# Contents at a Glance

**Introduction** . . . . . 1

**Book 1: Welcome to the Rust Side** . . . . . 5

CHAPTER 1: Getting to Know Rust . . . . . 7
CHAPTER 2: Getting Your Hands Rusty . . . . . 17
CHAPTER 3: Speaking Rust: The Basics . . . . . 39
CHAPTER 4: Crafting Expressions . . . . . 61
CHAPTER 5: Controlling the Flow . . . . . 77
CHAPTER 6: Pattern Matching: The Swiss Army Knife of Rust . . . . . 93
CHAPTER 7: Functions: Teaching Rust New Tricks . . . . . 107

**Book 2: Learning the Rust Way of Doing Things** . . . . . 131

CHAPTER 1: Ownership: Rust's Secret Sauce . . . . . 133
CHAPTER 2: Structs: Rolling Your Own Types . . . . . 159
CHAPTER 3: Enums: Embracing Possibilities . . . . . 177
CHAPTER 4: Storing Data in Rust . . . . . 197
CHAPTER 5: The String Section: Text in Rust . . . . . 225
CHAPTER 6: Functional Programming with Closures and Iterators . . . . . 253
CHAPTER 7: Organizing Your Code . . . . . 279

**Book 3: Deeper Rust Concepts** . . . . . 303

CHAPTER 1: Traits: Shared Behaviors . . . . . 305
CHAPTER 2: Generics: Code That Works for Almost Anything . . . . . 325
CHAPTER 3: Lifetimes: How Long Things Live . . . . . 343
CHAPTER 4: Handling Errors Like a Pro . . . . . 361
CHAPTER 5: Macros: When Code Writes Code . . . . . 381
CHAPTER 6: Parallelism and Concurrency: Rust Multitasking . . . . . 395
CHAPTER 7: Testing, Debugging, and Documenting . . . . . 421

**Book 4: Building Command-Line Tools** . . . . . 447

CHAPTER 1: Coding Your First Tools . . . . . 449
CHAPTER 2: File Processing and Configuration . . . . . 465
CHAPTER 3: Polishing Your Command-Line Tools . . . . . 487

COPYRIGHTED MATERIAL

**Book 5: WebAssembly: Rust in Your Browser** . . . . . 503
CHAPTER 1: WebAssembly Basics and Setup . . . . . 505
CHAPTER 2: Exchanging Data between Rust and JavaScript . . . . . 525
CHAPTER 3: Building Interactive WebAssembly Apps . . . . . 545

**Book 6: Networking with Rust** . . . . . 569
CHAPTER 1: Building a Web Server . . . . . 571
CHAPTER 2: Building a REST API . . . . . 593
CHAPTER 3: Building a Server-Side Web App . . . . . 617

**Index** . . . . . 645

# Table of Contents


INTRODUCTION . . . . . 1
About This Book . . . . . 2
Foolish Assumptions . . . . . 2
Icons Used in This Book . . . . . 3
Beyond the Book . . . . . 3

BOOK 1: WELCOME TO THE RUST SIDE . . . . . 5

CHAPTER 1: Getting to Know Rust . . . . . 7
Programming: Making a Computer Do Your Bidding . . . . . 8
What is a programming language? . . . . . 8
The role of programming languages . . . . . 9
Understanding how code is written and executed . . . . . 11
Why the World Needs Rust . . . . . 12
What Makes Rust Different . . . . . 14
Rust: Not Just for Systems Nerds . . . . . 15
Setting Realistic Expectations . . . . . 15

CHAPTER 2: Getting Your Hands Rusty . . . . . 17
Getting to the Command Line . . . . . 18
Launching Terminal on Windows . . . . . 18
Launching Terminal on macOS . . . . . 18
Launching Terminal on Linux . . . . . 19
Some useful command-line shortcuts . . . . . 19
Basic command-line survival skills . . . . . 20
Installing Rust: Easier Done Than Said . . . . . 20
Installing Rust on Windows . . . . . 21
Installing Rust on macOS and Linux . . . . . 22
What happens during installation . . . . . 22
Verifying the installation . . . . . 22
Updating Rust . . . . . 23
Meet Cargo: Your New Coding Friend . . . . . 24
What Cargo can do . . . . . 25
Essential Cargo commands . . . . . 26
Setting Up Your Development Environment . . . . . 26
Creating Your First Rust Project . . . . . 28
Understanding the Project Structure . . . . . 29
Exploring the directory structure . . . . . 29
Checking out the src directory . . . . . 29
Examining the Cargo.toml file . . . . . 31
Other files and directories you might encounter . . . . . 31

