

MODERN WEB DEVELOPMENT WITH **RUST** AND **AXUM**

BUILDING PRODUCTION-GRADE APIS AND SERVICES



A PRACTICAL, HANDS-ON GUIDE WITH A RUNNING EXAMPLE:
TASKFLOW – A TASK MANAGEMENT API



AUTHENTICATION
Secure your API with
modern auth patterns



DATABASE INTEGRATION
Work with SQLx and
migrations



REAL-TIME FEATURES
WebSockets and Server-
Sent Events



OBSERVABILITY
Logging, metrics and
tracing in production



DEPLOYMENT
Containerize and deploy
with confidence



AXUM
0.8.x



TOKIO
1.x



RUST
EDITION 2021

★
COMPLETE
EXAMPLES
COMPILABLE
EXPLAINED
IN CONTEXT

EDWARD PARKER

Modern Web Development with Rust and Axum

Building Production-Grade APIs and Services

Steve Publications

This book is available at

<https://leanpub.com/modernwebdevelopmentwithrustandaxum>

This version was published on 2026-08-28



This is a [Leanpub](#) book. Leanpub empowers authors and publishers with the Lean Publishing process. [Lean Publishing](#) is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

© 2026 Steve Publications

Contents

Building Production-Grade APIs and Services	1
Introduction	2
What You Will Learn	2
The Running Example: Taskflow API	3
How This Book Is Structured	4
Prerequisites and Versions	4
How to Read This Book	5
Chapter 1: Why Rust for Web Development	7
The Case for Rust in Web Services	7
Landscape of Rust Web Frameworks	11
Why Axum Stands Out	13
Introducing Our Running Example: A Task Management API	16
How to Read This Book	17
Summary	19
Chapter 2: Project Setup and Architecture	21
Rust Toolchain and Edition Selection	21
Cargo Configuration and Dependency Management	22
Directory Layout for Scalable Projects	25
Our First Axum Server: Hello World with Structure	33
Environment Configuration Strategy	37
Summary	38
Chapter 3: Tokio and Asynchronous Rust	40
How async/await Works in Rust	40
The Tokio Runtime: Multithreaded vs Current Thread	41
Tasks, Spawning and Joining	41
Channels and Concurrent Communication	42
Timeouts, Sleeps and Scheduling Control	43
Summary	44

Chapter 4: Tower, Hyper and Axum’s Architecture 45

 The Tower Service Trait: Composable Asynchronous Services 45

 Middleware as Service Layers 45

 Hyper: The Low-Level HTTP Engine 46

 How Axum Fits Into the Stack 47

 Understanding the Type-Driven Design Philosophy 47

 Summary 48

Chapter 5: Routing and Extractors 49

 Defining Routes with the Router Combinator 49

 Path Parameters and Their Types 49

 Query Parameters and Structured Extraction 50

 Application State and the State Extractor 51

 Nested Routers and Modular Route Organization 52

 Summary 52

Chapter 6: Request and Response Handling 53

 Response Types and Status Codes 53

 JSON Serialization with `serde_json` 53

 Form Data and URL-Encoded Bodies 54

 Headers: Reading and Writing 55

 Cookies and Session Identifiers 55

 Content Negotiation Patterns 56

 Summary 56

Chapter 7: Error Handling and Validation 57

 Designing Application Error Types 57

 The `IntoResponse` Trait for Error Conversion 57

 Centralized Error Handlers 58

 Input Validation with Validator Crates 58

 Separating Internal Errors from Client-Facing Messages 59

 Summary 59

Chapter 8: Authentication and Authorization 60

 Password Hashing with `Argon2` 60

 Session-Based Authentication 61

 JWT Authentication Flow 61

 Extractors for Authenticated Requests 62

 Role-Based Authorization Patterns 63

 Summary 63

Chapter 9: Web Security Practices 64

 Cross-Origin Resource Sharing (CORS) Configuration 64

 Security Headers and Helmet-Like Middleware 64

 CSRF Protection Strategies 65

 Rate Limiting and Abuse Prevention 66

 Secure Configuration and Secrets Management 67

 Summary 67

Chapter 10: Database Integration with SQLx 68

 Choosing a Database Driver 68

 Connection Pooling with SQLx Pools 68

 Running Queries and Mapping Results 69

 Database Migrations with sqlx-migrate 70

 Transactions and Atomic Operations 71

 Summary 71

Chapter 11: Repository and Service Patterns 72

 The Repository Pattern in Rust 72

 Service Layer Design 72

 Dependency Injection Through Application State 73

 Domain Modeling and Type Safety 73

 Structuring Large Applications 73

 Summary 74

Chapter 12: Real-Time Communication with WebSockets 75

 WebSocket Upgrade Flow 75

 Handling Bidirectional Messages 75

 Connection State and Broadcast Patterns 76

 Scaling WebSocket Applications 76

 Server-Sent Events as an Alternative 77

 Summary 77

Chapter 13: File Handling and Static Assets 78

 Serving Static Files Efficiently 78

 File Uploads with Multipart Forms 79

 Streaming Large Responses 79

 File Validation and Security Considerations 80

 Summary 81

Chapter 14: RESTful API Design and Documentation 82

CONTENTS

REST Principles in Practice	82
Pagination, Filtering and Sorting Patterns	82
API Versioning Strategies	83
Generating OpenAPI/Swagger Documentation	84
Request/Response Schema Design	85
Summary	85
Chapter 15: Testing Strategies	86
Unit Testing Handlers and Services	86
Integration Testing with axum_test or request	86
Database Testing Strategies	87
Mocking External Dependencies	87
Test Organization for Large Projects	88
Summary	88
Chapter 16: Observability and Performance	89
Structured Logging with Tracing	89
Distributed Tracing in Microservices	89
Metrics Collection and Export	90
Profiling and Identifying Bottlenecks	91
Caching Strategies: In-Memory and Distributed	91
Summary	92
Chapter 17: Deployment and Operations	93
Building Optimized Docker Images	93
Reverse Proxies: Nginx and Caddy	93
TLS Termination and Certificate Management	94
Graceful Shutdown and Health Checks	94
CI/CD Pipelines for Rust Services	95
Production Monitoring and Alerting	95
Summary	96
Conclusion: The Path Forward	97
What Makes This Stack Special	97
Key Principles to Remember	97
Where to Go From Here	97
Continuing Your Rust Journey	97
Final Thoughts	97
References	98

Official Documentation	98
Axum Repository and Examples	98
Axum Announcements and Blog Posts	98
Rust Web Framework Comparisons	98
Authentication and Security	98
Database and SQLx	98
Testing	99
OpenAPI and Documentation	99
Observability	99
Deployment and Docker	99
Tokio Runtime	99
Additional Resources	99
Community and Learning	99

Building Production-Grade APIs and Services

This book shows you how to design, build, test, deploy and operate production-ready web applications with Rust and the Axum framework. Using a task management API called Taskflow as a running example, you'll start with the fundamentals and gradually work your way into more advanced topics such as authentication, database integration, real-time communication, observability and containerized deployment. The examples are explained in context so you can follow along and understand how each part fits into the application. The book assumes you have some basic knowledge of Rust syntax, while introducing the web development and async concepts you need along the way. It targets Axum 0.8.x, Tokio 1.x and Rust edition 2021.

Introduction

There is a moment in every web developer's career when they realize that the frameworks they love are also the ones constraining them. Memory leaks in long-running services. Runtime panics caused by null references that should have been caught earlier. Performance cliffs that appear only under production load. Race conditions that manifest once per week and take hours to reproduce.

Rust was designed to eliminate these problems at their source, and Axum brings Rust's guarantees to web development without sacrificing developer experience. If you are reading this book, you probably already know something about Rust - perhaps you have written command-line tools or libraries and are curious about web services, or maybe you are coming from another language looking for a more reliable foundation for your backend systems.

This book will take you on a deliberate journey through the entire lifecycle of building production-grade web APIs with Axum, from your first route to deploying an observable, containerized service that can handle real traffic. We will not skim over details or hand-wave away complexity. Every concept will be explained thoroughly, every code example will be complete and compilable, and every architectural decision will be justified with reasoning you can apply to your own projects.

