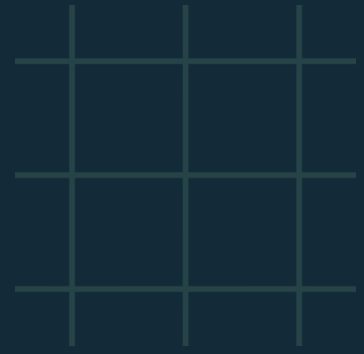


WORKING TITLE · VOLUME 2



From Lovable to WordPress

The Conversion

Yukesh Chaudhary

yukesh.com.np

BOOK NOTE

The Conversion

Turn an approved interface into a native, editable WordPress theme without losing the design that made the client say yes.

Working title

From Lovable to WordPress is the working title of this series. It can be changed before publication without changing the method taught inside.

Written by

Yukesh Chaudhary
yukesh.com.np

WHAT THIS VOLUME IS

A practical field guide for using AI as a coding partner. It teaches decisions, prompts, structure, and checks. It does not try to turn the reader into a PHP textbook.

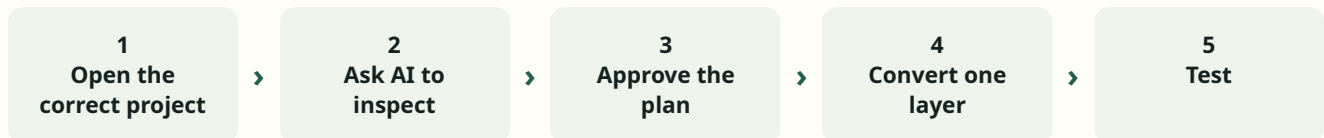
Important: Tool interfaces, model names, pricing, and free-tier limits can change. Check the current terms of every service. Use free tools and free tiers only in ways their providers allow.

Original edition. Website: <https://www.yukesh.com.np>

HOW TO USE THIS VOLUME

Inspect first. Convert in layers. Test every change.

The safest AI workflow is not one giant request. It is a controlled sequence with a clear checkpoint after each step.



The running example

ABC Construction is our fictional client. Volume 1 produced an approved Lovable interface with Home, About, Services, Projects, Team, and Contact pages. The Projects area has categories and project details. The client also needs menus, contact details, and calls to action they can manage.

THE VISUAL SOURCE

The Lovable project shows how the site should look and behave. It contains pages, components, styles, assets, routes, and front-end interactions.

THE WORDPRESS DESTINATION

The theme must connect that interface to WordPress content, templates, settings, menus, media, and admin workflows.

Read with a project open. After each chapter, perform the matching step on a copy or branch. Commit or back up before the next major change.

TABLE OF CONTENTS

From project folder to editable theme

01	Coding Without an AI Subscription	5
02	Giving AI the Correct Project	7
03	Converting the UI Into a WordPress Theme	9
04	The Master Conversion Prompt	12
05	Never Hard-Code What the Client Edits	14
06	Global Theme Settings	15
07	Precise Prompts for Admin Features	16
08	Pages Into Templates	17
09	Custom Post Types With AI	18
10	Slugs, Taxonomies, and Fields	19
11	Archives, Singles, and Listing Pages	20
12	Conversion Checkpoint and Prompt Library	22
13	In Volume 3, About the Author, References	23

My promise to the reader: I will explain what to ask for, why it matters, and how to judge the result. Code samples are small and purposeful.

A LOW-COST WORKING SETUP

Free tools can lower the cost, but discipline creates the result

I can use a free coding environment and a model such as DeepSeek to do serious work. I still have to supply the right project, the right context, and a testing loop.

A free tier is a limited offer, not a loophole. I use it within the service's current terms, limits, and acceptable-use rules. I do not create extra accounts to evade limits. I do not promise unlimited access. If a limit is reached, I stop, wait, change to another permitted tool, or pay for the work.

