Are you looking to unlock the full potential of TypeScript? Our Advanced TypeScript course might be just what you need. This 2-day course is designed to provide developers with an in-depth understanding of advanced language features, design patterns, and real-world applications. By mastering decorators and functional programming with TypeScript, you'll be able to elevate your skills and take on more sophisticated TypeScript development projects. Through challenging projects and hands-on experience, you'll optimize performance and explore advanced tooling. This course is perfect for developers who want to become TypeScript experts or enhance their development team's capabilities. Join us on this transformative journey and unleash the power of advanced TypeScript in your projects.

Objectives

The objectives of the course are to help students understand the benefits of using TypeScript over JavaScript, learn how to use TypeScript's type system to write maintainable code, and master advanced TypeScript concepts such as generics, utility types, and decorators.

Additionally, the course aims to provide students with a deep understanding of functional programming in TypeScript and how to use it to build scalable and maintainable applications.

By the end of the course, students will be able to effectively use TypeScript to build complex applications that are easy to maintain and scale.

Prerequisites

To succeed in this course, you will need:

- 6 months of working knowledge of TypeScript
- Familiarity with JavaScript ES6 syntax

Day 1

1 TypeScript and Tools

1.1 TypeScript IDEs and Tools

- IDEs and editors
  - → Code editors and extensions
  - → Debugging tools
  - → TypeScript compiler options
- Typescript Configuration
  - → What is tsconfig?
  - → Using tsconfig in a project
- Type Declaration in TypeScript
  - → What are .d.ts files?
  - → How to create .d.ts files?
  - → Importing and using the JS file
- Advantages of Type-Driven Development

2 Advanced Generics

2.1 Overview of Generics in TypeScript

- Languages that support generics
- Generic Functions and Classes
- Using Type Parameters
- Exercises:

...1 Generic Print Function
...2 Generic Get Length of An Array Function
...3 Generic Box Class
...4 Generic Pair Class

2.2 Using Constraints with Generics

- Introduction to Constraints
- Applying Constraints to Type Parameters
- Exercises:

...1 Generic Get Length of An Array
...2 Type-Safe Generic Company Class
...3 Generic Logging Length Function

2.3 Understanding Type Inference with Generics

- Overview of Type Inference
- Type Inference with Generic Functions
- Exercises:

...1 Inferring Types for a Generic Function

3 Advanced Types and Utility Types

3.1 Creating Generic Utility Types

- Introduction to Utility Types
- Overview of the Built-in Utility Types
- Creating Custom Utility Types
- Exercises:

...1 Creating a Partial Utility Type

3.2 Using Mapped Types

- Overview of Mapped Types
- Creating Mapped Types with Readonly, Excluded, and Partial
- Exercises:

...1 Creating a Read-Only Mapped Type
...2 Creating an Excluded Mapped Type
...3 Creating a Partial Mapped Type
...4 Creating a FilterPropertiesOfType of a Mapped Type

3.3 Using Conditional Types

- Overview of Conditional Types
- Exercises:

...2 Using extends for conditional types

4 TypeScript Decorators and Metadata

4.1 Introduction to decorators

- Overview of decorators
- Basic syntax and usage of decorators
- Different types of decorators: class decorators, method decorators, property decorators, parameter decorators
- Exercises:

...1 Basic Decorators
...2 Decorated Property, Method, Parameter

4.2 Creating custom decorators

- Implementing custom decorators using TypeScript syntax
- Decorator factories: creating decorators that accept parameters
- Use cases and scenarios for creating custom decorators
- Exercises:

...1 Log Class using a custom decorator
...2 Log Method using a custom decorator
...3 Validate String using a custom decorator

4.3 Lifecycle of decorators

- Understanding the order of decorator execution
- Initialization, instantiation, and destruction phases of decorators
- Interactions between multiple decorators applied to the same entity
- Exercises:

...1 Log Decorator using a simple class
...2 Log Lifecycle using a database service

4.4 Decorator composition:

- Combining multiple decorators to achieve complex behavior
- Nesting decorators within each other
- Best practices for composing decorators effectively
- Exercises:

...1 Nesting Caching with Log Decorator
...2 Validate Params Decorator
...3 Authorization Decorator
...4 Nesting Authorization and Validate Params Decorator

