MODUL PERKULIAHAN PEMROGRAMAN WEB

DASAR HTML DAN GIT

Tim Pengajar

BAB 1: Pengenalan HTML

1.1 Apa Itu HTML?

HTML (*HyperText Markup Language*) adalah bahasa markup yang berfungsi sebagai fondasi dalam membangun sebuah halaman web. Melalui kumpulan elemen berbasis tag, HTML memungkinkan pengembang untuk menentukan struktur, konten, dan keterhubungan antarkomponen dalam situs. Bahasa ini bukanlah bahasa pemrograman karena tidak mengeksekusi logika alur perintah seperti halnya JavaScript atau Python, melainkan hanya menata tampilan dan penyusunan konten.

HTML pertama kali diperkenalkan oleh Tim Berners-Lee pada tahun 1991 sebagai sarana berbagi dokumen ilmiah. Seiring perkembangannya, HTML terus berevolusi untuk mendukung kebutuhan pengembangan web yang kian kompleks, termasuk integrasi CSS, JavaScript, multimedia, dan banyak lagi.

Sejarah Singkat HTML:

- **1991:** Versi pertama HTML dirilis secara terbatas untuk kalangan peneliti.
- **1995:** Diresmikan HTML 2.0 yang mendukung formulir, sehingga interaksi antara pengguna dan halaman web kian dinamis.
- **1997:** Diperkenalkan HTML 4.0, yang membuka jalan bagi pemisahan struktur dan tampilan melalui CSS.
- **2014:** HTML5 lahir dengan penekanan pada elemen-elemen semantik, validasi formulir bawaan, dan dukungan multimedia seperti audio serta video.

Kemudahan dalam menulis HTML membuatnya menjadi gerbang masuk yang ideal bagi siapa saja yang baru memulai belajar pengembangan web. Anda dapat membuat halaman statis, lalu menambahkan CSS untuk tampilan yang lebih menarik, dan JavaScript untuk interaktivitas.

1.2 Struktur Dasar HTML

Struktur utama HTML mencakup deklarasi tipe dokumen, elemen `<html>`, `<head>`, dan `<body>`. Deklarasi tipe dokumen, `<!DOCTYPE html>`, menandakan bahwa kita menggunakan standar HTML modern. Kemudian di dalam `<html>`, terdapat dua bagian pokok:

1. `<head>`: Menyimpan metadata, judul halaman, link ke file CSS, atau skrip.
2. `<body>`: Menyajikan konten yang terlihat oleh pengguna, seperti teks, gambar, tautan, dan elemen-elemen lain.

Contoh struktur dasarnya:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Judul Halaman</title>
  </head>
  <body>
    <h1>Selamat Datang!</h1>
    <p>Ini adalah contoh halaman HTML dasar.</p>
  </body>
</html>
```

Penjelasan:

- `<!DOCTYPE html>`: Menyatakan bahwa dokumen ini adalah HTML5.
- `<html>`: Elemen induk yang membungkus seluruh elemen lain.
- `<head>`: Bagian untuk metadata, seperti `<title>` yang menentukan nama halaman di tab browser, atau `<meta charset="UTF-8">` untuk memastikan karakter teks ter-encode dengan benar.
- `<body>`: Bagian utama yang menampilkan konten kepada pengguna. Di sinilah Anda menempatkan elemen teks, gambar, tautan, daftar, tabel, dan sebagainya.

Hasil dari kode sebelumnya dapat dilihat pada gambar diawah ini

1.3 Tag-Tag HTML yang Umum Digunakan

Tag HTML berfungsi seperti instruksi untuk browser, menentukan bagaimana konten ditampilkan dan diorganisasi. Beberapa tag fundamental meliputi:

1. **Heading** (`<h1>` hingga `<h6>`): Digunakan untuk membuat judul dan subjudul.
   <h1>Judul Utama</h1>

<h2>Sub Judul</h2>
Semakin kecil angkanya, semakin penting heading tersebut di mata mesin pencari dan struktur dokumen.

2. **Paragraf** (`<p>`): Berfungsi menampilkan teks paragraf.
   Halo, ini adalah paragraf tentang perkenalan diri.
3. **Tautan** (`<a>`): Dipakai untuk membuat hyperlink ke halaman lain atau ke bagian tertentu.
   Kunjungi Situs
   Atribut `href` menentukan destinasi tautan.
4. **Gambar** (`<img>`): Menyisipkan gambar ke dalam halaman.
   <img src="foto.jpg" alt="Deskripsi foto">
   Atribut `alt` berguna untuk teks alternatif saat gambar gagal dimuat dan untuk aksesibilitas.
5. **Daftar** (`<ul>` dan `<ol>`):
   - `<ul>`: Daftar tidak berurutan (bullet list).
   - `<ol>`: Daftar berurutan (angka).

```html
<ul>
  <li>Buah-buahan</li>
  <li>Sayur-sayuran</li>
</ul>

<ol>
  <li>Pertama</li>
  <li>Kedua</li>
</ol>
```

6. **Tabel** (`<table>`, `<tr>`, `<td>`): Digunakan untuk menampilkan data secara terstruktur.

```html
<table>
  <tr>
    <td>Baris 1, Kolom 1</td>
    <td>Baris 1, Kolom 2</td>
  </tr>
  <tr>
    <td>Baris 2, Kolom 1</td>
    <td>Baris 2, Kolom 2</td>
  </tr>
</table>
```

1.4 Atribut HTML

Atribut menambahkan informasi atau perilaku khusus pada suatu elemen. Berikut adalah contoh atribut yang sering digunakan:

- `href`: Menentukan URL untuk tautan.
- `src`: Lokasi file gambar, audio, atau video.
- `alt`: Teks alternatif untuk aksesibilitas.
- `target`: Bagaimana tautan dibuka, misalnya `_blank` untuk tab baru.
- `class` dan `id`: Menandai elemen tertentu untuk kebutuhan styling dan scripting.
- `style`: Menambahkan gaya inline, meskipun sebaiknya CSS terpisah.

Contoh penerapan:

```
<a href="https://example.com" target="_blank">Kunjungi Website</a>
<img src="gambar.jpg" alt="Deskripsi gambar" class="img-responsive">
```

Atribut memberikan fleksibilitas yang lebih besar dalam menyesuaikan tampilan atau interaksi sebuah elemen. Contohnya, dengan menggunakan `class="img-responsive"`, Anda dapat menargetkan elemen ini lewat file CSS.

1.5 Semantic Tags di HTML5

HTML5 menghadirkan semantic tags untuk memperjelas struktur dokumen secara logis, membantu mesin pencari serta pembaca layar untuk memahami konteks konten. Beberapa tag semantik di antaranya:

- `<header>`: Bagian teratas halaman atau sebuah bagian besar, umumnya berisi logo, judul, atau navigasi utama.
- `<nav>`: Menu navigasi yang menautkan ke bagian lain dalam situs.
- `<section>`: Bagian utama yang mengelompokkan konten terkait.
- `<article>`: Konten yang dapat berdiri sendiri seperti artikel berita atau posting blog.
- `<aside>`: Informasi tambahan, misalnya sidebar atau catatan samping.
- `<footer>`: Bagian bawah halaman yang lazim berisi informasi hak cipta atau tautan penting.

Semantic tags tidak hanya membuat struktur dokumen lebih rapi, tetapi juga meningkatkan SEO karena mesin pencari dapat lebih mudah menafsirkan bagian-bagian penting dalam halaman.

1.6 Praktik Penulisan HTML yang Baik

Menulis HTML dengan kaidah yang benar mempermudah proses pengembangan dan pemeliharaan. Berikut beberapa praktik terbaik:

1. **Gunakan Indentasi Teratur:** Memakai dua atau empat spasi (atau tab) secara konsisten memudahkan kita meninjau dan memperbaiki kode.
2. **Gunakan Komentar:** Komentar seperti `<!-- Komentar -->` membantu programmer lain memahami maksud suatu bagian kode.
3. **Validasi Kode:** W3C menyediakan layanan validasi di validator.w3.org. Validasi membantu kita menemukan kesalahan dalam penulisan tag, atribut, atau struktur.
4. **Hindari Tag Usang:** Tag seperti `<font>` atau atribut yang tidak lagi direkomendasikan sebaiknya ditinggalkan demi standar modern. Gunakan CSS untuk styling.
5. **Gunakan Tag Semantik:** Lebih baik memakai `<header>` atau `<footer>` dibanding `<div>` semata.

1.7 Contoh Halaman HTML Sederhana

Di bawah ini contoh penulisan halaman HTML sederhana dengan pemakaian semantic tags:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Profil Saya</title>
  </head>
  <body>
    <header>
      <h1>Nambi Sembilu</h1>
      <nav>
        <ul>
          <li><a href="#tentang">Tentang Saya</a></li>
          <li><a href="#kontak">Kontak</a></li>
        </ul>
      </nav>
    </header>

    <section id="tentang">
      <h2>Profil Singkat</h2>
      <p>Saya adalah seorang dosen dan pengembang perangkat lunak. Dalam keseharian, saya mengajar mata kuliah terkait pemrograman dan sistem informasi. Di luar itu, saya aktif menulis artikel dan mengembangkan proyek open-source.</p>
      <img src="profil.jpg" alt="Foto Nambi" style="max-width:200px;">
    </section>

    <aside>
```

```
      <h3>Aktivitas Terbaru</h3>
      <ul>
        <li>Merilis artikel tentang HTML5</li>
        <li>Berpartisipasi dalam konferensi IT tahunan</li>
        <li>Menjadi pembicara webinar tentang keamanan web</li>
      </ul>
    </aside>

    <footer id="kontak">
      <p>Hubungi saya melalui surel: <a
href="mailto:nambi@domain.com">nambi@domain.com</a></p>
      <p>Hak Cipta (c) 2025 Nambi Sembilu</p>
    </footer>
  </body>
</html>
```

Dalam contoh di atas, `<header>` menempatkan judul dan navigasi. Bagian profil dan informasi utama ditempatkan di dalam `<section>`, sedangkan informasi tambahan seperti kegiatan terbaru dimasukkan ke `<aside>`. Terakhir, ada `<footer>` untuk informasi kontak dan hak cipta.

Gambar diatas adalah hasil pada source code sebelumnya

BAB 2: HTML Lanjutan

Di bab ini, kita akan mendalami aspek-aspek lanjutan HTML. Meskipun HTML sering dianggap sekadar "bahasa markup" dasar, seiring perkembangan web modern, peran HTML menjadi semakin kompleks dan menarik. Anda akan mempelajari cara membuat formulir interaktif, menyisipkan multimedia, menggunakan elemen-elemen khusus seperti SVG atau Canvas, mengoptimalkan aksesibilitas, dan masih banyak lagi.

