FULL-STACK DEVELOPMENT WITH CLAUDE CODE

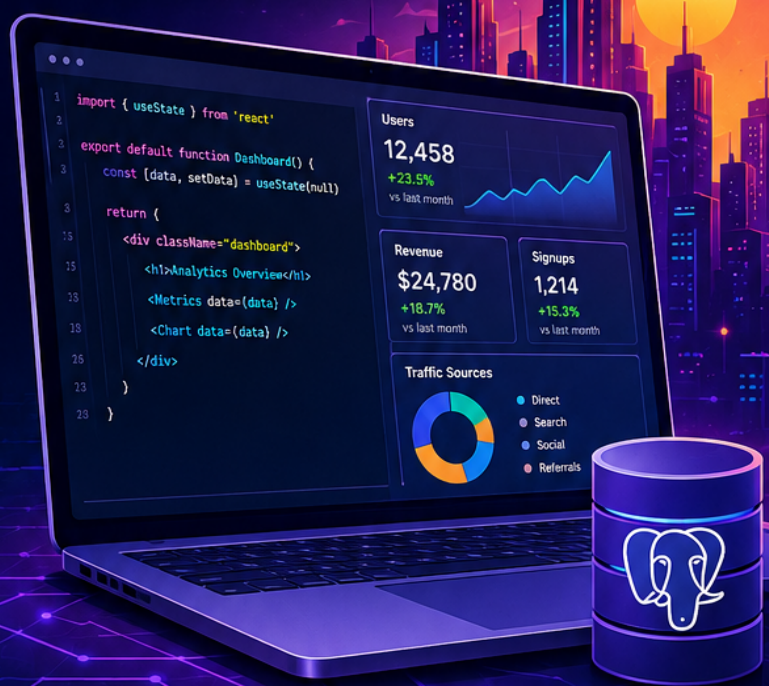
BUILDING PRODUCTION-READY WEB APPLICATIONS WITH AI-ASSISTED ENGINEERING

AI-ASSISTED DEVELOPMENT
Master Claude Code to accelerate your workflow and ship better software

MODERN TECH STACK
React, Next.js, TypeScript, PostgreSQL and more

SECURE, SCALABLE, PRODUCTION-READY
Build applications that are secure, reliable and ready for real users

END-TO-END PROJECTS
Learn by building. Ship a complete SaaS application from start to finish



JASON CRAWFORD

Full-Stack Development with Claude Code

Building Production-Ready Web Applications with AI-Assisted Engineering

Steve Publications

This book is available at
https://leanpub.com/full-stackdevelopmentwithclaudecode

This version was published on 2026-08-21

Contents

Building Production-Ready Web Applications with AI-Assisted Engineering

This book takes you from zero web development knowledge to advanced professional proficiency building secure, scalable, production-ready full-stack applications using Claude Code as your AI-assisted development partner. You will learn modern web technologies including React, Next.js, TypeScript, PostgreSQL, and more while mastering the workflows, prompting techniques, and engineering practices that make AI-assisted development genuinely productive rather than just novel. Through multiple hands-on projects culminating in a complete end-to-end SaaS application, you will gain both the conceptual understanding and practical skills needed to design, develop, test, deploy, and maintain real-world web applications.

Introduction: The New Era of Web Development

A few years ago, learning full-stack web development meant memorizing framework syntax, wrestling with build tools, debugging cryptic errors at 2 AM, and spending weeks just getting your environment configured correctly. You would read documentation, follow tutorials, make mistakes, slowly internalize patterns, and hope you were building things the right way. The barrier to entry was real, and the path from beginner to competent developer took months or years of focused effort.

Today, that landscape has fundamentally changed. AI-assisted development tools like Claude Code have not replaced the need for human understanding, but they have dramatically altered what it takes to go from idea to working application. You can now describe what you want to build in natural language and watch as code appears, tests run, and features materialize at a pace that would have been unimaginable just a few years ago.

This is both an opportunity and a challenge. The opportunity is obvious: you can build more, faster, with less friction than ever before. The challenge is subtler but more important. When AI generates code for you, it is tempting to treat it as black magic that produces working software without needing deep understanding. This is dangerous. Code you do not understand is a liability. Bugs in code you cannot reason about become mysteries. Security vulnerabilities in patterns you never learned to recognize stay hidden until they are exploited.

This book is built on one central thesis: AI-assisted development amplifies human capability, it does not replace the need for it. The developers who will thrive in this new era are not those who passively accept whatever code an AI generates, but those who understand web development deeply enough to guide the AI effectively, verify its output critically, and own the architecture and decisions behind the systems they build. Claude Code is your partner, not your replacement.

