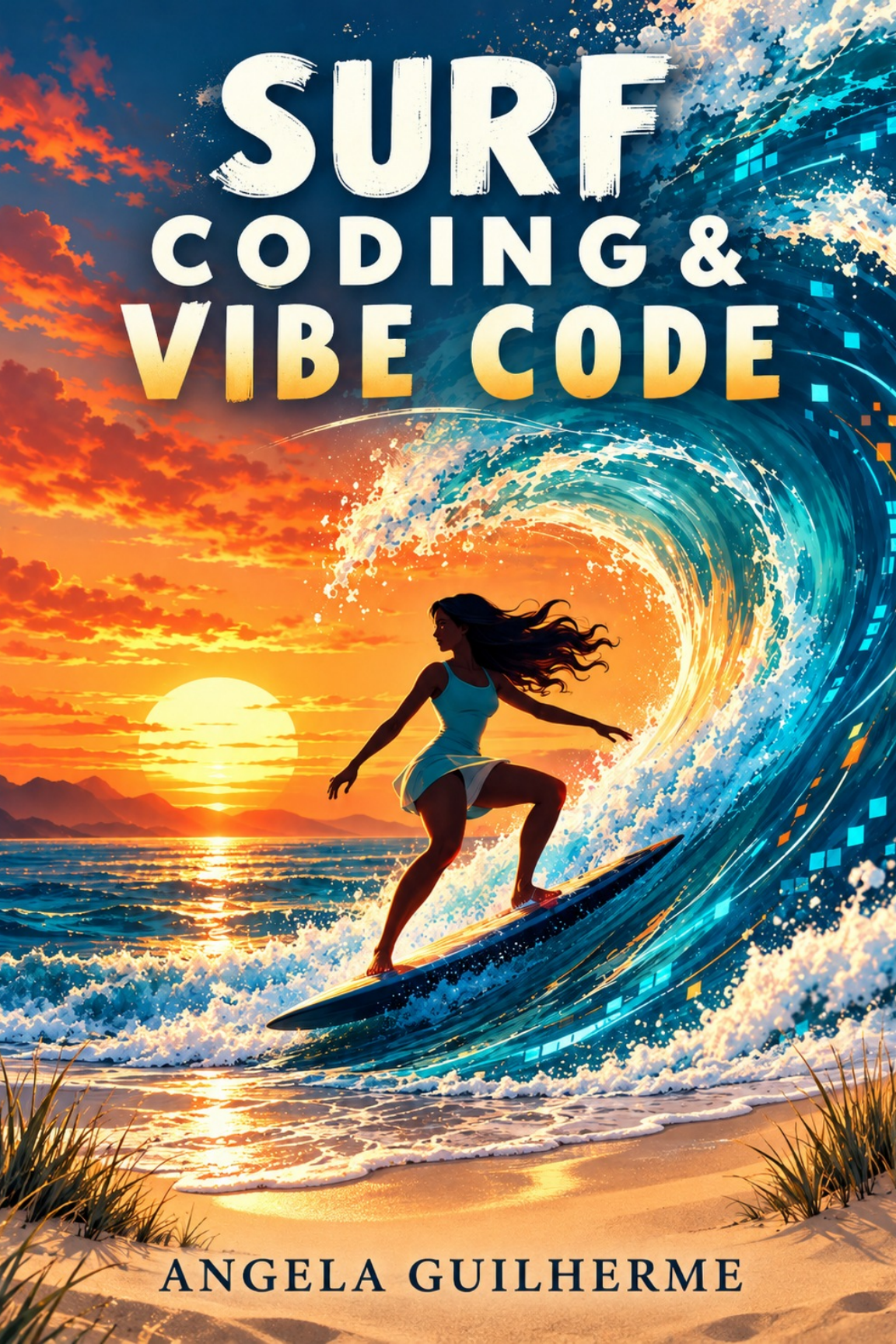

SURF
CODING &
VIBE CODE
ANGELA GUILHERME

SURF CODING AND VIBE-CODE

Ride the Wave of AI-Assisted Development Without Drowning in the Deep End

A Practical Guide for Creators, Entrepreneurs, and Curious Builders Who Want to Build Smarter

ANGELA GUILHERME

Revised Edition
2026

Surf Coding and Vibe-Code

Copyright © 2026 Angela Guilherme. All rights reserved. No part of this publication may be reproduced, distributed, or transmitted in any form without prior written permission.

First edition March 2026. Revised edition, October 2026.

Product names, prices, and regulations described in this book change quickly. They were checked in October 2026; readers should confirm current details with the original sources listed at the end of the book. Nothing here is legal advice.