Untuk memudahkan Anda, Bab 2 akan dibagi menjadi beberapa subtopik utama:

1. **Formulir dan Input Lanjutan**
2. **Multimedia dalam HTML**
3. **SVG dan Canvas**
4. **Struktur Lanjutan dengan Semantic Tags**
5. **Optimasi Aksesibilitas**
6. **HTML Entities dan Karakter Khusus**
7. **Penggunaan Tag `<meta>`**
8. **Microdata, RDFa, dan JSON-LD**
9. **Tips dan Praktik Terbaik**
10. **Studi Kasus dan Latihan**

Mari kita jelajahi satu per satu.

2.1 Formulir dan Input Lanjutan

Formulir adalah salah satu elemen terpenting dalam sebuah halaman web karena memungkinkan interaksi dengan pengguna. Mulai dari pendaftaran akun, login, pemesanan barang, hingga pengiriman komentar, semuanya difasilitasi oleh formulir HTML.

2.1.1 Elemen Dasar Formulir

Tag utama untuk membuat formulir adalah `<form>`, yang umumnya memiliki atribut:

- `action`: Menentukan ke mana data akan dikirim. Biasanya diisi URL server.
- `method`: Metode pengiriman data, umumnya `GET` atau `POST`.
- `enctype`: Jenis encoding data, misalnya `multipart/form-data` untuk unggah berkas.

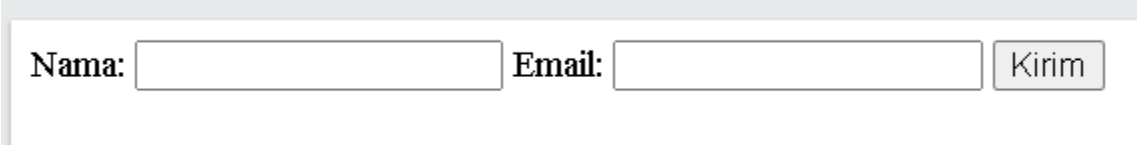
Contoh dasar:

```
<form action="/proses" method="POST">
  <label for="nama">Nama:</label>
  <input type="text" id="nama" name="nama">
```

```
  <label for="email">Email:</label>
  <input type="email" id="email" name="email">

  <button type="submit">Kirim</button>
</form>
```

Berikut adalah hasilnya



2.1.2 Jenis-jenis Input HTML5

HTML5 memperkenalkan berbagai tipe input yang membantu validasi langsung di sisi klien:

- `type="email"`: Memverifikasi format email.
- `type="url"`: Memverifikasi format URL.
- `type="number"`: Hanya menerima angka.
- `type="range"`: Menampilkan slider untuk rentang nilai.
- `type="date"`, `type="month"`, `type="week"`, `type="time"`: Memudahkan input tanggal dan waktu.
- `type="color"`: Menampilkan pemilih warna.
- `type="search"`: Input yang dioptimalkan untuk pencarian.

Dengan menggunakan tipe input ini, browser dapat membantu melakukan validasi awal sebelum data dikirim.

2.1.3 Validasi Form HTML5

HTML5 memungkinkan Anda menambahkan atribut khusus seperti `required`, `pattern`, `min`, `max`, atau `step` untuk membatasi nilai dan pola data.

Contoh:

```
<input type="text" name="username" required pattern="[a-zA-Z0-9]+">
```

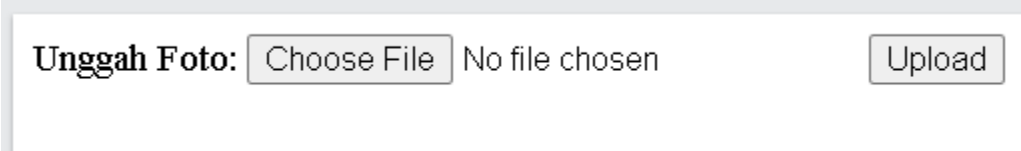
Atribut `required` memastikan input tidak boleh kosong, sedangkan `pattern` menentukan format regex yang harus dipenuhi (misalnya hanya alfanumerik). Ini mengurangi beban validasi di sisi server, meski tak sepenuhnya menggantikan validasi server-side.

2.1.4 File Upload

Untuk mengunggah berkas, Anda perlu menambahkan atribut `enctype="multipart/form-data"` di elemen `<form>`:

```
<form action="/upload" method="POST" enctype="multipart/form-data">
  <label for="photo">Unggah Foto:</label>
  <input type="file" id="photo" name="photo">
  <button type="submit">Upload</button>
</form>
```

Dengan kode di atas, pengguna dapat memilih dan mengunggah foto yang kemudian diproses di sisi server.



2.2 Multimedia dalam HTML

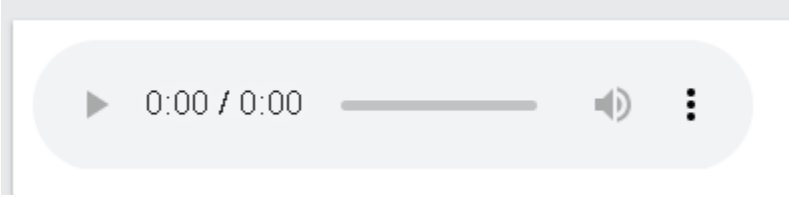
HTML5 memudahkan integrasi multimedia seperti audio dan video tanpa perlu plugin eksternal. Tag `<audio>` dan `<video>` memfasilitasi pemutaran file musik atau film secara native.

2.2.1 Audio

Tag `<audio>` mendukung berbagai format (misalnya MP3, WAV, OGG). Atribut `controls` menampilkan tombol play, pause, dan volume.

```
<audio controls>
  <source src="audio.mp3" type="audio/mpeg">
  Browser Anda tidak mendukung pemutar audio.
</audio>
```

Anda bisa menambahkan beberapa `<source>` dengan format berbeda untuk memastikan kompatibilitas lintas browser.

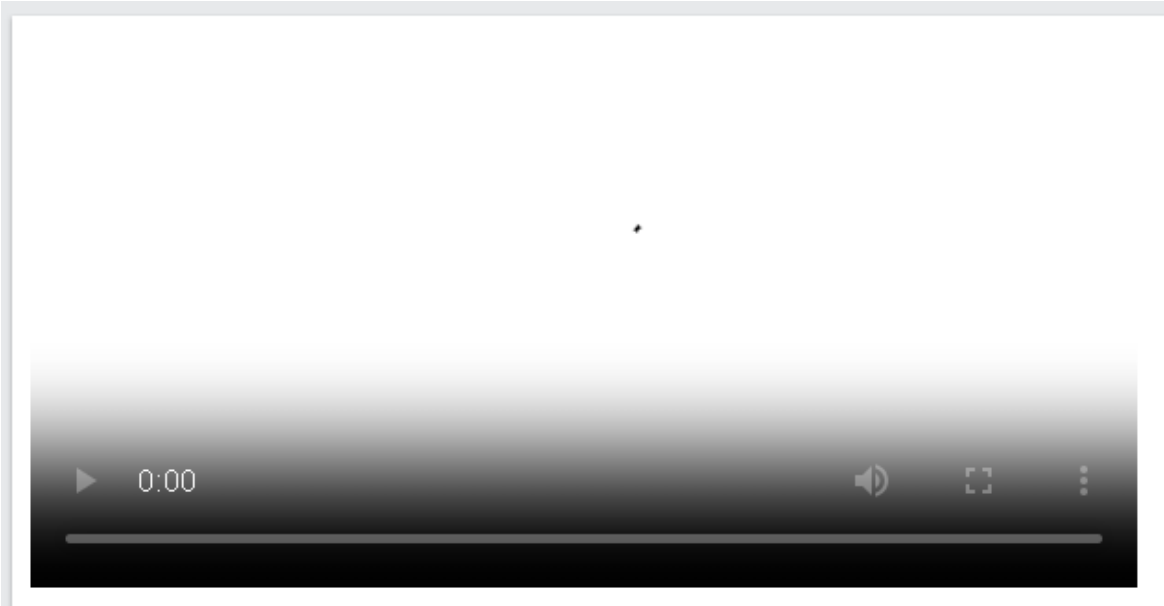


2.2.2 Video

Mirip dengan audio, `<video>` memungkinkan pemutaran klip video:

```
<video controls width="500" poster="thumbnail.jpg">
  <source src="video.mp4" type="video/mp4">
  <source src="video.webm" type="video/webm">
  Browser Anda tidak mendukung pemutar video.
</video>
```

Atribut `poster` menampilkan gambar mini sebagai pratinjau sebelum video diputar. Anda juga bisa mengatur `autoplay`, `loop`, dan `muted` sesuai kebutuhan.

