45-Day TypeScript Roadmap

Goal: Become proficient in TS for a full-stack role, focusing on consistency and practical mastery.

Week 1: Solidifying Foundations

- **Day 1: Type Annotations & Type Inference**
  - *Why:* Ensures you're crystal clear on explicitly defining types vs. letting TS infer them—core to writing clean, predictable code.
  - *Focus:* Practice with primitives (string, number, boolean) and complex objects.
- **Day 2: Interfaces & Type Aliases**
  - *Why:* These are your building blocks for defining object shapes; crucial for reusable, structured code in full-stack apps.
  - *Focus:* Differences between them and when to use each.
- **Day 3: Union Types & Intersection Types**
  - *Why:* Combines types flexibly—key for handling varied data (e.g., API responses).
  - *Focus:* Practical examples like combining object types or literals.
- **Day 4: Enums & Literal Types**
  - *Why:* Enums for fixed sets (e.g., status codes), literals for specific values—great for type safety.
  - *Focus:* Numeric vs. string enums, narrowing with literals.
- **Day 5: Function Types & Optional Parameters**
  - *Why:* Functions are everywhere in full-stack; typing them correctly avoids bugs.
  - *Focus:* Typing parameters, return values, and using `?` for optional args.
- **Day 6: Arrays & Tuples**
  - *Why:* Data manipulation is constant; tuples add strictness for fixed-length arrays.
  - *Focus:* Typing arrays with generics, tuple use cases (e.g., [string, number]).
- **Day 7: Type Assertions & Non-Null Assertion**
  - *Why:* Safely override TS when you know better (e.g., DOM elements, API data).
  - *Focus:* `as` keyword, `!` operator, and pitfalls to avoid.

Week 2: Intermediate Concepts

- **Day 8: Generics (Basics) & Generic Constraints**
  - *Why:* Reusable code with type safety—vital for components or utils in full-stack.
  - *Focus:* Simple generic functions, constraining with `extends`.

- **Day 9: Modules & Namespaces**
  - *Why:* Organize code across files—a must for larger TS projects.
  - *Focus:* `import/export` syntax, when namespaces still make sense.
- **Day 10: Type Guards & Narrowing**
  - *Why:* Runtime checks refine types—huge for handling dynamic data.
  - *Focus:* `typeof`, `instanceof`, and custom guards.
- **Day 11: keyof & Lookup Types**
  - *Why:* Work with object keys dynamically—an advanced tool for APIs/utils.
  - *Focus:* Extracting keys, typing dynamic property access.
- **Day 12: Mapped Types & Conditional Types**
  - *Why:* Transform types programmatically—powerful for reusable logic.
  - *Focus:* Basic mapping, simple `T extends U ? X : Y` examples.
- **Day 13: Utility Types (Part 1)**
  - *Why:* Built-in helpers like `Partial`, `Pick`, `Omit` save time.
  - *Focus:* Use cases in object manipulation.
- **Day 14: Utility Types (Part 2)**
  - *Why:* More advanced ones like `Record`, `Exclude` for flexibility.
  - *Focus:* Practical examples (e.g., typing configs or filters).

Week 3: Advanced TS

- **Day 15: Decorators & Metadata**
  - *Why:* Used in frameworks like NestJS (backend) or Angular (frontend).
  - *Focus:* Basic class decorators, enabling experimental support.
- **Day 16: Async/Await & Promise Typing**
  - *Why:* Full-stack means APIs—typing async code is critical.
  - *Focus:* Typing `Promise<T>`, handling errors with types.
- **Day 17: Index Signatures & Excess Property Checks**
  - *Why:* Handle dynamic keys (e.g., JSON) while keeping strictness.
  - *Focus:* `[key: string]: T`, pitfalls of loose typing.
- **Day 18: Discriminated Unions & Exhaustiveness Checking**
  - *Why:* Model complex states (e.g., Redux actions) with safety.
  - *Focus:* Using a discriminant (e.g., `type` field), `never` for checks.
- **Day 19: Advanced Generics & Higher-Order Types**
  - *Why:* Push generics further for reusable, abstract code.
  - *Focus:* Generic constraints with multiple bounds, type inference.
- **Day 20: Declaration Files & Ambient Types**
  - *Why:* Integrate JS libraries in TS projects seamlessly.
  - *Focus:* Writing `.d.ts` files, using `declare`.
- **Day 21: Type Compatibility & Structural Typing**
  - *Why:* Understand how TS matches types—avoids surprises.
  - *Focus:* Duck typing, assignability rules.