CONTENTS

Foreword: Why I Wrote This Guide v

1. From Fear to Fluency 1

1.1 The Wall I Almost Didn't Climb 1

1.2 Why Coding Feels Harder Than It Is 2

2. What Vibe-Code Is and What It Is Not 3

2.1 The Genuine Revolution 3

2.2 Two Families of Tools 4

2.3 Agents and the Model Context Protocol 4

2.4 The Honest Limitations 5

3. The Auth Problem 7

3.1 The Scenario 7

3.2 What Auth Actually Means 8

3.3 Eight Terms That Will Unlock Your Workflow 9

4. Twelve Scenarios Where Basic Knowledge Changes Everything . . 10

5. What to Actually Learn 14

5.1 Tier 1: The Absolute Essentials (One to Two Weeks) 14

5.2 Tier 2: Meaningful Competency (Two to Four Weeks Additional) . . 15

5.3 Tier 3: Genuine Leverage (Ongoing) 15

6. How to Learn 17

6.1 For Absolute Beginners 17
6.2 For Learning by Doing 17
6.3 For Specific Topics 18

7. Surf Coding 19
7.1 The Compounding Effect 20
7.2 You Are Already Closer Than You Think 20

8. Building Your First Hybrid Workflow 21
8.1 Start With a Clear Feature Description 21
8.2 Read What the AI Generates 21
8.3 Test With Intention 22
8.4 Debug With a Framework 22
8.5 Save Your Progress With Version Control 22
8.6 Review Security Before You Launch 22
8.7 Document as You Go 23

Closing: The Wave Is Here 24

Appendix A: Glossary 25

Appendix B: The OWASP Top 10:2025 27

Sources and Further Reading 29

FOREWORD

Why I Wrote This Guide

I want to be clear from the very first page: if you have built something using a vibe-coding platform, Cursor, GitHub Copilot, Claude Code, or any other AI-assisted development tool, you have absolutely nothing to apologize for. You are not cheating. You are not taking a shortcut. You are using the best tools available to bring your vision to life, and that is exactly what skilled builders have always done.

I say this as a full-stack web developer: someone who has written thousands of lines of HTML, CSS, JavaScript, Python, and more; who has spent late nights debugging authentication flows, wrestling with database schemas, and staring at error messages that seemed written in a language more cryptic than any programming language. I have been there, and I still use AI tools in my workflow today.

But I also want to be honest about something. There is a meaningful difference between using a power tool and understanding what the power tool is doing. Both people can build something. But one of them, when something goes wrong — and in web development, something always goes wrong — knows exactly what to do next. The other gets stuck.

> *"The goal of this guide is not to intimidate you into learning code. It is to show you how a little bit of knowledge goes an extraordinarily long way."*

This guide is for the entrepreneur building a first software-as-a-service product; the designer with brilliant ideas whose AI tool produces an error they do not understand; the content creator who wants to launch a

membership platform with proper user logins; and the freelancer who has been delivering projects with AI tools but wants to offer clients more.

A Note on the Revised Edition

The term *vibe coding* was introduced by the computer scientist Andrej Karpathy in early 2025 to describe building software by describing what you want and accepting what the AI produces, without closely reading the code (Karpathy, 2025). Within months it had become a mainstream phrase, and the dictionary publisher Collins named it its Word of the Year for 2025. The practice has matured quickly since then. The tools have moved from suggesting single lines to acting as *agents* that plan, edit many files, run commands, and fix their own errors. Security has become a central concern, and the industry's main reference list of web vulnerabilities, the OWASP Top 10, was revised in late 2025 for the first time in four years.

This revised edition updates the guide accordingly. It adds a map of the current tool landscape, a fuller treatment of security and of AI agents, two new scenarios, a longer hybrid workflow with a pre-launch checklist, and a glossary. The central argument has not changed, and I think it has only become more important: the more capable the tools become, the more valuable it is to understand what they are doing on your behalf.

My own journey began from a place of genuine fear. I want to tell you about that before we go any further, because I think it matters.

CHAPTER ONE

From Fear to Fluency

My Story

1.1 The Wall I Almost Didn't Climb

Before I became a developer, the idea of programming seemed completely out of reach. It felt like a secret language that brilliant people were born knowing, something reserved for the mathematically gifted or the obsessively technical. When I looked at even a few lines of code, my brain would immediately signal retreat: too complex, too foreign, not for you.

This was not a small fear. It was a conviction. I genuinely believed I would never be able to understand how software was built. The syntax alone looked alienating — all those semicolons, brackets, and functions nested inside functions. I had tried to read tutorials and they quickly felt overwhelming. I closed a lot of browser tabs in defeat.

> *"The first lie programming told me was that I had to understand everything at once. I didn't. Nobody does."*

What changed was a decision to stop trying to consume all of it at once and to start with just one small, useful thing. I learned what a variable was. Then what a function was. Then how a webpage was structured in HTML. I did not try to learn everything simultaneously. I focused on understanding enough to be dangerous, and then a little more than that.

That shift in approach changed everything. Within a few months I was building real things. Not perfect things, and not enterprise-grade systems, but real, working projects that solved real problems. And that momentum compounded.