Compiling and Running Your First Program . . . . . 32
Compiling and running in one fell swoop . . . . . 34
Compiling for release . . . . . 34
Compiling and running from VS Code . . . . . 35
Learning Rust: A Roadmap . . . . . 36

CHAPTER 3: **Speaking Rust: The Basics** . . . . . 39
main(): A Home for Your Code . . . . . 40
Variables: Storing Your Digital Stuff . . . . . 40
Make Mine a Mutable . . . . . 43
Annotating Your Code with Comments . . . . . 46
Outputting Text . . . . . 46
Talking about Types . . . . . 49
Scalar values . . . . . 49
String literals . . . . . 55
Type inference: The compiler knows all (most of the time) . . . . . 56
Constants: Values carved in stone . . . . . 56
Talking about type safety . . . . . 57
Working with the Standard Library . . . . . 57

CHAPTER 4: **Crafting Expressions** . . . . . 61
Understanding Expression Structure . . . . . 62
Building Numeric Expressions . . . . . 62
A quick look at the arithmetic operators . . . . . 62
What's going on with the remainder operator (%)? . . . . . 64
Using the arithmetic assignment operators . . . . . 65
Floating-point constants . . . . . 66
Floating-point methods . . . . . 67
Building Comparison Expressions . . . . . 68
Building Logical Expressions . . . . . 69
Understanding Operator Precedence . . . . . 70
The order of precedence . . . . . 70
Controlling the order of precedence . . . . . 72
Corralling Code into Expression Blocks . . . . . 74

CHAPTER 5: **Controlling the Flow** . . . . . 77
Making Decisions with if . . . . . 78
Branching with if. . .else . . . . . 79
Making multiple decisions . . . . . 80
Using if as an expression . . . . . 82
Handling Repetitive Code with Loops . . . . . 83
Looping with a basic loop . . . . . 84
Using loop as an expression . . . . . 85

Doing conditional repetition with while loops . . . . . 86
Iterating with for loops . . . . . 87
Bypassing loop statements using continue . . . . . 89
Nesting loops . . . . . 89
Labeling loops . . . . . 90

CHAPTER 6: **Pattern Matching: The Swiss Army Knife of Rust** . . . . . 93

Becoming a match Master . . . . . 94
Using match as an expression . . . . . 95
Matching a range of values . . . . . 96
Matching one value or another . . . . . 97
Matching with a condition . . . . . 97
Introducing Block Scope . . . . . 98
Project: Escape the Dungeon (a Tiny Text Adventure) . . . . . 101
Getting input from the user . . . . . 101
Setting up the game . . . . . 102
Introducing variable shadowing . . . . . 103
Pattern matching the user input . . . . . 104
Finishing up . . . . . 106

CHAPTER 7: **Functions: Teaching Rust New Tricks** . . . . . 107

What Is a Function? . . . . . 108
Why Functions Matter . . . . . 108
Writing Your First Function . . . . . 109
Arguments: How to Pass 'Em, How to Use 'Em . . . . . 111
Passing a single argument . . . . . 111
Passing multiple arguments . . . . . 112
Making an argument mutable . . . . . 113
Return Values: Bringing Data Back Alive . . . . . 114
The unit type return . . . . . 116
A practical return example . . . . . 116
Digging Deeper into Variable Scope . . . . . 117
Function scope . . . . . 117
Global scope . . . . . 119
Functions That Call Other Functions . . . . . 120
Separating responsibilities . . . . . 120
Using helper functions . . . . . 121
Getting a function to call itself . . . . . 122
Project: Temperature Converter . . . . . 124
Checking out the project's functions . . . . . 124
Putting everything together . . . . . 127

