Study Plan — Effective TypeScript (2nd Edition)

User Story Template & Examples mapped to the 10 chapters

Assumes TypeScript 5.x, Node 18+, pnpm/npm, VS Code + TS extension.

How to use this template

For each chapter, duplicate the **Template Card** and tailor the fields. Keep cards *user-centered*, *small*, and *testable*. Use the badges to mark non-functional focus areas.

Required data on every card

- **Epic / Feature** → where this card rolls up (e.g., "Type System Mastery").
- **Business Value** → outcomes this study work enables (e.g., safer APIs, faster PR reviews).
- **Priority / Estimate** → Must/Should/Could + SP (3–5 ideal for a study chapter).
- **Persona** → who benefits or acts (e.g., "TS app dev on a new repo").
- **Dependencies** → tools, repos, or pre-reading.
- **Assumptions / Risks** → things that could block you.
- **Story** → *As a <persona>, I want <action> so that <value>.*
- **Non-Functional** → tags like `Performance` `Security` `Reliability` `Accessibility` `Privacy` `i18n`.
- **Acceptance Criteria (BDD)** → Scenario with **Given/When/Then** that proves outcomes.
- **Tasks** → 4–8 concrete steps using checkboxes ☐.

Writing effective stories (quick guide)

- Prefer **observable outcomes** over internal activity.
- One behavior per scenario; keep steps short and readable.
- Cover **happy path**, **negative path**, and key **edge cases**.
- Include boundaries, error handling, and permissions where relevant.
- Tie outcomes to **DX**, **safety**, or **runtime correctness**.

TEMPLATE — <Concise Card Title>

Epic / Feature	<Parent initiative or feature>
Business Value	<Concrete value users/teams get from this learning>
Priority / Estimate	Priority: <Must/Should/Could> `SP: <3–5>`
Persona	<Who does this work?>
Dependencies	<Repos, tools, readings>
Assumptions / Risks	<What might go wrong or be unclear?>

Story *As a <persona>, I want <outcome> so that <value>.*

Non-Functional Performance Security Reliability Accessibility Privacy i18n

Acceptance Criteria (BDD)
Scenario Happy path
Given <Preconditions, repos and context available>
When <The hands-on objectives are executed>
Then <Observable outcome of the chapter appears in code, CI, or docs>

Definition of Ready: Persona clear; AC drafted; Dependencies known; Estimate set. • **Definition of Done:** All ACs pass; Tests green; Security/a11y checks; Docs updated; Deployed/flagged.

Tasks

- ☐ <Task 1 (concrete, 15–60 minutes)>
- ☐ <Task 2>
- ☐ <Task 3>
- ☐ <Task 4>

TS-1 — Getting to Know TypeScript

Epic / Feature	Onboarding and Fundamentals
Business Value	Shared understanding of TS vs JS, strictness posture, and tooling; reduces integration risk and confusion.
Priority / Estimate	Priority: Must SP: 3
Persona	Developer on a new repo
Dependencies	Node 18+, pnpm/npm, VS Code, TypeScript 5.x
Assumptions / Risks	Local vs CI toolchain drift; risk of initial failures after enabling strict options

Story *As a developer, I want to configure and explore TypeScript so that I correctly understand what the compiler checks and how to keep builds green.*

Non-Functional Reliability Security DX

Acceptance Criteria (BDD)
Scenario Happy path
Given a fresh repo with TypeScript installed
When strict compiler options are enabled and a sample file is compiled
Then a short `tsconfig-rationale.md` explains each option; build is green in CI

Definition of Ready: Persona clear; AC drafted; Dependencies known; Estimate set. • **Definition of Done:** All ACs pass; Tests green; Security/a11y checks; Docs updated; Deployed/flagged.