This book takes you on a complete journey from absolute beginner to advanced professional proficiency. We will start with the fundamentals of

how the web works, build up through frontend and backend development, databases, authentication, testing, deployment, and operations, and culminate in advanced topics like real-time systems, caching strategies, and scalable architecture patterns. Throughout this journey, Claude Code will be your constant companion: we will use it to scaffold projects, generate code, debug problems, write tests, refactor systems, and explore complex topics together.

But here is what makes this book different from a typical tutorial or documentation walkthrough: every piece of code you encounter, whether written by you or generated by Claude Code, will be explained in terms of why it works, not just how to type it. You will learn the reasoning behind architectural decisions, the trade-offs between different approaches, the security implications of common patterns, and the failure modes that turn working prototypes into production disasters. By the time you finish this book, you will not only be able to build impressive applications with Claude Code as your assistant, but you will understand them well enough to debug, extend, optimize, and defend them when things go wrong.

The technology stack we will use is deliberately chosen for coherence, modernity, and real-world relevance: TypeScript as our language across frontend and backend, React for building interactive user interfaces, Next.js as our unified full-stack framework, PostgreSQL as our database, Prisma as our type-safe data access layer, Tailwind CSS for styling, Auth.js for authentication, and a comprehensive testing stack including Vitest and Playwright. Each tool is explained not just in terms of what it does, but why it was chosen over alternatives, when it is appropriate to use, and what problems it solves.

As you read this book, you will encounter Claude Code interactions presented as example prompts with explanations of why they are effective. You will see complete directory structures, configuration files, environment variables, and working code that you can reproduce on your own machine. You will follow the incremental development of a substantial SaaS application called TaskFlow from initial requirements through architecture design, implementation, testing, security hardening, deployment, and production monitoring.

This book assumes no prior programming experience, but it does expect curiosity, patience, and a willingness to engage deeply with technical material. Some chapters will move quickly through concepts that may feel familiar if you have development experience. Others will slow down to explain foundational ideas in detail. The goal is for every reader, regardless of starting point, to finish with genuine proficiency rather than superficial familiarity.

Let us begin by understanding what actually happens when someone visits a website. This mental model will anchor everything we build from here forward.

Chapter 1: How the Web Works – The Mental Model Every Developer Needs

Before you write a single line of code, you need a clear mental picture of what happens behind the scenes when someone types a URL into their browser and clicks Enter. Understanding this journey is not academic trivia; it shapes how you design applications, debug problems, optimize performance, and make architectural decisions throughout your career as a developer.

What Happens When You Visit a Website – The Complete Journey from DNS to Rendered Page

Imagine a user types https://example.com into their browser and presses Enter. Here is what happens next, step by step:

First, the browser needs to find where example.com lives on the internet. Domain names like example.com are human-readable labels, but computers communicate using IP addresses – numerical identifiers like 93.184.216.34. The browser queries a DNS (Domain Name System) server to translate the domain name into an IP address. This is similar to looking up a phone number in a contact list: you remember the name, but your phone needs the number to make the call.

Once the browser knows the IP address, it establishes a secure connection to that server using HTTPS (HTTP Secure). The S matters: HTTPS encrypts all communication between the browser and the server so that no one eavesdropping on the network can read or modify the data being exchanged. This encryption is handled by TLS (Transport Layer Security), which negotiates cryptographic keys between browser and server before any application data is sent.

With the secure connection established, the browser sends an HTTP request to the server. At its simplest, this request says: "I am requesting the homepage at https://example.com." The server processes this request and

sends back an HTTP response containing HTML (HyperText Markup Language), which is essentially a structured description of the page content.

The browser receives this HTML and begins parsing it – reading through the markup to understand what elements are on the page: headings, paragraphs, images, buttons, links, and so on. As it encounters references to CSS (Cascading Style Sheets) files, JavaScript files, images, or other resources, it makes additional HTTP requests to fetch those assets.

The CSS tells the browser how everything should look – colors, fonts, spacing, layout, animations. The JavaScript adds interactivity – form validation, dynamic content updates, animations triggered by user actions. The browser applies all of this together to render the final page that the user sees and interacts with.

This entire process typically takes a fraction of a second on modern networks, but each step introduces potential points of failure, latency, or security concerns. As a developer, you will be responsible for optimizing some of these steps (like reducing the number of requests or the size of assets), securing others (like ensuring proper HTTPS configuration), and understanding all of them when things go wrong.

Clients and Servers – Understanding the Division of Labor in Web Architecture

The web operates on a client-server model. This is not just terminology; it is a fundamental architectural principle that determines where code runs, where data lives, and how applications are structured.