A practical tool choice

Choose an environment that can open the whole local project, search files, edit several files, show diffs, and run commands. Then choose a coding model available inside that environment. DeepSeek is a useful option for theme conversion, PHP troubleshooting, CSS changes, and WordPress functions. Other capable coding models can review or refactor the same work.

Tool-neutral rule: Keep the project in a normal folder and keep it under version control. If one tool changes or disappears, the work should move with you.

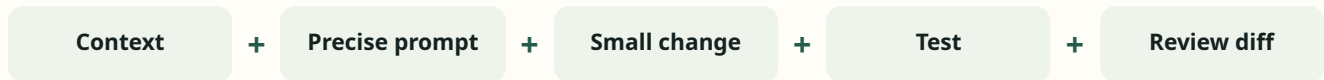
ABC Construction setup

- ✓ The approved Lovable project is downloaded or synced from its repository.
- ✓ A clean local copy exists for conversion.
- ✓ The project is committed before AI edits begin.
- ✓ The coding environment is opened at the project root.

MODEL SELECTION

The model is not the workflow

The same model can produce excellent work with clear context and weak work with a vague request. My real system is simple:



Task	Suggested starting model	How to use it
Theme conversion	DeepSeek or another strong coding model	Ask for a plan first, then convert in stages.
PHP debugging	DeepSeek	Give the exact error, file, line, and reproduction steps.
CSS changes	DeepSeek or the available front-end model	Name the component, breakpoint, and visual behavior to preserve.
WordPress functions	DeepSeek	Require native APIs, sanitization, escaping, and capability checks.
Refactoring	A second capable coding model	Compare its plan with the working implementation before editing.

My one-change loop

1. State the current behavior and the required result.
2. Name the files or component, if known.
3. Tell the AI what must remain unchanged.
4. Ask for the smallest complete change.
5. Review the diff before accepting it.
6. Run the site and test the exact behavior.
7. Commit the working result with a clear message.

Do not outsource judgment. A confident answer can still be wrong. The website running correctly is the evidence, not the model's explanation.

THE SETUP RITUAL

The first technical mistake is often opening the wrong folder

Before I ask for code, I make sure the AI can see the complete project and understands what it contains.

1. **1 Open the downloaded project.** Find the folder that contains the real application files, not a parent downloads folder.
2. **2 Copy the full folder path.** Use the path from the operating system, then open that exact directory in the coding environment.
3. **3 Confirm the project root.** Look for the package file, source folder, public assets, and configuration files expected by the project.
4. **4 Tell the AI what it is viewing.** Explain that this is the approved ABC Construction UI and the destination is a native WordPress theme.
5. **5 Ask for inspection only.** No edits yet.

Working directory: [PASTE THE PROJECT PATH]

This folder contains the approved ABC Construction UI exported from Lovable. The visual design has been accepted by the client. The destination is a native WordPress theme.

First confirm that you can see the project root. Do not modify any file yet.

Why this matters: If the AI sees only one component, it may invent missing routes, duplicate styles, or rewrite assets that already exist.

SIGNATURE PROMPT

Ask for a map before asking for construction

First inspect this entire project.

Do not modify anything yet.

Identify:

- framework and build system
- entry points
- pages and routes
- reusable components
- assets and fonts
- styling system and responsive rules
- forms and interactions
- repeated content
- areas that should become WordPress-driven
- external dependencies

Then explain how you would convert this project into a native WordPress theme. Map each important source file or component to its likely WordPress destination. List risks, unclear areas, and decisions you need from me. Finish with a staged conversion plan. Wait for approval before editing.

What a useful inspection report contains

Current project map

- Routes and pages
- Header and footer
- Card and gallery components
- Forms and scripts
- Image and font sources

WordPress mapping

- Theme templates
- Template parts
- Menus and settings
- Post types and fields
- Assets and dependencies

ABC Construction questions to resolve

Are Services separate entries or simple page sections? Do Projects need filters by project type and location? Should Team profiles have individual pages? Which homepage text changes often? Which contact details are global?

