi@mail.com" });
```

■ Output

```
Dewi 0 Jakarta
Andi 25 JS
Rudi | rudi@mail.com | user
```

6.3 Map, Set, WeakMap

map_set.js

```
// Map — seperti Object tapi key bisa apa saja
const map = new Map();
map.set( "nama", "Sari" );
map.set( 42, "angka sebagai key" );
map.set({ id: 1 }, "objek sebagai key" );
console.log( map.get( "nama" ); // Sari
console.log( map.size ); // 3

// Iterasi Map
map.forEach(( val, key ) => console.log( key, "→", val ));

// Set — kumpulan nilai UNIK
const set = new Set([ 1, 2, 2, 3, 3, 3 ]);
console.log([ ...set ]; // [1, 2, 3] — duplikat hilang!

// Hapus duplikat dari array dengan Set
```

```
const arr = [1, 2, 2, 3, 1];
const unik = [...new Set( arr )];
console.log( unik ); // [1, 2, 3]
```

Output

```
Sari
3
[1, 2, 3]
[1, 2, 3]
```

BAB 7

Modules (import/export)

Memecah Kode Menjadi File-file Terpisah

Modules memungkinkan kamu membagi kode ke dalam file-file terpisah yang saling berbagi fungsionalitas. Ini membuat kode lebih terorganisir, mudah di-maintain, dan bisa di-reuse.

7.1 Named Export & Import

utils/math.js

```
// ■■ File: utils/math.js ■■■■■■■■■■■■■■■■■■■■■■■

// Named export — bisa banyak per file
export const PI = 3.14159;

export function tambah( a, b ) { return a + b; }
export function kali( a, b ) { return a * b; }

export class Vektor {
constructor( x, y ) { this.x = x; this.y = y; }
panjang() { return Math.sqrt( this.x**2 + this.y**2 ); }
}
```

main.js

```
// ■■ File: main.js ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■

// Import named exports
import { PI, tambah, kali, Vektor } from "./utils/math.js";

console.log( PI ); // 3.14159
console.log( tambah( 5, 3 ); // 8

// Import dengan alias
import { kali as multiply } from "./utils/math.js";
console.log( multiply( 4, 3 ); // 12

// Import semua sekaligus
import * as MathUtils from "./utils/math.js";
console.log( MathUtils.tambah( 10, 5 ); // 15
```

■ Output

```
3.14159
8
12
15
```

7.2 Default Export & Import

default_export.js

```
// ■■ File: services/ApiService.js ■■■■■■■■■■■■■■
class ApiService {
constructor( baseURL ) { this.base = baseURL; }
async get( path ) {
const res = await fetch( this.base + path );
return res.json();
}
async post( path, data ) {
const res = await fetch( this.base + path, {
method: "POST", body: JSON.stringify( data )
});
return res.json();
}
}
// Default export — hanya SATU per file
export default ApiService;
// ■■ File: main.js ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
// Import default — bebas nama
import Api from "./services/ApiService.js";
const api = new Api( "https://api.example.com" );
api.get( "/users" ).then( data => console.log( data ));
```

■ Named vs Default Export

Gunakan Named export untuk utility functions/constants (bisa banyak). Gunakan Default export untuk 1 class/fungsi utama per file (seperti komponen React). Boleh kombinasikan keduanya dalam satu file.

BAB 8

Regular Expression (RegExp)

Mencari dan Memanipulasi Teks dengan Pola

Regular Expression (Regex) adalah pola yang digunakan untuk mencocokkan, mencari, dan memanipulasi teks. Meski terlihat rumit, regex sangat powerful untuk validasi form, parsing data, dan transformasi string.

8.1 Sintaks Dasar Regex

regex_dasar.js

```
// Dua cara membuat regex
const re1 = /pola/; // literal
const re2 = new RegExp( "pola" ); // konstruktor

// test() — mengembalikan true/false
const pola = /javascript/i; // i = case insensitive
console.log( pola.test( "Belajar JavaScript" ); // true
console.log( pola.test( "Belajar Python" ); // false

// match() — menemukan kecocokan
const teks = "Harga: Rp 150.000 dan Rp 75.000";
const angka = teks.match(/\\d+/g); // g = global (semua)
console.log( angka ); // ["150", "000", "75", "000"]

// replace() — mengganti
const hasil = "Hello World".replace(/o/g, "0");
console.log( hasil ); // Hell0 W0rld
```

■ Output

```
true
false
["150", "000", "75", "000"]
Hell0 W0rld
```

8.2 Character Classes & Quantifiers

Pola	Arti	Contoh
.	Karakter apa saja (kecuali newline)	/.at/ → cat, bat, hat
\d	Angka [0-9]	/\d+/ → 123

\w	Huruf, angka, underscore	/\w+/ → hello_1
\s	Whitespace (spasi, tab)	/\s+/ → spasi
^	Awal string	/^Hello/ → Hello world
$	Akhir string	/world$/ → Hello world
{n,m}	Repetisi min n, max m kali	/\d{2,4}/ → 12, 1234
?	0 atau 1 kali (opsional)	/colou?r/ → color/colour
*	0 atau lebih kali	/a*/ → '', a, aa
+	1 atau lebih kali	/a+/ → a, aa, aaa

8.3 Kasus Nyata: Validasi Form

validasi_regex.js