Today I work across the full stack: frontend interfaces, backend APIs, databases, authentication systems, and deployment pipelines. I did not arrive here by being naturally gifted. I arrived here by refusing to be intimidated by the parts I did not yet understand.

1.2 Why Coding Feels Harder Than It Is

There is a structural reason that coding feels overwhelming at first: the internet is full of expert-to-expert content. Most tutorials assume you already know quite a bit. Forum answers are written by and for people who are comfortable with technical jargon. Documentation is often thorough but not welcoming.

When you arrive in that environment as a beginner, it is easy to mistake ‘not yet knowing the vocabulary’ for ‘not being capable.’ These are completely different things. The vocabulary of programming is learnable. The logic of programming, once you grasp the underlying patterns, is actually very human.

The other intimidating factor is scope. There are hundreds of programming languages and dozens of frameworks. The beginner who tries to decide where to start by surveying all of it will feel lost before writing a single line.

There is good news here that did not exist when I started. AI assistants are patient explainers. You can paste in a line you do not understand and ask for a plain-English explanation as many times as you need, without feeling judged. Used this way, the same tools that generate code can also be the most approachable teacher a beginner has ever had.

> ***Key point.*** You do not need to know all of programming. You need to know enough to understand what is happening in your project, to communicate with AI tools effectively, and to solve the specific class of problems you are likely to encounter. That is a much smaller and more achievable target.

CHAPTER TWO

What Vibe-Code Is and What It Is Not

2.1 The Genuine Revolution

Let us acknowledge what AI-assisted development has achieved, because it is remarkable. Tools in the vibe-coding category — platforms and environments where you describe what you want in plain language and receive working code in return — have democratized the ability to build software to an unprecedented degree.

A small business owner can describe a customer booking form and receive a functional, styled component in seconds. A designer can turn a mockup into a working prototype without writing a line of code. A founder with a validated idea can spin up a working minimum viable product in a weekend rather than spending three months learning the fundamentals first.

This is not a trick, and it is not pretend development. The code these tools generate is real code. Anyone who dismisses vibe coding as 'cheating' or 'not real development' is either gatekeeping or missing the point.

> *"The most important thing a tool can do is lower the barrier between a person's vision and their ability to execute it. Vibe-code does exactly that."*

2.2 Two Families of Tools

It helps to know that ‘vibe-coding tools’ are not one thing. In 2026 they fall broadly into two families, and the family you choose determines how much of the underlying code you will see.

The first family, *prompt-to-app builders*, includes Lovable, Bolt, Replit, Base44, and v0. You describe an application in a chat window and receive a running, hosted result, often with a database, a login system, and a deployment button included. These tools are the fastest route from idea to something you can show people, and they hide most of the code by default.

The second family, *AI-native editors and coding agents*, includes Cursor, Windsurf, GitHub Copilot, Claude Code, OpenAI Codex, and Google Antigravity. These work inside a real codebase that you own. They read many files, propose changes, run commands and tests, and can work for long stretches with limited supervision. They assume more comfort with code, and they reward it.

Many teams combine the two: they validate an idea in a prompt-to-app builder in an afternoon, then move the project into an editor or agent for the work that follows. Many tools also sync to a code repository such as GitHub, so that a developer can take over later. Whichever family you use, the argument of this book applies. Tool rankings change month to month, so I have deliberately kept this section short; the principle is more durable than the product list.

2.3 Agents and the Model Context Protocol

The most significant change since this book was first written is the rise of *agents*. An agent is an AI system that does not merely answer a question but takes a series of actions toward a goal: it reads files, edits them, runs the application, reads the error, and tries again. Agents can be astonishingly productive. They also make more decisions on your behalf, which means that more of your project is shaped by choices you did not see.

Agents increasingly connect to outside services — databases, payment providers, calendars, code repositories — through an open standard

called the Model Context Protocol (MCP). Each connection is a form of access, and access is a security question. A tool that can read your database can also, if it is manipulated or simply mistaken, change or delete it. We return to this in Chapter 4.

2.4 The Honest Limitations

At the same time, honesty requires us to look at where AI-assisted development runs into real friction — not to diminish it, but to understand where complementary knowledge makes the difference between a project that ships and one that stalls.

The most common pattern I have observed is what I call the ‘opaque wall.’ Everything is going smoothly, the tool is generating excellent code, and then you hit an error or a limitation you do not understand. The tool says it cannot proceed. Or it offers a fix that introduces a new problem.

In these moments, the person with no programming knowledge is wholly dependent on the tool’s ability to explain and repair itself. Sometimes that works. Often it leads to a loop, and projects stall. The person with even a basic understanding of what is happening under the hood can step back, read the error message with comprehension, identify the category of the problem, and either direct the AI more precisely or apply a quick manual fix.