BOOK 2: LEARNING THE RUST WAY OF DOING THINGS . . . . . 131

CHAPTER 1: **Ownership: Rust's Secret Sauce** . . . . . 133
Understanding Ownership without Losing Your Mind . . . . . 134
The three rules of ownership club . . . . . 136
Copying owned values . . . . . 137
Introducing the String type . . . . . 138
Moving: How Rust Transfers Ownership . . . . . 140
Cloning: Deep-copying heap data . . . . . 141
Passing values into functions moves them . . . . . 142
Borrowing: Sharing Your Toys . . . . . 143
Borrowing and function calls . . . . . 145
Mutable borrowing: Looking *and* touching . . . . . 146
The golden rule of borrowing . . . . . 146
The golden rule promotes memory safety . . . . . 148
Working around the golden rule . . . . . 149
References: Pointing to Things Safely . . . . . 149
Dereferencing: Getting at the referenced value . . . . . 150
References and the println! macro . . . . . 152
Reference lifetimes . . . . . 153
Move? Borrow? Clone? Which One Should You Use? . . . . . 153
Project: Creating a Text Analyzer . . . . . 154

CHAPTER 2: **Structs: Rolling Your Own Types** . . . . . 159
You Need More Struct in Your Life . . . . . 160
Meet the Struct . . . . . 160
Creating an instance of a struct . . . . . 161
Getting at the data . . . . . 162
Making it mutable . . . . . 163
A Couple of Useful Struct Shortcuts . . . . . 164
The field init shorthand . . . . . 164
The struct update syntax . . . . . 165
Methods: Teaching Structs New Behaviors . . . . . 166
Implementing Associated Functions . . . . . 168
Project: Building a Simple Bank Account Manager . . . . . 170
Setting up the struct . . . . . 171
Implementing the struct methods . . . . . 171
Adding the user input functions . . . . . 173
Creating an instance of the struct . . . . . 173
Implementing the main menu loop . . . . . 175

CHAPTER 3: **Enums: Embracing Possibilities** ..... 177
Meet the Enum ..... 178
Creating an instance of an enum ..... 178
Pattern matching enum states ..... 179
Adding data to an enum ..... 180
Option: It's Always Something (or Is It?) ..... 181
Handling only Some with if let ..... 184
Assigning a default value to None ..... 185
Returning an Option in your own functions ..... 186
Result: Dealing with Errors Gracefully ..... 187
Assigning a default value to Err ..... 189
Returning a Result in your own functions ..... 190
Handling only Ok with if let ..... 192
Project: Modeling an Authentication Flow ..... 193

CHAPTER 4: **Storing Data in Rust** ..... 197
Tuples: Groups of Things ..... 198
Why tuples are useful ..... 199
Tuple destructuring ..... 200
Tuple limitations ..... 200
Arrays: Fixed-Sized Storage ..... 201
Making an array ..... 201
Getting an item from an array ..... 202
Array bounds checking: Rust has your back ..... 203
Iterating through an array ..... 203
Array types and functions ..... 204
When to use an array (and when not to) ..... 206
Slices: Versions of Your Stuff ..... 206
Creating a slice ..... 207
Modifying data with mutable slices ..... 208
Dereferencing with a mutable slice ..... 209
Using slices to make functions more flexible ..... 210
Vectors: Dynamic Arrays That Grow ..... 211
Creating a vector ..... 211
Accessing vector elements (a story about risk) ..... 213
Adding and removing elements ..... 214
Iterating through a vector ..... 216
Hash Maps: Key-Value Storage Made Easy ..... 217
Creating a hash map ..... 217
Accessing hash map values ..... 219
Adding and removing key-value pairs ..... 220
Iterating through a hash map ..... 221
Project: The Prime Explorer ..... 222