The client is the user's browser (or mobile app, but we will focus on browsers for now). The client is responsible for:

- Displaying content to the user in a visually appealing way
- Capturing user interactions like clicks, typing, scrolling
- Running JavaScript that adds dynamic behavior and interactivity
- Making requests to servers when it needs data or wants to perform actions

The server is a computer (or cluster of computers) running somewhere on the internet that is always online and waiting for requests. The server is responsible for:

- Receiving and processing HTTP requests from clients
- Querying databases to retrieve or store data
- Running business logic – the rules that determine what your application does
- Generating responses, which might be HTML pages, JSON data, images, or other content
- Managing authentication, authorization, and security

This division of labor is not arbitrary. The client runs in the user's browser, which means it has direct access to the screen, keyboard, mouse, and other input devices – making it ideal for rendering interfaces and capturing interactions. However, the browser environment is inherently untrusted: anyone can inspect and modify client-side code, so you should never put sensitive logic or secrets there.

The server runs in a controlled environment that you manage. It has access to databases, file systems, external APIs, and other resources that should not be exposed directly to users. Because you control the server environment, it is the right place for authentication logic, database credentials, payment processing, and any code that handles sensitive data or business-critical operations.

In modern web development, the line between client and server has become more nuanced than it used to be. Frameworks like Next.js allow you to write code that runs on either the server or the client (or both), often within the same file. But the fundamental principle remains: understand where your code is running, what environment it has access to, and what security implications that carries.

HTTP and HTTPS – How Browsers Talk to Servers (Methods, Status Codes, Headers)

HTTP (Hypertext Transfer Protocol) is the language that browsers and servers use to communicate. Understanding HTTP basics is essential because every web application you build will be built on top of it, even when frameworks abstract away most of the details.

An HTTP request consists of several parts:

- A method that indicates what action should be performed (GET, POST, PUT, DELETE, etc.)

- A URL specifying which resource is being requested
- Headers containing metadata about the request (like the browser type, accepted content types, authentication tokens)
- Optionally, a body containing data being sent to the server

HTTP methods have semantic meaning that you should respect when designing your applications:

- GET retrieves data from the server. It should be safe (not modify anything) and idempotent (calling it multiple times has the same effect as calling it once).
- POST creates new resources or submits data that causes side effects (like processing a payment).
- PUT replaces an existing resource entirely with new data.
- PATCH partially updates an existing resource.
- DELETE removes a resource.

When the server responds, it sends back:

- A status code indicating the result of the request
- Headers with metadata about the response
- Optionally, a body containing the requested data or content

Status codes are three-digit numbers that fall into categories:

- 2xx (Success): 200 OK means the request succeeded. 201 Created means a new resource was created.
- 3xx (Redirection): 301 Moved Permanently and 302 Found tell the browser to look elsewhere for the resource.
- 4xx (Client Error): 400 Bad Request means the server could not understand the request. 401 Unauthorized means authentication is required. 403 Forbidden means the user is authenticated but does not have permission. 404 Not Found means the requested resource does not exist.
- 5xx (Server Error): 500 Internal Server Error means something went wrong on the server. 502 Bad Gateway and 503 Service Unavailable indicate infrastructure problems.

HTTPS is HTTP wrapped in encryption via TLS. The encryption ensures confidentiality (eavesdroppers cannot read your data), integrity (data cannot be modified in transit without detection), and authentication (you are actually talking to the server you think you are). HTTPS should be used for essentially all modern web applications; it is not optional for anything handling user data, credentials, or sensitive information.

Frontend vs Backend vs Full Stack – Defining the Roles and Responsibilities

With the client-server model in mind, we can define the terms frontend, backend, and full stack:

Frontend development focuses on everything that runs in the browser and is directly visible or interactive to the user. This includes HTML structure, CSS styling, JavaScript interactivity, responsive design for different screen sizes, animations, accessibility, and performance optimization of the user interface. Frontend developers specialize in creating delightful, intuitive experiences that work across browsers and devices.

Backend development focuses on everything that runs on the server and is invisible to users but powers the application behind the scenes. This includes API design, database schema design and queries, authentication and authorization logic, business rules, integrations with external services, caching strategies, security hardening, logging, monitoring, and deployment infrastructure. Backend developers specialize in building reliable, secure, scalable systems that handle data correctly under load.

Full-stack development means working across both frontend and backend – understanding the entire request lifecycle from browser to server to database and back again. Full-stack developers can build complete applications end-to-end, making architectural decisions that consider implications across the entire stack rather than optimizing one layer in isolation.

This book is designed for full-stack development because it gives you the most flexibility and the deepest understanding of how web applications work as integrated systems. You do not need to be equally expert at everything from day one; specialization often develops naturally based on interest and experience. But having a working knowledge of the entire stack makes you a better developer regardless of where you eventually focus.