Tasks

- ☐ Initialize repo; run `tsc -init`; enable `strict`, `noUncheckedIndexedAccess`, `exactOptionalPropertyTypes`.
- ☐ Convert a small JS utility to TS; record surprises and fixes.
- ☐ Capture a "TS myths vs reality" note (5 bullets).
- ☐ Configure VS Code TS language service tips: "Go to Type Definition", quick fixes, refactors.

TS-2 — Type System (Structural Types and Assignability)

Epic / Feature	Type System Mastery
Business Value	Fewer runtime errors and clearer APIs via correct assignability and literal/widening control.
Priority / Estimate	Priority: Must SP: 5
Persona	Application developer
Dependencies	ESLint + `@typescript-eslint` configured
Assumptions / Risks	Misunderstanding variance or excess property checks can leak bugs

Story *As an app developer, I want to internalize structural typing and assignability so that functions and objects compose safely.*

Non-Functional Reliability Maintainability

Acceptance Criteria (BDD)
Scenario Happy path
Given a module with object and function types
When assignability puzzles and ESLint strict boolean checks are addressed
Then a `Result<T,E>` helper is in use and misuse causes compile errors

Definition of Ready: Persona clear; AC drafted; Dependencies known; Estimate set. • **Definition of Done:** All ACs pass; Tests green; Security/a11y checks; Docs updated; Deployed/flagged.

Tasks

- ☐ Create 10 small "type gym" puzzles covering unions, intersections, and excess property checks.
- ☐ Implement `Result<T,E>` and a function returning success/failure.
- ☐ Enable `strict-boolean-expressions`; fix violations.
- ☐ Document parameter vs return variance guidance in `type-design-checklist.md`.

TS-3 — Inference & Control Flow Analysis

Epic / Feature	Ergonomic Typing
Business Value	Less annotation noise; safer narrowing paths and clearer code.
Priority / Estimate	Priority: Must SP: 3
Persona	App developer refactoring an existing module
Dependencies	TypeScript 5.x, ESLint rules enabled
Assumptions / Risks	Over-annotation harms readability; brittle guards can be unsound

Story *As a developer, I want to rely on inference and control-flow narrowing so that code stays concise and correct.*

Non-Functional DX Reliability

Acceptance Criteria (BDD)
Scenario Happy path
Given a module with many explicit types
When redundant annotations are removed and guards/"satisfies" are introduced
Then the module compiles cleanly and has fewer lines of type noise with equal or better safety

Definition of Ready: Persona clear; AC drafted; Dependencies known; Estimate set. • **Definition of Done:** All ACs pass; Tests green; Security/a11y checks; Docs updated; Deployed/flagged.

Tasks

- ☐ Replace unnecessary annotations with inference; keep only clarifying ones.
- ☐ Implement user-defined guards: `isNonEmptyString`, `isISODate`, `isRecord<K,V>`.
- ☐ Use `satisfies` to prevent widening on literals.
- ☐ Refactor a discriminated-union `switch`; enable `noFallthroughCasesInSwitch`.

TS-4 — Type Design (APIs as Types)

Epic / Feature	API Design
Business Value	Stable, evolvable APIs; fewer breaking changes and safer refactors.
Priority / Estimate	Priority: Must SP: 5
Persona	Library/API author
Dependencies	Project with domain types (Users/Projects/Tickets)
Assumptions / Risks	Overly broad types leak implementation; missing read-only/optional semantics

Story *As an API author, I want to design composable, minimal types so that consumers enjoy a stable, discoverable surface.*

Non-Functional Reliability Maintainability DX

Acceptance Criteria (BDD)
Scenario Happy path
Given DTOs and service interfaces for a small domain
When branded IDs, `NonEmptyArray<T>`, and separate input/output types are introduced
Then an API review finds no accidental widenings; examples compile in consumer code

Definition of Ready: Persona clear; AC drafted; Dependencies known; Estimate set. • **Definition of Done:** All ACs pass; Tests green; Security/a11y checks; Docs updated; Deployed/flagged.