What You Will Learn

By the end of this book, you will be able to:

- Set up a properly structured Axum project with modern tooling and dependency management
- Understand how async Rust and the Tokio runtime work under the hood, not just how to use them
- Explain how Axum fits into the broader ecosystem of Tower services and Hyper HTTP handling
- Design type-safe routes and handlers using Axum's extractor pattern

- Implement robust error handling that separates internal failures from client-facing responses
- Build secure authentication and authorization systems using JWTs, sessions and password hashing
- Integrate relational databases with SQLx, including connection pooling, migrations and transactions
- Apply clean architecture patterns like repositories and service layers to keep your code maintainable
- Handle real-time communication with WebSockets for live updates and notifications
- Design RESTful APIs with proper pagination, filtering, sorting and OpenAPI documentation
- Write comprehensive tests - unit, integration and end-to-end - that give you confidence in your code
- Implement structured logging, distributed tracing, and metrics for production observability
- Optimize performance through caching, rate limiting, and concurrency patterns
- Deploy your services in optimized Docker containers behind reverse proxies with TLS termination
- Set up CI/CD pipelines and production monitoring for reliable operations

The Running Example: Taskflow API

Throughout this book we will build a single application called Taskflow - a task and project management API that gradually grows in complexity. Starting with basic CRUD operations for tasks, we will add user authentication, database persistence, real-time updates via WebSockets, file attachments, rate limiting, caching and full production readiness.

Using a running example serves several purposes. First, it provides continuity - you can see how decisions made early in the book affect later chapters. Second, it gives you a concrete reference implementation that you can study, modify and use as a starting point for your own projects. Third, it demonstrates how various patterns and techniques fit together in a realistic application rather than isolated snippets.

You do not need to follow along with every line of code if you already understand the concepts. The running example is meant to complement

focused examples that illustrate individual topics, not replace them. Each chapter balances both approaches.

How This Book Is Structured

The book is organized into seven parts that build progressively on each other:

Part I: Foundations establishes why Rust and Axum are compelling choices for web development and gets you from zero to a working project structure with proper tooling.

Part II: The Async Foundation explains Tokio and asynchronous Rust in depth, then demystifies the Tower and Hyper layers that Axum sits on top of. You cannot write good Axum code without understanding this foundation.

Part III: Building Web Applications covers the core mechanics - routing, extractors, request and response handling, error handling, and validation. This is where you learn to build actual functionality.

Part IV: Security and Authentication addresses authentication mechanisms, authorization patterns, and web security practices including CORS, CSRF protection, rate limiting, and secure configuration.

Part V: Data Persistence integrates relational databases using SQLx - connection pooling, migrations, type-safe queries, transactions and the repository/service patterns that keep data access clean and testable.

Part VI: Advanced Features tackles WebSockets for real-time communication, file handling, RESTful API design principles, pagination, filtering and OpenAPI documentation generation.

Part VII: Production Readiness brings everything together with testing strategies, observability through logging and tracing, performance optimization, containerization, reverse proxies, TLS, CI/CD pipelines, and production operations.

Each chapter is self-contained enough that you can return to it later as a reference, but the progression matters. Later chapters assume understanding of earlier material, so reading in order is recommended.

Prerequisites and Versions

This book assumes you are comfortable with basic Rust syntax - structs, enums, traits, generics, ownership, borrowing and lifetimes at a working level. If you have read The Rust Programming Language book or built a few Rust projects, you are ready. We will briefly review relevant concepts when they arise but will not teach Rust from scratch.

Some familiarity with web development concepts is helpful - HTTP methods, status codes, JSON, REST principles - but we will explain these along the way where needed.

The code examples target the following versions:

- **Rust edition:** 2021 (the current stable standard; all examples compile with Rust 1.75+)
- **Axum:** 0.8.x (the latest stable series as of this writing)
- **Tokio:** 1.x with rt-multi-thread feature
- **Tower:** 0.5.x
- **SQLx:** 0.9.x for database integration
- **tower-http:** 0.7.x for middleware layers

Rust web frameworks move quickly, and APIs can change between versions. We will note version-sensitive details where they matter and point you to the official documentation for the most current information. The design principles and patterns covered here are stable even if specific function signatures evolve.

How to Read This Book

Read this book actively. Type out the code examples rather than copying them - you will learn more through the friction of getting things right yourself. Experiment with variations: change a route, modify an extractor, break something on purpose and read the compiler errors until you understand why it failed.

When you encounter Rust concepts that are unfamiliar - async traits, Pin, Send + Sync bounds, lifetimes in generics - pause and study them. The compiler messages are your friend; they are often more precise than any explanation you will find elsewhere. Use rust-analyzer in your editor, read the standard library documentation, and come back to this book once the fundamentals click.

The code examples are organized with clear file paths when multiple files are involved. Every dependency needed is specified in Cargo.toml snippets. Database schemas and migrations are included where relevant. Commands for running tests, applying migrations, or building Docker images are shown explicitly. Nothing is left for you to guess.

Some chapters contain architectural diagrams rendered as ASCII art. These are meant to clarify relationships between components - services, layers, handlers, databases - not to be exhaustive technical drawings. Focus on the patterns they illustrate rather than exact layout.

Finally, do not feel pressured to implement every pattern in every project. The goal is to give you a toolkit of proven approaches so you can make informed decisions about what fits your context. A small internal service does not need the same observability infrastructure as a public API serving millions of requests per day. Understand the full spectrum so you can choose appropriately.

Let us begin.

Chapter 1: Why Rust for Web Development

Before writing a single line of Axum code, it is worth understanding why we are choosing this stack in the first place. The decision to build web services in Rust is not trivial - it involves trade-offs in developer experience, ecosystem maturity, and operational characteristics that deserve honest examination. This chapter establishes the motivation for using Rust and Axum, surveys the landscape of Rust web frameworks, explains what sets Axum apart, introduces our running example application, and outlines how to approach this book.

The Case for Rust in Web Services

Rust has gained a reputation as the language that lets you sleep at night. That is not marketing hyperbole; it is the consistent experience of developers who have shipped Rust services to production and compared them with services written in other languages. The reasons are concrete.

Memory Safety Without Garbage Collection

Most mainstream web development languages - Java, Go, Python, JavaScript, Ruby - rely on garbage collection for memory management. GC provides convenience but introduces nondeterministic pauses that can spike latency under load. For many applications this is acceptable; for latency-sensitive services it is not.

C and C++ give you manual control over memory but make it trivially easy to introduce use-after-free bugs, buffer overflows, data races, and dangling pointers - vulnerabilities that are routinely exploited in production systems. The Linux kernel alone has had thousands of such bugs fixed over the years.

Rust's ownership system eliminates these categories of bug at compile time without requiring garbage collection. When you write Rust code, the compiler tracks every value's lifetime and enforces rules that prevent multiple mutable

references to the same data, use-after-free, and data races. This is not a linter or a best-effort static analyzer; it is a mathematical guarantee enforced by the type system.

For web services this means:

- No GC pauses affecting your p99 latency
- No memory leaks from forgotten deallocations
- No buffer overflows exploitable via crafted HTTP requests
- No data races in concurrent request handling that cause intermittent crashes

These are not theoretical benefits. Companies running Rust services at scale - Cloudflare, Discord, AWS, Dropbox, Figma, Stripe - have reported dramatic reductions in production incidents related to memory safety and concurrency after migrating from other languages.

Performance That Matters for Cost

Rust compiles to native machine code with zero runtime overhead beyond what you explicitly write. There is no virtual machine to start up, no just-in-time compilation warmup phase, no garbage collector consuming CPU cycles during your request processing.

In practice this translates to:

- Higher requests per second per CPU core compared to managed languages
- Lower memory footprint, allowing more instances in the same infrastructure budget
- Faster cold starts for serverless and containerized deployments
- Predictable performance characteristics that do not degrade under sustained load

Benchmarks tell a clear story. In TechEmpower's framework benchmarks - widely regarded as one of the most comprehensive comparisons of web framework performance - Rust frameworks consistently rank at or near the top, often handling 2-5x more requests per second than Go equivalents and 10-50x more than Python or Node.js frameworks for equivalent workloads.

The cost implications are real. If your service handles millions of requests per day, a language that uses half the CPU and memory can reduce your infrastructure bill by thousands of dollars per month. Rust's performance is not just impressive in benchmarks; it compounds into operational savings at scale.

The Compiler as Your Safety Net

Rust's compiler is famous - or infamous, depending on your experience level - for being strict. It will reject code that JavaScript would happily run until it crashes three minutes later in production. This strictness is a feature, not a bug.

Consider what happens when you refactor a large codebase in most languages: you change a function signature, and everything that called it continues to compile because dynamic typing or weak static checking does not catch the mismatch. The bug manifests at runtime when someone calls that function with the wrong arguments.

In Rust, changing a function's input type causes every call site to fail compilation until you update them all. The compiler tells you exactly where the problems are and what types it expected versus what you provided. This makes refactoring large codebases dramatically safer and faster - you get immediate feedback about broken code paths instead of discovering them through production incidents or brittle test suites.