CHAPTER 5: **The String Section: Text in Rust** . . . . . 225
String Instruments: String and &str . . . . . 226
Text for the whole world: Rust and UTF-8 . . . . . 226
Text types in Rust: Why two? . . . . . 226
When to use which . . . . . 227
Making String Music: Creating Strings . . . . . 228
Creating string literals . . . . . 228
Creating Strings . . . . . 230
Converting String to &str . . . . . 231
String Works: The Greatest Hits . . . . . 231
Getting info about a string . . . . . 232
Adding to a string . . . . . 233
Concatenating a String and a string slice . . . . . 234
Replacing text in a string . . . . . 236
Clearing a string . . . . . 236
String Pieces: Slicing Strings . . . . . 237
Slicing a string with the range syntax . . . . . 237
Safer slicing with get() . . . . . 238
Understanding deref coercion . . . . . 239
String Movements: Common String Patterns . . . . . 240
Trimming whitespace from a string . . . . . 240
Searching and testing a string . . . . . 242
Converting case . . . . . 243
Splitting a string . . . . . 244
Project: Slugifying a String . . . . . 246

CHAPTER 6: **Functional Programming with Closures and Iterators** . . . . . 253
What on Earth Is Functional Programming? . . . . . 254
Closures: Anonymous Functions with Attitude . . . . . 254
Meet the closure . . . . . 255
Capturing local variables . . . . . 256
Sorting with a closure . . . . . 257
Different ways to write closures . . . . . 259
Meet the Iterator Pattern . . . . . 260
Creating an Iterator from a Collection . . . . . 261
Borrowing immutably with iter() . . . . . 261
Borrowing mutably with iter_mut() . . . . . 262
Owning with into_iter() . . . . . 263
Consuming Iterators . . . . . 264
Consuming methods that perform calculations . . . . . 265
find(): Finding the first item that matches . . . . . 266
collect(): The ultimate consumer . . . . . 268
Adapting Iterators . . . . . 269

filter(): Keeping only what you want . . . . . 269
map(): Transforming each item . . . . . 270
enumerate(): Adding index numbers . . . . . 270
Project: Word Wrangler . . . . . 271
Tokenizing the text . . . . . 272
Counting word frequencies . . . . . 274
Getting the most frequently used words . . . . . 275
Finding the longest word . . . . . 275
Putting it all together . . . . . 276

CHAPTER 7: **Organizing Your Code** . . . . . 279
How Rust Treats Code Organization . . . . . 280
The big picture: Packages, crates, and modules . . . . . 280
The module tree: Your code's family tree . . . . . 280
Files and modules are not necessarily the same . . . . . 281
Starting Small: Organizing Code in One File . . . . . 282
The main.rs versus lib.rs decision . . . . . 282
Creating modules in main.rs . . . . . 283
Nesting one module inside another . . . . . 284
Growing Up: Moving to Multiple Files . . . . . 287
Using main.rs and lib.rs . . . . . 287
Mapping rules for module files . . . . . 287
Creating your own module files . . . . . 288
Controlling Access: Public versus Private . . . . . 290
Understanding private by default . . . . . 290
Exposing only what's necessary when designing module interfaces . . . . . 292
Importing and Using Code . . . . . 293
Importing code with the use keyword . . . . . 293
Shortening the module paths . . . . . 294
Working with External Dependencies . . . . . 296
Adding a crate as a project dependency . . . . . 296
Importing the dependency . . . . . 297
Real-World Organization Patterns . . . . . 298
Organizing larger applications . . . . . 298
Re-exports and crate interfaces . . . . . 300
Splitting a project into multiple crates . . . . . 301

**BOOK 3: DEEPER RUST CONCEPTS** . . . . . 303

CHAPTER 1: **Traits: Shared Behaviors** . . . . . 305
Traits Are Promises . . . . . 305
When Your Types Won't Behave . . . . . 306
When your type won't display . . . . . 306
When your type won't compare . . . . . 307
When your type won't clone . . . . . 308

Deriving: Traits in the Fast Lane . . . . . 309
Deriving Debug for display . . . . . 310
Deriving PartialEq for comparison . . . . . 311
Deriving Clone for copying . . . . . 312
Behave Yourself: Implementing Traits for Your Own Types . . . . . 313
Printing Prettier with the Display Trait . . . . . 316
Setting Sensible Defaults with the Default Trait . . . . . 318
Project: Building a Media Library with Traits . . . . . 320
The traits . . . . . 320
The media . . . . . 321
Implementing the traits . . . . . 322

