SkillNewton

Website Development Checklist

A launch-ready checklist covering planning, HTML, accessibility, CSS, JavaScript, PHP, security, performance, SEO and deployment.

This checklist walks you through building a website from first plan to live launch. It is written for students and junior developers working with HTML, CSS, JavaScript, PHP and MySQL, and for freelancers who want a repeatable quality check before handing a site to a client. Work through it section by section during the build, then run the whole list again on the staging site before you go live.

How to use it

1. **Start with planning.** Complete the planning section before you write code, so scope, pages and hosting are agreed in writing.
2. **Check as you build.** Tick off the HTML, CSS, JavaScript and backend items as each part of the site is finished rather than leaving them all to the end.
3. **Test on staging.** Run the security, performance and SEO sections on a staging copy that matches the live server as closely as possible.
4. **Launch and verify.** Use the deployment section on launch day, then re-check forms, HTTPS and backups on the live site.

Planning

- [ ] Write down the site's goal, audience and main calls to action
  A one-page brief stops scope creep and tells you what each page must achieve.
- [ ] List every page and draw a simple sitemap
  Agree the navigation and URL structure before design starts; changing URLs later means redirects.
- [ ] Collect content, images and brand assets early
  Missing copy is the most common cause of launch delays. Confirm you have rights to every image and font.
- [ ] Choose the stack and hosting
  Check the host supports your PHP and MySQL versions, HTTPS, backups and SSH or SFTP access.
- [ ] Register the domain and plan DNS
  Keep domain registrar logins with the client, not just the developer.
- [ ] Set up version control from day one
  Create a Git repository and commit small, described changes. Never commit passwords or API keys.
- [ ] Agree browsers and devices you will support
  Usually the current versions of Chrome, Edge, Firefox and Safari, plus Android and iOS phones.

HTML, semantics and accessibility

- [ ] Start every page with `<!DOCTYPE html>` and `<html lang="en">`
  The `lang` attribute tells screen readers and browsers which language to use.

- [ ] Use landmarks: `header`, `nav`, `main`, `footer`
  Landmarks let assistive technology users jump straight to the main content. Use one `main` per page.
- [ ] Use one H1 and a logical heading order
  Do not skip levels for styling; headings form the page outline for screen reader users.
- [ ] Give every meaningful image useful alt text; use `alt=""` for decorative ones
  Describe what the image conveys in context, not "image of".
- [ ] Label every form field with a `<label>`
  Placeholders are not labels. Link them with `for` and `id`, and show errors in text, not colour alone.
- [ ] Use real `<button>` and `<a>` elements
  Buttons perform actions, links go somewhere. Clickable `div`s are not keyboard accessible by default.
- [ ] Check colour contrast meets WCAG 2.2 AA
  Normal text needs at least 4.5:1, large text 3:1, and UI components and focus indicators 3:1. Use a contrast checker.
- [ ] Test the whole page with the keyboard only
  Tab through every link and control: focus must be visible, in a logical order and never hidden behind sticky headers.
- [ ] Validate the HTML
  Use the W3C Nu HTML Checker to catch unclosed tags, duplicate IDs and invalid nesting.
- [ ] Run an automated accessibility scan, then check manually
  Tools such as Lighthouse or axe DevTools catch some issues only; keyboard and screen reader checks find the rest.

CSS and responsive layout

- [ ] Add the viewport meta tag to every page
  `<meta name="viewport" content="width=device-width, initial-scale=1">` makes mobile browsers use the device width.
- [ ] Build mobile-first with `min-width` media queries
  Start with the small-screen layout and add complexity as space grows.
- [ ] Use Flexbox and Grid instead of floats for layout
  They are supported in all modern browsers and make responsive layouts simpler.
- [ ] Use relative units for type and spacing
  `rem` for font sizes respects the user's browser settings; avoid fixing heights on text containers.
- [ ] Check there is no horizontal scrolling at 320px wide
  Overflowing tables, long URLs and fixed-width elements are the usual causes.
- [ ] Respect `prefers-reduced-motion`
  Turn off or reduce non-essential animation for users who request it.
- [ ] Remove unused CSS and keep one consistent set of styles
  Use custom properties (CSS variables) for colours and spacing so changes happen in one place.

JavaScript

- [ ] Make core content and navigation work without JavaScript where practical
  Progressive enhancement keeps the site usable if a script fails to load.
- [ ] Load scripts with `defer` or `type="module"`
  This stops scripts from blocking HTML parsing.
- [ ] Check the browser console for errors on every page
  Open DevTools and fix all errors and warnings before launch.

- [ ] Never trust client-side validation alone
  Validate again on the server; JavaScript checks are for convenience only.
- [ ] Use `textContent` rather than `innerHTML` for user data
  Inserting untrusted strings as HTML creates cross-site scripting (XSS) risks.
- [ ] Keep third-party scripts to a minimum
  Each tag, widget or tracker adds weight and a security dependency. Remove ones nobody uses.
- [ ] Manage focus in modals and dynamic content
  Move focus into an opened dialog, trap it there and return it when the dialog closes.

