TypeScript Mastery 2026

FROM FIRST PRINCIPLES TO PRODUCTION-READY APPLICATIONS

TYPE-LEVEL PROGRAMMING
Master TypeScript's powerful type system.

MODERN ARCHITECTURE PATTERNS
Build scalable and maintainable applications.

FULL-STACK DEVELOPMENT
Build end-to-end apps with modern tools and runtimes.

BUN, DENO AND BEYOND
Leverage modern runtimes and the best ecosystem tools.

COVERS TYPESCRIPT 6.0 AND THE UPCOMING GO-NATIVE TYPESCRIPT 7.0 COMPILER

BEA RINCK

TypeScript Mastery 2026

From First Principles to Production-Ready Applications

Steve T. Publications

This book is available at https://leanpub.com/typescriptmastery2026

This version was published on 2026-07-14

Contents

From First Principles to Production-Ready Applications

TypeScript has grown from a niche superset of JavaScript into the dominant language for building modern web applications, backend services, and everything in between. This book takes you from your first type annotation through advanced type-level programming, ecosystem mastery, and production-ready architecture patterns. Every concept is explained with production-quality code examples that progressively build from simple programs to full-stack applications. Whether you are a JavaScript developer looking to add types to your workflow or an experienced engineer seeking deep mastery of TypeScript's type system, this book provides the structured progression you need. The content reflects the state of the TypeScript ecosystem as of mid-2026, including TypeScript 6.0 and the upcoming Go-native TypeScript 7.0 compiler, modern runtimes like Bun and Deno, and current best practices for scalable applications.

Introduction: The TypeScript Promise

In 2012, a Microsoft engineer named Anders Hejlsberg posted a preview of a language that would change the JavaScript ecosystem forever. TypeScript was not the first attempt to add static typing to JavaScript. Flow, Closure Compiler, and CoffeeScript had all tried before it. What made TypeScript different was not its types alone, but its design philosophy: TypeScript would never get in your way. It would compile to plain JavaScript. It would work with any JavaScript library. You could adopt it incrementally, file by file, without rewriting your application.

That promise held true, and the results speak for themselves. By 2026, TypeScript is the language of choice for more than 70 percent of professional JavaScript developers. The React team wrote React 19 in TypeScript. Next.js ships with TypeScript support out of the box. Node.js added experimental type stripping so you can run TypeScript files directly without compilation. The Go-native TypeScript 7.0 compiler promises up to ten times faster type checking, removing the last remaining friction point for some teams: wait times during development.

But TypeScript is more than a tool. It is a way of thinking about your code that pays dividends as projects grow. This book will show you why, and how to do it right.

Why TypeScript Exists

JavaScript was designed to be simple and flexible. Its dynamic typing lets you write quick prototypes and experiment freely. But flexibility has a cost. Consider this JavaScript function:

```
function calculateTotal(items) {
  return items.reduce((sum, item) => sum + item.price, 0);
}
```

It looks correct. It works when you pass it an array of objects with numeric `price` properties. But what happens when a developer passes an empty array

by mistake? Or a string? Or an object whose prices are stored as strings from a CSV import? In JavaScript, these errors surface at runtime, often in production, sometimes hours after deployment.

TypeScript catches many of these problems before your code ever runs:

```
interface LineItem {
  price: number;
  quantity: number;
}

function calculateTotal(items: LineItem[]): number {
  return items.reduce((sum, item) => sum + item.price * item.quantity, 0);
}
```

Now the compiler tells you immediately if you pass the wrong shape. Your editor shows the expected type as you type. Refactoring becomes safe because the compiler finds every reference to a changed interface. These are not theoretical benefits; they are the daily experience of developers working on codebases with hundreds of thousands of lines of TypeScript.

What TypeScript Is (and Is Not)

TypeScript is a superset of JavaScript. Every valid JavaScript program is also a valid TypeScript program. TypeScript adds type annotations, interfaces, generics, and other constructs that the compiler uses to verify your code. At compile time, all type information is stripped away, producing plain JavaScript that runs anywhere JavaScript runs.

