DOMS | Page No.
Date / /

Multi-page Website

What is a multi-page website (MPA)?

- Defination: An MPA is a website each page has its own HTML file and the browser request a new HTML document from the server when the user navigates (eg:- index.HTML, about.html, contact.html).

- How it Contrasts with SPA: Single page Application dynamicdly update content on one HTML file (client-side routing) MPAs use full page reloads and server routing. MPAs are simpler, SEO friendly by default and often better for content driven sites.

Typical file/folder structure
A clean predictable structure avoids path errors and helps collaboration

```
/project-root
  /assets
    /css
      style.css
      about.css
    /js
      main.js
      contacts.js
```

/images
  logo.png
  hero.jpg
index.html
about.html
contact.html
/partials <-- Optional for templating / build tools
  header.html
  footer.html
404.html

- Use lowercase filenames, hyphens for space and keep extensions consistent
- Group assets by type (css, js, images) for large projects consider /assets/fonts, /assets/icons.

Linking between pages - paths explained
- Relative paths: href="about.html" (works for same level)
- Relative to subfolders: href="../index.html" to go up one level
- Absolute paths: href="/about.html" (root-relative) used for consistent linking on server.
- External links use full URLs: href="https://ex.com"

Common mistakes
- Using ./about.html vs about.html - both usually work but be consistent
- Forgetting ../ when linking from nested folder (eg. /pages/team/index.html linking to root assets).

Date / /

Reusable header/footer - Dry principle

- Many MPAs repeat header/footer. Options:
  - Server-side includes (PHP include, templating engines)
  - Build tools (Gulp, webpack, Eleventy) that assemble pages from partials
  - Client-side JS injection (fetch header.html and insert) - works but less ideal for SEO
- Example simple header include (pseudo Server-side)

```
<!-- header.html -->
<header>
  <nav>
    <a href="/index.html">Home</a>
    <a href="/about.html">About</a>
  <nav>
</header>
```

Then server templating injects that into each page

HTML essentials for each page

Each page should have:

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-
                                 width, initial-scale=1">
```

```
<title> Page Title - Site Name </title>
<link rel="stylesheet" href="/assets/css/style.css">
<link rel="icon" href="/assets/images/favicon.ico">
<meta name="description" content="short, unique description
                                  for this page.">
<link rel="canonical" href="https://yourdomain.com/about.
                                  html">
</head>
<body>
  <!-- header -->
  <!-- main content -->
  <!-- footer -->
  <script src="/assets/js/main.js" defer></script>
</body>
</html>
```

- Meta description is critical for Search Snippets
- Canonical tag prevents duplicate content issues for similar pages
- Use defer on scripts to avoid render-blocking unless immediate execution is needed.

CSS Organization and Page-Specific Styling

- Keep global styles in style.css but page-specific rules in about.css and only load them on that page
- Use a CSS reset or normalize.css at top of stylesheet to smooth browser differences
- Use semantic class names (BEM if team scale demands) .nav_item, .hero-title
- Avoid inline styles keep layout in CSS

DOMS | Page No.
Date / /

Navigation & active state

- Show which page is active with class `.active` on the nav link
- Use server logic or templating to add `.active` on the current page or use JS to match
- Example CSS:

```
nav a.active {
    font-weight: 700;
    border-bottom: 2px solid;
}
```

Forms and Contact pages

- Use proper `label` element and `name` attributes:

```
<form action="/contact-submit" method="POST">
  <label for="email">Email</label>
  <input id="email" name="email" type="email" required>
  <button type="submit">Send</button>
</form>
```

- Add server validation and client validation. (HTML 5.+ JS)
- Protect with CAPTCHA or rate-limiting on the server side

Accessibility (a11y) - non negotiable

- alt text on images, Semantic tags (<main>, <nav>, <section>).
- Keyboard navigation: ensure focus styles, logical tab order
- ARIA only when necessary; prefer native Semantic HTML
- Contrast ratio checks for text readability

SEO basics for MPAs
- Unique <title> and <meta description> per page
- Use heading (<h1> per page, then h2/h3) in logical order.
- Create and Submit sitemap.xml with all pages.
- Use readable URLs: /about.html -> /about (Server rewrite) or /about.html.
- Add structured data (JSON-LD) for rich results when relevant.

Performance tips
- Minify CSS/JS, compress images (webP where supported)
- Use link rel="preload" for important fonts or hero images when needed
- Serve assets with proper cache headers; consider versioning filenames (style.v1.css) to bust cache on updates.
- Avoid large JS on basic content pages.

Progressive enhancement & graceful degradation
- Build with HTML first so content works without JS
- Add interactivity progressively with JS
- For critical features, ensure Server-side fallback exists (eg. form submits without JS)

DOMS | Page No.
Date / /

When to use MPAs vs SPA (Quick decision guide)

- Use MPA if: Content heavy, SEO important, Simple hosting, less JS complexity
- Use SPA if: highly interactive app, authenticated dashboards, real-time UI where client routing is beneficial

Deployment basics

- For Static MPAs: GitHub pages, Netlify, Vercel or any static host
- For Server routes / forms: host with Node/PHP/ Netlify functions or similar backend
- Ensure 404 handling and redirect rules (eg:- /about → /about.html or rewrite to /about)

Debugging checklist (when pages don't load)

(1) check console for JS errors (DevTools → console)
(2) inspect network tab for 404s (missing CSS/JS/ images)
(3) verify paths are correct (relative vs absolute)
(4) Confirm server rewrite rules if using root relative links
(5) check <base> tag if present - it affects relative paths
(6) Test on different devices / screen sizes
(7) check robots.txt doesn't block important pages

Quick Sample : Simple nav with active link (HTML+JS)

```
<nav>
  <a href="/index.html">Home</a>
  <a href="/about.html">About</a>
  <a href="/contact.html">Contact</a>
</nav>
<script>
  // small helper to mark current link active
  document.querySelectorAll('nav a').forEach(a => {
    if (a.href === window.location.href) a.classList.add
                                          ('active');
    });
</script>
```

Classroom Exercises (mini labs)

(1) Build a 3-page site (Home, About, Contact) with shared header/footer using a templating method (Server includes or build tool)

(2) Add responsive navigation (hamburger for mobile) and test keyboard access

(3) Create a Contact form that posts to a mock endpoint and shows a success message; add client and server validation

(4) Create sitemap.xml and test with a local robots tester.