A second limitation is less visible and more serious: AI-generated code is not automatically secure. Code that ‘works’ in a demonstration can still leave a door open. Prompt-to-app builders increasingly offer built-in security scanning, which is welcome, but a scanner cannot know what your application is *supposed* to allow. Only someone who understands the rules of the application can judge that. Table 2.1 summarizes the difference a little knowledge makes.

Table 2.1. Common situations with and without foundational knowledge

Situation	Vibe-code only	Vibe-code with the basics
An authentication error stops the project	Pastes the error into the AI chat, receives a new error, and repeats for hours.	Recognizes a missing OAuth callback, directs the AI precisely, and resolves it in minutes.
A privacy law requires a consent record	The AI says it cannot add a consent flow without authentication, and the project halts.	Understands why authentication is required and scaffolds the solution with AI assistance.
Performance degrades	Cannot identify the cause; tries random suggestions with no clear direction.	Reads the network tab, finds the slow query, and directs the AI to a specific fix.
An API integration fails	Unclear whether the fault lies in the key, the endpoint, or the data format.	Reads the status code (401 or 422) and narrows the fix immediately.
The AI adds a package you have not heard of	Accepts it without question and ships it.	Checks that the package is genuine, maintained, and necessary before installing it.
A secret key appears in the code	Does not notice; the key is published with the project.	Recognizes the key, moves it to an environment variable, and replaces the exposed key.

CHAPTER THREE

The Auth Problem

A Case Study in Why Basics Matter

3.1 The Scenario

Here is a real situation that many builders encounter. You are working on a web application — perhaps a community platform, a software-as-a-service tool, or a client portal. Everything is coming together nicely. Then you decide to add functionality that requires storing sensitive or personal information about your users, and you begin to think about compliance with the General Data Protection Regulation (GDPR).

You ask your platform to add a consent checkbox and record which users accepted your terms. It replies that it cannot do so, because you do not have authentication set up. You have no ‘Auth’ integration.

If you do not know what authentication means in a technical context, this response is completely opaque. You might think: I only want a checkbox; why does this require something called ‘Auth’? You are stuck, and the only way forward is to ask more questions, which generate more answers, which generate more questions.

3.2 What Auth Actually Means

Authentication, or ‘Auth’, is the mechanism by which your application knows who a user is. When someone logs in, the application checks their credentials, confirms their identity, and creates a session or token that travels with their subsequent requests. Modern products increasingly support passkeys, which replace passwords with cryptographic keys stored on the user’s device, as well as multi-factor authentication and ‘sign in with’ options.

Consent records are meaningful precisely because they are tied to an identified individual. Under the GDPR, an organization that relies on consent must be able to demonstrate that the person consented (Regulation (EU) 2016/679, Article 7). A record that says ‘someone checked a box at 3:42 pm’ demonstrates very little. A record that says ‘user 7842, whose email address is verified, accepted version 2 of the Privacy Policy at 3:42 pm on March 15, 2026’ is far closer to what the regulation expects.

> *“Auth is not a technicality. It is the foundation of any feature that has legal, financial, or personal significance attached to it.”*

Authentication has a companion concept that is easy to confuse with it. *Authentication* answers ‘who are you?’ *Authorization* answers ‘what are you allowed to do?’ A user may be perfectly authenticated and still have no right to see another customer’s invoice. As we will see in Chapter 4, failures of authorization are among the most common and most damaging weaknesses in web applications.

Once you understand this — even without ever having written an authentication flow yourself — you can return to your platform with a different kind of request: ‘First, help me integrate an authentication provider such as Clerk, Auth0, or Supabase Auth. Once users can log in and I can reference their user ID, we will add the consent table and connect it to their profile.’ That is a solvable, well-scoped instruction.

3.3 Eight Terms That Will Unlock Your Workflow

Authentication is one example. Below are eight concepts that, once understood even loosely, will markedly improve your ability to work with AI coding tools.

API. An Application Programming Interface is how two pieces of software talk to each other. Knowing that API responses carry status codes — 200 means success, 401 means not authenticated, 403 means not permitted, and 500 means the server failed — helps you diagnose an enormous category of problems.

Database. A database is where your application's data lives persistently. Knowing the difference between a relational database (tables of rows and columns) and a document database (flexible JSON records) helps you understand why some AI suggestions work and others do not.

Environment variables. These are configuration values — API keys, database addresses, secret tokens — that your application needs in order to run but that should never be written directly into source code. Not knowing this pattern leads to broken deployments or to accidentally publishing sensitive keys.

Frontend and backend. The frontend is everything the user sees in the browser. The backend is the server-side code handling data and logic. This split explains why certain things, such as charging a card, must happen on the server, while others can safely happen in the browser.

Deployment. Deployment is the process of making your application available on the internet. Understanding it prevents the classic confusion when AI-generated code 'works locally but breaks in production.'

Authorization. The rules that decide what each authenticated user may see or do. Every feature that shows user-specific data depends on it.

Version control. A system, usually Git, that records every change to your code so you can compare versions and undo mistakes. When an AI agent edits dozens of files, version control is your safety net.