This erasure model is both TypeScript's greatest strength and its most common source of confusion. Because types disappear at runtime, TypeScript cannot guarantee anything about the shape of data from external sources: API responses, user input, database queries, or files read from disk. TypeScript checks your code against its own type annotations; it does not validate data at runtime. For that, you need a separate validation library like Zod. This distinction between compile-time types and runtime values is fundamental to understanding TypeScript correctly.

TypeScript is also not a replacement for testing. Types catch structural errors, but they cannot verify business logic. A function that sorts an array in descending order instead of ascending will pass type checking perfectly. You

still need tests for behavioral correctness. Types and tests are complementary tools, each catching different classes of bugs.

The State of TypeScript in 2026

The TypeScript landscape in 2026 is defined by several major developments. TypeScript 6.0, released in March 2026, is the final version built on the original JavaScript codebase. It introduces stricter defaults, deprecates legacy module systems, and prepares codebases for the upcoming TypeScript 7.0, a complete rewrite of the compiler in Go that delivers up to ten times faster type checking. The 7.0 release candidate arrived in June 2026, signaling that the transition is imminent.

The runtime ecosystem has also evolved dramatically. Bun, the all-in-one JavaScript toolkit built on Apple's JavaScriptCore engine, was acquired by Anthropic in late 2025 and has become production-ready for most workloads. Deno 2.x achieved full npm compatibility while maintaining its security-first design and native TypeScript support. Node.js 22 LTS added experimental TypeScript type stripping, narrowing the gap between runtimes.

On the frontend, React 19 stabilized Server Components and Server Actions, changing how TypeScript developers structure their applications. Next.js 16 made the App Router the default, with TypeScript deeply integrated into its routing and data-fetching model. Vite remains the default build tool for new projects, while Turbopack has become the production bundler of choice within the Next.js ecosystem.

This book covers all of these developments. You will learn TypeScript as it exists today, with awareness of where the language and ecosystem are heading.

How to Read This Book

The chapters follow a deliberate progression. Part 1 (Chapters 1 through 3) establishes foundations: installation, the type system, and the everyday types you use in every program. Part 2 (Chapters 4 through 6) covers advanced language features: classes, generics, and type-level programming. Part 3 (Chapters 7 through 9) addresses project structure, configuration, and tooling. Part 4 (Chapters 10 through 12) covers application development with Node.js,

React, and Next.js. Part 5 (Chapters 13 through 15) tackles runtimes, publishing, and production concerns.

Each chapter builds on the previous ones. If you are already comfortable with TypeScript basics, you can skim Chapters 1 through 3 and jump to the advanced material. If you are new to TypeScript, work through the chapters sequentially. The code examples are designed to be read in order; later examples reference patterns introduced earlier.

All code examples use modern TypeScript features available in TypeScript 5.9 and 6.0. Where a feature requires a specific version, the example notes it. You can run all examples with any of the major runtimes: Node.js, Deno, or Bun.

Prerequisites

This book assumes you know JavaScript at an intermediate level. You should be comfortable with functions, objects, arrays, promises, and ES module syntax. If you have built a web application in JavaScript, even a small one, you have the background needed. If TypeScript is entirely new to you, that is exactly where this book begins.

No prior experience with static typing is required. The concepts of types, interfaces, and generics are explained from first principles. If you have used Java, C#, or Go before, you will find many parallels; if you have not, the explanations stand on their own.

Chapter 1: Getting Started – Installation, Tooling, and Your First Program

Before writing a single line of TypeScript, you need the toolchain in place. This chapter covers installation, project configuration with `tsconfig.json`, compilation, and the development workflow that professional TypeScript developers use daily. By the end of this chapter, you will have a working TypeScript project, understand how the compiler transforms your code, and know which tools to reach for in different scenarios.

Installing TypeScript and Setting Up Your Environment

TypeScript is distributed as an npm package. The recommended installation method depends on whether you are adding TypeScript to an existing project or starting fresh.

For a new project, initialize it first:

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

Then install TypeScript as a development dependency:

```
npm install -D typescript
```

The `-D` flag (shorthand for `--save-dev`) records TypeScript in the `devDependencies` section of your `package.json`. This is important because TypeScript is a build-time tool, not a runtime dependency. Your compiled JavaScript runs without TypeScript installed.