For web services that grow over years, this characteristic is invaluable. The codebase stays consistent with itself because the compiler enforces it.

Concurrency Without Fear

Web services are inherently concurrent - they handle many requests simultaneously. Managing concurrency correctly is one of the hardest problems in software engineering. Data races, deadlocks, priority inversion, livelock - these bugs are notoriously difficult to reproduce and debug because they depend on precise timing between threads.

Rust's ownership model extends to concurrency through the Send and Sync traits. The compiler guarantees that:

- Only data types safe to send between threads can be sent (Send)

- Only data types safe to share via references can be shared (Sync)
- No two mutable references to the same data can exist simultaneously, even across threads

This eliminates entire classes of concurrency bugs at compile time. You cannot accidentally share a non-thread-safe object between request handlers and cause a data race that crashes your service under load. The compiler catches it before you ship.

Combined with Tokio's async runtime - which we will explore in depth later - Rust provides a concurrency model that is both powerful and safe, enabling high-throughput web services without the fear of heisenbugs.

A Mature Ecosystem for Production Work

Five years ago, choosing Rust for web development was an act of faith. The ecosystem was young, frameworks were experimental, and documentation was sparse. That has changed dramatically.

The Rust web ecosystem now includes:

- Multiple production-grade web frameworks (Axum, Actix Web, Rocket)
- Async runtime (Tokio) used by companies processing billions of requests daily
- Database tooling (SQLx, Diesel) with compile-time query checking
- Serialization libraries (serde) that are the standard across the ecosystem
- Testing, tracing and observability tools on par with other languages
- Containerization and deployment patterns that work smoothly

Tokio alone is a testament to the ecosystem's maturity. It is used by Discord for their backend services, by Cloudflare for Workers, by AWS in production infrastructure, and by thousands of open-source projects. When your async runtime is battle-tested at that scale, you can trust it with your service.

The Trade-offs

No technology is perfect, and Rust's trade-offs are real:

- **Steeper learning curve:** Ownership, borrowing, lifetimes and async traits take time to internalize. Expect a few weeks or months of frustration before things click. This is the most common reason developers abandon Rust early.
- **Longer compile times:** Complex Rust projects can take minutes to compile, slowing your edit-test cycle. Incremental compilation helps but does not eliminate this.
- **Smaller ecosystem than established languages:** While mature for web development, Rust still has fewer libraries overall than Python or JavaScript. You may need to write some infrastructure code yourself.
- **Verbose error messages:** The compiler is helpful but verbose. Reading multi-paragraph errors becomes a skill in itself.

These trade-offs are worth it if your priorities include reliability, performance and long-term maintainability. They are not worth it if you need to prototype something in an afternoon and never touch it again.

Landscape of Rust Web Frameworks

Rust has several capable web frameworks, each with different design philosophies. Understanding the landscape helps explain why Axum is a strong choice.

Actix Web

Actix Web was for many years the most popular Rust web framework and remains one of the fastest. It uses an actor-based concurrency model inspired by Erlang, where actors are lightweight concurrent entities that communicate via messages.

Strengths:

- Excellent raw performance - frequently tops TechEmpower benchmarks
- Mature ecosystem with many supporting crates
- Large community and extensive documentation

Weaknesses:

- Actor model adds conceptual complexity for developers unfamiliar with it
- Steeper learning curve than some alternatives
- Macro-heavy API that can be opaque

Actix Web is an excellent choice, particularly if raw throughput is your primary concern. Its actor model provides powerful patterns for managing stateful connections and distributed systems. For many use cases, however, the added complexity is unnecessary.

Rocket

Rocket emphasizes developer experience with a clean, macro-based API that feels familiar to developers coming from Ruby on Rails or Flask. It uses compile-time validation of routes and type-safe parameter extraction.

Strengths:

- Very ergonomic for simple applications
- Fairings system provides powerful middleware hooks
- Strong focus on developer productivity

Weaknesses:

- Historically slower to adopt new Rust features (nightly dependencies in older versions)
- Less flexible than lower-level frameworks for advanced use cases
- Smaller ecosystem compared to Actix and Axum

Rocket is a great choice for developers who prioritize rapid development and clean syntax. Recent versions have stabilized on stable Rust, addressing one of its historical concerns.

Warp

Warp takes a functional approach with composable filters – small units that extract data from requests or transform responses, chained together to build routes. It sits directly on top of Hyper and Tokio without an actor layer.

Strengths:

- Highly composable filter system
- Close to the metal for fine-grained control
- Good integration with the Tokio ecosystem

Weaknesses:

- Filter composition can become hard to read at scale
- Less structured than Router-based frameworks
- Smaller community and fewer examples

Warp appeals to developers who like functional programming patterns and want maximum flexibility. Its filter system is powerful but can become unwieldy in large applications.

Other Notable Frameworks

Several other frameworks deserve mention:

- **Poem:** A full-featured framework inspired by Flask, with built-in Web-Socket, SSE and GraphQL support.
- **Salvo:** Emphasizes simplicity and performance with a clean routing API.
- **Leptos Server Functions / Axum integration:** For full-stack applications combining server and client rendering.

The Rust web ecosystem is healthy and competitive. Each framework has valid use cases, and the choice often comes down to team preferences and specific requirements.

Why Axum Stands Out

Axum emerged in 2021 from the Tokio project maintainers with a clear goal: create a web framework that feels natural in modern Rust while leveraging the existing ecosystem rather than reinventing it. The result is a framework that has quickly become one of the most popular choices for new Rust web projects.

Type-Driven Design

Axum's defining characteristic is its type-driven design philosophy. Routes, extractors and handlers are built using Rust's type system so that correctness is enforced at compile time rather than runtime.

Consider how routing works. In many frameworks, you define routes as strings and parameters as loosely typed values:

```
1 # Python Flask example
2 @app.route('/users/<user_id>')
3 def get_user(user_id):
4     # user_id is a string; type errors happen at runtime
```

In Axum, the route handler's argument types declare what it expects:

```
1 async fn get_user(Path(user_id): Path<u64>) -> Json<User> {
2     // user_id is guaranteed to be u64 or the request returns 400
3 }
```

If a request arrives with a non-numeric `user_id`, Axum automatically rejects it with a 400 Bad Request before your handler runs. The type system encodes your routing contract.

This extends to extractors for JSON bodies, query parameters, headers, cookies, WebSocket upgrades, and custom data types. Each extractor declares what it needs from the request as a type, and Axum's compiler-enforced rules ensure everything fits together correctly.

Built on Tower and Hyper

Axum does not reinvent middleware or HTTP handling. Instead, it sits on top of two battle-tested libraries:

- **Hyper** handles the low-level HTTP protocol - parsing requests, serializing responses, managing connections for HTTP/1.1 and HTTP/2.
- **Tower** provides a universal abstraction for services and middleware through its Service and Layer traits.

This design decision is profound. It means:

- Every Tower middleware works with Axum out of the box - timeouts, tracing, compression, rate limiting, retries, circuit breakers. You do not need Axum-specific versions.
- Middleware written for Axum can be reused in Hyper-based services, gRPC servers using Tonic, or any other Tower-compatible framework.
- When you learn Tower's patterns, they transfer across the ecosystem.

Axum essentially provides ergonomic routing and request handling on top of this foundation without locking you into its own middleware system. This is unlike many frameworks that create proprietary middleware APIs you cannot use elsewhere.

Ergonomics for Modern Rust

Axum was designed with modern Rust idioms in mind:

- **No macros for routing:** Routes are defined using combinators (`Router::new().route(...)`) rather than `#[get("/")]` attributes. This means routes are regular Rust code that your editor can autocomplete, refactor and navigate.
- **async/await throughout:** Handlers are plain async functions. No special trait implementations or macro wrappers required for basic use.
- **State sharing via the State extractor:** Application-wide state (database pools, configuration, caches) is injected into handlers through a type-safe extractor rather than global variables or closures.
- **IntoResponse for flexible returns:** Handlers can return strings, JSON, HTML, custom structs implementing `IntoResponse`, or `Result` types with automatic error conversion.

These design choices make Axum code read like natural Rust rather than framework-specific DSL syntax.

Official Tokio Support

Axum is maintained by the same team behind Tokio. This provides several advantages:

- Deep integration with Tokio's runtime and ecosystem

- Priority bug fixes and feature development aligned with Tokio releases
- Access to Tokio's extensive experience running production workloads
- Consistent design philosophy across the stack

When a framework is backed by the maintainers of its underlying runtime, you get coordination that independent projects rarely achieve. Tokio and Axum evolve together rather than chasing each other.