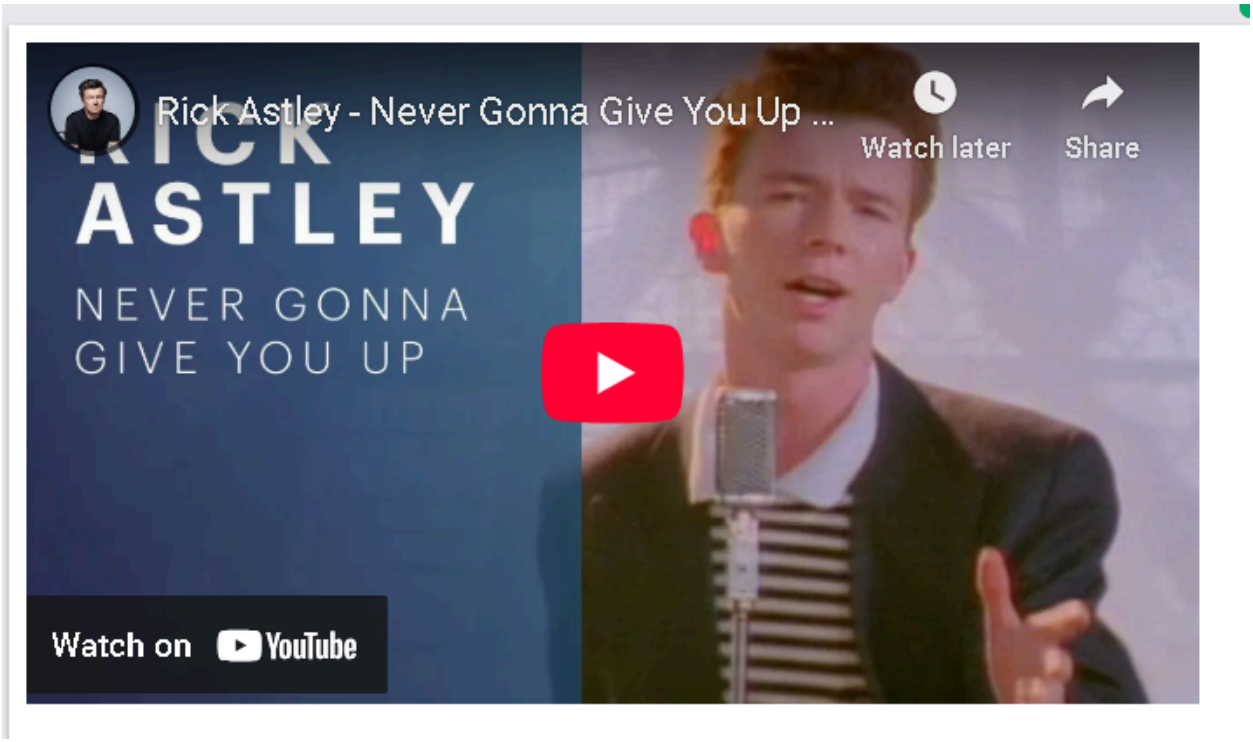


2.2.3 Iframe untuk Konten Eksternal

`<iframe>` memungkinkan Anda menanamkan konten eksternal seperti video YouTube, peta Google, atau halaman situs lain. Atribut `src` diisi dengan URL konten.

```
<iframe width="560" height="315"
src="https://www.youtube.com/embed/dQw4w9WgXcQ" frameborder="0"
allowfullscreen></iframe>
```

Jika menampilkan iframe dari sumber eksternal, pastikan memahami ketentuan keamanan (misalnya Content Security Policy).



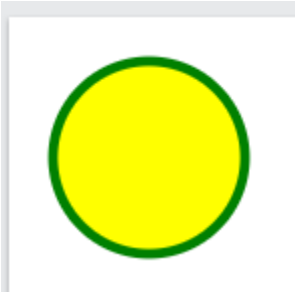
2.3 SVG dan Canvas

2.3.1 Scalable Vector Graphics (SVG)

SVG adalah format gambar berbasis vektor yang didefinisikan dalam XML. Keunggulannya adalah gambar tidak akan pecah ketika diperbesar karena bukan berbasis piksel. Anda dapat menuliskan SVG langsung di HTML:

```
<svg width="100" height="100">
  <circle cx="50" cy="50" r="40" stroke="green" stroke-width="4"
fill="yellow" />
</svg>
```

Anda juga bisa menampilkan file SVG dengan `<img src="gambar.svg">`, namun menuliskan SVG inline memudahkan interaksi serta animasi dengan CSS atau JavaScript.



2.3.2 Canvas

Elemen `<canvas>` memberikan area menggambar yang dapat dimanipulasi lewat JavaScript, cocok untuk grafik dinamis, animasi, bahkan game sederhana. Contoh penggunaan:

```
<canvas id="myCanvas" width="300" height="200"></canvas>
<script>
  const canvas = document.getElementById('myCanvas');
  const ctx = canvas.getContext('2d');
  ctx.fillStyle = 'blue';
  ctx.fillRect(10, 10, 50, 50);
</script>
```

Dengan Canvas API, Anda dapat menggambar bentuk, teks, atau grafik sesuai kebutuhan.

2.4 Struktur Lanjutan dengan Semantic Tags

Sementara di Bab 1 kita sudah membahas garis besar semantic tags, di sini kita akan memperdalam penerapannya. Menggunakan `<section>` untuk tiap bagian topik, `<article>` untuk tiap konten mandiri, dan `<aside>` untuk informasi pendukung akan memudahkan penyusunan dokumen.

Misalnya, untuk halaman berita:

```
<main>
  <article>
    <header>
      <h2>Judul Berita</h2>
      <p>Tanggal: 18 Februari 2025</p>
    </header>
    <p>Isi berita di sini...</p>
    <footer>
      <p>Penulis: Tim Redaksi</p>
    </footer>
  </article>

  <aside>
    <h3>Berita Terkait</h3>
    <ul>
      <li>Topik 1</li>
      <li>Topik 2</li>
    </ul>
  </aside>
</main>
```

Dengan struktur semantik, pembaca layar dan mesin pencari dapat lebih cepat menafsirkan konten.

2.5 Optimasi Aksesibilitas

Aksesibilitas (Accessibility) adalah aspek penting dalam pengembangan web modern. Dengan merancang konten yang mudah diakses oleh semua orang, termasuk penyandang disabilitas, kita mematuhi standar internasional seperti WCAG (Web Content Accessibility Guidelines). Beberapa prinsip pokok:

1. **Teks Alternatif:** Setiap elemen gambar (`<img>`) harus memiliki atribut `alt`.

2. **Label pada Form:** Gunakan `<label for="id">` yang sesuai dengan input.
3. **Kontras Warna:** Pastikan teks mudah dibaca.
4. **Heading Terstruktur:** Gunakan `<h1>` untuk judul utama, diikuti `<h2>`, `<h3>` secara hierarkis.
5. **Navigasi yang Konsisten:** Buat menu yang sama di tiap halaman, jadikan site map jelas.
6. **Penggunaan ARIA (Accessible Rich Internet Applications):** Untuk elemen yang memerlukan peran (role) atau status (state) lebih spesifik.

Contoh penggunaan ARIA:

```
<div role="alert" aria-live="assertive">
  Pemberitahuan penting untuk pengguna.
</div>
```

ARIA attributes membantu teknologi asisten seperti screen reader memahami fungsi dan status elemen halaman.

2.6 HTML Entities dan Karakter Khusus

HTML entities adalah representasi khusus untuk karakter yang tidak dapat ditulis langsung atau menimbulkan bentrokan dengan sintaks HTML. Misalnya, `<` dipakai sebagai tanda pembuka tag, sehingga jika ingin menampilkannya secara literal, kita gunakan `&lt;`.

Beberapa contoh HTML entities umum:

- `&lt;` = <
- `&gt;` = >
- `&amp;` = &
- `&quot;` = "
- `&apos;` = '
- `&copy;` = ©
- `&reg;` = ®

Entities ini juga bermanfaat saat menulis simbol matematika (`&Sigma;`, `&Pi;`) atau karakter khusus lainnya.

2.7 Penggunaan Tag `<meta>`

Elemen `<meta>` berada di dalam `<head>` dan menyediakan metadata tentang dokumen HTML. Metadata berpengaruh pada SEO, media sosial, dan perilaku browser. Beberapa `<meta>` tag yang penting:

1. **`<meta charset="UTF-8">`** Mendefinisikan encoding karakter. Penting agar teks berbahasa non-Inggris ditampilkan benar.
2. **`<meta name="description" content="Deskripsi singkat halaman">`** Menjelaskan isi halaman yang sering ditampilkan di hasil pencarian.
3. **`<meta name="keywords" content="HTML, modul, tutorial">`** Dulu penting untuk SEO, kini banyak mesin pencari mengabaikannya.
4. **`<meta name="viewport" content="width=device-width, initial-scale=1.0">`** Mengatur tampilan responsif di perangkat seluler.

Open Graph / Twitter Cards (untuk media sosial)

```
<meta property="og:title" content="Judul Konten">
<meta property="og:description" content="Deskripsi singkat">
<meta property="og:image" content="url_gambar.jpg">
<meta property="og:url" content="https://example.com">
```

Pengaturan `<meta>` yang tepat dapat meningkatkan visibilitas situs Anda di hasil pencarian dan tampilan pratinjau media sosial.

2.8 Microdata, RDFa, dan JSON-LD

Struktur data terorganisir (structured data) membantu mesin pencari memahami konteks konten Anda. HTML5 mendukung Microdata, sementara RDFa adalah spesifikasi lain yang menumpangkan metadata pada elemen HTML. JSON-LD (JavaScript Object Notation for Linked Data) biasanya ditempatkan dalam tag `<script>` di `<head>`.

2.8.1 Microdata

Contoh microdata untuk menandai info produk:

```
<div itemscope itemtype="http://schema.org/Product">
  <h2 itemprop="name">Laptop XYZ</h2>
  <span itemprop="brand">Brand ABC</span>
  <span itemprop="price">Rp 10.000.000</span>
</div>
```

Dengan menandai data menggunakan schema.org, mesin pencari dapat menampilkan "rich snippet" di hasil pencarian.

2.8.2 JSON-LD

Google lebih merekomendasikan JSON-LD untuk pemasangan structured data. Contoh:

```
<script type="application/ld+json">
{
  "@context": "http://schema.org",
  "@type": "Person",
  "name": "Nambi Sembilu",
  "jobTitle": "Dosen & Developer",
  "url": "http://contoh.com"
}
</script>
```

JSON-LD memisahkan data dari markup HTML, sehingga lebih rapi dan mudah dikelola.

2.9 Tips dan Praktik Terbaik

1. **Selalu Gunakan DOCTYPE Terbaru:** Pastikan `<!DOCTYPE html>` berada di baris pertama.
2. **Konsisten Dalam Penulisan Tag:** Tag dibuka (`<tag>`) dan ditutup (`</tag>`). Untuk elemen void seperti `<img>` atau `<br>`, cukup satu tag.
3. **Manfaatkan Semantic Tags:** Memudahkan navigasi, SEO, dan aksesibilitas.
4. **Validasi Kode Secara Berkala:** Mencegah eror sintaks.
5. **Optimalkan Ukuran Gambar dan Multimedia:** Performa halaman akan meningkat.
6. **Perhatikan Aksesibilitas Sejak Awal:** Gunakan `alt` pada gambar, `label` pada form, dan struktur heading yang jelas.
7. **Pertimbangkan Responsivitas:** Gunakan `<meta name="viewport" content="width=device-width, initial-scale=1.0">` dan desain layout agar adaptif di berbagai ukuran layar.

BAB 3: Validasi pada HTML

Setelah mempelajari berbagai aspek dasar dan lanjutan HTML, tahap berikutnya yang sangat penting adalah **validasi**. Validasi pada HTML bertujuan untuk memastikan bahwa kode yang kita tulis sesuai dengan standar yang ditetapkan oleh World Wide Web Consortium (W3C). Dengan melakukan validasi, kita dapat meminimalisir kesalahan sintaks, meningkatkan kompatibilitas lintas browser, sekaligus membuat halaman web lebih mudah diakses. Bab 3 ini akan mengulas beragam topik seputar validasi HTML, mulai dari apa itu validasi, mengapa penting, metode validasi, hingga penanganan error yang umum.