Stop if the report is vague. If it does not name actual files and components from the project, the AI may not be looking at the right place.

TWO DIFFERENT SYSTEMS

A Lovable interface is the design source, not yet the WordPress content system

LOVABLE UI PROJECT

- Routes show page views
- Components repeat interface patterns
- Data may be written directly in files
- Forms may be visual only
- A build tool may produce front-end assets

NATIVE WORDPRESS THEME

- Templates respond to requested content
- Template parts repeat interface patterns
- Content comes from WordPress
- Forms need real processing
- Assets are loaded through WordPress

The goal is not to place the old app inside WordPress. The goal is to make WordPress own the content while preserving the approved interface.

What must survive

Typography, spacing, colors, navigation behavior, responsive layouts, component relationships, image treatment, and the visual rhythm approved by the client.

What must change

Hard-coded page data, fake links, static listings, visual-only forms, route assumptions, direct asset paths, and any front-end feature that bypasses WordPress content or permissions.

ABC Construction decision: The project cards keep their Lovable design. Their titles, images, locations, years, links, and categories will come from WordPress Projects.

THE DESTINATION MAP

Give every part of the interface a WordPress home

```
abc-construction/  
├── style.css  
├── functions.php  
├── index.php  
├── header.php  
├── footer.php  
├── front-page.php  
├── page.php  
├── single.php  
├── archive.php  
├── 404.php  
├── page-templates/  
├── template-parts/  
│   ├── cards/  
│   ├── sections/  
│   └── content/  
├── assets/  
│   ├── css/  
│   ├── js/  
│   └── images/  
└── inc/  
    ├── setup.php  
    ├── post-types.php  
    ├── taxonomies.php  
    ├── meta.php  
    └── settings.php
```

Core files

style.css identifies the theme and may hold global styles.

functions.php loads setup modules and connects features.

header.php and **footer.php** carry global structure.

front-page.php controls the front page.

Content files

page.php is the general page fallback.

single.php shows a single standard post.

archive.php shows a content collection.

404.php helps visitors recover from a missing URL.

This is a practical classic-theme map. WordPress also supports block themes. Choose one architecture at the start and keep it consistent unless the project requirements clearly justify a hybrid.

FROM COMPONENTS TO RESPONSIBILITIES

Map the accepted UI before writing templates

Approved UI part	WordPress destination	Content owner
Header, logo, desktop and mobile navigation	header.php plus a registered menu	Site administrator
Footer, address, phone, social links	footer.php plus global settings	Site administrator
Homepage hero and featured projects	front-page.php plus page content and a Project query	Editor
Reusable project card	template-parts/cards/project-card.php	Theme
Projects overview	archive-project.php	Project entries
Project detail	single-project.php	Project entry and fields
Team grid	Team query in a page template or archive-team.php	Team entries
Contact form	WordPress form solution styled to match	Administrator

Convert in this order

1. Theme identity, asset loading, and basic setup.
2. Header, footer, and navigation.
3. Front page and normal pages.
4. Projects and Team content models.
5. Archives, single views, and listing queries.
6. Forms, settings, responsive behavior, and cleanup.

Do not convert every component blindly. A presentational button can stay a template part. A project card needs real WordPress data. A decorative divider needs no admin field.

SIGNATURE RESOURCE

Define the destination, the boundaries, and the evidence of success

Convert this existing ABC Construction UI project into a fully functional native WordPress theme.

Preserve the approved visual design, layout, typography, spacing, responsive behavior, components, interactions, and assets wherever practical. Do not redesign the website.

Do not embed the existing application inside WordPress. Rebuild it as proper WordPress theme files and functionality.

Create and connect the appropriate theme files, including style.css, functions.php, header.php, footer.php, front-page.php, page.php, single.php, archive.php, 404.php, template parts, page templates, CSS, JavaScript, and organized setup files under inc/.