Growing Ecosystem

Since its release, Axum has attracted significant ecosystem support:

- **axum-extra**: Official companion crate with additional extractors (Multi-part file uploads, TypedHeader, SignedCookieJar)
- **utoipa-axum**: Tight integration with utoipa for OpenAPI documentation generation
- **axum-test**: Testing utilities specifically designed for Axum applications
- **axum-session** / **axum-security**: Session management and security utilities
- Numerous community crates for authentication, rate limiting, GraphQL and more

The ecosystem is not as mature as Actix Web's after many more years of development, but it is growing rapidly and covers most production needs.

When Axum Might Not Be the Best Choice

Axum is an excellent default choice, but it is not universally optimal:

- If you need absolute maximum throughput and are willing to accept more complexity, Actix Web's actor model may give you an edge.
- If your team prefers macro-based routing syntax similar to Rails or Flask, Rocket might feel more natural.
- If you are building a very small prototype and want minimal setup, a simpler framework may suffice.

For most production web services and APIs, however, Axum provides the best balance of performance, ergonomics, ecosystem integration, and long-term maintainability.

Introducing Our Running Example: A Task Management API

Throughout this book we will build Taskflow - a task and project management API that demonstrates progressively more advanced patterns. Here is what Taskflow will support by the end of the book:

Core features:

- User registration and authentication with secure password hashing
- JWT-based session management with refresh tokens
- CRUD operations for tasks (create, read, update, delete)
- Projects with many-to-many relationships to tasks
- Role-based authorization (admin, member, viewer)

Advanced features:

- Real-time task updates via WebSockets
- File attachments on tasks with multipart uploads
- Full-text search across tasks and projects
- Rate limiting to prevent abuse
- Response caching for frequently accessed data

Production readiness:

- PostgreSQL database with SQLx and migrations
- Structured logging with tracing
- OpenTelemetry integration for distributed tracing
- Comprehensive test suite (unit, integration, end-to-end)
- Optimized Docker images for deployment
- Health check endpoints for orchestration
- Graceful shutdown on termination signals

We will start simple - a basic server with a few routes - and add complexity chapter by chapter. Each addition will be explained in the context of why it matters for production services, not just how to implement it.

The Taskflow codebase is designed to be realistic rather than minimal. Real applications have configuration management, error handling across layers, database transactions, middleware stacks, and testing requirements. We will address all of these as they arise naturally.

How to Read This Book

This book is organized to take you from foundational understanding through production deployment in a logical progression. Here is how to approach it:

Read in Order for Best Results

While individual chapters can serve as references, the material builds cumulatively. Chapter 3 on Tokio and async Rust is essential background for understanding Axum's behavior. Chapter 4 on Tower explains the middleware model used throughout later chapters. Skip ahead at your own risk of encountering unexplained concepts.

Type the Code, Do Not Just Copy It

The best way to learn Rust web development is to write the code yourself. Each chapter contains complete, compilable examples. Type them out rather than copy-pasting - the friction of dealing with imports, types and compiler errors is where real learning happens.

When the compiler complains, do not immediately search for a fix online. Read the error message carefully. Rust's compiler messages are among the best in any language; they often explain exactly what went wrong and how to fix it. Learning to read them is a valuable skill.

Experiment with Variations

Once an example compiles and runs, try modifying it:

- Change a route path or HTTP method
- Add a new field to a request body struct
- Replace one extractor with another
- Introduce an error condition and see how it propagates

These small experiments build intuition faster than passive reading.

Use the Official Documentation as a Companion

This book explains concepts thoroughly, but official documentation is always the authoritative source for API details. Bookmark these links:

- Axum docs: <https://docs.rs/axum/latest/axum/>
- Tokio docs: <https://docs.rs/tokio/latest/tokio/>
- Tower docs: <https://docs.rs/tower/latest/tower/>
- SQLx docs: <https://docs.rs/sqlx/latest/sqlx/>
- Rust reference: <https://doc.rust-lang.org/reference/>

When you encounter a type or function mentioned in this book, look it up in the official docs to see the full signature, examples and related types.

Focus on Patterns, Not Just Syntax

APIs change between versions; patterns endure. Pay attention to the architectural decisions and design patterns explained in each chapter:

- Why we structure errors a certain way
- Why middleware is layered in a particular order
- Why we separate repositories from services
- Why tests are organized into specific categories

Understanding the why behind patterns will serve you longer than memorizing specific function calls.

It Is Okay to Go Slow

Rust web development involves many interconnected concepts: ownership, async/await, traits, generics, lifetimes, HTTP semantics, database design, security practices. Absorbing all of this takes time. Do not feel pressured to rush through chapters. Re-read sections that are unclear. Work through compiler errors patiently. The investment pays off in code that is correct, performant and maintainable.

Summary

This chapter has established why Rust and Axum form a compelling stack for building production web services:

- Rust provides memory safety without garbage collection, excellent performance, compile-time guarantees about correctness, safe concurrency primitives, and a mature ecosystem for web development.
- The Rust web framework landscape includes Actix Web, Rocket, Warp and others, each with valid trade-offs.
- Axum stands out through its type-driven design, integration with Tower and Hyper rather than reinventing middleware, ergonomic modern Rust API, official Tokio maintainership, and rapidly growing ecosystem.
- Our running example - the Taskflow task management API - will evolve throughout the book to demonstrate progressively advanced patterns in a realistic context.

The next chapter sets up your development environment and creates the initial project structure that we will use throughout the book. We will configure Cargo.toml with all necessary dependencies, establish a directory layout that scales, write our first Axum server with proper structure, and implement an environment configuration strategy that works from local development through production deployment.

Chapter 2: Project Setup and Architecture

A well-structured project is the foundation of maintainable software. In this chapter we set up everything needed to start building with Axum: the Rust toolchain, Cargo configuration with proper dependencies, a directory layout that scales as our application grows, our first working server with clean structure, and an environment configuration approach that separates local development from production settings.

By the end of this chapter you will have a Taskflow project ready for development, with all the plumbing in place to add functionality in subsequent chapters.

Rust Toolchain and Edition Selection

Before creating our project, ensure your Rust toolchain is up to date. Open a terminal and run:

```
1 rustup update stable
2 rustup default stable
```

This installs or updates to the latest stable Rust release. Verify the installation:

```
1 rustc --version
2 cargo --version
```

You should see versions at least as recent as rustc 1.75+ and cargo 1.75+. Newer is better.

Choosing the Edition

Rust uses editions - periodic updates to the language standard library and some syntax rules that do not require breaking changes. The editions are:

- **2015:** The original edition, now legacy
- **2018:** Introduced major improvements (modules, macros, keywords)
- **2021:** Current stable standard; adds const generics improvements, try traits, and lifetime elision refinements
- **2024:** Emerging edition with strict provenance and other changes

We will use the 2021 edition for this book. It is widely supported across the ecosystem, all our dependencies target it, and it provides a stable foundation. The 2024 edition is still being adopted; while interesting, it may introduce compatibility friction with some crates during this book's lifetime.

When you create a new project with `cargo new`, `Cargo.toml` will include `edition = "2021"` by default in recent versions of Rust. If not, add it manually.

Essential Tools

Beyond the base toolchain, install these tools that we will use throughout the book:

```
1 # Rust Analyzer is built into most editors; ensure it is installed
2 rustup component add rust-analyzer
3
4 # clippy for linting (usually included with stable)
5 rustup component add clippy
6
7 # rustfmt for code formatting (usually included with stable)
8 rustup component add rustfmt
9
10 # sqlx-cli for database migrations (needed later in Chapter 10)
11 cargo install sqlx-cli --features postgres,runtime-tokio-rustls
```

Rust Analyzer provides IDE-like features - autocomplete, go-to-definition, inline errors - in editors like VS Code, Neovim and IntelliJ. It is invaluable for Rust development. Clippy catches common mistakes and suggests idiomatic improvements. rustfmt ensures consistent formatting across the codebase.

Cargo Configuration and Dependency Management

Create the Taskflow project:

```
1 cargo new taskflow-api --lib
2 cd taskflow-api
```

We start with `--lib` rather than `--bin` because our application will grow into multiple modules, and separating library code from the binary entry point is a good pattern for larger projects. We will add a binary later.