3.1 Apa Itu Validasi HTML?

Validasi HTML adalah proses memeriksa kode HTML untuk memastikan bahwa seluruh tag, atribut, dan struktur dokumen mengikuti kaidah yang telah disepakati dalam spesifikasi HTML atau HTML5. Ketika kode HTML Anda valid, browser dapat menafsirkan halaman tersebut secara lebih konsisten. Hasilnya, tampilan dan interaksi dapat berjalan sebagaimana mestinya.

Menulis kode HTML yang valid juga erat kaitannya dengan **aksesibilitas**. Misalnya, jika Anda lupa menutup tag tertentu atau salah meletakkan atribut, screen reader dan perangkat asisten lain mungkin kesulitan menafsirkan informasi. Selain itu, validasi membantu mengantisipasi potensi masalah di masa depan. Karena spesifikasi HTML terus berkembang, kode yang valid cenderung lebih tahan terhadap perubahan teknologi.

Contoh sederhana masalah validasi: Anda menggunakan `<img src="gambar.jpg">` namun lupa menutup tag dengan benar atau lupa menambahkan `alt`. Meski browser modern mungkin masih menampilkan gambar, hal ini tetap dianggap tidak sesuai standar.

3.2 Mengapa Validasi Penting?

1. **Kompatibilitas Antar Browser (Cross-Browser Compatibility)**
    - Masing-masing browser memiliki "mesin rendering". Jika kode Anda penuh kesalahan, hasil interpretasi bisa berbeda-beda. Validasi membantu mengurangi inkonsistensi ini.
2. **Mudah Dipelihara (Maintainability)**
    - Dengan menulis kode yang valid, Anda akan lebih mudah menavigasi dan memperbaiki kode. Bug yang timbul akibat sintaks salah juga lebih cepat diidentifikasi.
3. **Aksesibilitas**
    - Pengguna dengan keterbatasan tertentu biasanya mengandalkan teknologi bantu. Kode HTML yang tidak valid dapat membuat konten sulit diakses oleh mereka.
4. **SEO (Search Engine Optimization)**

    - Mesin pencari seperti Google tidak selalu memprioritaskan halaman yang valid. Namun, kode yang bersih dan rapi memudahkan perayapan (crawling) dan pemahaman konteks oleh search engine.
5. **Kepatuhan Terhadap Standar**
    - Standar web dibuat agar ada keseragaman. Dengan validasi, Anda membuktikan bahwa situs Anda mematuhi best practice yang diharapkan dalam pengembangan web profesional.

3.3 Alat Validasi (Validators)

3.3.1 W3C Markup Validation Service

Layanan gratis yang paling umum adalah W3C Validator. Anda dapat menggunakannya dengan tiga cara:

1. **By URL:** Memasukkan URL halaman web yang sudah daring (online). Validator akan mengunduh halaman dan memeriksa apakah terdapat kesalahan.
2. **By File Upload:** Mengunggah berkas HTML yang ingin divalidasi. Cocok untuk halaman yang belum dipublikasikan.
3. **By Direct Input:** Menyalin langsung kode HTML ke form di situs validator.

Setelah Anda mengirim, validator akan menampilkan daftar peringatan dan kesalahan. Contoh kesalahan umum adalah tag yang tak ditutup, atribut yang tidak dikenal, atau penggunaan elemen deprecated.

3.3.2 Editor Tekstual dan IDE

Banyak *editor* dan *IDE* modern seperti Visual Studio Code, Sublime Text, atau WebStorm memiliki ekstensi atau plugin yang secara otomatis memeriksa kesalahan HTML. Dengan demikian, Anda bisa mendapatkan umpan balik *real-time* saat menulis kode, tanpa harus bolak-balik ke situs W3C Validator.

3.3.3 ESLint / HTMLHint

Untuk proyek yang lebih besar, Anda dapat menggunakan alat linting seperti HTMLHint yang diintegrasikan ke proses build. HTMLHint memindai kode HTML dan mengidentifikasi potensi kesalahan atau pelanggaran aturan konvensi.

3.4 Cara Kerja Validator

Ketika Anda mengirim kode ke validator, sistem akan:

1. Membaca DOCTYPE: Menentukan versi HTML, misalnya HTML5.
2. Membaca setiap tag, atribut, dan konten.

3. Mencocokkan struktur dengan aturan spesifikasi HTML.
4. Menghasilkan laporan jika ada kesalahan sintaks, atribut tidak valid, atau struktur yang kurang pas.

Penting untuk dipahami bahwa validator tidak selalu mengenali niat Anda. Terkadang kita sengaja memakai atribut non-standar (misalnya untuk framework tertentu). Anda bisa menambahkan pengaturan khusus jika validasi default menandai hal tersebut sebagai error.

3.5 Jenis Kesalahan dan Peringatan Umum

1. **Tag Tidak Ditutup**
   - Contoh: `<p>Paragraf` tanpa `</p>`. HTML modern kerap "memperbaiki" sendiri, tetapi hasilnya bisa tak terduga.
2. **Tag Salah Letak**
   - Misal, `<span>` di dalam `<p>` yang tidak sesuai penempatan logis, atau `<li>` di luar `<ul>`.
3. **Atribut Tidak Dikenal**
   - HTML5 mendukung custom data attribute seperti `data-`. Namun atribut lain yang tidak terdokumentasi bisa dipandang salah.
4. **Penggunaan Tag Usang (Deprecated)**
   - Seperti `<center>`, `<font>`, `<marquee>`, yang sudah digantikan dengan CSS.
5. **Masalah Karakter dan Encoding**
   - Tidak menyertakan `<meta charset="UTF-8">` bisa menyebabkan karakter spesial muncul keliru.
6. **Struktur Tag `<table>`**
   - Tabel dengan `<tr>` tidak lengkap atau `<td>` tidak berurutan.
7. **Penggunaan Elemen Blok di Dalam Inline**
   - Misalnya `<div>` ditempatkan di dalam `<span>`.

Setiap pesan error biasanya disertai baris kode yang bermasalah. Perbaiki dari atas ke bawah, karena satu error sering menimbulkan error lanjutan.

3.6 Praktik Terbaik Validasi

1. **Validasi Secara Berkala**: Jangan menunggu proyek selesai. Lebih baik memvalidasi setelah perubahan signifikan di kode.
2. **Gunakan Editor dengan Fitur Linting**: Hal ini memudahkan Anda menghindari kesalahan sederhana.
3. **Catat Kesalahan dan Perbaiki**: Terkadang, satu peringatan mungkin bukanlah masalah besar. Namun kesalahan lintas halaman menunjukkan pola keliru yang perlu dibenahi secara menyeluruh.
4. **Manfaatkan DOCTYPE HTML5**: `<!DOCTYPE html>` cenderung lebih ringkas dan toleran, namun tetap peduli pada standar utama.

5. **Periksa Aksesibilitas**: Sebagian validator juga menandai ketidaksesuaian WCAG. Makin cepat diperbaiki, makin baik.
6. **Konsultasi Dokumentasi**: Ketika error muncul, buka dokumentasi resmi MDN (Mozilla Developer Network) atau W3C untuk penjelasan lebih lanjut.

3.7 Validasi Formulir dan Input

Validasi tidak terbatas pada struktur halaman, tetapi juga menyangkut cara Anda menggunakan elemen form. Dalam Bab 2, kita sudah membahas bahwa HTML5 menambahkan validasi bawaan (seperti `required`, `pattern`, dsb.). Namun, pemanfaatan ini pun bisa menimbulkan error jika:

- Atribut `pattern` salah sintaks.
- Penempatan elemen `<label>` tidak terhubung dengan `<input>` via `for` atau `id`.
- Menyertakan tipe input yang tidak dikenali oleh browser lebih lama (misalnya `type="datetime-local"`).

Validator akan memperingatkan jika Anda memakai tipe input yang belum distandardisasi. Meski kadang masih didukung secara eksperimental oleh sebagian browser, sebaiknya ikuti standar yang sudah mapan untuk menjaga kompatibilitas.

3.8 Menangani Pesan Error dan Peringatan

1. **Baca Pesan dengan Teliti**: Validator biasanya menampilkan "Line XX, Column YY" dan teks kesalahan. Periksa baris tersebut.
2. **Pahami Konteks**: Kesalahan satu bisa memicu kesalahan lain. Jika ada rangkaian error, perbaiki yang paling mendasar lebih dulu.
3. **Gunakan Sumber Daya Belajar**: Apabila pesan error kurang jelas, rujuk ke MDN atau W3C. Atau Anda bisa bertanya di forum pengembang web.
4. **Gunakan Opsi "Show Outline"**: Beberapa validator menampilkan outline dari struktur dokumen. Anda bisa mengecek apakah heading dan elemen lain terstruktur rapi.

3.9 Contoh Studi Kasus Validasi

Misalkan Anda punya halaman HTML seperti ini:

```
<!DOCTYPE html>
<html>
```

```
<head>
  <title>Halaman Contoh</title>
</head>
<body>
<h1>Selamat Datang!</h1>
<p>Paragraf pertama.

<p>Paragraf kedua.

<img src="gambar.png">
<a href="#">Tautan</a>
</body>
</html>
```

Ketika divalidasi, mungkin muncul kesalahan:

1. **Missing `</p>`**: Tag paragraf kedua tidak ditutup.
2. **`img` element missing `alt` attribute**: Anda belum menambahkan `alt`.

Untuk memperbaikinya:

```
<!DOCTYPE html>
<html>
<head>
  <title>Halaman Contoh</title>
  <meta charset="UTF-8">
</head>
<body>
  <h1>Selamat Datang!</h1>
  <p>Paragraf pertama.</p>
  <p>Paragraf kedua.</p>

  <img src="gambar.png" alt="Deskripsi gambar">
  <a href="#">Tautan</a>
</body>
</html>
```

Setelah pembaruan, kemungkinan besar kesalahan akan hilang di validator.

3.10 Integrasi Validasi ke Dalam Alur Kerja (Workflow)

Untuk proyek berskala besar atau tim, sangat disarankan agar validasi HTML menjadi bagian dari pipeline build. Misalnya, Anda bisa memiliki skrip seperti:

1. **Linting**: Memeriksa kesalahan JavaScript/TypeScript.
2. **HTML Validation**: Menggunakan HTMLHint atau plugin lain.
3. **Testing**: Jalankan unit test atau end-to-end test.
4. **Deployment**: Jika validasi gagal, hentikan proses.