Static vs Dynamic Websites – The Fundamental Distinction That Shapes Everything

One of the most important distinctions in web development is between static and dynamic websites, because this choice influences your technology selection, deployment strategy, performance characteristics, and even security considerations.

A static website serves the same content to every visitor, every time. The HTML files are pre-built and stored on a server or CDN (Content Delivery Network). When someone requests a page, the server simply sends back the file that already exists. Static websites are fast (no computation needed), cheap to host, simple to deploy, and relatively secure (limited attack surface). Examples include documentation sites, portfolios, marketing pages, and blogs with pre-generated content.

A dynamic website generates content on-the-fly in response to each request. When someone requests a page, the server runs code that might query a database, check the user's authentication status, apply business logic, and then construct a personalized response. Dynamic websites are necessary for applications where content varies by user (like social media feeds), where data changes frequently (like e-commerce inventory), or where users interact with the application to create or modify data (like project management tools). Examples include Gmail, Netflix, GitHub, and virtually any web application requiring user accounts.

Most modern web applications are dynamic at their core but incorporate static elements strategically. Your Next.js application will dynamically generate pages based on database content and user sessions, but it will also serve static assets like images, JavaScript bundles, and CSS files efficiently from CDNs.

Understanding this distinction matters because some tools and deployment platforms optimize for one or the other. Vercel, our primary deployment platform, excels at both but has different optimization strategies depending on whether a page is statically generated (built once and cached globally) or dynamically rendered (computed per-request). You will learn to make these decisions deliberately as we build applications throughout this book.

Why Type Safety Matters – A Brief Case for TypeScript From Day One

Before we dive into writing code, let us address a decision that shapes everything else: why we will use TypeScript rather than plain JavaScript.

JavaScript is the language of the web browser. Every modern browser can execute JavaScript natively, which is why it powers frontend development. Node.js extended JavaScript to run on servers, making it possible to use the same language across frontend and backend. JavaScript is flexible, widely supported, and has an enormous ecosystem of libraries and frameworks.

But JavaScript has a fundamental design characteristic that causes problems in large projects: it is dynamically typed with very loose type coercion. Variables can hold any value and change types at runtime. You can accidentally pass a string where a number is expected, access properties on null without getting compile-time warnings, or have functions return unexpected types that break downstream code. These issues often manifest as runtime errors – bugs that only appear when users interact with your application in unexpected ways.

TypeScript addresses these problems by adding a type system to JavaScript. With TypeScript, you declare what types of values variables, function parameters, and return values should have. The TypeScript compiler checks your code before it runs and catches type-related errors at development time rather than letting them reach production. For example, if you define a function that expects a user object with an email field, TypeScript will warn you immediately if you try to call that function with an object missing the email field.

The benefits of TypeScript are especially pronounced in full-stack development:

- Shared types between frontend and backend prevent mismatches in API contracts
- Better IDE support with autocomplete, inline documentation, and refactoring tools
- Self-documenting code where types clarify what data structures look like
- Fewer runtime errors in production because many bugs are caught during development
- Safer refactoring when changing large codebases

TypeScript compiles down to JavaScript, so it runs everywhere JavaScript runs. You do not sacrifice compatibility or performance for type safety. The learning curve is modest, especially if you approach it from the beginning rather than retrofitting it into existing habits.

Throughout this book, we will use TypeScript consistently across frontend and backend code. You will see how types make your code more reliable, your development experience smoother, and your collaboration with AI tools like Claude Code more effective – because precise type definitions give Claude clearer constraints and expectations for the code it generates.

Now that you understand how the web works at a conceptual level, let us set up your development environment so you can start building real applications.

Chapter 2: Setting Up Your Development Environment with Claude Code

Writing software requires tools, and setting those tools up correctly is an investment that pays dividends throughout every project you build. This chapter will guide you through installing everything you need: the JavaScript runtime, package manager, version control system, code editor, and most importantly, Claude Code – your AI-assisted development partner for the rest of this book.

The Tools You Need – Node.js, npm, Git, VS Code, and Why Each Matters

Before diving into installation commands, let us understand what each tool does and why it is part of our stack:

Node.js is a runtime environment that allows JavaScript to run outside the browser, on your local machine or on servers. Since we will be building full-stack applications with JavaScript and TypeScript, Node.js is essential for running backend code, build tools, testing frameworks, and development utilities. It transforms JavaScript from a language that only runs in web pages into a general-purpose programming environment.

npm (Node Package Manager) comes bundled with Node.js and is the tool you use to install, manage, and share JavaScript libraries and packages. The JavaScript ecosystem produces thousands of open-source packages for virtually every task – HTTP servers, database clients, testing frameworks, UI components, utility functions, and more. npm lets you add these dependencies to your projects with simple commands rather than downloading and configuring them manually.