Use WordPress APIs to enqueue assets, register menus, enable theme support, load content, and handle administrator-controlled values.

Convert static areas that should be editable into suitable WordPress content, menus, featured images, custom fields, post types, taxonomies, or settings. Do not hard-code information the administrator should reasonably change.

For ABC Construction:

- register Projects and Team as custom post types
- give Projects a Project Type and Location taxonomy
- support project location, completion year, budget range, gallery, and featured image
- create project archive and single-project templates
- preserve the existing project-card design
- make the header menu and global contact information editable

Use secure WordPress practices. Sanitize input, escape output, check capabilities, and protect custom admin actions.

Work in stages. First provide the file plan and content model. Wait for approval. Then implement one stage at a time. After every stage, list changed files and give exact tests. Do not delete unrelated working code.

PROMPT ANATOMY

A master prompt works because each paragraph has one job

1. Outcome

Name a native WordPress theme, not a vague conversion.

2. Preserve

List the accepted visual and responsive behavior that must survive.

3. Reject shortcuts

Say not to embed the app, iframe the result, or replace the design.

4. Architecture

Name the theme files, template parts, assets, and setup modules.

5. Content ownership

Define what WordPress editors need to manage.

6. Verification

Require changed files, tests, and staged approval.

The build-up sequence

1. **Start with intent:** “Convert this approved UI into a native WordPress theme.”
2. **Add constraints:** preserve design, do not redesign, do not embed.
3. **Add architecture:** name templates and the content model.
4. **Add client specifics:** Projects, Team, taxonomies, fields, global settings.
5. **Add quality rules:** native APIs, security, accessibility, responsive checks.
6. **Add control:** plan first, wait, change one stage, report tests.

One prompt does not mean one uncontrolled run. The prompt defines the full destination. The AI should still implement it in reviewable stages.

My review question

Can I explain why each generated file exists, which WordPress responsibility it owns, and how I will test it? If not, I ask the AI to explain before I accept the code.

CONTENT OWNERSHIP

If the client will reasonably change it, give it a proper control

A theme should not force the client to edit PHP for ordinary business updates. It should also not bury every decorative word inside a settings screen. I choose the control by asking who owns the value and how often it changes.

BAD: FIXED IN THE TEMPLATE

```
<h1>Welcome to ABC Construction</h1>
```

BETTER: CONTROLLED IN WORDPRESS

```
<h1><?php
echo esc_html(
    get_theme_mod(
        'abc_hero_title',
        'Build with confidence'
    )
);
?></h1>
```

Need	Use	ABC Construction example
Site-wide choice	Customizer or settings page	Header CTA label and phone number
Navigation order	WordPress menu	Primary and footer navigation
Primary image for an entry	Featured image	Project cover image
Structured entry detail	Custom field	Completion year or location
Flexible editorial section	Blocks	About-page story and callout

Decision test

- ✓ Will the client change it more than once?
- ✓ Is it content rather than visual structure?
- ✓ Should the value appear in several places?
- ✓ Does it need validation or a specific data type?
- ✓ Would changing it in code create avoidable risk?

A SMALL CONTROL CENTER

Expose the decisions that are truly global

A useful Theme Settings area is short enough to understand and complete enough to prevent repeated editing.

Branding

- Logo and site icon
- Brand name
- Approved color choices

Header

- CTA label and URL
- Menu behavior choices

Footer and contact

- Footer text
- Phone, email, address
- Social profile links

SEO and sharing

- Default social image
- Organization details

ABC Construction rule

The phone number appears in the header, footer, and Contact page. Store it once and reuse it. Project titles do not belong in global settings because each Project entry already owns its title.

Do not make everything editable. Spacing values, border radii, card markup, and icon placement belong to the design system unless the client has a real need to change them.