Dengan integrasi semacam ini, setiap kali ada pengembang yang mengubah HTML, build akan gagal jika terjadi pelanggaran serius terhadap standar. Hal ini menjaga kualitas kode tetap tinggi.

3.11 Hubungan Validasi dengan Penggunaan Framework

Banyak pengembang menggunakan framework Front-End seperti React, Vue, atau Angular. Framework ini sering "membungkus" HTML dalam sintaks khusus (JSX, template, dsb.). Meskipun begitu, saat di-compile atau di-build, hasil akhirnya tetap HTML murni.

Jika Anda menggunakan framework, periksa apakah library tersebut mengikuti standar HTML. Kesalahan seperti penggunaan atribut yang salah masih mungkin terjadi, misalnya saat menuliskan `class` di JSX Anda pakai `className`, atau mengikat data pada elemen yang tidak mendukung atribut tersebut.

Hasil build final biasanya bisa diekspor menjadi berkas HTML, yang kemudian dapat divalidasi. Walau mungkin ada beberapa peringatan "non-standar" (tergantung seting bundler), langkah ini tetap membantu menjaga kepatuhan kode.

3.12 Tips Tambahan untuk Mempertahankan Kode yang Valid

1. **Rutin Mengecek Pembaruan Standar**: Spesifikasi HTML5 terus hidup (living standard). Ada elemen baru yang kadang diusulkan.
2. **Perhatikan Elemen Tertentu Seperti `<main>`**: Pastikan hanya sekali dipakai di halaman, sesuai tujuan semantiknya.
3. **Jangan Lupa ARIA**: Menambahkan ARIA attribute tanpa memahami benar fungsinya bisa menimbulkan error. Baca dokumentasi.
4. **Hindari Penggunaan Inline CSS Berlebihan**: Meskipun bukan kesalahan validasi HTML, banyak peringatan yang muncul terkait gaya inline.
5. **Validasi Sisi Server**: Walau ini di luar cakupan HTML itu sendiri, ingat bahwa validasi HTML di sisi klien tak menggantikan validasi server.

3.13 Studi Kasus Lengkap

Bayangkan Anda membangun situs berita sederhana dengan beberapa halaman, masing-masing memiliki:

- Header dan navigasi.
- Artikel (`<article>`) menampilkan konten berita.

- Sidebar (`<aside>`) untuk berita terkait.
- Footer dengan informasi hak cipta.

Di setiap halaman, Anda perlu memastikan:

1. Struktur heading yang konsisten. Judul utama (`<h1>`), sub judul (`<h2>`), dan seterusnya.
2. Tak ada tag paragraf yang terlupa menutup.
3. Gambar menyertakan `alt`.
4. Link internal (`<a href="index.html">`) bekerja dan format URL benar.
5. Penggunaan `<section>` dan `<article>` di tempat tepat.
6. Tak ada *duplicated ID* pada elemen yang berbeda.
7. Tidak ada *broken link*.

Setelah seluruh halaman diperiksa dengan validator, hasilnya menampilkan "No errors or warnings to show." Anda boleh berlega hati, namun tetap waspada. Sekalipun valid, bukan berarti semuanya sempurna. Masih ada area lain seperti performa, SEO, dan konten yang relevan bagi pengguna.

BAB 4: Pengenalan Git

Setelah kita membahas dasar-dasar HTML, fitur lanjutan, hingga validasi kode agar sesuai standar, kini saatnya beralih ke topik yang tak kalah penting dalam dunia pengembangan perangkat lunak modern, yaitu **Git**. Git adalah sistem pengontrol versi (*version control system*) yang dirancang untuk melacak perubahan pada kode sumber di berbagai tahap pengembangan. Dengan Git, Anda dapat bekerja secara kolaboratif bersama tim, memelihara riwayat setiap perubahan, dan memudahkan proses rollback jika terjadi kesalahan.

Bab 4 ini akan mengupas tuntas aspek-aspek fundamental Git, mencakup sejarah, prinsip kerja, cara instalasi, alur kerja dasar, hingga strategi lanjutan seperti branching dan merging. Tak ketinggalan, kita juga akan membahas bagaimana Git berkolaborasi dengan layanan hosting kode seperti GitHub dan GitLab. Materi di bab ini dirancang untuk memastikan Anda siap mengelola proyek secara efisien dan terstruktur.

4.1 Sejarah Singkat Git dan Motivasi Utama

Git diciptakan oleh Linus Torvalds pada tahun 2005, terinspirasi dari kebutuhan untuk mengelola pengembangan kernel Linux secara kolaboratif. Sebelum Git, komunitas kernel Linux menggunakan sistem kontroversial bernama BitKeeper. Namun, ketika masa lisensi BitKeeper berakhir, Linus memutuskan untuk membangun sistem pengontrol versi baru yang bersifat *open source*, cepat, andal, dan terdistribusi.

Motivasi utama pembuatan Git meliputi:

1. **Kinerja dan Kecepatan**: Git memprioritaskan operasi lokal, seperti membuat commit dan memeriksa riwayat, agar sangat cepat.
2. **Desain Terdistribusi**: Setiap klien Git menyimpan salinan lengkap dari repositori, memudahkan kerja offline.
3. **Integritas Data**: Git memastikan setiap commit diberi tanda tangan kriptografis (SHA-1 atau SHA-256 dalam beberapa implementasi terbaru), sehingga perubahan tidak bisa dimanipulasi tanpa terdeteksi.
4. **Alur Kerja Fleksibel**: Branching dan merging dipermudah sehingga tim dapat mengembangkan fitur secara terpisah dan menggabungkannya kemudian.

Hingga hari ini, Git menjadi standar de facto dalam pengembangan perangkat lunak, banyak digunakan oleh perusahaan besar maupun individu.

4.2 Instalasi Git

Sebelum menggunakan Git, Anda perlu menginstalnya pada sistem operasi Anda. Proses instalasi berbeda tergantung platform:

1. **Windows**:
    - Unduh installer resmi dari git-scm.com.

    - Jalankan installer, ikuti petunjuk, dan pilih opsi default kecuali Anda memiliki preferensi khusus.
    - Setelah selesai, Anda bisa membuka *Git Bash* atau menggunakan Git dari *Command Prompt / PowerShell*.
2. **macOS**:
    - Git sering kali sudah terpasang bersamaan dengan *Xcode Command Line Tools*.
    - Jika belum, Anda dapat menginstal *Xcode Command Line Tools* atau menggunakan *Homebrew* dengan perintah: `brew install git`.
3. **Linux (Ubuntu/Debian)**:
    - Jalankan perintah: `sudo apt-get update && sudo apt-get install git`.
4. **Linux (Fedora/RedHat)**:
    - Jalankan perintah: `sudo dnf install git`.

Setelah instalasi, Anda bisa memeriksa keberhasilan instalasi dengan mengetik `git --version` di terminal/command prompt. Jika menampilkan versi Git, maka instalasi sukses.

Konfigurasi Awal
Langkah berikutnya adalah menambahkan identitas Anda:

```
git config --global user.name "Nama Anda"
git config --global user.email "email@domain.com"
```

Penetapan *user.name* dan *user.email* memastikan setiap commit dapat dikaitkan dengan identitas pembuatnya. Opsi `--global` membuat konfigurasi ini berlaku di seluruh repositori Git pada akun dan komputer yang sama.

4.3 Konsep Dasar Git

4.3.1 Repositori

Repositori (disingkat repo) adalah tempat semua riwayat perubahan kode disimpan. Anda bisa membuat repositori di folder mana saja. Ada dua cara:

1. **Membuat Repositori Baru**: `git init` di folder kosong.
2. **Mengkloning Repositori yang Sudah Ada**: `git clone <URL>`.

Begitu repositori dibuat, Git akan membuat folder tersembunyi bernama `.git` yang menyimpan semua metadata dan riwayat.

4.3.2 *Working Directory*, *Staging Area*, dan *Repository*

Git mengenal tiga area penting:

1. **Working Directory**: File dan folder tempat Anda bekerja.
2. **Staging Area** (Index): Area penampung yang menampung perubahan terpilih yang siap di-commit.
3. **Repository (History)**: Riwayat permanen berupa commit yang telah dibuat.

Ketika Anda mengubah file, Git akan menandainya sebagai *modified*. Setelah itu, Anda perlu menambahkan perubahan tersebut ke *staging area* menggunakan `git add <namafile>` atau `git add .` (untuk semua file). Lalu, Anda melakukan `git commit -m "pesan"` agar perubahan resmi tercatat dalam riwayat repositori.

4.3.3 Commit

Setiap commit bagaikan "snapshot" kondisi kode pada saat tertentu. Hal ini mencakup informasi penulis, waktu, dan ringkasan perubahan. Commit idealnya kecil dan fokus pada satu *task* tertentu agar histori perubahan mudah ditelusuri. Penggunaan pesan commit yang jelas juga sangat membantu kolaborasi tim.

Contoh:

```
git add index.html
git commit -m "Menambahkan halaman index dengan struktur awal"
```

4.3.4 Branch

Branch adalah cabang pengembangan independen. Git memudahkan pembuatan branch:

git checkout -b fitur-login

Kini Anda berada di branch `fitur-login`. Semua commit yang Anda buat di sini tidak akan memengaruhi branch lain. Setelah fitur rampung, Anda bisa menggabungkan branch `fitur-login` ke `main` (branch utama).

4.4 Branching dan Merging

4.4.1 Motivasi Branching

Branching memungkinkan beberapa pengembang bekerja paralel pada fitur atau perbaikan bug tanpa mengganggu kode di branch utama. Branching juga mempermudah konsep *release*, *hotfix*, dan manajemen versi.

4.4.2 Membuat Branch

1. **Membuat Branch**: `git branch nama-branch`.
2. **Berpindah ke Branch**: `git checkout nama-branch`, atau sekaligus `git checkout -b nama-branch`.

3. **Melihat Daftar Branch**: `git branch`.

4.4.3 Menggabungkan Branch (Merging)

Untuk menggabungkan branch ke dalam branch aktif, Anda bisa melakukan:

```
git checkout main
git merge fitur-login
```

Jika tidak ada konflik, Git akan menggabungkan perubahan dan menampilkan pesan "Merge made by...". Jika terjadi *merge conflict*, Git meminta Anda untuk menentukan bagaimana menyesuaikan file yang bentrok.