You can verify the installation by checking the version:

```
npx tsc --version
# Version 6.0.2
```

The `npx` command runs the locally installed version of `tsc` (the TypeScript compiler) from your project's `node_modules/.bin` directory. This ensures you always use the version specified in your project rather than a potentially outdated global installation.

Choosing a package manager. In 2026, you have several options beyond npm. The `pnpm` package manager is the default recommendation for new TypeScript projects because of its speed, disk efficiency through hard linking, and strict dependency isolation that prevents phantom import bugs. Bun's built-in package manager is the fastest option, installing packages up to thirty times faster than npm in benchmarks. Yarn remains viable, especially with Plug'n'Play mode, though its adoption has declined. All four managers work identically with TypeScript; the choice affects install speed and disk usage, not compilation behavior.

To use `pnpm` instead of npm, install it globally and replace `npm install -D typescript` with `pnpm add -D typescript`. The rest of this book uses npm commands for familiarity, but the same principles apply regardless of your package manager.

Node.js version. TypeScript compiles to any version of JavaScript, from ES3 (deprecated in TypeScript 6.0) to the latest ECMAScript proposal. However, your runtime must support the target output. For new projects, targeting ES2022 or later is recommended, which requires Node.js 16 or newer. Node.js 22 LTS is the current long-term support release as of mid-2026 and provides excellent TypeScript compatibility.

Understanding tsconfig.json – The Compiler Configuration File

The `tsconfig.json` file tells the TypeScript compiler how to process your project. It controls which files are included, what JavaScript version to target, which type-checking rules to enforce, and where output goes. Without it, TypeScript falls back to minimal defaults that are rarely appropriate for real projects.

Generate a starting configuration with:

```
npx tsc --init
```

In TypeScript 5.9 and later, this command produces a clean, prescriptive configuration rather than the verbose commented-out options of earlier versions. A typical output looks like this:

```
{
  "compilerOptions": {
    "target": "ES2022",
    "module": "NodeNext",
    "strict": true,
    "esModuleInterop": true,
    "skipLibCheck": true,
    "outDir": "./dist"
  },
  "include": ["src/**/*"]
}
```

Let us break down each option and explain why it matters.

`target` determines the ECMAScript version of the emitted JavaScript. Setting `"target": "ES2022"` means TypeScript will preserve modern syntax like optional chaining (`?.`), nullish coalescing (`??`), and `async/await`, while transforming anything newer into compatible code. For browser projects, you might target a lower version to support older browsers. For Node.js 18+, ES2022 is safe. TypeScript 6.0 changed the default to ES2025.

`module` controls how imports and exports are handled. `"NodeNext"` tells TypeScript to follow Node.js module resolution rules, which distinguish between CommonJS (`require`) and ES modules (`import`). This is the correct setting for most Node.js projects. For browser projects using a bundler like Vite, use `"ESNext"` with `"moduleResolution": "bundler"`.

`strict` is the single most important option in your `tsconfig.json`. It enables a suite of type-checking rules that catch the vast majority of common errors. When `strict` is true, TypeScript enforces:

- `noImplicitAny`: Variables and parameters must have explicit types or be inferable
- `strictNullChecks`: `null` and `undefined` are distinct types, not subtypes of everything

- `strictFunctionTypes`: Function parameter types are checked contravariantly
- `strictBindCallApply`: The `bind`, `call`, and `apply` methods check argument types
- `noImplicitThis`: The `this` keyword must have a known type
- `alwaysStrict`: Emitted JavaScript includes `"use strict"`
- `strictPropertyInitialization`: Class properties must be initialized in the constructor

There is no good reason to disable `strict` in a new project. It is the default in TypeScript 6.0. If you are migrating an existing JavaScript codebase, you can enable individual strict flags gradually.

`esModuleInterop` enables interoperability between CommonJS and ES modules. When true, you can import a CommonJS module using ES module syntax: `import express from 'express'` instead of `import * as express from 'express'`. This option adds a helper at the top of your compiled JavaScript to handle the interop. TypeScript 6.0 makes this the default and removes the ability to disable it.