CHAPTER 2: **Generics: Code That Works for Almost Anything** . . . . . 325
Why Write the Same Code Twice? . . . . . 326
Introducing Generics . . . . . 327
Making a function generic . . . . . 328
Making a struct generic . . . . . 329
Making an enum generic . . . . . 331
Specifying multiple generic type parameters . . . . . 332
Type Bounds and Constraints . . . . . 334
Adding a trait bound to a generic parameter . . . . . 336
Working with multiple trait bounds . . . . . 336
Making complex constraints more readable . . . . . 337
Traits and Generics = Chef's Kiss . . . . . 338
Project: Build a Generic Logger That Works with Any Type . . . . . 340

CHAPTER 3: **Lifetimes: How Long Things Live** . . . . . 343
Introducing Lifetimes . . . . . 344
When Rust Demands Lifetime Annotations . . . . . 346
Adding Lifetime Annotations to Functions . . . . . 347
A first look at lifetime annotations . . . . . 348
Annotating a function with a lifetime . . . . . 348
Typing a lifetime to a single input . . . . . 350
Adding multiple lifetime annotations . . . . . 350
Lifetimes don't solve everything . . . . . 351
Lifetimes in Structs: Keeping References Alive . . . . . 352
Lifetime Elision: When the Compiler Does the Work . . . . . 354
Rule 1: Every input reference gets its own lifetime parameter . . . . . 354
Rule 2: If there's just one input reference, any output reference gets the same lifetime . . . . . 354
Rule 3: If &self is an input, its lifetime is used for the output . . . . . 355
The 'static Lifetime: Data That Lives (Sort of) Forever . . . . . 356
Project: A Zero-Copy Config Parser . . . . . 357

CHAPTER 4: **Handling Errors Like a Pro** . . . . . 361
Recoverable versus Unrecoverable: Know When to Panic! . . . . . 362
Recoverable errors: Handle gracefully mode . . . . . 362
Unrecoverable errors: Panic mode . . . . . 362
Result<T, E>: Your Go-to Type for Errors . . . . . 364
Result: A review . . . . . 364
Pattern matching on Result . . . . . 365
Saving Time with Helper Methods . . . . . 366
Living dangerously: unwrap() and expect() . . . . . 366
Living alternatively I: unwrap_or() . . . . . 367
Living alternatively II: unwrap_or_else() . . . . . 369
Living sensibly: unwrap_or_default() . . . . . 370
Propagating Errors with the ? Operator . . . . . 372
Grokking Rust's Error Types . . . . . 373
Magically Creating a Result from an Option . . . . . 374
Converting None with ok_or() . . . . . 375
Lazily converting with ok_or_else() . . . . . 376
Creating Your Own Error Types . . . . . 377

CHAPTER 5: **Macros: When Code Writes Code** . . . . . 381
Useful Macro Magic: println! and vec! . . . . . 382
Using Macros in Your Own Code . . . . . 383
Macro Syntax: Learning to Read the Hieroglyphics . . . . . 384
Writing Macros . . . . . 387
Example 1: Adding any number of values . . . . . 389
Example 2: Converting a list of key-value pairs to a vector . . . . . 390
Project: Building a Timer Macro . . . . . 391

CHAPTER 6: **Parallelism and Concurrency: Rust Multitasking** . . . . . 395
Parallelism versus Concurrency: What's the Difference? . . . . . 396
Spawning Threads: Your First Parallel Adventure . . . . . 397
Creating and joining a thread . . . . . 397
Moving data into threads . . . . . 400
Passing Messages through Channels . . . . . 403
Passing messages between a sender and a receiver . . . . . 404
Multiple producers, single consumer . . . . . 406
Sharing State Safely with Arc and Mutex . . . . . 407
Sharing ownership with Arc . . . . . 407
Modifying shared data safely with Mutex . . . . . 410
Borrowing Data with Scoped Threads . . . . . 411
Iterating in Parallel with Rayon . . . . . 413
Async/Await: Concurrency without the Thread Overhead . . . . . 414
Project: Collatz Conjecture Explorer . . . . . 416