Safe implementation prompt

Create a Theme Settings area for ABC Construction with only the approved global fields. Use native WordPress APIs. Sanitize each saved value according to its type, check user capabilities, escape values when displayed, and provide clear labels and sensible defaults. Do not move page-specific content into global settings.

THE FIVE-PART FORMULA

Tell AI what changes, where, who controls it, what stays, and how to test



Replace the hard-coded primary navigation in header.php with a registered WordPress navigation menu. I will assign it from the WordPress admin.

Preserve the current HTML structure, class names, visual appearance, spacing, hover states, active state, keyboard focus styles, and responsive mobile-menu behavior. Do not redesign the header.

Provide a safe fallback when no menu is assigned. After implementation, list the changed files and verify the menu on desktop, keyboard navigation, mobile opening and closing, nested items, and the unassigned state.

WEAK

"Make the menu editable."

The AI must guess the location, method, design boundary, and test.

STRONG

The full prompt above leaves fewer decisions to chance and gives a clear definition of done.

Use the same formula elsewhere

- **Footer:** replace fixed contact values with the global settings, preserve layout, test missing values.
- **Homepage:** query three featured Projects, preserve card markup, define fallback order.
- **Team:** load profiles from the Team post type, preserve grid behavior, handle no image.

CHOOSE BY RESPONSIBILITY

A route from the old UI becomes a template only when its job requires one

Page need	Practical template choice	Use it when
Front page	front-page.php	The homepage has a unique composition.
Normal content page	page.php	The same content layout works across pages.
Specific About layout	page-about.php	One known page has a stable special layout.
Reusable landing format	page-templates/template-landing.php	Editors should choose the layout for several pages.
Standard post	single.php	A normal blog article is requested.

ABC Construction mapping

Home uses front-page.php because it combines a hero, selected services, featured projects, proof points, and a contact call to action. **About** can use normal page content if blocks reproduce the design. A custom page-about.php is justified only if its team and history structure cannot be managed cleanly with the general template.

Inspect the approved About route. Decide whether its layout should be handled by page.php with editable blocks, a dedicated page-about.php, or a reusable page template. Explain the trade-off. Prefer the least specialized solution that preserves the design and gives the editor appropriate control. Do not create a unique template only because the old project had a separate route.

Remember the hierarchy. WordPress chooses templates according to the requested content. A specialized template should solve a real presentation need, not duplicate an existing file.

MODEL REPEATABLE CONTENT

A post type is only the first link in a complete content chain

Post type



Fields



Taxonomies



Archive



Single view

ABC Construction: Projects

Projects repeat the same kind of information and need their own archive and detail URL. Each Project gets a title, summary, main content, featured image, gallery, location, completion year, budget range, Project Type, and Location terms.

ABC Construction: Team

Team members also repeat a structure: name, role, portrait, short biography, credentials, and profile order. Decide whether each person needs a public single page. If not, the post type can support the team grid without exposing individual URLs.

Create Projects and Team as native WordPress custom post types for this theme prototype. Before coding, show the proposed labels, slugs, supported editor features, public behavior, archive choice, REST visibility, menu icons, capabilities, and template files.

Keep registration code separate under inc/. Do not create fields, taxonomies, or templates until I approve the model. Explain whether this content should later move into a small site plugin so it survives a theme change.

Architecture note: Business content often belongs in a plugin because a theme can be replaced. For a client delivery, decide this deliberately instead of hiding the decision inside generated code.

LET THE INTERFACE REVEAL THE DATA

The approved UI tells you what WordPress must store

URLs

A Project archive can live at:

```
example.com/projects/
```

A single Project can live at:

```
example.com/projects/project-alpha/
```

Taxonomies

Project Type: Commercial, Residential, Renovation.

Location: the reusable geographic grouping needed by filters and archives.