Agent and MCP. An agent is an AI that takes actions rather than only giving answers; the Model Context Protocol is a common way of connecting agents to outside tools and data. Both increase what the AI can do, and therefore what can go wrong.

CHAPTER FOUR

Twelve Scenarios Where Basic Knowledge Changes Everything

The following scenarios are drawn from situations that builders encounter regularly. For each, we consider what happens without foundational knowledge and what becomes possible with even a surface-level understanding. The first ten appeared in the original edition; the last two address risks that have grown in importance with the arrival of AI agents.

01. Your App Breaks After Deployment

Your project works perfectly in preview. You deploy it, and it breaks. The error mentions environment variables that are not defined.

Without basic knowledge, you have no idea what an environment variable is or why it would work in one context and not another. With it, you open your hosting provider's settings, add the missing variables, and redeploy: ten minutes instead of several hours.

02. Payments Are Not Processing

You have integrated Stripe. In preview it worked; in production, payments silently fail and users see no error.

With basic knowledge, you know that Stripe has separate test keys and live keys, and that webhooks are how Stripe tells your application a

payment succeeded. You check the dashboard, see a 401 error, and correct the key or the webhook signature verification within an hour.

03. Users Are Seeing Each Other's Data

A user reports that changing a number in the web address shows them another user's information. Without basic knowledge, you may not even recognize this as a critical security vulnerability.

With it, you identify a missing authorization check. Broken access control has ranked first in the OWASP Top 10 in both the 2021 and 2025 editions (OWASP, 2025). You direct your AI tool to enforce access rules on every data request, for example with row-level security in the database. This is a legal and ethical necessity, not merely a bug fix.

04. Your App Is Slow

Users complain that pages take a long time to load. Without basic knowledge, you ask the AI to 'make it faster' and receive generic suggestions.

With it, you open the browser's developer tools, examine the Network tab, and see one database query returning 50,000 records when it should return 20. You instruct the AI: 'This query needs pagination and an index on the `user_id` column.' A precise problem receives a precise fix.

05. A Third-Party Integration Stops Working

An API your application depends on suddenly returns errors. With basic knowledge, you read the HTTP status code. A 429 means a rate limit, so add throttling. A 401 means your key has expired, so regenerate it. A 503 means their servers are down, so wait. You identify the category instantly.

06. Complying With a Legal Requirement

A lawyer tells you your application must comply with the CCPA, the GDPR, or HIPAA. With basic knowledge, you can translate legal requirements into technical specifications — consent records, data deletion, audit logs, encryption — and direct your AI tool to implement them precisely.

07. AI Generates Conflicting Code

The code generated for a new feature conflicts with something already in your project. With basic knowledge, you understand the nature of the conflict and give a precise instruction: ‘Rename this function and update all references, without touching the existing authentication module.’ With version control, you can also undo the change entirely if it goes wrong.

08. Handing a Project to a Developer

You have built a prototype and want to bring in a developer. With basic knowledge, you can describe your architecture in developer terms, evaluate estimates, and ask intelligent questions. You become a better client.

09. Controlling Infrastructure Costs

Your hosting bill is higher than expected. With basic knowledge, you understand the difference between serverless functions and persistent servers, find a webhook being called thousands of times unnecessarily, add a deduplication check, and cut the bill significantly. The same applies to AI usage: knowing that model calls are billed by volume lets you spot a loop that is quietly spending money.

10. Migrating Between Platforms

With basic knowledge, you understand your application’s components — the frontend framework, the database, the authentication provider, and the hosting layer. You know which are portable and which require reconfiguration, and you can plan and carry out the migration yourself.

11. A Dependency You Did Not Choose

To add a feature, your AI tool installs a package you have never heard of. Without basic knowledge, you assume it is fine. But AI tools can occasionally suggest packages that do not exist, or that are poorly maintained, and attackers have exploited the public package registries that developers rely on. The 2025 OWASP Top 10 reflects this by adding Software Supply Chain Failures as a new category (OWASP, 2025).

With basic knowledge, you know that a project is made of many outside components. You check the package's name, its download history, and its maintainers before accepting it; you ask the AI whether a built-in alternative exists; and you keep the list of dependencies small.

12. Your AI Agent Has Too Much Access

You connect an agent to your database, your email, or your payment provider so that it can 'just handle things.' Without basic knowledge, you grant full access because it is the quickest option. If the agent misunderstands an instruction, or is manipulated by hostile text hidden in a web page or document it reads — an attack known as *prompt injection*, which the OWASP project ranks as the leading risk for AI applications — it can act with every permission you gave it.

With basic knowledge, you apply a principle as old as computing: grant the least access that does the job. You give the agent a read-only database role for a reporting task, keep production credentials out of its reach, work on a copy of your data when experimenting, and require human approval before anything irreversible, such as deleting records or sending money.