4.4.4 Rebase

Selain *merge*, Git memiliki opsi `rebase`. Fungsinya membuat riwayat commit lebih "linear" dan rapi. Namun, *rebasing* perlu kehati-hatian karena berpotensi mengubah riwayat publik. Penggunaan rebase sebaiknya disesuaikan dengan kebijakan tim.

4.5 Bekerja dengan Repository Remote

Agar bisa berkolaborasi, Anda biasanya menyimpan repositori di layanan hosting seperti GitHub, GitLab, atau Bitbucket. Repositori di server disebut *remote repository*. GitHub adalah yang paling populer berkat komunitas besar dan fitur sosial seperti *pull request*.

4.5.1 Menambahkan Remote

Setelah membuat repositori di GitHub, Anda bisa menambahkan *remote* lokal:

```
git remote add origin https://github.com/username/repo.git
```

Dengan perintah ini, `origin` menjadi alias untuk URL repositori. Anda dapat memberi nama lain selain "origin" jika mau.

4.5.2 Push

Untuk mengunggah commit baru ke *remote*, gunakan:

```
git push -u origin main
```

Parameter `-u` (*upstream*) membuat Git tahu bahwa branch `main` lokal terhubung dengan branch `main` di *remote*.

4.5.3 Pull dan Fetch

- `git pull`: Mengambil perubahan dari remote dan langsung menggabungkannya ke branch aktif.
- `git fetch`: Hanya mengambil commit baru, tapi tidak menggabungkan otomatis. Anda harus melakukan merge manual.

Pull adalah kombinasi fetch dan merge. Terkadang, tim lebih suka fetch + merge agar bisa memeriksa perubahan sebelum benar-benar digabung.

4.5.4 Fork dan Pull Request

Jika Anda ingin berkontribusi pada proyek publik di GitHub:

1. *Fork* repositori ke akun Anda.
2. Clone fork milik Anda.
3. Buat branch baru, kerjakan perubahan.
4. Dorong (push) commit ke repositori fork.
5. Buat *pull request* ke repositori asli agar pemilik bisa meninjau dan *merge* perubahan Anda.

4.6 Alur Kerja (Workflow)

Ada beberapa alur kerja (workflow) Git yang populer:

1. **Git Flow**: Diperkenalkan oleh Vincent Driessen, Git Flow membagi branch utama menjadi `master` dan `develop`. `master` menampung rilis produksi stabil, `develop` menampung kode integrasi dari banyak fitur. Ada pula branch `feature`, `release`, dan `hotfix`.
2. **GitHub Flow**: Lebih sederhana, umumnya hanya `main` (atau `master`) dan branch fitur. Setelah selesai, branch fitur di-*merge* ke `main` via *pull request*.
3. **GitLab Flow**: Kombinasi konsep Git Flow dan GitHub Flow, menekankan environment-based flows.

Memilih workflow tergantung skala proyek, jumlah kontributor, dan aturan tim.

4.7 Git dan Kolaborasi Tim

Git tidak hanya mengatur versi kode, tetapi juga memfasilitasi kolaborasi. Beberapa poin penting:

1. **Code Review**: Sebelum kode di-*merge*, tim bisa melakukan peninjauan (review) lewat *pull request*.
2. **Continuous Integration (CI)**: Layanan seperti GitHub Actions, GitLab CI/CD, atau Jenkins dapat menguji kode secara otomatis setiap kali ada commit baru.

3. **Issue Tracking**: GitHub dan GitLab menyediakan fitur *issues* agar tim dapat melacak bug, tugas, dan permintaan fitur.
4. **Project Management**: Ada *milestones*, *labels*, dan *projects* di GitHub untuk pengelolaan proyek.

4.8 Fitur Lanjutan Git

Tagging: Tag menandai titik penting di riwayat, misalnya versi rilis.
git tag v1.0.0

```
git push origin v1.0.0
```

Stashing: Menyimpan sementara perubahan tanpa membuat commit. Cocok saat Anda ingin berpindah branch tapi belum selesai mengedit.
git stash

```
git push origin v1.0.0
```

Bisect: Mendeteksi commit penyebab bug. Git akan melakukan pencarian biner di antara commit.

```
git bisect start
git bisect bad
```

1. git bisect good <commit-lama>
   Lalu Git menuntun Anda menguji commit satu per satu.
2. **Cherry-Pick**: Memindahkan commit tertentu dari satu branch ke branch lain.
3. **Submodules**: Mengelola repositori di dalam repositori, berguna jika Anda punya proyek terpisah tapi saling terkait.

4.9 Praktik Terbaik dalam Menggunakan Git

1. **Buat Commit yang Kecil dan Spesifik**: Mudah direview dan mempermudah rollback.
2. **Gunakan Pesan Commit Deskriptif**: Sebaiknya format mencakup ringkasan singkat di baris pertama, diikuti detail jika perlu.
3. **Rajin Periksa Status dan Log**: `git status` dan `git log --oneline --graph` membantu memantau perkembangan.
4. **Perbarui Branch Secara Berkala**: Lakukan `git pull` (atau fetch + merge) untuk menghindari konflik.
5. **Gunakan Branch untuk Fitur Baru**: Jangan campurkan banyak perubahan acak di branch yang sama.
6. **Periksa Conflict Sebelum Merge**: Kolaborasi intens memicu merge conflict. Selesaikan dengan hati-hati.

7. **Pahami *Rebase* vs *Merge***: Rebase cocok untuk menjaga riwayat tetap bersih, tetapi hindari me-rebase branch publik.
8. **Backup**: Meski Git menyimpan riwayat, tak ada salahnya menjaga backup di beberapa remote, jaga-jaga terjadi insiden.

4.10 Studi Kasus: Mengelola Proyek Website dengan Git

Misalkan Anda memiliki proyek website yang melibatkan beberapa pengembang. Alur kerjanya:

1. **Inisialisasi dan Setup**:
   - Buat repositori di GitHub bernama `website-project`.
   - Clone repositori ke komputer lokal: `git clone https://github.com/user/website-project.git`.
2. **Membuat Branch Fitur**:
   - Anda ditugaskan membuat halaman "Tentang Kami".
   - Buat branch baru: `git checkout -b about-page`.
   - Edit file `about.html`, tambahkan konten.
   - `git add . && git commit -m "Menambahkan halaman Tentang Kami"`.
3. **Menjalankan Tes dan Linter**:
   - Pastikan HTML sudah lolos validasi.
   - Jalankan tool linting (jika tersedia).
4. **Push dan Pull Request**:
   - `git push -u origin about-page`.
   - Buka GitHub, buat pull request dari `about-page` ke `main`.
5. **Code Review**:
   - Rekan tim meninjau perubahan, memberi saran perbaikan.
   - Anda melakukan revisi jika perlu. Commit lagi, push, pull request diperbarui otomatis.
6. **Merge**:
   - Setelah disetujui, rekan tim atau Anda menggabungkan `about-page` ke `main`.
7. **Deploy**:
   - Setiap commit di `main` memicu CI/CD, misalnya men-*deploy* ke server staging atau production.

Dengan Git, proses ini terdokumentasi. Anda dapat mundur ke commit sebelumnya jika terjadi kesalahan. Setiap pengembang bisa membuat branch sendiri untuk fitur lain, lalu menggabungkan setelah siap. Hal ini meminimalkan kekacauan di branch `main`.

4.11 Error dan Konflik Umum dalam Git

1. **Merge Conflict**: Terjadi jika dua branch mengubah baris yang sama di file yang sama. Anda harus membuka file, mencari penanda conflict seperti `<<<<<<< HEAD`, lalu memutuskan versi mana yang diambil.
2. **Detached HEAD**: Saat Anda check out commit spesifik, HEAD tidak menunjuk ke cabang mana pun. Hasil commit tak terhubung ke branch tertentu.
3. **Rebase Bermasalah**: Mengubah riwayat yang sudah dibagikan ke orang lain bisa menimbulkan kebingungan.
4. **Tidak Sinkron dengan Remote**: Terjadi ketika Anda mengedit file yang sudah diubah orang lain namun belum melakukan pull.
5. **Menimpa Commit Orang Lain**: Jika Anda melakukan *force push* (`git push --force`), commit rekan bisa terhapus.

Melewati kesulitan ini memerlukan praktik rutin dan pemahaman mendalam. Kesalahan wajar terjadi, tetapi Git memiliki banyak mekanisme pemulihan.

BAB 5: Studi Kasus dan Praktik

Pada bab ini, kita akan menggabungkan semua elemen yang telah dipelajari—mulai dari HTML dasar, fitur-fitur lanjutannya, validasi kode untuk memastikan kepatuhan terhadap standar, hingga penggunaan Git dalam mengelola versi kode. Bab 5 difokuskan pada sebuah studi kasus yang menyeluruh. Kita akan mencoba membuat sebuah *mini project* berbasis HTML, melakukan validasi, dan mengatur alur kerja (workflow) dengan Git. Selain itu, kita akan mempersiapkan proyek ini agar siap untuk di-*deploy* atau dikembangkan lebih lanjut. Tujuan akhirnya adalah agar Anda memahami bagaimana cara mengintegrasikan seluruh materi secara praktis dalam skenario dunia nyata.

5.1 Deskripsi Proyek dan Tujuan

Kita akan membangun sebuah **situs web sederhana** berisi halaman-halaman berikut:

1. **Halaman Beranda (index.html)**: Menampilkan ringkasan konten situs.
2. **Halaman Profil (about.html)**: Menjelaskan pemilik atau tim pengembang situs, termasuk foto dan deskripsi singkat.
3. **Halaman Portofolio (portfolio.html)**: Berisi daftar proyek atau contoh karya, lengkap dengan gambar dan tautan.
4. **Halaman Kontak (contact.html)**: Menyediakan formulir bagi pengunjung untuk mengirim pesan atau pertanyaan.

Selain itu, kita akan melakukan hal-hal berikut:

- Menggunakan **semantic tags** dalam HTML5 untuk struktur yang lebih baik.
- Memastikan setiap halaman **valid** menurut validator.w3.org.
- Mengelola berkas-berkas proyek dengan **Git**.
- Menerapkan alur kerja kolaborasi minimal (branching, commit, merge) untuk memudahkan pengembangan jangka panjang.

Di akhir bab, Anda diharapkan mampu:

- Menulis kode HTML yang rapi dan mudah dipahami.
- Menyiapkan form kontak dengan sedikit validasi bawaan HTML5.
- Menyimpan riwayat perubahan menggunakan Git.
- Membuat branch, melakukan merge, dan memeriksa potensi konflik.
- Menyusun situs yang siap diunggah (deploy) ke hosting atau layanan static site.