Fields for one Project

UI element	Stored as	Reason
Project name	Post title	Native and searchable
Main image	Featured image	Works in cards and sharing
Location	Taxonomy or field	Taxonomy if visitors filter by it
Completion year	Number field	Structured and sortable
Budget range	Controlled field	Consistent display without false precision
Gallery	Gallery field or blocks	Supports the single layout

Principle: The UI determines what information the content model needs. Do not invent fields simply because WordPress can store them.

After changing slugs

Save permalinks, test the archive and a single entry, confirm pagination, and check old URLs before launch. A slug choice affects both visitors and future redirects.

DESIGN BOTH SIDES OF THE CONTENT MODEL

The archive helps people choose. The single page helps them understand.

Project archive

- Clear archive title and introduction
- Project cards from real entries
- Project Type and Location filters
- Useful result count
- Pagination
- Meaningful empty state

Single Project

- Hero image and title
- Location, year, and budget range
- Project story and challenge
- Gallery
- Related Projects
- Contact call to action

Build `archive-project.php` and `single-project.php` from the approved ABC Construction components.

Archive:

- query published Projects
- preserve the approved project-card markup
- paginate the results
- support Project Type and Location filters
- keep filters in the URL
- show a helpful no-results state

Single:

- show the featured image, content, location, completion year, budget range, gallery, and terms
- show related Projects by Project Type
- preserve the approved responsive layout
- hide optional sections cleanly when their data is missing

Use native WordPress template functions. Escape output. Reset custom queries. List all test cases.

Design for missing data. A Project without a gallery should not show an empty frame. A missing budget should not produce a blank label. Optional content disappears cleanly.

QUERIES ARE PRODUCT BEHAVIOR

Describe the visitor experience before asking for a query

A request such as “show Projects” leaves too much open. I specify which Projects appear, in what order, how many per page, how filters behave, and what happens when there are no results.

ABC Construction listing contract

Decision	Chosen behavior
Content	Published Projects only
Order	Newest completion year, then title
Page size	Nine cards
Filters	Project Type and Location
State	Selected filters remain visible in the URL
No results	Clear filters button and contact link
Card data	Image, title, type, location, year

Create the Projects listing behavior described above. Use the existing project-card template part. Keep WordPress pagination working with filters. Validate incoming filter values against allowed taxonomy terms. Avoid duplicate queries. Preserve accessible labels, keyboard operation, and focus visibility. Explain how to test page 2, combined filters, invalid filter values, no results, and a Project with missing optional fields.

What I inspect in the result

- ✓ The listing uses the main query when practical.
- ✓ Pagination keeps the selected filters.
- ✓ Cards link to the correct single Project.
- ✓ No raw database values are printed without escaping.
- ✓ The layout still matches the Lovable design at each breakpoint.

BEFORE LOCAL TESTING

A converted theme should already have a clear content system

Theme foundation

- ✓ Theme identity is complete.
- ✓ Assets load through WordPress.
- ✓ No absolute local paths remain.
- ✓ Header, footer, and 404 work.
- ✓ Menus have a safe fallback.
- ✓ Visual tokens match the approved UI.

Pages

- ✓ Front page uses real content.
- ✓ Normal pages have a useful fallback.
- ✓ Custom templates solve real needs.
- ✓ Links point to WordPress URLs.

Content model

- ✓ Projects and Team have clear responsibilities.
- ✓ Fields match visible UI needs.
- ✓ Taxonomies support grouping or filtering.
- ✓ Slugs are intentional.
- ✓ Archives and singles handle missing data.

Admin experience

- ✓ Global values are stored once.
- ✓ Labels explain what editors should enter.
- ✓ Input is sanitized and output escaped.
- ✓ Changes do not require PHP editing.

Fast content test

Create two Projects with different types and locations. Leave one optional field empty. Change the phone number once. Reorder the menu. If the site updates correctly without code edits and the design stays stable, the architecture is doing its job.