`skipLibCheck` skips type checking of declaration files (`.d.ts` files) in `node_modules`. This speeds up compilation significantly and avoids errors from third-party packages with imperfect type definitions. The trade-off is that bugs in library types go undetected. For most projects, the speed benefit outweighs the risk.

`outDir` specifies where compiled JavaScript files are written. Setting `"outDir": "./dist"` means all output goes into a `dist` directory, keeping compiled files separate from source files. This is essential for clean project structure and for telling your version control system to ignore the output directory.

Beyond these core options, several others are worth enabling in production projects. The `sourceMap` option generates `.map` files that let debuggers map compiled JavaScript back to your original TypeScript source. The `declaration` option produces `.d.ts` type definition files alongside your JavaScript, which is essential if you are publishing a library. The `resolveJsonModule` option lets you import JSON files with full type inference:

```
import packageJson from './package.json' assert { type: 'json' };
console.log(packageJson.name); // Type-safe access
```

The `include` and `exclude` fields control which files the compiler processes. The `"include": ["src/**/*"]` pattern tells TypeScript to compile all files under the `src` directory. The `exclude` field lets you remove specific patterns, such as test files or build artifacts:

```
{
  "include": ["src/**/*"],
  "exclude": ["node_modules", "dist", "**/*.test.ts"]
}
```

By default, TypeScript excludes `node_modules`, `bun.lockb`, and various build output directories. You rarely need to specify these explicitly.

Your First TypeScript Program – From Source to JavaScript

With installation complete and configuration in place, write your first TypeScript file. Create a `src` directory and add `src/index.ts`:

```
// src/index.ts
interface User {
  id: number;
  name: string;
  email: string;
}

function greetUser(user: User): string {
  return `Hello, ${user.name}! Your account ID is ${user.id}.`;
}

const currentUser: User = {
  id: 1,
  name: "Alice",
  email: "alice@example.com"
};

console.log(greetUser(currentUser));
```

This program introduces several TypeScript concepts simultaneously. The `interface User` defines a type that describes the shape of a user object. The parameter `user: User` annotates the function parameter with that type. The variable declaration `const currentUser: User` explicitly types the constant.

Compile it with:

```
npx tsc
```

The compiler reads your `tsconfig.json`, processes all files matching the `include` pattern, and writes JavaScript to the `outDir`. Check the output in `dist/index.js`:

```
// dist/index.js
"use strict";
Object.defineProperty(exports, "__esModule", { value: true });
function greetUser(user) {
    return `Hello, ${user.name}! Your account ID is ${user.id}.`;
}
const currentUser = {
    id: 1,
    name: "Alice",
    email: "alice@example.com"
};
console.log(greetUser(currentUser));
```

Notice what happened. All type annotations (`: User`, `: string`, the `interface` declaration) are gone. The output is valid JavaScript that any runtime can execute. Run it with:

```
node dist/index.js
# Hello, Alice! Your account ID is 1.
```

Now try introducing an error. Change the `id` property to a string:

```
const currentUser: User = {
  id: "one", // Error: Type 'string' is not assignable to type 'number'
  name: "Alice",
  email: "alice@example.com"
};
```

Run the compiler again:

```
npx tsc
# src/index.ts:12:3 - error TS2322: Type 'string' is not assignable to type
↪  'number'.
#   12   id: "one",
#        ~~~~~~~~~~
# Found 1 error in the file. No output emitted.
```

TypeScript refuses to produce JavaScript when it finds errors. This is the default behavior; the compiler treats errors as fatal. You can override this with `noEmitOnError: false`, but doing so defeats the purpose of type checking. If the compiler cannot verify your code, you should not ship it.

Running TypeScript Directly with ts-node and Alternative Runners

Compiling to a separate directory before running works fine for production, but during development it adds friction. Every change requires recompilation before you can test. Several tools let you run TypeScript files directly, transpiling on the fly.

tsx is the current recommendation for running TypeScript scripts. It is built on esbuild, making it dramatically faster than older alternatives:

```
npm install -D tsx
npx tsx src/index.ts
# Hello, Alice! Your account ID is 1.
```

The `tsx` tool transpiles TypeScript to JavaScript in memory and executes it. It does not perform type checking, so errors like the `"one"` string above would pass silently. This is intentional: fast iteration during development, with full type checking reserved for your build pipeline and CI.

ts-node was the dominant tool for years but has largely been superseded by `tsx` due to performance. It uses the TypeScript compiler directly, which means it performs full type checking during execution. This can be useful when you want type errors to stop your script immediately, but the overhead makes it slower for frequent runs.