CHAPTER 7: **Testing, Debugging, and Documenting** . . . . . 421
Testing Your Code . . . . . 422
Writing your first tests . . . . . 423
Meeting the rest of the assertion family . . . . . 427
Testing functions that return Result . . . . . 427
Testing panics . . . . . 428
Creating a module for your tests . . . . . 429
Creating a library automatically includes a test module . . . . . 431
Integration tests: Testing how things work together . . . . . 432
Debugging Your Code . . . . . 433
The dbg! macro: The Watson to your Holmes . . . . . 434
Debugging in VS Code . . . . . 435
Using println! debugging . . . . . 438
Cleaning Your Code . . . . . 439
rustfmt: Your automatic code cleaner-upper . . . . . 439
Clippy: Your friendly code reviewer . . . . . 441
Documenting Your Code . . . . . 443
Adding documentation comments . . . . . 443
Running documentation examples as tests . . . . . 445
Building your program's docs . . . . . 446

**BOOK 4: BUILDING COMMAND-LINE TOOLS** . . . . . 447

CHAPTER 1: **Coding Your First Tools** . . . . . 449
Understanding What Makes a Good CLI Tool . . . . . 450
Touring a Typical CLI . . . . . 450
Parsing Command-Line Arguments . . . . . 451
Getting the arguments . . . . . 451
Parsing arguments manually . . . . . 452
Parsing arguments idiomatically . . . . . 455
Reading from Standard Input . . . . . 457
Reading buffered input . . . . . 458
Reading redirected input . . . . . 458
Basic File Operations for CLI Tools . . . . . 459
Project: A Word Replacer Tool . . . . . 461

CHAPTER 2: **File Processing and Configuration** . . . . . 465
Learning File System Foundations . . . . . 466
Manipulating paths without breaking a sweat . . . . . 466
Fiddling with the filesystem . . . . . 470
Reading and Writing Files Safely . . . . . 471
Reading files in one fell swoop . . . . . 472
Reading files incrementally . . . . . 472
Writing files safely . . . . . 474

Appending text to a file . . . . . 475
Reading and writing CSV files . . . . . 477
Serializing and Deserializing Data . . . . . 480
Project: Creating a File Organizer . . . . . 482

CHAPTER 3: **Polishing Your Command-Line Tools** . . . . . 487
Organizing Your Tools with Subcommands . . . . . 488
Adding subcommands . . . . . 488
Working with nested subcommands . . . . . 491
Fancifying Output with Colors . . . . . 493
Implementing Progress Bars . . . . . 494
Project: Building a Multicommand File Organizer . . . . . 496
Building the command structure . . . . . 498
Implementing main() . . . . . 498
Implementing the scan command . . . . . 499
Implementing the archive command . . . . . 500
Implementing the stats command . . . . . 501

**BOOK 5: WEBASSEMBLY: RUST IN YOUR BROWSER** . . . . . 503

CHAPTER 1: **WebAssembly Basics and Setup** . . . . . 505
Web *What* Now? Understanding WebAssembly . . . . . 506
Why Rust and WebAssembly Are Best Friends . . . . . 507
Setting Up Your Rust-to-WASM Toolchain . . . . . 508
Installing the wasm32 target . . . . . 508
Installing wasm-pack . . . . . 509
Installing Node.js (optional but recommended) . . . . . 509
Creating a test project . . . . . 510
Running Your First Rust Function in the Browser . . . . . 510
Getting your project off the ground . . . . . 511
Building your project . . . . . 512
Creating a web page . . . . . 512
Spinning up a local server for your page . . . . . 513
What just happened? . . . . . 514
Running JavaScript Functions in Rust . . . . . 515
Common WASM Build Commands . . . . . 519
Project: Building a Rust-Powered Password Analyzer . . . . . 520