The Cargo.toml File

Replace the generated Cargo.toml with this configuration:

```
1 [package]
2 name = "taskflow-api"
3 version = "0.1.0"
4 edition = "2021"
5
6 [dependencies]
7 # Web framework and HTTP handling
8 axum = { version = "0.8", features = ["macros"] }
9 tokio = { version = "1", features = ["full"] }
10 tower = { version = "0.5", features = ["util"] }
11 tower-http = { version = "0.7", features = [
12     "cors",
13     "trace",
14     "fs",
15     "compression-full",
16 ] }
17
18 # Serialization and data handling
19 serde = { version = "1", features = ["derive"] }
20 serde_json = "1"
21
22 # Logging and tracing
23 tracing = "0.1"
24 tracing-subscriber = { version = "0.3", features = ["env-filter"] }
25
26 # Error handling
27 anyhow = "1"
28 thiserror = "2"
29
```

```
30 [dev-dependencies]
31 # Testing utilities
32 axum-test = "18"
33 tokio = { version = "1", features = ["macros", "rt-multi-thread"] }
```

Let us walk through each dependency:

axum: The web framework. We enable the `macros` feature for convenience macros like `debug_handler`. Version 0.8 is the current stable series.

tokio: The async runtime. The `full` feature includes everything - multi-threaded runtime, I/O, time, signals, process management. For production you might select specific features to reduce binary size, but `full` simplifies development and the size difference is modest for most applications.

tower: Provides the Service and Layer traits that Axum uses for middleware. The `util` feature includes utility services like `map_response`.

tower-http: HTTP-specific middleware layers built on Tower. We enable:

- `cors` for Cross-Origin Resource Sharing configuration
- `trace` for request/response logging
- `fs` for serving static files
- `compression-full` for gzip/brotli response compression

serde and serde_json: The standard Rust serialization ecosystem. `serde` derives `Serialize/Deserialize` traits for converting between Rust structs and JSON, YAML, TOML, etc. `serde_json` handles JSON specifically. Nearly every Rust web framework uses `serde`.

tracing and tracing-subscriber: Structured logging and diagnostic instrumentation. `tracing` emits events with structured fields; `tracing-subscriber` collects and formats them. The `env-filter` feature allows runtime filtering via the `RUST_LOG` environment variable.

anyhow: Error handling for application code (`main.rs`, top-level functions). Provides a flexible Error type that wraps other errors without requiring exhaustive enum definitions.

thiserror: Error handling for libraries and defined error types. Generates `Display` and `From` implementations for custom error enums with concise macros.

axum-test: Testing utility for integration tests against Axum routers. We place it in `dev-dependencies` since it is not needed at runtime.

Feature Flags and Version Pinning

Note that we specify version ranges ("`0.8`" means `>=0.8.0`, `<0.9.0`) rather than exact versions. Cargo's semver resolution ensures you get compatible updates while avoiding breaking changes. The `Cargo.lock` file in your repository pins exact versions for reproducible builds.

Commit `Cargo.lock` to version control for binaries and services - this ensures everyone builds with identical dependencies. For libraries, the convention is mixed; many Rust library authors do not commit `Cargo.lock`. For our API service, we will commit it.

Optional Dependencies for Later Chapters

As we progress through the book, we will add more dependencies. Here is a preview of what comes:

- Chapter 8 (Authentication): `argon2`, `jsonwebtoken`, `uuid`
- Chapter 10 (Database): `sqlx` with `postgres` feature
- Chapter 12 (WebSockets): Already included via `axum`'s `ws` feature when needed
- Chapter 14 (OpenAPI): `utoipa`, `utoipa-swagger-ui`
- Chapter 16 (Observability): `opentelemetry`, `axum-tracing-opentelemetry`

We will add each when it is needed rather than front-loading dependencies we may not use.

Directory Layout for Scalable Projects

A flat `src/` directory works fine for small projects but becomes unwieldy as complexity grows. We will establish a structure that scales from our initial simple server through the full Taskflow application.

Create this directory layout:

```
1 taskflow-api/  
2 |─ Cargo.toml  
3 |─ Cargo.lock  
4 |─ .env.example  
5 |─ src/  
6 |   |─ main.rs           # Binary entry point  
7 |   |─ lib.rs           # Library root, module declarations  
8 |   |─ config.rs        # Configuration loading and types  
9 |   |─ error.rs         # Application-wide error types  
10 |  |─ state.rs          # Shared application state  
11 |  |─ routes.rs         # Route registration  
12 |  |─ handlers/        # HTTP handlers by domain  
13 |    |─ mod.rs  
14 |─ migrations/         # SQLx database migrations (added later)  
15 |─ tests/              # Integration tests (added later)
```

This structure separates concerns:

- **main.rs:** Wires everything together - loads config, creates state, builds router, starts server. Minimal logic; mostly orchestration.
- **lib.rs:** Declares public modules and re-exports shared types. This allows integration tests to import from the library rather than duplicating code.
- **config.rs:** Handles environment-based configuration with typed structs.
- **error.rs:** Centralized error types used across handlers and services.
- **state.rs:** Defines the AppState struct that holds shared resources (database pool, config, caches) injected into handlers.
- **routes.rs:** Registers all routes with the router in one place for easy navigation.
- **handlers/:** Domain-specific HTTP handlers organized by resource (tasks, users, projects).

Let us create each file now.

src/lib.rs

```

1 pub mod config;
2 pub mod error;
3 pub mod state;
4 pub mod routes;
5 pub mod handlers;

```

This declares our modules and makes them accessible both within the crate and to integration tests in the `tests/` directory.

src/handlers/mod.rs

```

1 // Handlers will be organized by domain as we add functionality:
2 // pub mod tasks;
3 // pub mod users;
4 // pub mod projects;

```

For now this is empty with comments showing where handlers will go. We will uncomment and create modules as needed.

src/config.rs

Configuration management is critical for applications that run in multiple environments - local development, staging, production - each with different database URLs, ports, secret keys, and feature flags. We will use environment variables exclusively, loaded from a `.env` file during development and set explicitly in production deployments.

```

1 use serde::Deserialize;
2 use std::net::SocketAddr;
3
4 /// Application configuration loaded from environment variables.
5 #[derive(Debug, Deserialize, Clone)]
6 pub struct Config {
7     /// Server bind address, e.g., "127.0.0.1:3000" or "0.0.0.0:8080"
8     pub server_addr: SocketAddr,
9
10    /// Application environment: "development", "staging", or "production"
11    pub env: Environment,
12
13    /// Database URL for SQLx connections
14    pub database_url: String,
15

```

```

16     /// Secret key for JWT signing and cookie encryption
17     pub jwt_secret: String,
18
19     /// CORS allowed origins (comma-separated)
20     #[serde(default = "default_cors_origins")]
21     pub cors_origins: Vec<String>,
22 }
23
24 #[derive(Debug, Deserialize, Clone, PartialEq)]
25 #[serde(rename_all = "lowercase")]
26 pub enum Environment {
27     Development,
28     Staging,
29     Production,
30 }
31
32 fn default_cors_origins() -> Vec<String> {
33     vec!["http://localhost:5173".to_string()]
34 }
35
36 impl Config {
37     /// Load configuration from environment variables.
38     ///
39     /// In development, this can be combined with a .env file loaded
40     /// by tools like dotenvy or by setting RUST_LOG and other vars manually.
41     pub fn from_env() -> anyhow::Result<Self> {
42         Ok(Self {
43             server_addr: std::env::var("SERVER_ADDR")
44                 .unwrap_or_else(|_| "127.0.0.1:3000".to_string())
45                 .parse()?,
46             env: std::env::var("APP_ENV")
47                 .unwrap_or_else(|_| "development".to_string())
48                 .parse()?,
49             database_url: std::env::var("DATABASE_URL")
50                 .expect("DATABASE_URL must be set"),
51             jwt_secret: std::env::var("JWT_SECRET")
52                 .expect("JWT_SECRET must be set"),
53             cors_origins: std::env::var("CORS_ORIGINS")
54                 .ok()
55                 .map(|s| s.split(',').map(str::to_string).collect())
56                 .unwrap_or_else(default_cors_origins),
57         })
58     }
59 }
60
61 impl Default for Config {
62     fn default() -> Self {
63         Self::from_env().expect("Failed to load configuration from environment")
64     }
65 }

```

Key design decisions in this config module:

- **Typed fields:** `server_addr` is a `SocketAddr`, not a `String`. Parse errors fail fast at startup rather than causing confusing errors later when the address is used.
- **Environment enum:** Using an enum for `APP_ENV` allows pattern matching on environment-specific behavior (different logging levels, feature flags) throughout the application.
- **Defaults with safety:** `SERVER_ADDR` has a sensible default; `DATABASE_URL` and `JWT_SECRET` do not - they must be explicitly set to avoid accidentally using wrong credentials.
- **Clone trait:** Config is cloned into `AppState` so handlers can access configuration without shared mutable state.