5.2 Persiapan dan Struktur Folder

Struktur folder untuk proyek kita dapat disusun seperti ini:

```
my-website-project/
```

```
├── index.html
├── about.html
├── portfolio.html
├── contact.html
├── assets/
│   ├── images/
│   └── css/
└── README.md
```

Penjelasan singkat:

- **index.html**: Halaman beranda.
- **about.html**: Halaman profil.
- **portfolio.html**: Halaman portofolio.
- **contact.html**: Halaman kontak.
- **assets/images/**: Menampung berbagai gambar.
- **assets/css/**: Menyimpan file CSS (jika dibutuhkan).
- **README.md**: Dokumentasi proyek (bisa berformat Markdown) yang menjelaskan tujuan, cara penggunaan, dan lain-lain.

Anda bebas memodifikasi struktur sesuai kebutuhan, misalnya menambahkan folder **scripts** untuk JavaScript. Namun, contoh di atas cukup bagi kita untuk memulai.

5.3 Membuat Repositori Git

Sebelum menulis kode, mari **inisialisasi** repositori Git dan menyiapkan environment:

Buat Folder Proyek
mkdir my-website-project

1. cd my-website-project
2. **Inisialisasi Git**

   ```
   git init
   ```

   Perintah ini membuat folder tersembunyi `.git`.

3. Buat File Awal

   ```
   touch index.html about.html portfolio.html contact.html
   mkdir -p assets/images
   mkdir -p assets/css
   ```

4. **Buat Commit Pertama**

```
git add . git commit -m "Inisialisasi struktur folder untuk proyek web"
```

5. Buat Repositori Remote (Opsional)**

   - Jika Anda menggunakan GitHub, buat repositori baru di GitHub.

   - Tambahkan remote:

```
git remote add origin https://github.com/username/my-website-project.git
git push -u origin main
```

Dengan langkah-langkah di atas, proyek Anda sudah terlacak oleh Git. Setiap perubahan yang Anda buat pada file dapat di-*commit* dan dilihat riwayatnya.

5.4 Membangun Halaman Beranda (index.html)

Mari mulai dengan `index.html`, yang akan menjadi pintu utama ke situs kita. Kode contoh minimal:

```
<!DOCTYPE html>
<html lang="id">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Selamat Datang - My Website Project</title>
</head>
<body>
  <header>
    <h1>Selamat Datang di My Website Project</h1>
    <nav>
      <ul>
        <li><a href="index.html">Beranda</a></li>
        <li><a href="about.html">Tentang</a></li>
        <li><a href="portfolio.html">Portofolio</a></li>
        <li><a href="contact.html">Kontak</a></li>
      </ul>
    </nav>
  </header>

  <main>
    <section>
```

```
      <h2>Pengantar</h2>
      <p>Selamat datang di situs ini. Anda dapat menelusuri beragam
informasi dan proyek di sini.</p>
    </section>

    <section>
      <h2>Highlight</h2>
      <p>Bagian ini dapat memuat konten unggulan, gambar slider, atau
informasi penting lainnya.</p>
    </section>
  </main>

  <footer>
    <p>&copy; 2025 My Website Project</p>
  </footer>
</body>
</html>
```

Penjelasan dan Tips:

1. **Semantic Tags**: Penggunaan `<header>`, `<main>`, `<section>`, dan `<footer>` akan membantu mesin pencari dan pembaca layar memahami struktur konten.
2. **Menu Navigasi**: Setiap tautan mengarah ke halaman lain. Anda dapat menambahkan *class* atau *id* untuk styling.
3. **Meta Tags**: `viewport` penting agar halaman responsif di perangkat seluler.
4. **Bahasa Dokumen**: `lang="id"` menandakan bahwa dokumen ditulis dalam bahasa Indonesia.

Setelah menambahkan kode ini, lakukan commit:

```
git add index.html
git commit -m "Menambahkan struktur dasar halaman beranda"
```

5.5 Menulis Halaman Profil (about.html)

Halaman ini menjelaskan latar belakang pemilik situs atau tim pengembang. Contoh sederhana:

```
<!DOCTYPE html>
<html lang="id">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
  <title>Tentang Kami - My Website Project</title>
</head>
<body>
  <header>
    <h1>Tentang Kami</h1>
    <nav>
      <ul>
        <li><a href="index.html">Beranda</a></li>
        <li><a href="about.html">Tentang</a></li>
        <li><a href="portfolio.html">Portofolio</a></li>
        <li><a href="contact.html">Kontak</a></li>
      </ul>
    </nav>
  </header>

  <main>
    <section>
      <h2>Profil Singkat</h2>
      <p>Kami adalah tim pengembang web yang berfokus pada pembuatan
aplikasi modern dan responsif.</p>
      <img src="assets/images/team.jpg" alt="Foto Tim">
    </section>

    <section>
      <h2>Visi dan Misi</h2>
      <p>Visi kami adalah menciptakan solusi digital yang memudahkan hidup
masyarakat. Misi kami antara lain:</p>
      <ul>
        <li>Menyediakan layanan pengembangan berkualitas tinggi</li>
        <li>Memprioritaskan pengguna dalam setiap keputusan desain</li>
        <li>Terus berinovasi sesuai tren teknologi terbaru</li>
      </ul>
    </section>
  </main>

  <footer>
    <p>&copy; 2025 My Website Project</p>
  </footer>
</body>
</html>
```

1. **Penggunaan `<img>`**: Selalu sertakan `alt`. Misalnya, `alt="Foto Tim"`.
2. **Struktur**: Masih memanfaatkan `<header>`, `<main>`, `<section>`, dan `<footer>`.

Lakukan commit lagi:

```
git add about.html
git commit -m "Menambahkan halaman Tentang Kami"
```

5.6 Membuat Halaman Portofolio (portfolio.html)

Halaman ini menampilkan daftar proyek atau karya yang ingin Anda tonjolkan. Misalnya:

```
<!DOCTYPE html>
<html lang="id">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Portofolio - My Website Project</title>
</head>
<body>
  <header>
    <h1>Portofolio</h1>
    <nav>
      <ul>
        <li><a href="index.html">Beranda</a></li>
        <li><a href="about.html">Tentang</a></li>
        <li><a href="portfolio.html">Portofolio</a></li>
        <li><a href="contact.html">Kontak</a></li>
      </ul>
    </nav>
  </header>

  <main>
    <section>
      <h2>Karya Kami</h2>
      <article>
        <h3>Aplikasi Pemesanan Tiket</h3>
        <img src="assets/images/app1.jpg" alt="Screenshot Aplikasi
Pemesanan Tiket">
        <p>Aplikasi berbasis web untuk memesan tiket transportasi.</p>
      </article>

      <article>
        <h3>Website Edukasi Interaktif</h3>
        <img src="assets/images/app2.jpg" alt="Screenshot Website Edukasi">
```

```
        <p>Platform pembelajaran dengan kuis dan modul interaktif.</p>
      </article>
    </section>
  </main>

  <footer>
    <p>&copy; 2025 My Website Project</p>
  </footer>
</body>
</html>
```

- `<article>`: Cocok untuk menandai satu entitas portofolio.
- Anda dapat menambahkan tautan "Demo" atau "Source Code".

Commit perubahan:

```
git add portfolio.html
git commit -m "Menambahkan halaman Portofolio"
```

5.7 Formulir Kontak (contact.html) dan Validasi HTML5

Untuk interaksi dengan pengunjung, kita butuh halaman kontak:

```
<!DOCTYPE html>
<html lang="id">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Kontak - My Website Project</title>
</head>
<body>
  <header>
    <h1>Kontak</h1>
    <nav>
      <ul>
        <li><a href="index.html">Beranda</a></li>
        <li><a href="about.html">Tentang</a></li>
        <li><a href="portfolio.html">Portofolio</a></li>
        <li><a href="contact.html">Kontak</a></li>
      </ul>
    </nav>
  </header>
```

```
  <main>
    <section>
      <h2>Hubungi Kami</h2>
      <form action="#" method="POST">
        <label for="name">Nama:</label>
        <input type="text" id="name" name="name" required>

        <label for="email">Email:</label>
        <input type="email" id="email" name="email" required>

        <label for="message">Pesan:</label>
        <textarea id="message" name="message" rows="4" required></textarea>

        <button type="submit">Kirim</button>
      </form>
    </section>
  </main>

  <footer>
    <p>&copy; 2025 My Website Project</p>
  </footer>
</body>
</html>
```

1. **Atribut `required`**: Mewajibkan input pada *field* tertentu.
2. **`type="email"`**: Memverifikasi format email di sisi klien.
3. **`<textarea>`**: Untuk pesan yang lebih panjang.

Commit lagi:

```
git add contact.html
git commit -m "Menambahkan halaman Kontak beserta form"
```

5.8 Validasi Kode HTML dengan W3C

Setelah semua halaman dibuat, mari cek validasinya:

1. Buka validator.w3.org
2. Pilih **Validate by File Upload**.
3. Unggah `index.html`, lihat hasilnya.
4. Perbaiki error jika ada, lalu ulangi.

Lakukan hal yang sama untuk `about.html`, `portfolio.html`, dan `contact.html`. Beberapa error atau peringatan umum:

- Lupa menutup tag `<p>`.
- Gambar `<img>` tanpa atribut `alt`.
- Penggunaan karakter khusus tanpa *entity*.

Jika sudah bersih dari error, kita dapat yakin struktur HTML kita cukup sesuai standar. Validasi ini tak menjamin kesempurnaan, tapi mengeliminasi kesalahan sintaksik mendasar.