Week 4: Full-Stack Relevance

- **Day 22: Typing REST APIs & Request/Response**
  - *Why:* Backend-frontend glue; ensures data contracts.
  - *Focus:* Interfaces for payloads, typing fetch/axios calls.
- **Day 23: Middleware & Express Typing**
  - *Why:* Common in Node.js backends—type safety in routing.
  - *Focus:* Typing `req`, `res`, `next` with `@types/express`.
- **Day 24: React Props & State Typing**
  - *Why:* Frontend full-stack staple—TS shines here.
  - *Focus:* Typing functional components, hooks like `useState`.
- **Day 25: Event Handling & DOM Typing**
  - *Why:* Frontend interactivity needs precise types.
  - *Focus:* `React.MouseEvent`, TS DOM types (e.g., `HTMLElement`).
- **Day 26: Redux/Action Typing**
  - *Why:* State management is common in full-stack apps.
  - *Focus:* Action creators, reducers with TS.
- **Day 27: Database Typing (e.g., Prisma/Mongoose)**
  - *Why:* Backend data layer—type your models.
  - *Focus:* Typing schemas, query results.
- **Day 28: Error Handling & Custom Errors**
  - *Why:* Robust apps need typed errors.
  - *Focus:* Extending `Error`, typing `try/catch`.

Week 5: Practical Application

- **Day 29: Configuring tsconfig.json**
  - *Why:* Control TS behavior for your project.
  - *Focus:* Key options (`strict`, `target`, `module`).
- **Day 30: Strict Mode & NoImplicitAny**
  - *Why:* Enforce discipline for team-ready code.
  - *Focus:* Fixing common `any` issues.
- **Day 31: Type Inference in Libraries**
  - *Why:* Leverage TS with tools like Lodash/zod.
  - *Focus:* Using `@types`, inferring from libs.
- **Day 32: Testing with TS (Jest)**
  - *Why:* Full-stack roles often include testing.
  - *Focus:* Typing mocks, assertions.
- **Day 33: Code Splitting & Lazy Loading**
  - *Why:* Optimize full-stack app performance.
  - *Focus:* Typing dynamic imports.
- **Day 34: Generics in Real-World Utils**
  - *Why:* Write sharable, typed helpers.

    - *Focus:* Example: generic API fetcher.
- **Day 35: Debugging TS Errors**
    - *Why:* Master fixing cryptic TS messages.
    - *Focus:* Common errors (e.g., `TS2345`), solutions.

Week 6 + 3 Days: Mastery & Polish

- **Day 36: Refactoring JS to TS**
    - *Why:* Real-world skill—migrate legacy code.
    - *Focus:* Step-by-step conversion example.
- **Day 37: Performance & TS Trade-offs**
    - *Why:* Know when TS helps or hinders.
    - *Focus:* Compile-time vs. runtime considerations.
- **Day 38: TS with WebSockets**
    - *Why:* Real-time full-stack apps (e.g., chat).
    - *Focus:* Typing messages, events.
- **Day 39: Combining Types for Complex State**
    - *Why:* Model app state realistically.
    - *Focus:* Nested unions, intersections.
- **Day 40: TS in CI/CD Pipelines**
    - *Why:* Team workflows rely on automation.
    - *Focus:* Linting, type checking in CI.
- **Day 41: Best Practices Recap**
    - *Why:* Solidify habits for team code.
    - *Focus:* Naming, structure, comments.
- **Day 42: Mock Project (Frontend)**
    - *Why:* Apply TS in a React mini-app.
    - *Focus:* Component typing, API calls.
- **Day 43: Mock Project (Backend)**
    - *Why:* TS in Node.js/Express API.
    - *Focus:* Routes, middleware typing.
- **Day 44: Code Review Simulation**
    - *Why:* Prep for team feedback.
    - *Focus:* Spotting/fixing TS issues in sample code.
- **Day 45: Final Q&A & Confidence Check**
    - *Why:* Tie up loose ends, solidify knowledge.
    - *Focus:* Review weak spots, discuss scenarios.