`.env.example`

Create this file as a template for developers:

```

1 # Server configuration
2 SERVER_ADDR=127.0.0.1:3000
3 APP_ENV=development
4
5 # Database (PostgreSQL)
6 DATABASE_URL=postgresql://localhost:5432/taskflow_dev
7
8 # Security
9 JWT_SECRET=change-me-to-a-random-secret-in-production
10
11 # CORS (comma-separated origins)
12 CORS_ORIGINS=http://localhost:5173
13
14 # Logging
15 RUST_LOG=info,taskflow_api=debug,tower_http=debug

```

Add `.env` to your `.gitignore` - never commit secrets. The `.env.example` file shows required variables without exposing actual values.

`src/error.rs`

Error handling deserves its own module from the start. We will define application-wide error types that handlers can return, with automatic conversion into appropriate HTTP responses.

```

1  use axum::http::StatusCode;
2  use axum::response::{IntoResponse, Response};
3  use serde::Serialize;
4  use thiserror::Error;
5
6  /// Application-level errors that can occur during request handling.
7  #[derive(Debug, Error)]
8  pub enum AppError {
9      #[error("Resource not found")]
10     NotFound,
11
12     #[error("Unauthorized: {0}")]
13     Unauthorized(String),
14
15     #[error("Forbidden")]
16     Forbidden,
17
18     #[error("Validation error: {0}")]
19     Validation(String),
20
21     #[error("Internal server error")]
22     Internal(#[source] anyhow::Error),
23 }
24
25 /// JSON error response body sent to clients.
26 #[derive(Debug, Serialize)]
27 pub struct ErrorResponse {
28     pub error: String,
29     pub message: String,
30 }
31
32 impl IntoResponse for AppError {
33     fn into_response(self) -> Response {
34         let (status, error_type, message) = match &self {
35             AppError::NotFound => (
36                 StatusCode::NOT_FOUND,
37                 "not_found",
38                 self.to_string(),
39             ),
40             AppError::Unauthorized(msg) => (
41                 StatusCode::UNAUTHORIZED,
42                 "unauthorized",
43                 msg.clone(),
44             ),
45             AppError::Forbidden => (
46                 StatusCode::FORBIDDEN,
47                 "forbidden",
48                 "You do not have permission to perform this action".to_string(),
49             ),
50             AppError::Validation(msg) => (

```

```

51         StatusCode::BAD_REQUEST,
52         "validation_error",
53         msg.clone(),
54     ),
55     AppError::Internal(inner) => {
56         // Log the full internal error; do not expose details to clients
57         tracing::error!(error = ?inner, "Internal server error");
58         (
59             StatusCode::INTERNAL_SERVER_ERROR,
60             "internal_error",
61             "An unexpected error occurred".to_string(),
62         )
63     }
64 };
65
66 let body = ErrorResponse {
67     error: error_type.to_string(),
68     message,
69 };
70
71 (status, axum::Json(body)).into_response()
72 }
73 }
74
75 /// Convenience: convert anyhow::Error into AppError for use in handlers.
76 impl From<anyhow::Error> for AppError {
77     fn from(err: anyhow::Error) -> Self {
78         AppError::Internal(err)
79     }
80 }

```

This error module establishes patterns we will use throughout the book:

- **Enum variants map to HTTP status codes:** `NotFound` -> 404, `Unauthorized` -> 401, etc. This makes it easy to see what each error means for the client.
- **Internal errors hide details:** The `Internal` variant wraps an `anyhow::Error` for full context in logs but returns a generic message to clients. Never leak stack traces or internal details.
- **Consistent JSON format:** All errors return the same `ErrorResponse` structure with error type and human-readable message.
- **IntoResponse implementation:** Axum automatically converts `AppError` into HTTP responses, so handlers can simply return `Result<T, AppError>`.

src/state.rs

Application state holds shared resources that all handlers need access to - configuration, database pools, caches, background task handles. Axum's State extractor provides type-safe injection of this shared data.

```
1 use crate::config::Config;
2 use std::sync::Arc;
3
4 /// Shared application state accessible to all handlers via the State extractor.
5 ///
6 /// Wrapped in Arc for cheap cloning across async tasks and threads.
7 #[derive(Clone)]
8 pub struct AppState {
9     pub config: Config,
10 }
11
12 impl AppState {
13     /// Create a new AppState from configuration.
14     ///
15     /// Additional fields (database pool, caches, etc.) will be added
16     /// in later chapters as we integrate more functionality.
17     pub fn new(config: Config) -> Arc<Self> {
18         Arc::new(Self { config })
19     }
20 }
```

The AppState struct is intentionally minimal for now. As we add database integration (Chapter 10), caching (Chapter 16), and other shared resources, they will be added as fields here. Wrapping in Arc allows cheap cloning - handlers receive an Arc reference, not ownership of the underlying data.

src/routes.rs

Route registration centralizes all route definitions in one place, making it easy to see the full API surface at a glance.

```

1  use axum::{routing::get, Router};
2  use crate::state::AppState;
3  use std::sync::Arc;
4
5  /// Build the application router with all routes registered.
6  pub fn create_router(state: Arc<AppState>) -> Router {
7      Router::new()
8          .route("/", get(root_handler))
9          .route("/health", get(health_check))
10         .with_state(state)
11     }
12
13     /// Root endpoint - simple confirmation that the server is running.
14     async fn root_handler() -> &'static str {
15         "Taskflow API is running"
16     }
17
18     /// Health check endpoint for load balancers and orchestration systems.
19     async fn health_check() -> &'static str {
20         "ok"
21     }

```

The `create_router` function takes our `AppState` and returns a fully configured `Router`. The `with_state` call makes state available to all handlers via the `State` extractor. We will add more routes here as we build out Taskflow's functionality.

Our First Axum Server: Hello World with Structure

Now we have all the pieces in place. Let us create the binary entry point that ties everything together.

Create `src/main.rs`:

```

1  mod config;
2  mod error;
3  mod state;
4  mod routes;
5  mod handlers;
6
7  use axum::serve;
8  use tokio::net::TcpListener;
9  use tower_http::trace::TraceLayer;
10 use tracing_subscriber::{layer::SubscriberExt, util::SubscriberInitExt};
11
12 #[tokio::main]
13 async fn main() -> anyhow::Result<()> {
14     // Initialize structured logging
15     tracing_subscriber::registry()
16         .with(
17             tracing_subscriber::EnvFilter::try_from_default_env()
18                 .unwrap_or_else(|_|
19 ↪ "taskflow_api=debug,tower_http=debug,info".into()),
20         )
21         .with(tracing_subscriber::fmt::layer())
22         .init();
23
24     // Load configuration from environment
25     let config = config::Config::from_env()?;
26     tracing::info!(
27         env = %config.env,
28         addr = %config.server_addr,
29         "Starting Taskflow API"
30     );
31
32     // Create shared application state
33     let state = state::AppState::new(config);
34
35     // Build the router with middleware
36     let app = routes::create_router(state.clone())
37         .layer(TraceLayer::new_for_http());
38
39     // Bind to the configured address and start serving
40     let listener = TcpListener::bind(state.config.server_addr).await?;
41     tracing::info!(addr = %listener.local_addr(), "Server listening");
42
43     serve(listener, app).await?;
44
45     Ok(())
46 }

```

Let us walk through this file section by section:

Module declarations: The mod statements at the top bring our library

modules into the binary crate. This is why we structured things with a `lib.rs` - `main.rs` stays thin and focused on orchestration.

#[tokio::main]: This macro sets up the Tokio runtime and marks `main` as an async function. By default it creates a multi-threaded runtime suitable for I/O-heavy web servers. The runtime manages the event loop, spawns worker threads, and schedules async tasks.

Tracing initialization: We configure tracing-subscriber with:

- `EnvFilter` that reads from `RUST_LOG` environment variable (set in `.env`) or defaults to debug level for our app and `tower_http`
- `fmt` layer that outputs structured logs in a readable format

This gives us production-quality logging from the start. In development you can set `RUST_LOG=trace` for maximum verbosity; in production `RUST_LOG=info` reduces noise.

Configuration loading: `Config::from_env()` reads environment variables and parses them into typed fields. If required variables are missing, it panics with a clear message - failing fast at startup is better than failing mysteriously under load.

State creation: `AppState::new(config)` wraps our config in `Arc` for shared access.

Router building: `routes::create_router(state.clone())` constructs the router with all routes and injects state. We then layer `TraceLayer` on top, which logs every incoming request and outgoing response with method, path, status code, and latency.

Server binding: `TcpListener::bind()` creates a TCP listener on the configured address. The `?` operator propagates errors - if the port is already in use, the server fails to start with a clear error rather than silently ignoring the problem.

serve(): `axum::serve()` starts the HTTP server using Hyper under the hood. It runs until an error occurs or until we implement graceful shutdown (Chapter 17).