Git is a distributed version control system that tracks changes to your code over time. It allows you to create snapshots of your project at different

points (commits), branch off to experiment with new features without affecting working code, collaborate with other developers, and revert changes when mistakes happen. Git is the universal standard for source control in professional software development, and platforms like GitHub, GitLab, and Bitbucket build on top of it to add collaboration features like pull requests, code reviews, and continuous integration.

VS Code (Visual Studio Code) is a free, open-source code editor from Microsoft that has become the most popular development environment for web developers. It provides syntax highlighting, intelligent autocomplete, integrated terminal, debugging tools, Git integration, and thousands of extensions. While you can use any text editor or IDE, VS Code offers an excellent balance of power and simplicity, plus strong support for JavaScript, TypeScript, and all the frameworks we will use.

Claude Code is Anthropic's agentic coding tool that runs in your terminal (and integrates with VS Code) to assist with writing, understanding, debugging, and refactoring code through natural language conversation. Unlike simple autocomplete tools that suggest a few lines at a time, Claude Code understands your entire codebase context, can create and modify files, run commands, explain complex code, write tests, and help you work through architectural decisions. It is the central AI tool we will use throughout this book, and setting it up correctly will define your development experience.

Installing Node.js and Verifying Your Setup – Getting the JavaScript Runtime Running

Node.js installation depends on your operating system. Here are the recommended approaches:

For macOS users, the simplest approach is using Homebrew, a package manager for macOS. If you do not have Homebrew installed, open Terminal and run:

```
1 /bin/bash -c "$(curl -fsSL
  ↪ https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

Then install Node.js:

```
brew install node
```

For Windows users, download the LTS (Long Term Support) installer from https://nodejs.org/. Choose the LTS version rather than Current – LTS versions are more stable and receive longer support. Run the installer and accept the default options, which include adding Node.js to your system PATH.

For Ubuntu/Debian Linux users:

```
curl -fsSL https://deb.nodesource.com/setup_lts.x | sudo -E bash -
sudo apt-get install -y nodejs
```

After installation, verify that Node.js is working by opening a terminal and running:

```
node --version
```

You should see a version number like v20.x.x or higher. As of this writing, Node.js 20 LTS is the recommended minimum version, with Node.js 22+ preferred for some tools including Claude Code when installed via npm.

Also verify that npm is available:

```
npm --version
```

You should see a version number like 10.x.x or higher. npm comes bundled with Node.js, so if node works, npm should work too.

To test that your installation is functioning correctly, create a simple test file. In your terminal, run:

```
echo 'console.log("Hello from Node.js!");' > test-node.js
node test-node.js
```

You should see "Hello from Node.js!" printed to the terminal. If you do, your JavaScript runtime is working and ready for development.

Understanding npm and Package Management – How JavaScript Projects Share Code

JavaScript projects use a file called package.json to track their dependencies – the external packages they rely on. When you install a package with npm, it downloads the package code into a node_modules directory in your project and records the dependency in package.json. This allows anyone else to reproduce your project's dependencies by running npm install, which reads package.json and installs everything listed there.

Let us create a simple project to see how this works. In your terminal:

```
mkdir my-first-project
cd my-first-project
npm init -y
```

The npm init -y command creates a basic package.json file with default values. Open it in VS Code or any text editor; you will see something like:

```
{
  "name": "my-first-project",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

The key fields to understand are:

- name: identifies your project
- version: follows semantic versioning (major.minor.patch)
- main: the entry point file for your project
- scripts: custom commands you can run with npm run
- dependencies and devDependencies (not yet present): will list packages your project uses

Now let us install a package. We will use chalk, a popular library for styling terminal output:

```
npm install chalk
```

After installation, you will see:

- A node_modules directory containing chalk and its dependencies
- An updated package.json with chalk listed under dependencies
- A package-lock.json file that records exact versions of all installed packages

The package-lock.json file is important: it ensures that everyone working on the project installs the exact same versions of every dependency, preventing subtle bugs caused by version differences. Always commit package-lock.json to your Git repository.

To use chalk in your code, create a file called index.js:

```
const chalk = require('chalk');
console.log(chalk.blue('Hello from chalk!'));
console.log(chalk.red.bold('This text is red and bold.'));
```

Run it with:

```
node index.js
```

You should see colored output in your terminal. This demonstrates the fundamental pattern you will use throughout this book: install packages with npm, import them in your code, and leverage their functionality to avoid reinventing common solutions.