Bun runs TypeScript natively without any additional tooling:

```
bun src/index.ts
# Hello, Alice! Your account ID is 1.
```

Bun includes a built-in TypeScript transpiler that strips types at runtime. Like `tsx`, it does not perform type checking; use `tsc --noEmit` separately for that.

Node.js with type stripping. Starting with Node.js 22, you can run TypeScript files directly using the `--experimental-strip-types` flag:

```
node --experimental-strip-types src/index.ts
# Hello, Alice! Your account ID is 1.
```

This approach requires no additional packages. The trade-off is that it only strips erasable syntax; features like enums, namespaces, and experimental decorators are not supported. TypeScript 5.8 introduced the `--erasableSyntaxOnly` flag to help you audit your codebase for compatibility.

Deno also runs TypeScript natively:

```
deno run src/index.ts
# Hello, Alice! Your account ID is 1.
```

Deno performs type checking by default, which means errors will be reported before execution. You can disable this with `--no-check` for faster iteration.

The development workflow looks like this: use `tsx`, Bun, or Deno for fast iteration during coding; use `tsc --noEmit` in your CI pipeline and pre-commit hooks for type checking; use `tsc` with full compilation for production builds. Each tool serves a different purpose, and using them together gives you both speed and safety.

Integrated Development Environment Setup

Your editor is where you spend most of your time, and TypeScript's language service integrates deeply with modern editors to provide real-time type checking, auto-completion, go-to-definition, and refactoring tools.

Visual Studio Code has the best TypeScript integration because both are Microsoft products. The TypeScript language service runs inside VS Code, providing:

- Real-time error underlining as you type
- Hover tooltips showing types and documentation
- Auto-import suggestions
- Rename symbol across the entire project
- Go to definition and find all references
- Quick fixes for common errors (add missing properties, convert to async, etc.)

VS Code uses the TypeScript version installed in your project by default. If you want to ensure it uses the right version, open the Command Palette (`Ctrl+Shift+P` / `Cmd+Shift+P`) and select "TypeScript: Select TypeScript Version", then choose "Use Workspace Version".

Neovim users can get excellent TypeScript support through the combination of `nvim-lspconfig` and the `typescript-language-server` package. The setup requires installing the language server separately:

```
npm install -g typescript-language-server
```

Then configure your LSP client to use it. The experience approaches VS Code quality, with type hints, diagnostics, and code actions.

JetBrains IDEs (WebStorm, IntelliJ IDEA Ultimate) include TypeScript support out of the box. They use their own indexing engine rather than the official TypeScript language server, which means some features work differently. JetBrains IDEs are particularly strong for large refactoring operations and cross-file navigation in massive codebases.

Regardless of your editor, enable format-on-save with Prettier or the built-in TypeScript formatter. Consistent formatting eliminates visual noise and lets you focus on logic during code review.

Common Beginner Pitfalls and How to Avoid Them

Every TypeScript beginner encounters the same set of frustrations. Understanding them in advance saves hours of debugging.

The "any" trap. When TypeScript cannot infer a type, it defaults to `any` (unless `noImplicitAny` is enabled). The `any` type disables all type checking for that value, effectively turning TypeScript back into JavaScript for that variable. Never use `any` as a first resort. If you do not know the type, use `unknown`, which forces you to narrow the type before using the value. If you need to suppress an error temporarily, add a comment explaining why, and follow up later.

Forgetting that TypeScript does not validate runtime data. A common mistake is assuming that because a variable is typed as `User`, any value assigned to it is guaranteed to have the correct shape. TypeScript checks assignments in your code, but data from APIs, databases, and user input bypasses the type system entirely. Always validate external data at runtime using a library like Zod.

Over-annotating. TypeScript's type inference is remarkably powerful. In many cases, explicit annotations add no value:

```
// Unnecessary annotation
const name: string = "Alice";

// TypeScript infers the type correctly
const name = "Alice"; // name has type string
```

Annotate function return types, complex expressions, and any place where inference might be ambiguous. For simple variable declarations initialized with a literal value, let TypeScript infer the type.