CHAPTER FIVE

What to Actually Learn

The Minimum Effective Dose

One of the most common mistakes people make when deciding to learn programming is trying to learn too much at once. They enroll in a comprehensive course, encounter a scope that stretches to the horizon, and quit before they have learned anything useful. This guide takes a different position.

You do not need to become a developer. You need to become a sufficiently informed collaborator with AI development tools. Those are very different goals, and the second requires dramatically less investment.

> *"Surf Coding is not about mastering the ocean. It is about learning to read the waves, staying upright, and knowing when to paddle and when to let the water do the work."*

5.1 Tier 1: The Absolute Essentials (One to Two Weeks)

These are the concepts that will immediately improve your ability to use any AI coding tool and to get unstuck faster.

- **HTML structure**: what tags are, how nesting works, and what the Document Object Model (DOM) is.
- **CSS basics**: how styles are applied to elements, what a class and an ID are, and the box model.
- **Variables and functions**: the most fundamental concepts in any programming language.

- **Reading an error message**: stack traces, line numbers, and the difference between syntax and runtime errors.
- **HTTP basics**: requests and responses, and status codes (200, 401, 403, 404, 429, 500).
- **Reading a change**: understanding what a ‘diff’ is, so you can see what an AI tool has altered.

5.2 Tier 2: Meaningful Competency (Two to Four Weeks Additional)

- **JavaScript fundamentals**: arrays, objects, loops, conditionals, and asynchronous code (`async/await`).
- **How databases work**: tables, rows, columns, relationships, and simple `SELECT` queries.
- **Authentication and authorization**: sessions, tokens, JWTs, OAuth, passkeys, and access rules.
- **Environment variables and secrets**: where they live and how to keep them out of your code.
- **Version control**: what Git is, what a commit is, and why branching matters.
- **Writing a specification**: describing a feature clearly enough that an AI tool can build it correctly the first time.

5.3 Tier 3: Genuine Leverage (Ongoing)

- **A backend framework**: Node.js with Express, Python with FastAPI, or similar.
- **A specific database**: PostgreSQL, SQLite, or a managed service such as Supabase.
- **Testing**: writing or requesting automated tests so that changes can be verified quickly.

- **Deployment basics**: continuous integration and delivery, containers, and content delivery networks.
- **Security fundamentals**: the OWASP Top 10, including access control, injection, misconfiguration, and supply chain risks (see Appendix B), and the separate OWASP list for AI applications.
- **Working safely with agents**: permissions, approval steps, and sandboxes.

Recommendation. Start with Tier 1 only. Build something with it. Notice how much more you understand about what your AI tool is generating. Then let genuine curiosity and specific problems drive you into Tier 2.

CHAPTER SIX

How to Learn

Resources and Approaches That Work

The resources for learning to code are better today than they have ever been. The challenge is no longer access to information. It is choosing the right starting point and maintaining the discipline to go narrow and deep rather than wide and shallow.

6.1 For Absolute Beginners

- **freeCodeCamp** (freecodecamp.org): free, project-based, and well structured.
- **The Odin Project** (theodinproject.com): a free, open-source full-stack curriculum.
- **MDN Web Docs** (developer.mozilla.org): the definitive reference for HTML, CSS, and JavaScript.
- **CS50** (cs50.harvard.edu): Harvard's introduction to computer science, widely regarded as one of the best available, and free to follow online.

6.2 For Learning by Doing

- Build something immediately. Do not complete a course before touching a project.

- Use AI tools to explain code to you. Paste a confusing piece of code and ask: 'Explain what each line does, in plain English.'
- Ask the AI to quiz you. After it generates a feature, ask it to ask *you* three questions about how the code works.
- Read error messages carefully before asking for fixes. Train yourself to understand the error before seeking the solution.

6.3 For Specific Topics

- **Authentication:** the documentation for Clerk, Auth0, and Supabase Auth is excellent.
- **Databases:** Supabase publishes tutorials that are readable for beginners.
- **APIs and payments:** the Stripe and Resend developer documentation is among the best written in the industry.
- **Security:** the OWASP Top 10 is the standard awareness reference for web application security, and the OWASP GenAI Security Project covers risks specific to AI applications.
- **Your own tools:** the official documentation for Cursor, Claude Code, Copilot, Lovable, Replit, or whichever tool you use. Features change faster than any book can follow.

CHAPTER SEVEN

Surf Coding

A Philosophy for the AI Age

The title of this guide uses a metaphor I find genuinely useful: surfing. Not because programming involves a beach, but because of what surfing requires and represents.

A surfer does not control the ocean. They cannot decide when the waves come, how tall they are, or in which direction they move. What the surfer does is develop a deep enough understanding of how water moves to take the right position, read what is coming, and respond with skill rather than panic.

AI development tools are the wave. They are powerful, they are available, and they will continue to grow. You can either be thrown by them or ride them. The difference is not whether you are willing to use them; it is whether you understand enough about what is beneath you to stay upright.