This is not the final launch test. Volume 3 covers local activation, debugging, responsive review, accessibility, search engine optimization, performance, migration, and client handover.

DO NOT LET AI DO THIS

Shortcuts create expensive maintenance

Do not say

"Convert this to WordPress."

Say instead

Name the native theme architecture, visual boundary, editable areas, and tests.

Do not say

"Make everything editable."

Say instead

Identify the content the administrator will reasonably control. Keep design structure in the theme.

Do not accept

An iframe or a bundled app that bypasses normal WordPress content.

Require instead

Templates, parts, queries, menus, media, settings, and content models that WordPress owns.

Do not accept

A new framework or plugin for every small feature.

Require instead

Native APIs first, with every dependency justified.

Watch for silent breakage

- CSS selectors changed while templates still use the old class names.
- Asset paths work locally but fail inside a theme directory.
- Mobile-menu JavaScript targets an element that no longer exists.
- Queries forget pagination or fail to reset post data.
- Optional values produce empty headings or broken links.
- Custom content disappears when the theme changes.
- Admin fields save without capability checks or sanitization.

My response to a large surprise diff: I do not ask the AI to explain it away. I revert or isolate the change, reduce the task, and test again.

COPY, ADAPT, AND TEST

Five compact prompts for the conversion desk

1. Preserve a component

Move [COMPONENT] into [TEMPLATE PART]. Preserve its HTML relationships, class names, responsive behavior, and visual states. Replace only the hard-coded data with the named WordPress values. List the changed files and tests.

2. Make a field intentional

The [UI ELEMENT] needs [VALUE]. Propose the simplest WordPress owner for it: post title, content, featured image, taxonomy, custom field, block, menu, or global setting. Explain the choice before coding.

3. Debug with evidence

Expected: [BEHAVIOR]. Actual: [BEHAVIOR]. Reproduction: [STEPS]. Error: [EXACT MESSAGE]. Inspect the likely files, state the root-cause hypothesis, and make the smallest fix. Do not redesign or refactor unrelated code.

4. Review security

Review only the new admin and output code from this stage. Check capability controls, nonces where appropriate, input validation, sanitization, output escaping, and safe URLs. Report each issue with file and line before changing it.

5. Verify the conversion

Compare the current WordPress page with the approved UI. List differences in structure, typography, spacing, responsive behavior, states, and content ownership. Separate visual differences from functional defects. Do not edit until I approve the list.

Prompt pattern: Context + location + action + requirement + constraint + test.

IN VOLUME 3

The site leaves the code editor and proves itself

The next volume follows ABC Construction through local activation, debugging, quality checks, migration, launch, and client delivery.

You will learn

- ✓ Install and activate the theme in LocalWP
- ✓ Debug PHP, CSS, and JavaScript problems
- ✓ Test desktop, tablet, mobile, forms, and content
- ✓ Configure SEO and performance basics
- ✓ Migrate to production hosting
- ✓ Run a client handover

About the author

Yukesh Chaudhary writes practical guides about search, websites, and AI-assisted digital work. His approach is built around clear requirements, controlled prompts, and testing real outcomes.

Website: <https://www.yukesh.com.np>

Official references

WordPress Theme Handbook:

Template Hierarchy

<https://developer.wordpress.org/themes/templates/template-hierarchy/>

WordPress Theme Handbook:

Custom Post Type Template Files

<https://developer.wordpress.org/themes/classic-themes/templates/custom-post-type-template-files/>

WordPress Plugin Handbook:

Working with Custom Post Types

<https://developer.wordpress.org/plugins/post-types/working-with-custom-post-types/>

WordPress Theme Handbook:

Security

<https://developer.wordpress.org/themes/advanced-topics/security/>

WordPress Common APIs:

Nonces

<https://developer.wordpress.org/apis/security/nonces/>

Volume 2 outcome: You now have a repeatable way to turn an approved UI into a native WordPress content system without surrendering control to the model.