PHP, backend and database

- [ ] Run a supported PHP version
  Check the PHP supported versions page on php.net; versions past end of life receive no security fixes.
- [ ] Use prepared statements for every database query
  PDO or MySQLi prepared statements with bound parameters prevent SQL injection.
- [ ] Escape output with `htmlspecialchars()`
  Escape for the context you are writing into (HTML, attribute, URL or JavaScript).
- [ ] Hash passwords with `password_hash()`
  Verify with `password_verify()`. Never store plain text or use MD5 or SHA-1 for passwords.
- [ ] Keep credentials out of code
  Store database passwords and API keys in environment variables or a config file outside the web root, and add it to `.gitignore`.
- [ ] Turn off `display_errors` in production
  Log errors to a file instead so visitors never see paths or queries.
- [ ] Design tables with correct types, keys and indexes
  Use primary keys, foreign keys where relationships matter and `utf8mb4` for full Unicode support in MySQL.
- [ ] Give the application database user only the privileges it needs
  The web app rarely needs `DROP` or `GRANT`; do not connect as root.

Security basics

- [ ] Serve the whole site over HTTPS
  Install a TLS certificate (for example from Let's Encrypt) and redirect all HTTP requests to HTTPS with a 301.
- [ ] Add CSRF tokens to every state-changing form
  Check the token on the server for each POST request.
- [ ] Set secure session cookie flags
  Use `Secure`, `HttpOnly` and `SameSite=Lax` (or `Strict`), and regenerate the session ID on login.
- [ ] Add basic security headers
  Content-Security-Policy, X-Content-Type-Options: nosniff, Referrer-Policy and Strict-Transport-Security once HTTPS is stable.
- [ ] Restrict file uploads
  Check type and size on the server, rename files and store them where they cannot be executed.
- [ ] Rate-limit login and contact forms
  Limit attempts and consider a CAPTCHA or honeypot field to cut spam and brute-force attacks.
- [ ] Remove test files, `phpinfo()` pages and directory listings
  Anything left on the server can be found and used by attackers.

Performance

- [ ] Compress and resize images; use WebP or AVIF where supported
  Serve images no larger than they are displayed, with `width` and `height` set to prevent layout shift.
- [ ] Lazy-load below-the-fold images with `loading="lazy"`
  Do not lazy-load the main hero image, which is often the Largest Contentful Paint element.
- [ ] Enable compression and browser caching
  Turn on Brotli or gzip and set cache headers for static files such as CSS, JS and fonts.
- [ ] Minify CSS and JavaScript
  Use a build step or your framework's asset pipeline.
- [ ] Check Core Web Vitals
  Aim for good LCP, INP and CLS. Test with PageSpeed Insights and Lighthouse on a mobile profile.
- [ ] Limit web fonts
  Load only the weights you use and set `font-display: swap`.

SEO basics

- [ ] Write a unique title tag and meta description for every page
  Keep titles descriptive and close to 60 characters so they are less likely to be cut short.
- [ ] Use clean, readable URLs
  Lowercase words separated by hyphens, for example `/services/web-design/`.
- [ ] Add a canonical tag to each indexable page
  It tells search engines the preferred URL when duplicates exist.
- [ ] Create `robots.txt` and an XML sitemap
  Make sure robots.txt does not block CSS, JS or pages you want indexed, and list the sitemap URL in it.
- [ ] Remove `noindex` left over from staging
  A forgotten noindex tag or blocking robots.txt is a classic launch mistake.
- [ ] Add structured data where it fits
  For example Organization or LocalBusiness markup. Validate with Google's Rich Results Test.
- [ ] Set up Google Search Console and submit the sitemap
  Verify the domain property and watch the indexing reports after launch.

Deployment and backups

- [ ] Deploy from version control, not by editing live files
  Use Git-based deployment or a repeatable SFTP process so you always know what is live.
- [ ] Test every form, email and payment flow on the live site
  Send real test submissions and confirm emails arrive and are not marked as spam.
- [ ] Set up 301 redirects for any changed URLs
  Map old URLs to the closest new page to keep links and rankings.
- [ ] Configure automatic backups of files and database
  Keep copies off the hosting server and test a restore before you rely on them.
- [ ] Add uptime monitoring
  A free uptime monitor alerts you when the site goes down or the certificate is about to expire.

- [ ] Set up analytics and respect consent requirements
  Install analytics and add a privacy policy that explains what you collect.
- [ ] Hand over documentation and logins securely
  Share credentials through a password manager, not email or chat.

Online version, with the matching free guide: skillnewton.com/resources/templates/website-development-checklist/
Learn this hands-on: PHP Web Development: Foundation & Practical Implementation – skillnewton.com/courses/php-foundation-course/