Running the Server

Create a `.env` file based on `.env.example` with actual values:

```
1 SERVER_ADDR=127.0.0.1:3000
2 APP_ENV=development
3 DATABASE_URL=postgresql://localhost:5432/taskflow_dev
4 JWT_SECRET=my-development-secret-change-in-production
5 CORS_ORIGINS=http://localhost:5173
6 RUST_LOG=info,taskflow_api=debug,tower_http=debug
```

Run the server:

```
1 cargo run
```

You should see output like:

```
1 2026-08-27T21:00:00.000Z INFO taskflow_api: Starting Taskflow API
   ↳ env=development addr=127.0.0.1:3000
2 2026-08-27T21:00:00.001Z INFO taskflow_api: Server listening
   ↳ addr=127.0.0.1:3000
```

Test the endpoints:

```
1 curl http://localhost:3000/
2 # Returns: Taskflow API is running
3
4 curl http://localhost:3000/health
5 # Returns: ok
```

Check the logs - you should see TraceLayer output for each request showing method, path, status code, and elapsed time.

Understanding What Just Happened

When you made that curl request, here is the flow through our application:

1. Hyper (under Axum) receives the TCP connection and parses the HTTP request
2. TraceLayer records the incoming request details
3. The Router matches the path against registered routes
4. For “/” it calls `root_handler()`, which returns a static string
5. Axum converts the string into an HTTP response via `IntoResponse`
6. TraceLayer records the response status and latency

7. Hyper serializes the response and sends it back over TCP

This flow is the skeleton of every request your application will handle. As we add extractors, middleware, database queries, and authentication, they slot into this pipeline at specific points - but the fundamental structure remains the same.

Environment Configuration Strategy

Configuration management is one of those topics that seems simple until production forces you to confront its complexity. Different environments need different values. Secrets must not be in source control. Configuration changes should not require code redeployment in many cases. Let us establish patterns that handle all of this correctly.

The Twelve-Factor Approach

We follow the Twelve-Factor App methodology for configuration: store config in environment variables. This approach has several advantages:

- **No secret leakage:** Environment variables are not committed to git (when properly managed)
- **Environment parity:** Same code runs everywhere; only environment differs
- **Easy rotation:** Change a secret by updating the environment, not redeploying code
- **Orchestration friendly:** Docker, Kubernetes and cloud platforms all support env vars natively

Development vs Production

For local development:

- Use a .env file loaded manually or via a tool like dotenvy
- Use weak secrets (just something non-empty) since there is no real threat model locally

- Set RUST_LOG to debug or trace for maximum visibility
- Point DATABASE_URL at a local PostgreSQL instance or Docker container

For production:

- Never use .env files - set variables through your deployment platform
- Use strong, randomly generated secrets (at least 256 bits for JWT signing)
- Set RUST_LOG to info or warn to reduce log volume
- Use managed database services with connection pooling

Secrets Management in Practice

For the JWT_SECRET specifically:

```
1 # Generate a secure random secret
2 openssl rand -hex 32
3 # Output example: a1b2c3d4e5f6... (64 hex characters = 256 bits)
```

Use this value in production. Never use the example values from code or documentation. In Kubernetes, secrets are stored in Secret resources. In Docker Compose, they go in separate files not committed to git. On cloud platforms like AWS or GCP, use their secret management services.

Configuration Validation

Our Config::from_env() function validates configuration at startup:

- Required fields panic if missing (DATABASE_URL, JWT_SECRET)
- Parse failures propagate via anyhow::Result
- The server refuses to start with invalid configuration

This is critical - a server that starts with wrong configuration and fails under load is worse than one that refuses to start. Fail fast, fail loudly, fail at startup.

Summary

This chapter established the foundation for our Taskflow API:

- We configured the Rust toolchain with essential components (rust-analyzer, clippy, rustfmt, sqlx-cli)
- We created a Cargo.toml with all core dependencies - Axum 0.8, Tokio 1.x, Tower, tower-http, serde, tracing and error handling crates
- We designed a directory layout that scales: separate modules for config, errors, state, routes and handlers
- We implemented typed configuration loading from environment variables with sensible defaults and strict required fields
- We defined application-wide error types with IntoResponse conversion for consistent HTTP error handling
- We created an AppState struct wrapped in Arc for type-safe shared state injection into handlers
- We built our first working server with tracing middleware, health check endpoint, and proper startup logging
- We established configuration management practices following the Twelve-Factor methodology

The project structure we have created is not over-engineered for a small application; it is appropriately structured from the start so that adding complexity later does not require painful refactoring. Every file serves a clear purpose, and the separation of concerns will become increasingly valuable as Taskflow grows.

In the next chapter we dive into the async foundation - understanding how Tokio's runtime works, how async/await operates in Rust, task scheduling and spawning, channels for concurrent communication, and the concurrency patterns that underpin all Axum applications. You cannot write effective Axum code without this understanding, so we will go deep.

Chapter 3: Tokio and Asynchronous Rust

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

How async/await Works in Rust

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Futures Are Lazy

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

The Polling Model

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

await Suspends and Resumes

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Pin and Self-Referential Types

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

async fn vs impl Future

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

The Tokio Runtime: Multithreaded vs Current Thread

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Multi-Threaded Runtime (Default)

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Current-Thread Runtime

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

enable_all() vs Selective Features

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Important Runtime Gotchas

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Tasks, Spawning and Joining

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

tokio::spawn

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Joining Multiple Tasks

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Selecting Among Futures

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Task Abortion

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Tasks in Axum Handlers

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Channels and Concurrent Communication

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

mpsc: Multiple Producer, Single Consumer

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

broadcast: One Producer, Many Consumers

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

oneshot: Single Message, Point-to-Point

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

sync: Observing Changing Values

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Timeouts, Sleeps and Scheduling Control

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

tokio::time::sleep

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

tokio::time::timeout

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

tokio::time::interval

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

tokio::time::missed_tick_behavior

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

spawn_blocking for CPU-Intensive Work

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Chapter 4: Tower, Hyper and Axum's Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

The Tower Service Trait: Composable Asynchronous Services

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

What Is a Service?

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Why This Abstraction Matters

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Service Builder and Composition

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Middleware as Service Layers

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

The Layer Trait

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Example: A Logging Layer

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Tower-HTTP: Production Middleware for HTTP

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Middleware Ordering Matters

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Hyper: The Low-Level HTTP Engine

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

What Hyper Does

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Hyper and the http Crate

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

When You Might Use Hyper Directly

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

How Axum Fits Into the Stack

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

The Stack Diagram

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Axum as a Tower Service

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

The `axum::serve` Function

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Understanding the Type-Driven Design Philosophy

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Routes Are Types, Not Strings

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Extractors Encode Contracts

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

IntoResponse: Flexible Returns

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Compile-Time Safety Over Runtime Errors

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

When Type-Driven Design Has Trade-offs

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Chapter 5: Routing and Extractors

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Defining Routes with the Router Combinator

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Basic Route Definition

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Multiple Methods on One Route

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Catch-All and Fallback Routes

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Route Nesting with nest

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Route Merging with merge

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Path Parameters and Their Types

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Single Path Parameter

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Multiple Path Parameters

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Capture-All Parameters

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Type Choices for Path Parameters

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Escaping Literal Braces

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Query Parameters and Structured Extraction

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Basic Query Extraction

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Required vs Optional Parameters

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Default Values with serde

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Query Parameter Validation

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Application State and the State Extractor

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Defining and Injecting State

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Why Arc and Clone?

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Substate with FromRef

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

State in Nested Routers

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Nested Routers and Modular Route Organization

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Domain-Based Module Structure

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

API Versioning with Nesting

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Route Module Testing

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Chapter 6: Request and Response Handling

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Response Types and Status Codes

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Simple Responses

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Status Codes with Responses

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

HTML Responses

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Tuple Responses with Headers

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

JSON Serialization with `serde_json`

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Returning JSON Responses

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

JSON Arrays and Collections

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Customizing JSON Output

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Deserializing JSON Request Bodies

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Handling Large JSON Bodies

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Form Data and URL-Encoded Bodies

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Extracting Form Data

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

When to Use Form vs JSON

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Headers: Reading and Writing

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Reading Headers

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Typed Headers with axum-extra

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Writing Headers

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Important Headers for APIs

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Cookies and Session Identifiers

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Setting Cookies

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Reading Cookies

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Signed Cookies with axum-extra

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Content Negotiation Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Simple Accept Header Handling

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

API-First: Default to JSON

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Chapter 7: Error Handling and Validation

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Designing Application Error Types

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

The AppError Enum Pattern

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Using AppError in Handlers

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

The IntoResponse Trait for Error Conversion

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

How IntoResponse Works

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Implementing IntoResponse for Custom Types

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Centralized Error Handlers

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Using axum's Default Error Handler

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Error Handler Middleware Pattern

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Including Request IDs in Errors

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Input Validation with Validator Crates

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Setting Up validator

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Declarative Validation with Derive Macros

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Integrating Validation with Axum Handlers

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Using axum-valid for Cleaner Integration

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

garde as an Alternative

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Separating Internal Errors from Client-Facing Messages

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

The Logging-First Approach

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Environment-Specific Error Detail Levels

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Chapter 8: Authentication and Authorization

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Password Hashing with Argon2

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Why Argon2id?