```
// Koleksi regex validasi
const VALIDASI = {
email: /^[^\\s@]+@[^\\s@]+\\.[^\\s@]+$/,
telepon: /^(\\+62|0)[0-9]{8,12}$/, // format Indonesia
password: /^(?=.*[a-z])(?=.*[A-Z])(?=.*\\d).{8,}$/,
url: /^https?:\\/\\/[\\w-]+(\\.[\\w-]+)+/,
kodePos: /^[1-9][0-9]{4}$/, // kode pos 5 digit
};
function validasiField( tipe, nilai ) {
const valid = VALIDASI[ tipe ].test( nilai );
console.log( `${tipe}: '${nilai}' → ${valid ? '■' : '■'}` );
return valid;
}
validasiField( "email", "budi@gmail.com" );
validasiField( "email", "emailsalah" );
validasiField( "telepon", "08123456789" );
validasiField( "password", "Kuat123" );
validasiField( "password", "Kuat1234" );
```

■ Output

```
email: 'budi@gmail.com' → ■
email: 'emailsalah' → ■
telepon: '08123456789' → ■
password: 'Kuat123' → ■ (kurang dari 8 karakter)
password: 'Kuat1234' → ■
```

BAB 9

Web Storage & Cookies

Menyimpan Data di Browser

Web Storage API memungkinkan aplikasi web menyimpan data langsung di browser pengguna. Data ini tetap ada bahkan setelah halaman ditutup (localStorage) atau hilang saat tab ditutup (sessionStorage).

9.1 localStorage — Permanen

localStorage.js

```
// ■■■ MENYIMPAN DATA ■■■
// String langsung
localStorage.setItem( "tema", "gelap" );
// Object/Array — WAJIB di-JSON.stringify dulu
const user = { id: 1, nama: "Budi", role: "admin" };
localStorage.setItem( "user", JSON.stringify( user ));
// ■■■ MEMBACA DATA ■■■
const tema = localStorage.getItem( "tema" );
console.log( tema ); // gelap
// Parse balik Object
const dataUser = JSON.parse( localStorage.getItem( "user" ));
console.log( dataUser.nama ); // Budi
// ■■■ HAPUS DATA ■■■
localStorage.removeItem( "tema" ); // hapus satu
localStorage.clear(); // hapus SEMUA
// Cek jumlah item
console.log( localStorage.length );
```

9.2 Wrapper Utility — Aman & Praktis

localStorage bisa lempar error jika nilai bukan string yang valid. Buat wrapper untuk keamanan:

storage_utils.js

```
// Storage Utility — reusable & safe
const storage = {
simpan( key, value ) {
try {
```

```
localStorage.setItem( key, JSON.stringify( value ));
return true;
} catch { return false; }
},
ambil( key, defaultVal = null ) {
try {
const item = localStorage.getItem( key );
return item !== null ? JSON.parse( item ) : defaultVal;
} catch { return defaultVal; }
},
hapus( key ) { localStorage.removeItem( key ); }
bersih() { localStorage.clear(); }
};
// Penggunaan
storage.simpan( "preferensi", { tema: "gelap", bahasa: "id" });
const pref = storage.ambil( "preferensi", { tema: "terang" });
console.log( pref.tema ); // gelap
```

■ Output

```
gelap
```

9.3 sessionStorage & Perbandingan

sessionStorage.js

```
// sessionStorage — API SAMA, tapi hilang saat tab ditutup
sessionStorage.setItem( "token", "abc123xyz" );
const token = sessionStorage.getItem( "token" );
console.log( token ); // abc123xyz
// Cocok untuk: session login, data form sementara
// TIDAK untuk: preferensi user, cart belanja
```

Fitur	localStorage	sessionStorage	Cookie
Kapasitas	~5-10 MB	~5 MB	~4 KB
Lifetime	Permanen	Tab aktif	Bisa di-set
Akses Server	Tidak	Tidak	Ya (header)
API	Mudah	Mudah	Agak rumit
Keamanan	Rentan XSS	Rentan XSS	HttpOnly aman

BAB 10

Proyek: Todo App Lengkap

Menggabungkan Semua yang Telah Dipelajari

Di bab terakhir ini kita akan membangun aplikasi **Todo List** yang lengkap menggunakan semua konsep yang telah dipelajari: Class, Array methods, DOM manipulation, Event, Error handling, dan localStorage untuk persistensi data.

■■ Fitur yang Akan Dibangun

■ Tambah todo baru | ■ Tandai selesai/belum | ■ Hapus todo | ■ Filter (semua/aktif/selesai) | ■ Counter todo aktif | ■ Data tersimpan di localStorage | ■ Validasi input

10.1 Struktur & HTML

todo.html