Commit jika ada perbaikan:

```
git add .
git commit -m "Memperbaiki kesalahan hasil validasi HTML"
```

5.9 Branching: Penambahan Fitur Baru

Dalam skenario nyata, pengembangan situs jarang berhenti sampai di sini. Anggaplah Anda ingin menambahkan **Halaman Blog** di `blog.html`. Untuk memisahkan pengembangan dari branch `main`, lakukan:

```
git checkout -b blog-page
```

Buat file `blog.html`:

```
<!DOCTYPE html>
<html lang="id">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Blog - My Website Project</title>
</head>
<body>
  <header>
    <h1>Blog</h1>
    <nav>
      <ul>
        <li><a href="index.html">Beranda</a></li>
        <li><a href="about.html">Tentang</a></li>
        <li><a href="portfolio.html">Portofolio</a></li>
        <li><a href="blog.html">Blog</a></li>
        <li><a href="contact.html">Kontak</a></li>
```

```
      </ul>
    </nav>
  </header>

  <main>
    <section>
      <article>
        <h2>Judul Blog Post Pertama</h2>
        <p>Tuliskan konten blog post di sini.</p>
      </article>

      <article>
        <h2>Judul Blog Post Kedua</h2>
        <p>Konten blog post kedua.</p>
      </article>
    </section>
  </main>

  <footer>
    <p>&copy; 2025 My Website Project</p>
  </footer>
</body>
</html>
```

Sesuaikan *navigation* agar menambahkan tautan ke blog. Setelah selesai:

```
git add blog.html
git commit -m "Menambahkan halaman Blog"
```

Jika Anda menggunakan GitHub, lakukan `git push origin blog-page`. Tim lain bisa meninjau lewat *pull request* sebelum digabungkan ke `main`. Setelah disetujui, gabungkan:

```
git checkout main
git merge blog-page
```

5.10 Menyiapkan Proyek untuk Deploy

Ketika situs siap dipublikasikan, Anda dapat mengunggahnya ke layanan hosting statis seperti GitHub Pages atau Netlify. Misalnya, jika memilih GitHub Pages:

1. **Buka Pengaturan (Settings)** di repositori GitHub.
2. Pilih **Pages**.

3. Atur branch yang ingin dijadikan sumber (misal `main`).
4. Tunggu sejenak, GitHub Pages akan menghasilkan URL di mana situs statis bisa diakses.

Anda juga bisa memakai Netlify, Vercel, atau layanan hosting konvensional. Pastikan file HTML berada di struktur yang sesuai agar server dapat menayangkannya.

5.11 Peningkatan Aksesibilitas dan Performa

1. **Aksesibilitas**:
    - Pastikan setiap gambar punya teks alternatif.
    - Gunakan label yang jelas pada form.
    - Atur *tab index* untuk navigasi keyboard jika diperlukan.
2. **Optimasi Gambar**:
    - Kompres ukuran gambar agar situs lebih cepat dimuat. Format modern seperti WebP bisa dipertimbangkan.
3. **Responsivitas**:
    - Tambahkan CSS responsif. Anda bisa memanfaatkan *media queries* agar tampilan di perangkat seluler lebih optimal.
4. **Penggunaan `<meta>`**:
    - Tambahkan `<meta name="description" content="Deskripsi singkat situs">` untuk SEO.

5.12 Kontributor, Issue, dan Roadmap

Jika proyek ini dikelola tim:

- Gunakan **GitHub Issues** untuk mencatat bug atau tugas.
- Ciptakan **Milestone** dan **Project Board** agar rencana pengembangan lebih terorganisir.
- Menerima **Pull Request** dari kontributor eksternal.

Dokumentasi di `README.md` perlu menjelaskan:

- Cara mengkloning, memasang, dan menjalankan proyek.
- Daftar fitur saat ini dan fitur yang akan datang.
- Panduan kontribusi (CONTRIBUTING.md).

5.13 Studi Kasus Lanjutan

Sebagai variasi, bayangkan Anda ingin menambahkan konten video dan audio di halaman `index.html`. Anda bisa menambah:

```
<video controls width="320" height="240">
```

```
  <source src="assets/videos/intro.mp4" type="video/mp4">
  Browser Anda tidak mendukung pemutar video.
</video>

<audio controls>
  <source src="assets/audios/background-music.mp3" type="audio/mpeg">
  Browser Anda tidak mendukung pemutar audio.
</audio>
```

Atau Anda ingin menambahkan elemen `<canvas>` untuk animasi interaktif. Anda bahkan dapat memasukkan `<svg>` langsung ke HTML untuk ikon custom. Pastikan setiap penambahan tetap mengikuti standar.

Satu hal yang tidak kalah penting adalah menulis **pesan commit** yang deskriptif dan melakukan **branch** setiap kali Anda menambah fitur atau memperbaiki bug. Hindari menjejalkan terlalu banyak perubahan berbeda dalam satu commit.

5.14 Kesalahan dan Tantangan Umum

1. **Terjadi Merge Conflict**: Dapat diatasi dengan menyunting file yang bentrok secara manual.
2. **Link Internal Rusak**: Pastikan file HTML saling merujuk dengan benar. Terkadang, kita lupa memperbarui tautan di menu navigasi.
3. **Gambar Tidak Muncul**: Salah folder atau penulisan path.
4. **Validasi HTML Gagal**: Tag `<p>` tidak ditutup, *attribute* salah eja, dsb.
5. **Masalah Responsivitas**: Tanpa CSS atau *framework*, halaman mungkin kurang optimal di layar kecil.
6. **Belum Memperhatikan Aksesibilitas**: Lupa atribut `alt`, form tanpa label, dsb.
7. **Menimpa Commit Rekan**: Saat kolaborasi, jangan gunakan `git push --force` kecuali Anda mengerti konsekuensinya.

Pemahaman dan pengalaman praktek langsung akan membantu Anda mengatasi tantangan-tantangan ini.

Lampiran

Lampiran A: Daftar Perintah Git yang Umum

1. **Konfigurasi Pengguna**
    - `git config --global user.name "Nama Anda"`
    - `git config --global user.email "email@domain.com"`
2. **Inisialisasi dan Kloning**
    - `git init`
    - `git clone <URL>`
3. **Menambahkan dan Men-commit Perubahan**
    - `git add <file>` atau `git add .`
    - `git commit -m "pesan"`
4. **Melihat Status dan Riwayat**
    - `git status`
    - `git log` atau `git log --oneline`
5. **Branching**
    - `git branch <nama-branch>`
    - `git checkout -b <nama-branch>`
6. **Merging**
    - `git checkout <branch-tujuan>`
    - `git merge <branch-sumber>`
7. **Push dan Pull**
    - `git push -u origin <branch>`
    - `git pull`
8. **Lainnya**
    - `git stash` / `git stash pop`
    - `git tag <nama-tag>` / `git push origin <nama-tag>`

Lampiran B: Cheat Sheet HTML dan Validasi HTML

Struktur Dasar HTML

```
<!DOCTYPE html>
<html lang="id">
<head>
  <meta charset="UTF-8">
  <title>Judul Halaman</title>
</head>
<body>
```

```
  <h1>Contoh Heading</h1>
  <p>Paragraf contoh.</p>
</body>
</html>
```

1. **Tag Umum**
    - `<h1>` hingga `<h6>`: Heading
    - `<p>`: Paragraf
    - `<a>`: Tautan
    - `<img>`: Gambar
    - `<ul>` / `<ol>`: Daftar
    - `<table>`, `<tr>`, `<td>`: Tabel
2. **Atribut Penting**
    - `src`: Lokasi file (untuk `<img>`)
    - `alt`: Teks alternatif (untuk `<img>`)
    - `href`: Target tautan (untuk `<a>`)
    - `target`: Cara membuka tautan (untuk `<a>`)
3. **Semantic Tags**
    - `<header>`, `<section>`, `<article>`, `<nav>`, `<footer>`
4. **Validasi HTML**
    - validator.w3.org
    - Pastikan semua tag terbuka dan tertutup dengan benar.
    - Gunakan atribut `alt` pada gambar.
    - Hindari penggunaan tag usang (deprecated).

Lampiran C: Glosarium

- **HTML**: HyperText Markup Language, bahasa markup untuk membuat halaman web.
- **Form**: Elemen untuk input data user, seperti `<input>` dan `<textarea>`.
- **DOCTYPE**: Deklarasi di awal dokumen yang menandakan versi HTML.
- **Semantic Tags**: Tag yang memberikan makna jelas, seperti `<header>`, `<footer>`, `<article>`.
- **Validasi HTML**: Proses pengecekan kode agar sesuai standar W3C.
- **Git**: Sistem pengontrol versi yang memudahkan pelacakan perubahan pada kode.
- **Branch**: Cabang pengembangan yang independen.
- **Merge**: Proses penggabungan branch.
- **Repo (Repository)**: Tempat menyimpan seluruh riwayat perubahan kode.

Lampiran D: Referensi dan Bacaan Lanjutan

1. **Dokumentasi Resmi HTML5**

    - W3C HTML (Spesifikasi resmi)
    - MDN Web Docs: HTML
2. **Validasi HTML**
    - W3C Markup Validation Service
    - HTML Living Standard (WHATWG)
3. **Git**
    - Pro Git (Buku Gratis)
    - Git Reference
    - Atlassian Git Tutorials
4. **Tutorial dan Panduan Lain**
    - FreeCodeCamp (Kursus dan proyek HTML/CSS/JS)
    - W3Schools (Tutorial interaktif)
    - CSS-Tricks (Kiat dan trik seputar CSS dan front-end)
5. **Tools dan Sumberdaya**
    - Can I Use (Mengecek dukungan browser untuk fitur HTML5/CSS/JS)
    - Stack Overflow (Forum tanya-jawab pengembang)

Referensi

Duckett, J. (2011). *HTML & CSS: Design and Build Websites*. John Wiley & Sons.

Castro, E., & Hyslop, B. (2013). *HTML5 & CSS3: Visual QuickStart Guide* (8th ed.). Peachpit Press.

Chacon, S., & Straub, B. (2014). *Pro Git* (2nd ed.). Apress.
Diakses dari https://git-scm.com/book/en/v2

Bird, C., Rigby, P. C., Devanbu, P., & German, D. M. (2009). The Promises and Perils of Mining Git. *Proceedings of the 6th IEEE International Working Conference on Mining Software Repositories*, 1–10. IEEE.

Lima, J., & Sethi, S. (2019). Evaluating the Impact of Collaborative Version Control in Large-Scale Software Projects. *ACM Transactions on Software Engineering*, 38(2), 15–29.

Addison, E. (2020). Analyzing Web Standards Adoption and Validation Tools in Modern Web Development. *Journal of Web Engineering*, 17(3), 185–207.

Freedman, A., & Boren, M. (2018). Ensuring Code Quality through Continuous Integration and Continuous Deployment: A Git-based Approach. *International Journal of Computer Science & Applications*, 45(4), 321–335.

Niederst Robbins, J. (2018). *Learning Web Design: A Beginner's Guide to HTML, CSS, JavaScript, and Web Graphics* (5th ed.). O'Reilly Media.

Richards, B. (2019). Enhancing Accessibility and Semantic Structure in HTML5-based Systems. *IEEE Access*, 7, 118–129.

Norton, L. (2021). Effective Collaboration with GitHub Flow: A Case Study. In *Proceedings of the 23rd International Conference on Cooperative Computing* (pp. 77–88). Springer.