> *"Surf Coding means being comfortable enough with the fundamentals that the wave amplifies your ability rather than overwhelming it."*

Basic coding knowledge is not about slowing down. It is about the kind of speed that does not create debt you must repay later.

7.1 The Compounding Effect

When you understand what authentication is, you can ask your AI tool to implement a specific authentication pattern, with a specific provider, meeting specific security requirements. The AI executes that in a fraction of the time it would take to write by hand. Without your knowledge directing it, however, the AI might implement something that technically works but does not meet your actual requirements.

Knowledge gives AI tools context. Context makes AI tools precise. Precision makes everything faster, better, and less prone to the rework that kills momentum. As tools become more autonomous, this matters more, not less: an agent that can change a hundred files in a minute needs a person who can say clearly what the result should be and who can tell whether it is.

7.2 You Are Already Closer Than You Think

If you have used a vibe-coding platform and shipped something, you already understand more about programming than you may realize. You understand that computers execute instructions. You have seen variable names, function calls, and return values, even if you did not know those were the terms for them.

The conceptual distance between where you are and where you need to be is much smaller than it appears from the outside. The vocabulary is learnable. The patterns are recognizable once you have met them a few times. The logic, once you internalize it, will feel natural, because it is.

CHAPTER EIGHT

Building Your First Hybrid Workflow

What does the surf coding approach look like in practice? How do you combine basic knowledge with AI tools to build more effectively? Here is a workflow that has served me well, extended in this edition with two further steps.

8.1 Start With a Clear Feature Description

Before touching your AI tool, write out what you want to build in plain English. Be specific. Instead of ‘user login,’ write: ‘I want users to create an account with an email address and password, verify their email, log in, and stay logged in across browser sessions using a persistent token. I want to use Clerk as the authentication provider.’ Add what must *not* happen: ‘A user must never be able to see another user’s data.’

8.2 Read What the AI Generates

Do not accept AI-generated code blindly. Read it. You do not need to understand every line, but you should be able to identify the major sections. If you cannot, ask the AI to annotate the code with comments explaining each section. This practice improves your understanding and catches errors.

8.3 Test With Intention

When you test a feature, test it like someone trying to break it, not someone hoping it works. Create an account, then try to access someone else's account. Test the failure cases, not just the happy path. Ask the AI to write automated tests for the rules that matter most, and run them after every significant change.

8.4 Debug With a Framework

When something does not work, approach debugging with a framework: isolate, identify, fix. Isolate means finding the smallest situation in which the problem occurs. Identify means reading the actual error — the message, the line number, the status code. Fix means giving the AI a precise, targeted instruction based on your identification of the problem.

8.5 Save Your Progress With Version Control

Commit your work to Git whenever something works. Small, frequent commits turn every AI session into a series of reversible experiments. If an agent makes a mess, you return to the last good version in seconds. Most tools can create commits for you and describe what changed; ask them to.

8.6 Review Security Before You Launch

Before real users arrive, pause and run through the checklist below. You can ask your AI tool to help with each item, but you should be the one who confirms the answer.

1. Are all secret keys stored as environment variables, and absent from the code and from the repository?
2. Does every page and every data request check who the user is and what they are allowed to see?
3. Have you tried, as a logged-in user, to open another user's data by changing an address or an ID?

4. Are the packages the AI installed genuine, necessary, and kept up to date?
5. Do any agents or integrations have more access than their task requires?
6. Are errors handled so that failures do not expose internal details or leave actions half-completed?
7. Is there a backup of your data, and have you tested restoring it?
8. Does your privacy notice match what the application actually collects and stores?

8.7 Document as You Go

Once a feature is working, ask your AI tool to generate a brief explanation of what the code does, which external services it depends on, and what could go wrong. Store this in a simple README file in your project. Future you, and any developer you later hire, will be grateful.

CLOSING

The Wave Is Here. Learn to Ride It.

The tools available to builders today are genuinely extraordinary. We are living through a period in which the distance between an idea and a working application has compressed more dramatically than at any previous point in the history of software. That is worth celebrating, and worth taking advantage of.

But the builders who will get the most from this era are not the ones who outsource all thinking to their tools. They are the ones who bring enough understanding to the collaboration to make it powerful: to direct the AI with precision, to catch its mistakes, and to build things that are not merely functional but safe, compliant, performant, and maintainable.

You do not need to become a developer. You need to become someone who is not afraid of the underlying reality of what you are building; someone who can read an error message without panicking. That is surf coding. It is not about conquering the ocean. It is about learning the water well enough to ride it gracefully.

> *"Every developer you admire was once completely lost. They are not on the other side of confusion; they are simply comfortable sitting in it while they find their way through. You can learn that comfort too."*

Start small. Pick one concept. Build one thing. Let genuine problems drive genuine learning. And use every tool available to you, including the remarkable AI tools of this moment, to build things you are proud of.