Tasks

- ☐ Model Users/Projects/Tickets with DTOs and service interfaces.
- ☐ Add branded/opaque IDs and a `NonEmptyArray<T>` helper.
- ☐ Introduce `readonly` and `exactOptionalPropertyTypes` where appropriate.
- ☐ Draft a “type API review” checklist and run it.

TS-5 — Unsoundness & the any Type

Epic / Feature	Risk Management
Business Value	Contain escape hatches; improve safety without blocking delivery.
Priority / Estimate	Priority: Must SP: 3
Persona	Maintainer migrating legacy code
Dependencies	ESLint rules; optional runtime validator (e.g., Zod)
Assumptions / Risks	Overuse of `any` and unsafe casts

Story *As a maintainer, I want to fence and document unsoundness so that risks are visible and limited.*

Non-Functional Security Reliability

Acceptance Criteria (BDD)
Scenario Happy path
Given a codebase with `any`-heavy areas
When a quarantine adapter and runtime validation are introduced
Then all remaining `any` usages are documented and isolated; `noImplicitAny` enforced

Definition of Ready: Persona clear; AC drafted; Dependencies known; Estimate set. • **Definition of Done:** All ACs pass; Tests green; Security/a11y checks; Docs updated; Deployed/flagged.

Tasks

- ☐ Scan for `any` hotspots; move interop behind a small typed adapter.
- ☐ Replace `any` with `unknown`+refinement or generics where feasible.
- ☐ Add a minimal runtime validator for external data.
- ☐ Enable `noImplicitAny` and fix top offenders.

TS-6 — Generics & Type-Level Programming

Epic / Feature	Reusable Abstractions
Business Value	Safer, reusable helpers without IDE slowdowns.
Priority / Estimate	Priority: Should SP: 5
Persona	Library and app developers
Dependencies	TS 5.x features; benchmarking editor responsiveness
Assumptions / Risks	Overly clever types harm readability or performance

Story *As a developer, I want to write constrained generics and moderate type-level code so that helpers are powerful yet maintainable.*

Non-Functional DX Maintainability

Acceptance Criteria (BDD)
Scenario Happy path
Given a utilities module
When helpers like `DeepReadonly<T>` and a typed event emitter are implemented
Then examples compile; editor responsiveness remains good; code is documented with examples

Definition of Ready: Persona clear; AC drafted; Dependencies known; Estimate set. • **Definition of Done:** All ACs pass; Tests green; Security/a11y checks; Docs updated; Deployed/flagged.

Tasks

- ☐ Implement `DeepReadonly<T>`, `PickByValue<T,V>`, `TupleToObject<T>` with tests.
- ☐ Build `typedEventEmitter<TEvents>` with per-event handler types.
- ☐ Record compile/editor times before/after; refactor if too slow.
- ☐ Add docs with usage snippets.

TS-7 — Recipes (Applied Patterns)

Epic / Feature	Practical Patterns
Business Value	Fewer runtime surprises in configs, async flows, and state management.
Priority / Estimate	Priority: Should SP: 5
Persona	App developer
Dependencies	Fetch wrapper, runtime validator, state mgmt (Redux/RTK or TanStack Query)
Assumptions / Risks	Loss of type information across boundaries without care

Story *As an app dev, I want applied TS recipes so that common app paths remain type-safe without casts.*

Non-Functional Reliability Security

Acceptance Criteria (BDD)
Scenario Happy path
Given config, async fetch, and reducer examples
When typed config loader, typed fetch wrapper, and action unions are implemented
Then no `as` casts exist in those paths and misuse fails to compile

Definition of Ready: Persona clear; AC drafted; Dependencies known; Estimate set. • **Definition of Done:** All ACs pass; Tests green; Security/a11y checks; Docs updated; Deployed/flagged.

