Page No.
Date / /

Multipurpose / Multipage website

1. What is a multi-page website (MPA)?

- A MPA is a website where each page has its own HTML file & the browser requests a new HTML document from the server when the user navigates.

  eg. index.html, about.html, contact.html

- Single page Applications dynamically update content on one HTML file (client side routing). MPAs use full page reloads & server routing. MPAs are simpler, SEO-friendly by default & often better for content driven sites

2. Typical file / folder structure

- A clean predictable structure avoids path errors & helps collaboration.

- User lowercase filenames, hyphens for spaces, & keep extensions consistent

- Group assests by type (css, js, images). for large projects consider

  /assets/fonts, /assets/icons

```
/project-root
  /asset
    /css
      style.css
      about.css
    /js
      main.js
      contact.js
    /images
      logo.png
      hero.jpg
  index.html
  about.html
  contact.html
  /partials  <-- optional for templating/build tools
    header.html
    footer.html
  404.html
```

3: Linking between Pages - path 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) - useful for linking on server for consistent
- External links use full URLs href = "https://example.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).

4. Reusable header/footer - DRY Principle

- Many MPAs repeat header/footer. options
  - Server side includes PHP includes, templating engines
  - Builds tools (Gulp, webpack, Eleventy) that assemble pages from partials
  - Client-side js injection (fetch header.html & insert) - works but less ideal for SEO.

Eg.

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

Than Server templating injects that into each pages

5. 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, scale=1">
    <title> Page Title - Site Name </title>
    <link rel="stylesheet" href="/assets/css/style.css" >
    <link rel="icon" href="/assets/image/favicon.ico">
    <meta name="description" content="short, unique description for page">
    <link rel="canonical" href="https://yourdomain.com/about.html">
</head>
<body>
        header
        main content
        foter
    <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

Date / /

6. Css organisation & page specific Styling

- Keep global styles in style.css Put pages-specific rules in about.css & only load them on that page.
- Use a Css reset or normalise.css at top of stylesheet to smooth browser differences.
- Use semantic class names
  BEM if team scale demands

  .nav_items
  .hero-title

- Avoid inline styles. Keep layout in Css

7. Navigation & active state

- Show which pages is active with a class .active on the nav link
- Use Server logic or templating to add .active on the current page, or use Js to match window.location.pathname

Eg

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

8. forms & contact pages

use proper label elements & name attributes

```
<form action="/contact-submit" method="post">
    <label for="email@gmail.com">Email</label>
    <input id="email" name="email" type="email" required>
    <button type="submit">Send</button>
</form>
```

- Add server validation & client validation (HTML5 + JS)
- Protect with CAPTCHA or rate-limiting on the server side.

9. Accessibility ('a11y') - non-negotiable

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

10 SEO basics for MPAs

- Unique <title> & <meta description> per page
- use headings (<h1> per page, then h2/h3) in logical order
- Create & 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 with relevant

11 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 header ; consider versioning filenames (style.v1.css) to bust cache on update
- Avoid large JS on basic content pages.

12 Progressive enhancement & grateful 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)

13. When to use MPA'S vs SPA (quick decision guide)

- Use MPA if ! content heavy, SEO important, simpler hosting, less JS complexity
- Use SPA if ! highly interactive app, authenticated dashboards real-time UI where client routing is beneficial.

DOMS | Page No.
Date / /

14 Deployment basics

- for static MPAs: Github Pages, Netlify, Vercel, or any static host.
- for server routes / forms: host with Node.js / PHP / Netlify function or similar backend
- Ensure 404 handling and redirect rules

eg. (/about → /about.html or rewrite to /about/)

15. Debugging checklists (when pages don't load or links break)

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 diff. devices / screen sizes.
7. Check robots.txt doesn't block important pages.

16 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>
```

DOMS | Page No.
Date / /

17 Classroom exercises (mini labs)

- Builds a 3 page site (Home, About, Contact) with shared header/footer using a templating method (Server includes or build tool).
- Add responsive navigation (hamburger for mobile) & test everyone keyboard access
- Create a contact form that posts to a mock end point and shows a success message; add client & server validation
- Create sitemap.xml; test with a local robots tester.

18 Final best Practice (Summary)

- Keep structure logical & consistent
- favor semantics, accessibility, & clean paths.
- Split CSS into global + page specific.
- Use server includes or build tooling for shared parts
- Optimize assets & add SEO metadata per page
- Test thoroughly on mobile, desktop & with assistive tech