Installing Claude Code – Step-by-Step Setup and First Run

Claude Code is available through multiple installation methods. The recommended approach depends on your operating system and preferences.

Method 1: Native installer (recommended for most users)

The native installer includes automatic updates and works across platforms.

On macOS or Linux, open Terminal and run:

```
curl -fsSL https://claude.ai/install.sh | bash
```

On Windows with PowerShell:

```
irm https://claude.ai/install.ps1 | iex
```

After installation completes, verify it worked:

```
claude --version
```

Method 2: Homebrew (macOS)

If you prefer managing tools through Homebrew:

```
brew install --cask claude-code
```

Method 3: npm (requires Node.js 22+)

If you already have Node.js installed and prefer npm:

```
npm install -g @anthropic-ai/claude-code
```

Note that the npm installation method requires Node.js version 22 or higher. If you are on an older Node version, use one of the other methods.

Authentication

Claude Code requires an Anthropic account with a paid plan (Pro, Max, Team, or Enterprise) or an Anthropic Console account for API billing. The free Claude plan does not include Claude Code access.

Run claude in your terminal for the first time:

```
claude
```

You will be prompted to authenticate. Follow the on-screen instructions, which typically involve opening a browser window to sign in with your Anthropic account credentials. Once authenticated, you will see the Claude Code interface – a conversational prompt where you can start giving it instructions.

Running diagnostics

If you encounter issues, Claude Code includes a diagnostic tool:

```
claude doctor
```

This command checks your installation, authentication status, and environment configuration, reporting any problems and suggesting fixes.

Configuring Claude Code for Development – Settings, Model Selection, and Project Context

Once Claude Code is installed and authenticated, some initial configuration will make it significantly more effective as a development partner.

Understanding permission modes

Claude Code operates in different permission modes that control how much autonomy it has when making changes:

- Manual mode: Every file edit or command requires your explicit approval before executing. This is the safest mode for learning and gives you full visibility into what Claude proposes to do.
- Accept Edits mode: File edits are automatically approved, but commands still require confirmation. Good balance of speed and control for routine development work.
- Auto mode: Both edits and commands are auto-approved. Fastest but requires trust in Claude's output – use only when you are comfortable with its behavior on your project.
- Plan mode (read-only): Claude can read files and suggest changes but cannot modify anything or run commands. Useful for exploring codebases, understanding architecture, or getting recommendations without risk of unintended changes.

You can cycle through permission modes by pressing Shift+Tab in the Claude Code interface. For most of this book's examples, we will assume manual or accept edits mode so you can see and understand each change Claude proposes.

Creating CLAUDE.md for project context

One of the most powerful features of Claude Code is CLAUDE.md – a markdown file in your project root that contains persistent instructions loaded

at the start of every session. This is where you define project conventions, tech stack details, coding standards, and any rules Claude should follow.

In any project directory, run:

```
claude /init
```

This creates a CLAUDE.md file with starter content. You will customize this as your projects grow. For now, here is an example of what a well-structured CLAUDE.md might look like for a Next.js TypeScript project:

```
# Project Instructions

# Tech Stack
- Next.js 15 with App Router
- React 19 with TypeScript
- Prisma ORM with PostgreSQL
- Tailwind CSS v4
- Auth.js v5 for authentication
- Vitest + Playwright for testing

# Conventions
- All components use functional components with TypeScript interfaces for props
- Server Components by default; add "use client" only when necessary
- Use Zod for all input validation (forms, API routes, environment variables)
- Naming: PascalCase for components, camelCase for functions and variables
- Always include error handling in async operations
- Write tests for all business logic and critical UI components

# Commands
- dev: npm run dev
- build: npm run build
- test: npm run test
- lint: npm run lint
- db:migrate: npx prisma migrate dev
```

This file does not need to be perfect initially; you will refine it as you learn what instructions are most helpful. The key principle is: anything Claude needs to know about your project that is not obvious from reading the code belongs in CLAUDE.md.

Model selection

Claude Code supports different Anthropic models with varying capabilities and costs. You can check or change your current model with:

```
/model          # Shows current model
/model sonnet   # Switch to Sonnet
/model opus     # Switch to Opus (most capable, highest cost)
/model haiku    # Switch to Haiku (fastest, lowest cost)
```

For most development tasks described in this book, Sonnet provides an excellent balance of capability and efficiency. Opus is useful for complex architectural decisions or particularly tricky problems, while Haiku can handle straightforward tasks like simple refactoring or documentation generation at lower cost.

You can also set a default model via environment variable:

```
export ANTHROPIC_MODEL=sonnet
```

Add this to your shell configuration file (like ~/.zshrc on macOS) to make it persistent.