Confusing `interface` and `type`. Both define types, and for most purposes they are interchangeable. The differences are subtle: interfaces can be merged (declaration merging), extended with `implements`, and reopened to add properties. Type aliases support union types, intersection types, mapped types, and other advanced constructs. The practical guideline is: use `interface` for object shapes that might be extended (APIs, class contracts) and `type` for everything else.

Ignoring the compiler output directory. If you forget to add `dist/` (or whatever your `outDir` is) to `.gitignore`, you will accidentally commit

compiled JavaScript alongside your TypeScript source. This causes confusion when team members see changes in both `.ts` and `.js` files. Always configure `.gitignore` immediately after creating your project.

Chapter Summary

This chapter covered the complete setup process for a TypeScript project. You installed TypeScript, created a `tsconfig.json` with the essential compiler options, wrote and compiled your first TypeScript program, learned how to run TypeScript directly during development, and configured your editor for maximum productivity. You also saw common beginner mistakes and how to avoid them.

The key takeaway is that TypeScript's value comes from the combination of the compiler, your editor's language service, and a thoughtful configuration. Invest time in getting these right at the start of every project, and they will pay dividends throughout development. In the next chapter, you will dive deep into TypeScript's type system, understanding how types work, how inference operates, and how to model real-world data structures with precision.

Chapter 2: The Type System Foundations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Primitive Types – Strings, Numbers, Booleans, BigInt, Symbol, and More

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Structural Typing – Duck Typing in a Statically Typed World

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Type Annotations vs Type Inference – When to Annotate and When Not To

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Interfaces vs Type Aliases – The Design Trade-offs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Union Types and Discriminated Unions – Modeling Real-World Data

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Literal Types and Their Practical Applications

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Chapter Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Chapter 3: Functions, Objects, and Arrays

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Function Type Annotations – Parameters, Return Types, and Overloads

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Object Types – Properties, Optional Fields, and Index Signatures

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Array Types and Tuples – Fixed-Length Sequences with Type Safety

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

The `any` Type and the `unknown` Type – Escape Hatches and Their Costs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Void, Never, and Undefined – Understanding Absence in TypeScript

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Practical Patterns – Configuration Objects, Event Handlers, and Callbacks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Chapter Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Chapter 4: Classes, Interfaces, and Object-Oriented Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Classes and Access Modifiers – Public, Private, Protected

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Abstract Classes and Implementation Contracts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Interfaces and Implementation – Multiple Inheritance of Type

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Enums – String Enums, Numeric Enums, and Const Enums

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Readonly Properties and Immutability Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Mixins and Decorators – Advanced Class Composition

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Chapter Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Chapter 5: Generics – The Heart of Reusable Code

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Generic Functions – Writing Type-Safe Reusable Logic

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Generic Classes and Interfaces – Parameterized Types

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Generic Constraints – Bounding Type Parameters with `extends`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

The Default Type Parameter – When Generics Need a Fallback

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Conditional Types – If/Else for the Type System

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Mapped Types – Transforming Types Programmatically

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Chapter Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Chapter 6: Advanced Type Programming

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Built-in Utility Types – Partial, Required, Pick, Omit, Record, and More

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Template Literal Types – String Manipulation at the Type Level

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

The `infer` Keyword – Extracting Types from Conditional Expressions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Recursive Conditional Types – Before and After TypeScript 4.5

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Variance – Covariance, Contravariance, and Bivariance in Practice

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Type-Level Programming Patterns – Build Your Own Utility Types

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Chapter Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Chapter 7: Modules, Namespaces, and Project Structure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

ES Modules vs CommonJS – Module Systems in the TypeScript World

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Import and Export Patterns – Named Exports, Default Exports, Re-exports

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Path Aliases and tsconfig.json Configuration for Large Projects

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Namespaces – Legacy Patterns and Modern Alternatives

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Declaration Files – .d.ts Files and Ambient Declarations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Monorepo Architecture – Managing Multiple Packages with TypeScript

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Chapter Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Chapter 8: Asynchronous Programming in TypeScript