CHAPTER 2: **Exchanging Data between Rust and JavaScript** . . . . . 525
Understanding the wasm-bindgen Bridge . . . . . 526
Exchanging Data through the Wall . . . . . 527
Swapping Strings . . . . . 527
Passing Arrays, Vectors, and Other List-Like Things . . . . . 529

Passing numeric vectors and arrays . . . . . 530
Passing string vectors and arrays . . . . . 533
Passing structs and objects . . . . . 535
Project: Building a Remote Markdown Renderer . . . . . 538

CHAPTER 3: **Building Interactive WebAssembly Apps** . . . . . 545
Controlling DOM from Rust . . . . . 545
Downcasting elements . . . . . 549
Working with DOM . . . . . 550
Referencing DOM elements . . . . . 552
Handling Browser Events . . . . . 553
The event listener pattern . . . . . 553
Handling click events . . . . . 554
Getting event information . . . . . 556
Handling keyboard events . . . . . 557
Managing Application State . . . . . 559
Project: Populating DOM from Rust . . . . . 563

**BOOK 6: NETWORKING WITH RUST** . . . . . 569

CHAPTER 1: **Building a Web Server** . . . . . 571
Understanding HTTP from the Ground Up . . . . . 572
What is HTTP, anyway? . . . . . 572
The anatomy of an HTTP request . . . . . 572
The anatomy of an HTTP response . . . . . 574
What you're building in this chapter . . . . . 575
Getting a Minimal Web Server Off the Ground . . . . . 576
Starting the project . . . . . 576
Building the cutest little web server ever . . . . . 576
Testing your server . . . . . 578
Implementing Routing and Handler Functions . . . . . 580
How routing works in Axum . . . . . 580
Writing handler functions . . . . . 580
Returning HTML responses . . . . . 581
Reading HTML from files . . . . . 582
Reading URL query strings . . . . . 585
Organizing Your Web Server Code . . . . . 586
Serving Static Files . . . . . 589
Logging and Debugging . . . . . 591

CHAPTER 2: **Building a REST API** . . . . . 593
What's an API? . . . . . 594
Okay, So What's a REST API? . . . . . 594
How REST APIs differ from HTML web servers . . . . . 595
Why JSON instead of HTML . . . . . 595

From HTML to JSON: Changing Your Mindset . . . . . 597
Getting Started on the API . . . . . 597
POST, PUT, DELETE: The REST of the HTTP Verbs . . . . . 600
Building a Resource: The Book Library API . . . . . 600
Defining the Book struct . . . . . 601
Setting up in-memory storage . . . . . 602
Testing your API . . . . . 603
Implementing CRUD Operations . . . . . 604
GET a single book with path parameters . . . . . 604
PUT to update a book . . . . . 605
DELETE a book . . . . . 606
Validating Data and Handling Errors . . . . . 607
Migrating to a Real Database with SQLx . . . . . 609
Setting up SQLite . . . . . 609
Updating your code for SQLx . . . . . 610
Testing Your API . . . . . 614
Deployment Considerations . . . . . 615

CHAPTER 3: **Building a Server-Side Web App** . . . . . 617
Setting Up Your Web App Foundation . . . . . 618
Creating the project structure . . . . . 618
Database design for LinkLocker . . . . . 619
Initializing the database . . . . . 620
Templates: HTML with Superpowers . . . . . 621
Meet Askama: Rust-flavored templates . . . . . 621
Creating your first template . . . . . 622
Rendering templates from routes . . . . . 624
Serving Static Files (CSS and Friends) . . . . . 625
Connecting to the Database . . . . . 626
Setting up the database pool . . . . . 626
Authentication: Who Are You? . . . . . 628
Setting up session management . . . . . 629
User registration . . . . . 630
User login . . . . . 634
User logout . . . . . 636
Updating the home page with login status . . . . . 637
Managing Bookmarks . . . . . 637
Displaying bookmarks . . . . . 637
Adding bookmarks . . . . . 639
Editing bookmarks . . . . . 641
Deleting bookmarks . . . . . 643
Extending the App . . . . . 643

**INDEX** . . . . . 645