Your First Command-Line Session – Navigating Directories and Running Commands With Confidence

Before we start building applications, let us make sure you are comfortable with basic terminal commands, since Claude Code lives in the terminal and much of your development workflow will involve command-line operations.

Open your terminal and try these fundamental commands:

```
pwd             # Print working directory -- shows your current location in the
↪  filesystem
ls              # List files and directories (ls -la for detailed view
↪  including hidden files)
cd <directory>   # Change directory (cd .. to go up one level, cd ~ to go home)
mkdir <name>     # Create a new directory
touch <file>     # Create an empty file
cat <file>       # Display file contents
rm <file>        # Delete a file (use with caution -- no undo)
cp <source> <dest>   # Copy a file or directory
mv <source> <dest>   # Move or rename a file or directory
```

Now let us practice with Claude Code. Create a new project directory:

```
mkdir hello-claude
cd hello-claude
claude /init
```

Then start a Claude Code session:

```
claude
```

Try asking Claude to create a simple HTML file for you:

> Create an index.html file with a centered heading that says "Hello from Claude Code!" and a styled paragraph below it explaining this was created with AI assistance. Use some basic CSS to make it look nice.

Claude will respond with the proposed file content, and if you are in manual mode, you will need to approve the edit. Once accepted, open index.html in your browser to see the result.

This simple interaction demonstrates the core pattern you will use throughout this book: describe what you want in natural language, Claude generates or modifies code, you review the changes, and you iterate until the result matches your expectations. The key skill is learning to write prompts that give Claude enough context and constraints to produce useful output on the first try – a topic we will explore in depth as we build more complex applications.

With your development environment configured and Claude Code ready to assist, you are now prepared to start building real web applications. Next, we will learn the foundational technologies of the web: HTML for structure, CSS for styling, and responsive design principles that make websites work on every device.

Chapter 3: Frontend Fundamentals – HTML, CSS, and Responsive Design

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

HTML Structure – Elements, Attributes, Semantics, and Building Meaningful Pages

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

CSS Basics – Selectors, Properties, Box Model, and Making Things Look Right

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Responsive Design Principles – Making Websites Work on Every Screen Size

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Building Your First Page With Claude Code – From Prompt to Working HTML and CSS

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Understanding What Claude Code Generated – Reading AI-Written Frontend Code

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Common Frontend Mistakes And How To Avoid Them

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Chapter 4: JavaScript and TypeScript – The Language of the Web

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Variables, Types, and Data Structures – Storing and Organizing Information in Code

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Functions and Control Flow – Making Your Code Do Things Conditionally and Repeatedly

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Objects, Arrays, and Modern JavaScript Features – Working With Complex Data

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

The DOM and Browser APIs – How JavaScript Interacts With Web Pages

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

TypeScript Fundamentals – Type Annotations, Interfaces, and Catching Errors Early

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Using Claude Code to Learn Programming – Asking for Explanations, Examples, and Refactoring

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Chapter 5: React and Component-Based Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Why React – The Problem That Components Solve in Modern Web Applications

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Your First React Component – JSX, Props, and Composing UI From Pieces

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

State and Interactivity – Making Components Respond to User Actions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Effects and Side Effects – Data Fetching, Subscriptions, and Lifecycle Management

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Building a Small Interactive App With Claude Code – A Task List or Similar Project

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Reviewing and Understanding AI-Generated React Code

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Chapter 6: Introduction to Next.js – Bridging Frontend and Backend

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

The Problem With Pure Client-Side React – SEO, Performance, and User Experience

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

What Next.js Is and Why It Matters – Server Components, Routing, and Full-Stack in One Framework

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Creating Your First Next.js Project With Claude Code – Vite vs Next.js vs Other Options

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Understanding the App Router – File-Based Routing and Layouts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Server Components vs Client Components – When to Use Each and Why It Matters

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Your First Full-Stack Page – Rendering Data From Server to Browser

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Chapter 7: Databases and Data Modeling with Prisma and PostgreSQL

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Why You Need a Database – Persisting Data Beyond Browser Refresh

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Relational Databases Explained – Tables, Rows, Columns, Keys, and Relationships

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Setting Up PostgreSQL – Local Installation or Cloud Options for Development

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Prisma Schema Design – Defining Your Data Model in Type-Safe Declarations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Migrations – Evolving Your Database Structure Without Losing Data

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Querying Data With Prisma in Next.js – Reading and Writing From Your Application

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Chapter 8: APIs, Authentication, and Authorization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

RESTful API Design – Resources, Endpoints, Methods, and HTTP Semantics

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Building API Routes in Next.js – Serverless Functions That Handle Requests

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Authentication vs Authorization – Who You Are versus What You Can Do

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Implementing Authentication With Auth.js – OAuth Providers and Email Login