Thank you for reading. Now go build something.

APPENDIX A

Glossary

Agent. An AI system that takes a sequence of actions — reading files, running commands, calling services — toward a goal, rather than only answering questions.

API. Application Programming Interface: a defined way for one program to request services or data from another.

Authentication. Verifying who a user is. Compare *authorization*.

Authorization. Deciding what an authenticated user is permitted to see or do.

Backend. The server-side part of an application that stores data and applies business rules.

Commit. A saved snapshot of a project's files in version control, with a short description.

Deployment. Publishing an application so that others can use it on the internet.

Diff. A display of exactly which lines of code were added, removed, or changed.

DOM. Document Object Model: the browser's structured representation of a web page.

Environment variable. A named configuration value, such as a key or database address, kept outside the source code.

Frontend. The part of an application that runs in the user's browser or device.

Git. The most widely used version control system.

HTTP status code. A three-digit number returned with every web response: 2xx success, 4xx a problem with the request, 5xx a problem on the server.

JWT. JSON Web Token: a compact, signed token commonly used to carry a user's identity between systems.

MCP. Model Context Protocol: an open standard for connecting AI applications to external tools and data sources.

OAuth. A standard that lets a user grant one application limited access to their account on another service without sharing a password.

OWASP. The Open Worldwide Application Security Project, a non-profit that publishes widely used security guidance.

Passkey. A password replacement based on cryptographic keys stored on the user's device and unlocked by biometrics or a PIN.

Prompt injection. An attack in which hostile instructions hidden in content an AI reads cause it to behave in unintended ways.

Row-level security. A database feature that restricts which rows each user may read or change.

Supply chain (software). The collection of outside packages, tools, and services on which an application depends.

Vibe coding. Building software by describing the desired result in natural language and relying on AI to write the code.

Webhook. A message one service sends automatically to your application when an event occurs, such as a completed payment.

APPENDIX B

The OWASP Top 10:2025 in Plain Language

The OWASP Top 10 is a widely used awareness document that ranks the most critical categories of web application security risk. The 2025 edition, announced in November 2025, was the first since 2021 (OWASP, 2025). Table B.1 restates each category in everyday terms, together with a question you can put to your AI tool or developer. Because the list is an awareness guide rather than a complete checklist, treat it as a starting point.

Table B.1. OWASP Top 10:2025 categories, with plain-language explanations

Rank and category	In plain language	Question to ask
A01 Broken Access Control	Users can reach data or actions they should not.	Is every request checked against the user's permissions?
A02 Security Misconfiguration	Insecure settings, defaults, or exposed services.	Have we reviewed every setting before launch?
A03 Software Supply Chain Failures	Compromised or untrustworthy outside packages and tools.	Do we know and trust what we depend on?
A04 Cryptographic Failures	Sensitive data poorly protected in storage or transit.	Is sensitive data encrypted, and are the keys safe?

Rank and category	In plain language	Question to ask
A05 Injection	Untrusted input is run as commands or queries.	Is all user input validated and handled safely?
A06 Insecure Design	Missing protections at the planning stage.	Did we consider misuse before building?
A07 Authentication Failures	Weak login, session, or password handling.	Is login robust, with multi-factor or passkeys?
A08 Software or Data Integrity Failures	Code or data trusted without verifying its source.	Can updates and data be verified as genuine?
A09 Security Logging and Alerting Failures	Attacks go unnoticed because nothing is recorded or reported.	Would we know if something went wrong?
A10 Mishandling of Exceptional Conditions	Errors and unusual situations are handled badly or unsafely.	What happens when things fail?

Note: AI-specific risks, including prompt injection, are covered by the separate OWASP list for large language model and generative AI applications.

REFERENCES

Sources and Further Reading

Product features, prices, and rankings change quickly. Readers should check the current documentation of any tool before relying on the descriptions in this book.

Collins Dictionary (2025). *Word of the Year 2025: vibe coding.* HarperCollins Publishers.

European Parliament and Council (2016). *Regulation (EU) 2016/679 (General Data Protection Regulation)*, Article 7, Conditions for consent.

Karpathy, A. (2025). Post on the social platform X introducing the term “vibe coding,” February 2025.

Model Context Protocol. *Official documentation and specification.* modelcontextprotocol.io.

MDN Web Docs. *Learn web development.* developer.mozilla.org.

OWASP Foundation (2025). *OWASP Top 10:2025.* owasp.org/Top10.

OWASP GenAI Security Project. *OWASP Top 10 for Large Language Model Applications.* genai.owasp.org.

Harvard University. *CS50: Introduction to Computer Science.* cs50.harvard.edu.

freeCodeCamp. *Free coding curriculum.* freecodecamp.org.

The Odin Project. *Full-stack open-source curriculum.* theodinproject.com.

Stripe. *Developer documentation: API keys, webhooks, and testing.* docs.stripe.com.