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Setting Up argon2

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Hashing and Verifying Passwords

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Tuning Argon2 Parameters

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Rehashing on Login

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Session-Based Authentication

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Session Flow

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Implementing Sessions with axum-session

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Session Storage Options

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Session Security Considerations

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

JWT Authentication Flow

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

JWT Structure

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Setting Up jsonwebtoken

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Creating and Validating JWTs

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

JWT Login Handler

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Extracting JWT from Authorization Header

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

JWT vs Sessions: Trade-offs

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Extractors for Authenticated Requests

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Pattern: Extractor with State Access

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Optional Authentication

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Role-Based Authorization Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Defining Roles

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Authorization in Handlers

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Authorization Middleware

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Resource-Level Authorization

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Chapter 9: Web Security Practices

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Cross-Origin Resource Sharing (CORS) Configuration

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

How CORS Works

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Basic CORS Setup

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Production CORS Configuration

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Per-Route CORS with Nested Routers

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Common CORS Mistakes

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Security Headers and Helmet-Like Middleware

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Essential Security Headers

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Content Security Policy (CSP)

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Strict-Transport-Security (HSTS)

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Using tower-http for Headers

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

CSRF Protection Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

When CSRF Matters

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Synchronizer Token Pattern

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

SameSite Cookie Attribute as Defense

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Rate Limiting and Abuse Prevention

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Why Rate Limit?

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

In-Memory Rate Limiter with `from_fn`

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Production Rate Limiting with External Storage

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Per-Endpoint Rate Limiting

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Rate Limit Response Headers

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Secure Configuration and Secrets Management

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Environment Variables for Secrets

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

.env Files for Development Only

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Secrets in Production

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Validating Configuration at Startup

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Chapter 10: Database Integration with SQLx

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Choosing a Database Driver

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Adding SQLx Dependencies

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Runtime Selection

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Connection Pooling with SQLx Pools

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Creating a Pool

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Pool Configuration Tuning

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Using the Pool in Application State

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Running Queries and Mapping Results

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Defining Domain Types

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Simple Queries with `sqlx::query`

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Querying Single Rows

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Querying Multiple Rows

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Dynamic Queries with `sqlx::query`

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

SQL Injection Prevention

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Database Migrations with `sqlx-migrate`

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Installing `sqlx-cli`

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Creating the Migrations Directory

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Creating Migration Files

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Running Migrations

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Embedding Migrations in the Binary

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Migration Best Practices

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Offline Mode for CI/CD

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Transactions and Atomic Operations

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Basic Transactions

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Automatic Rollback on Drop

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Savepoints for Nested Transactions

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Chapter 11: Repository and Service Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

The Repository Pattern in Rust

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Defining Repository Traits

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Domain Types vs Database Types

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Service Layer Design

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Defining Task Service

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Using Services in Handlers

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Dependency Injection Through Application State

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Wiring Everything in AppState

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Per-Module State with FromRef

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Domain Modeling and Type Safety

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Using Types to Enforce Rules

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

State Machines with Enums

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Structuring Large Applications

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Chapter 12: Real-Time Communication with WebSockets

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

WebSocket Upgrade Flow

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Adding WebSocket Support

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Basic Echo Server

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Understanding the Upgrade Flow

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Handling Bidirectional Messages

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Splitting the Socket for Concurrent I/O

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Typed Messages with Enums

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Connection State and Broadcast Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Managing Connections with broadcast

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Integrating with Task Updates

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

WebSocket Handler Using the Hub

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Scaling WebSocket Applications

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Single-Instance Deployment

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Redis Pub/Sub for Multi-Instance

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Connection Limits and Resource Management

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Server-Sent Events as an Alternative

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

SSE Handler in Axum

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Chapter 13: File Handling and Static Assets

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Serving Static Files Efficiently

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Basic Static File Serving

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Serving a Single File

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

SPA Fallback Pattern

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Static File Security

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Caching Headers for Static Assets

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

File Uploads with Multipart Forms

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Adding Multipart Support

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Basic Upload Handler

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Streaming Uploads Directly to Disk

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Setting Upload Size Limits

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Storing Uploads in Cloud Storage

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Streaming Large Responses

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Streaming a File Download

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Custom Streaming Responses

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

File Validation and Security Considerations

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Validate File Types

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Check File Content, Not Just Extension

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Isolate Uploaded Files

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Scan for Malware

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Chapter 14: RESTful API Design and Documentation

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

REST Principles in Practice

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Resource-Oriented Design

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Proper Use of HTTP Methods

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Status Codes That Mean Something

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Consistent Response Envelopes

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Pagination, Filtering and Sorting Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Offset-Based Pagination

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Cursor-Based Pagination

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Filtering

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Sorting

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

API Versioning Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

URL Path Versioning (Recommended)

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Header-Based Versioning

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Recommendation

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Generating OpenAPI/Swagger Documentation

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Adding utoipa

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Annotating Types and Handlers

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Defining the OpenAPI Schema

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Serving Swagger UI

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Using utoipa-axum for Tighter Integration

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Request/Response Schema Design

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Use Consistent Naming

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Include Timestamps

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Use Nullability Explicitly

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Return Errors with Structure

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Avoid Breaking Changes

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Chapter 15: Testing Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Unit Testing Handlers and Services

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Testing Service Logic

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Testing Utility Functions

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Integration Testing with `axum_test` or `request`

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Using `axum-test`

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Using Tower Service Directly (No HTTP Server)

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Testing with Authentication

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Database Testing Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

In-Memory SQLite for Fast Tests

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Dedicated Test Database with PostgreSQL

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Transaction-Based Test Isolation

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Docker-Based Test Database

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Mocking External Dependencies

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Trait-Based Mocking

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Using mockall for Automatic Mocks

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Test Organization for Large Projects

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Directory Structure

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Shared Test Helpers

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Running Tests in Parallel

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Chapter 16: Observability and Performance

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Structured Logging with Tracing

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Basic Tracing Setup

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Using Tracing in Code

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Log Levels

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Structured Fields

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Distributed Tracing in Microservices

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

OpenTelemetry Integration

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Trace Propagation

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Viewing Traces

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Metrics Collection and Export

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Using axum-prometheus

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Custom Metrics

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Alerting on Metrics

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Profiling and Identifying Bottlenecks

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Using flamegraph

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Using tokio-console

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Using perf (Linux)

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Common Performance Bottlenecks

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Caching Strategies: In-Memory and Distributed

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

In-Memory Caching with moka

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Distributed Caching with Redis

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Cache Invalidation

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Chapter 17: Deployment and Operations

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Building Optimized Docker Images

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Basic Multi-Stage Build

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Further Optimization with cargo-chef

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Static Binary with musl

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Reverse Proxies: Nginx and Caddy

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Nginx Configuration

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Caddy as a Simpler Alternative

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Docker Compose for Local Development

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

TLS Termination and Certificate Management

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Let's Encrypt for Free Certificates

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Certificates in Containerized Environments

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

mTLS for Service-to-Service Communication

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Graceful Shutdown and Health Checks

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Graceful Shutdown Implementation

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Health Check Endpoints

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

CI/CD Pipelines for Rust Services

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

GitHub Actions Example

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Deployment Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Production Monitoring and Alerting

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Essential Metrics to Monitor

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Alerting Rules

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Log Aggregation

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Conclusion: The Path Forward

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

What Makes This Stack Special

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Key Principles to Remember

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Where to Go From Here

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Continuing Your Rust Journey

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Final Thoughts

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

References

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Official Documentation

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Axum Repository and Examples

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Axum Announcements and Blog Posts

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Rust Web Framework Comparisons

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Authentication and Security

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Database and SQLx

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Testing

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

OpenAPI and Documentation

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Observability

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Deployment and Docker

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Tokio Runtime

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Additional Resources

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.

Community and Learning

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/modernwebdevelopmentwithrustandaxum>.