



Build and Publish a Website with AI from Your Phone

A complete mobile-first workflow for planning, coding, testing, securing,
and publishing a real website

What this book delivers

This book takes a beginner from an empty folder to a published, maintainable website using a phone, an AI assistant, Termux, Git, and GitHub Pages. AI is treated as a coding partner, not as an authority. Every generated change is inspected, tested, and committed so a broken result can be reversed.

By the end, you will have:

- A clear project brief and page map.
- A semantic HTML structure.
- A responsive design system written in CSS.
- Small, optional JavaScript enhancements.
- A bilingual content strategy.
- Accessibility, security, SEO, and performance checks.
- A Git history and a GitHub Pages deployment.
- A process for fixing 404 errors, broken paths, and regressions.

Static-site boundary. GitHub Pages publishes public static files. It cannot safely store secret API keys, process private server logic, or protect a paid download merely because the URL is hard to guess.

Contents

1. Define the website before asking AI
2. Prepare a phone-based toolchain
3. Create a durable project structure
4. Give AI enough context
5. Build semantic HTML
6. Create a responsive CSS system
7. Add JavaScript progressively
8. Build bilingual pages correctly
9. Design accessible interactions
10. Handle forms without exposing secrets
11. Add SEO and social metadata
12. Protect privacy and reduce security risk
13. Use Git as a safety system
14. Publish with GitHub Pages
15. Diagnose 404 and path failures
16. Connect a custom domain and HTTPS
17. Test performance and compatibility
18. Maintain the site without losing features
19. Build credible content and product pages
20. Prepare images and structured content
21. Applied website repair cases
22. Complete starter project
23. Choose a hosting model and commercial boundary
24. Automate quality checks from a phone
25. Measure traffic without losing user trust
26. Recover from a broken deployment
27. Release checklist and source notes

1. Define the website before asking AI

AI produces better code when the requested outcome is concrete. Begin with a one-page brief rather than a vague request such as “make me a cool website.”

Project brief

Write answers to these questions:

- Who is the primary visitor?
- What single action should that visitor take?
- What pages are required at launch?
- Which information must be easy to find on a phone?
- Which features are essential, optional, or prohibited?
- What content already exists?
- What languages are required?
- Where will the site be hosted?
- Who will maintain it after launch?

Build a page map

A small project might use:

```
Home
├── About
├── Lessons
│   ├── Lesson listing
│   └── Lesson page
├── Resources
├── FAQ
└── Contact
```

Each page needs one purpose. If two pages repeat the same content, combine them or give them clearly different jobs.

Define acceptance criteria

Acceptance criteria turn taste into tests:

- Navigation works by keyboard and touch.
- Layout fits at 320 CSS pixels without horizontal scrolling.
- Every image has appropriate alternative text or an empty `alt` when decorative.
- English and Greek contain equivalent information.
- No secret appears in source files.
- Every internal link is valid in the deployed path.

- A user can understand the homepage before scrolling.

2. Prepare a phone-based toolchain

You need a terminal, editor, browser, Git client, and an AI interface. Termux can provide the local development tools while the browser previews the site.

```
pkg update
pkg install git python nano
termux-setup-storage
mkdir -p "$HOME/projects"
cd "$HOME/projects"
```

Create the project:

```
mkdir mobile-site
cd mobile-site
git init
```

Editor choices

- `nano` is simple and always available in the terminal.
- A trusted Android code editor provides tabs, syntax highlighting, and file browsing.
- GitHub's web editor can make small emergency changes but is less comfortable for broad refactors.

Keep the authoritative project under `$HOME`, not shared storage. Shared Android storage is useful for importing assets and exporting ZIP files, but it may not preserve Unix file behavior.

Local preview

```
python -m http.server 8000 --bind 127.0.0.1
```

Open `http://127.0.0.1:8000/`. A local web server catches path and browser behavior that may not appear when opening an HTML file directly.