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Promises and the Promise Type – Understanding Asynchronous Values

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Async/Await with Type Safety – Error Handling Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Generators and Iterators – Lazy Evaluation in TypeScript

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Event Emitters and Observables – Reactive Patterns with Types

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Cancellation and Timeout Patterns – AbortController and Beyond

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Concurrency – Parallel Execution, Race Conditions, and Type-Safe Coordination

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Chapter Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Chapter 9: Configuration, Linting, Formatting, and Build Tools

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

tsconfig.json Deep Dive – Every Compiler Option Explained

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Strict Mode – Enabling Maximum Type Safety

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

ESLint with TypeScript – Rulesets, Plugins, and Custom Rules

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Prettier Integration – Consistent Code Formatting at Scale

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Build Tools – Vite, esbuild, Turbopack, and Bun for TypeScript Projects

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

CI/CD Pipeline Integration – Type Checking in Continuous Integration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Chapter Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Chapter 10: Testing and Debugging TypeScript Applications

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Unit Testing with Vitest – Type-Safe Test Frameworks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Integration Testing – End-to-End Tests with Playwright and Cypress

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Mocking and Stubbing – Type-Safe Test Doubles

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Debugging TypeScript – Source Maps, Breakpoints, and the Debugger

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Type-Level Testing – Expect Types and Runtime Verification

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Performance Profiling – Finding Bottlenecks in TypeScript Applications

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Chapter Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Chapter 11: TypeScript with Node.js – Backend Development

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Setting Up a TypeScript Node.js Project – Tooling and Configuration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Express with TypeScript – Type-Safe Routes, Middleware, and Controllers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Database Integration – TypeORM, Prisma, and Drizzle ORM

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

API Design – RESTful APIs, GraphQL, and Type-Safe Request/Response

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Authentication and Authorization – JWT, Sessions, and OAuth with Types

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Deployment – Docker, Serverless, and Production Configuration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Chapter Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Chapter 12: TypeScript with React and Next.js – Frontend Development

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

React with TypeScript – Props, State, and Type-Safe Components

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Hooks and Context – Typing useReducer, useContext, and Custom Hooks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Next.js App Router – Server Components, Client Components, and Routing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

State Management – Zustand, Redux Toolkit, and Jotai with Types

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Forms and Validation – React Hook Form, Zod, and Type-Safe Schemas

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Styling with TypeScript – CSS Modules, Tailwind, and Styled Components

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Chapter Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Chapter 13: Modern Runtimes – Node.js, Deno, and Bun

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Node.js – The Established Runtime – LTS Releases and Ecosystem

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Deno – Secure by Default – Native TypeScript Support and Permissions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Bun – Speed First – Built-in Bundler, Test Runner, and Package Manager

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Cross-Runtime Compatibility – Writing Code That Runs Everywhere

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Performance Benchmarks – Real-World Comparisons in 2026

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Choosing Your Runtime – Decision Framework for Different Projects

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Chapter Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Chapter 14: Publishing Libraries and Package Management

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Package Managers in 2026 – npm, Yarn, pnpm, and Bun Install

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Building a TypeScript Library – Project Structure and Configuration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Publishing to npm – Versioning, Tags, and Access Levels

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Dual ESM/CommonJS Packages – Supporting Both Module Systems

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Distributing Type Definitions – .d.ts Files and Declaration Maps

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Maintaining Libraries – Changelogs, Breaking Changes, and SemVer

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Chapter Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Chapter 15: Performance, Optimization, and Production Best Practices

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Compilation Performance – Project References, Incremental Builds, and Caching

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Bundle Size Optimization – Tree Shaking, Code Splitting, and Dead Code Elimination

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Runtime Performance – How Types Affect Generated JavaScript

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Error Handling Strategies – Global Error Boundaries and Recovery Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Logging, Monitoring, and Observability – Type-Safe Telemetry

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Architectural Patterns for Scale – Domain-Driven Design with TypeScript

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Chapter Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Conclusion: The Future of TypeScript

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

Where TypeScript Is Heading

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

The Broader Trend: Typed Web Development

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

What Mastery Looks Like

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

How to Keep Growing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.

References

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/typescriptmastery2026.