```
<!DOCTYPE html>
<html lang="id">
<head>
<title>Todo App</title>
<style>
body { font-family: sans-serif; max-width: 500px; margin: 40px auto; padding: 0 20px }
.todo-item { display:flex; align-items:center; gap:10px; padding:8px 0;
border-bottom: 1px solid #eee }
.selesai span { text-decoration:line-through; color:#999 }
</style>
</head>
<body>
<h1>■ Todo App</h1>
<p>Tersisa: <strong id="counter">0</strong> tugas</p>
<div style="display:flex;gap:8px;margin-bottom:16px">
<input id="input-todo" placeholder="Tambah tugas baru..." style="flex:1;padding:8px">
<button id="btn-tambah">Tambah</button>
</div>
<div id="filter-btns">
<button data-filter="semua">Semua</button>
<button data-filter="aktif">Aktif</button>
<button data-filter="selesai">Selesai</button>
</div>
<ul id="daftar-todo"></ul>
```

```
<script src="todo.js"></script>
</body></html>
```

10.2 Class TodoApp — Logika Utama

todo.js (Bagian 1)

```
// ■■ todo.js ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
class TodoApp {
constructor() {
// Load dari localStorage atau array kosong
this.todos = JSON.parse(
localStorage.getItem( "todos" ) ?? []
);
this.filter = "semua";
this.nextId = this.todos.reduce(( max, t ) => Math.max( max, t.id ), 0) + 1;

// Simpan referensi elemen DOM
this.inputEl = document.getElementById( "input-todo" );
this.daftarEl = document.getElementById( "daftar-todo" );
this.counterEl = document.getElementById( "counter" );

this._bindEvents();
this.render();
}

_bindEvents() {
// Tombol tambah
document.getElementById( "btn-tambah" ).addEventListener( "click", () =>
this.tambah());

// Enter di input
this.inputEl.addEventListener( "keypress", e => {
if ( e.key === "Enter" ) this.tambah();
});

// Tombol filter
document.getElementById( "filter-btns" ).addEventListener( "click", e => {
if ( e.target.dataset.filter ) {
this.filter = e.target.dataset.filter;
this.render();
}
});
}
```

todo.js (Bagian 2 — Selesai)

```
tambah() {
const teks = this.inputEl.value.trim();
if (! teks ) return; // abaikan jika kosong

this.todos.push({
id: this.nextId++,
teks,
selesai: false,
dibuat: new Date().toLocaleDateString( "id-ID" ),
});
this.inputEl.value = "";
this._simpan();
this.render();
}

toggle( id ) {
const todo = this.todos.find( t => t.id === id );
if ( todo ) todo.selesai = !todo.selesai;
this._simpan(); this.render();
}

hapus( id ) {
this.todos = this.todos.filter( t => t.id !== id );
this._simpan(); this.render();
}

_simpan() {
localStorage.setItem( "todos", JSON.stringify( this.todos ));
}

render() {
// Filter berdasarkan mode aktif
const tampil = this.todos.filter( t => {
if ( this.filter === "aktif" ) return !t.selesai;
if ( this.filter === "selesai" ) return t.selesai;
return true;
});

// Render HTML untuk setiap todo
this.daftarEl.innerHTML = tampil.map( t => `

",
${t.teks}
${t.dibuat}
■

`).join( "" );
```

```
// Update counter
const aktif = this.todos.filter( t => !t.selesai ).length;
this.counterEl.textContent = aktif;

// Event: checkbox toggle
this.daftarEl.querySelectorAll( "input[type=checkbox]" ).forEach(( cb, i ) => {
cb.addEventListener( "change", () => this.toggle( tampil[ i ].id ));,
});

// Event: tombol hapus (event delegation)
this.daftarEl.addEventListener( "click", e => {
if ( e.target.classList.contains( "btn-hapus" ) {
this.hapus( +e.target.dataset.id );
}
});
}
}

// Inisialisasi aplikasi saat DOM siap
document.addEventListener( "DOMContentLoaded", () => {
new TodoApp();
});
```

Selamat! Kamu telah menyelesaikan E-book JavaScript Level Menengah. Kini kamu menguasai OOP, Async/Await, Fetch API, Modules, RegExp, dan Web Storage. Terus berlatih dengan membuat proyek nyata — semakin banyak kamu membangun, semakin mahir kamu menjadi! ■

Langkah Selanjutnya (Level Lanjut)

- React / Vue.js — Framework UI modern
- Node.js & Express — Backend JavaScript
- TypeScript — JavaScript dengan type safety
- Testing (Jest) — Unit & integration testing
- Design Patterns — Singleton, Observer, Factory
- Webpack / Vite — Build tools & bundling