Tasks

- ☐ Implement a validated `Config` loader returning precise inferred types.
- ☐ Create a `createReducer` helper using discriminated unions.
- ☐ Add a typed fetch wrapper returning `Result<T,E>`; handle errors idiomatically.
- ☐ Add unit tests for edge cases (missing keys, bad JSON, network failures).

TS-8 — Declarations & `@types`

Epic / Feature	Public Typings
Business Value	Consumers get great IntelliSense and fail fast on misuse.
Priority / Estimate	Priority: Should SP: 3
Persona	Library author
Dependencies	`tsd` or `dtslint`; sample package
Assumptions / Risks	Incorrect augmentations or breakage across versions

Story *As a library author, I want solid declaration files so that users have accurate types and documentation.*

Non-Functional DX Maintainability

Acceptance Criteria (BDD)
Scenario Happy path
Given a package with runtime JS and `index.d.ts`
When typings are tested and a third-party augmentation is added
Then consumers see accurate types; bad usage fails in `tsd` tests

Definition of Ready: Persona clear; AC drafted; Dependencies known; Estimate set. • **Definition of Done:** All ACs pass; Tests green; Security/a11y checks; Docs updated; Deployed/flagged.

Tasks

- ☐ Create a small library with runtime JS and matching declarations.
- ☐ Add `typesVersions` if needed; write `tsd` tests to lock behavior.
- ☐ Augment a third-party lib for a missing method/type; cover with tests.
- ☐ Document your public types in README/API docs.

TS-9 — Writing & Running Code (Toolchain)

Epic / Feature	Build, Test, Ship
Business Value	Boring, reliable DX and fast CI; fewer integration surprises.
Priority / Estimate	Priority: Should SP: 5
Persona	Maintainer
Dependencies	Vitest/Jest, Vite/ESBuild/Rspack; CI pipeline
Assumptions / Risks	Misaligned module/target settings, slow CI

Story *As a maintainer, I want a stable toolchain so that builds, tests, and types are consistent locally and in CI.*

Non-Functional Performance Reliability

Acceptance Criteria (BDD)
Scenario Happy path
Given build, typecheck, lint, and test scripts wired to CI
When project references and caching are enabled
Then CI is consistently green and runs in under 2 minutes for the sample monorepo

Definition of Ready: Persona clear; AC drafted; Dependencies known; Estimate set. • **Definition of Done:** All ACs pass; Tests green; Security/a11y checks; Docs updated; Deployed/flagged.

Tasks

- ☐ Add `build`, `typecheck`, `test`, `lint` scripts with fast feedback.
- ☐ Configure path aliases and project references; split into two small packages.
- ☐ Enable incremental builds and caching in CI; measure run times.
- ☐ Ensure source maps and declaration files are emitted as needed.

TS-10 — Modernization & Migration

Epic / Feature	Evolution
Business Value	Controlled migration of legacy code; repeatable upgrade playbook.
Priority / Estimate	Priority: Should SP: 3
Persona	Maintainer on a legacy app
Dependencies	JSDoc `// @ts-check`, codemod tooling
Assumptions / Risks	Risk of churn or breaking changes during upgrades

Story *As a maintainer, I want stepwise migrations so that legacy JS becomes strict TS with minimal disruption.*

Non-Functional Reliability Maintainability

Acceptance Criteria (BDD)
Scenario Happy path
Given a legacy JS folder
When code is migrated via JSDoc checks and codemods, then moved to TS files
Then `any`/`@ts-ignore` counts drop measurably; an upgrade checklist is published

Definition of Ready: Persona clear; AC drafted; Dependencies known; Estimate set. • **Definition of Done:** All ACs pass; Tests green; Security/a11y checks; Docs updated; Deployed/flagged.

Tasks
- ☐ Start with JSDoc `// @ts-check`; fix surfaced issues.
- ☐ Convert to TS files; turn on strict flags progressively.
- ☐ Write a codemod for a deprecated pattern and run repo-wide.
- ☐ Publish an "upgrade playbook" for minor/major TS versions.