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Session Management and Protected Routes – Keeping Users Logged In Securely

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Securing Your Application Secrets – Environment Variables and Never Hardcoding Credentials

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Chapter 9: The Project – Designing a Full-Stack SaaS Application

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

From Idea to Architecture – Requirements Analysis and Feature Prioritization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Designing the Data Model – Entities, Relationships, and Scalability Considerations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Project Structure and Conventions – Organizing Files for Maintainability

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Scaffolding the Application With Claude Code – Generating Boilerplate and Initial Setup

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Setting Up Development Workflows – Hot Reloading, Database Seeding, and Debugging

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Establishing Coding Standards and AI Prompting Patterns for the Project

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Chapter 10: Building the Frontend – Components, State, and User Experience

Layout Architecture – Navigation, Headers, Sidebars, and Responsive Shell Components

Form Handling – Controlled Inputs, Validation, Error Display, and Submission Patterns

Data Fetching and Loading States – Suspense, Optimistic Updates, and User Feedback

State Management Strategies – When to Use Local State, Context, or External Libraries

Accessibility Best Practices – Keyboard Navigation, Screen Readers, and Inclusive Design

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Iterative UI Development With Claude Code – Prompting for Components and Refinements

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Chapter 11: Building the Backend – Business Logic, APIs, and Data Operations

RESTful Endpoints for Your Application – Designing Consistent API Routes

Complex Queries and Database Relationships – Joins, Filtering, Pagination, and Aggregation

Input Validation and Error Handling – Rejecting Bad Data Gracefully and Securely

File Uploads and Media Handling – Storing and Serving User Content

Background Jobs and Async Processing – When to Defer Work Outside Request Handlers

Using Claude Code for Complex Backend Logic – Breaking Down Hard Problems Into Prompts

Chapter 12: Testing – Confidence at Every Level

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Why Testing Matters – Catching Bugs Early and Enabling Confident Refactoring

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Unit Testing With Vitest – Testing Functions and Business Logic in Isolation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Component Testing With React Testing Library – Verifying UI Behavior From the User's Perspective

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Integration Testing API Routes – Ensuring Endpoints Work With Real Data

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

End-to-End Testing With Playwright – Automating Browser-Level User Flows

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Using Claude Code to Write and Debug Tests – Generating Test Cases and Fixing Failures

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Chapter 13: Security, Performance, and Production Readiness

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Web Application Security Fundamentals – OWASP Top Ten and How to Defend Against Each Threat

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Input Sanitization and Output Encoding – Preventing Injection Attacks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Authentication Security – Password Hashing, Token Management, and Session Security

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Performance Optimization – Bundle Size, Image Optimization, Caching Headers, and Database Indexes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Environment Configuration for Production – Secrets, Feature Flags, and Multi-Environment Setup

Preparing for Deployment – Linting, Type Checking, and Build Validation

Chapter 14: Deployment, CI/CD, and Operations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Choosing a Deployment Platform – Vercel, Railway, Self-Hosted Options Compared

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Deploying Your Next.js Application – Build Process, Environment Variables, and Database Connection

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Continuous Integration With GitHub Actions – Automated Testing on Every Commit

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Domains, HTTPS, and DNS Configuration – Making Your Application Accessible to the World

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Monitoring and Observability – Logs, Error Tracking, Performance Metrics, and Alerts

Production Troubleshooting – Diagnosing Issues When Things Go Wrong

Chapter 15: Advanced Claude Code Workflows and Patterns

Context Management in Large Projects – Keeping Claude Code Focused and Accurate

Custom Instructions and Project Conventions – Teaching Claude Code Your Standards

Advanced Prompting Techniques – Chain-of-Thought, Few-Shot Examples, and Role-Based Prompts

MCP Servers and External Tool Integration – Extending Claude Code With Specialized Capabilities

Multi-Agent Workflows and Subagents – Delegating Tasks Across Specialized AI Agents

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Refactoring Large Codebases With Confidence – Using Claude Code for Architectural Changes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Chapter 16: Scaling Architecture and Future Directions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

When to Scale – Signals That Your Architecture Needs to Evolve

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Monolith vs Microservices – Tradeoffs and Practical Guidance for Most Teams

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Caching Strategies – Redis, CDN Caching, and Database-Level Optimization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

Real-Time Features – WebSockets, Server-Sent Events, and Live Updates

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

AI-Assisted Development in Teams – Collaboration Patterns, Code Review, and Knowledge Sharing

The Future of Full-Stack Development With AI – Trends, Opportunities, and What Developers Should Learn Next

Conclusion: From Learning to Building

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.

References

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/full-stackdevelopmentwithclaudecode.