5 Advanced Functions

5.1 Function Overloading

- Understanding function overloading syntax
- Practical examples demonstrating function overloading
- Benefits and limitations of function overloading
- Exercises:

...1 Addition
...2 Process Input
...3 Formatting Date
...4 Greet with Optional parameters

5.2 Higher-Order Functions

- Defining higher-order functions

- Passing functions as arguments
- Returning functions from other functions
- Implementing common higher-order functions (map, filter, reduce)
- Exercises:

...1 Using Callback in high-order function
...2 Passing functions as arguments
...3 Returning a function
...4 Implementing the Mapper function to double the numbers
...5 Using Numbers, Map, Filter, and Reduce
...6 Event Handling with a Callback in an HTML page

5.3 Types Alias

- Type Alias
- Interface vs Type Alias
- Exercises:

...1 Using type alias

6 Functional Programming in TypeScript

6.1 Type inference and type safety

- Leveraging TypeScript's type inference for functional programming
- Ensuring type safety in functional programming with TypeScript's type system
- Handling complex type scenarios in functional programming
- Exercises:

...1 Using infer in parameters
...2 Using infer in return types

6.2 Union types and pattern matching

- Understanding union types and discriminated unions
- Implementing pattern matching with union types
- Use cases and examples of pattern matching in functional programming
- Exercises:

...1 Phone Union Type
...2 Print Id Method Argument with Union Type
...3 Shape Interface with Discriminated Union Type
...4 Shape and Vehicle Types with Discriminated Union Types
...5 Welcome Message Based on User Type

6.3 Higher-kinded types and type-level programming

- Introduction to higher-kinded types and type-level programming
- Leveraging type-level programming techniques in TypeScript

- Advanced type manipulations for functional programming scenarios
- Exercises:

...1 Option Type
...2 Type Safe API response
...3 Type Manipulations on Tuple

Day 2

7 Functional Composition in TypeScript

7.1 Functional composition

- Definition of functional composition
- Composing functions together to create new functions
- Techniques for composing functions
- Function chaining, function composition operators
- Exercises:

...1 Compose add and multiply by 2 functions
...2 Compose calculate total price and apply discount functions

7.2 Currying and partial application

- Understanding currying and partial application
- Transforming functions into a curried form
- Use cases and benefits of currying and partial application
- Exercises:

...1 Curried add function
...2 Curried calculate Total Price function

7.3 Function purity and immutability

- Explaining function purity and immutability
- Benefits of pure functions and immutability in functional programming
- Techniques for ensuring purity and immutability in TypeScript
- Exercises:

...1 Square Pure function with immutability
...2 Cart Item Pure function with immutability

8 Exhaustiveness Checking, Type Assertions, Type Guards

8.1 Exhaustiveness Checking

- Using the **never** type
- Exercises:

...1 Using the **never** Type in a Switch Statement
...2 Animal Type with **never** usage
...3 Direction with **never**-usage
...4 Fruit type with **never** usage

8.2 Type assertions

- What are type assertions (aka type casting)?
- Why are they sometimes necessary?
- When can they be unsafe?
- When to use **unknown** instead of **any**
- When to use **as** versus **unknown**
- Using the **never** type to catch unsafe type assertions at compile time
- Exercises:

...1 Asserting unknown to string

...2 Using Animal with Type Assertion

8.3 Using type guards for safer type assertions

- What are type guards?
- How to write type guards for common scenarios
- Using **typeof**, **instanceof**, and custom type guards
- Combining type guards with type intersections and unions
- Exercises:

...1 Up Case function with t**ypeof** usage

...2 Greet user function within operator check

...3 Log Area function with an **instanceof** check

...4 Divide safely

...5 Type Guards with Bird or Fish

9 Advanced Object-Oriented Design Patterns

9.1 Implementing Singleton Pattern

- Overview of the Singleton pattern
- Implementing a Singleton in TypeScript using a private constructor and a static instance
- Ensuring thread safety and lazy initialization in a Singleton
- Exercises:

...1 Cache Manager

...2 Configuration Manager

9.2 Implementing Factory Pattern

- Introduction to the Factory Pattern
- Implementing a Factory pattern in TypeScript to create objects without exposing the instantiation logic
- Different variations of the Factory pattern: simple factory, factory method, abstract factory
- Exercises:

...1 Shape Factory

...2 Document Factory

9.3 Implementing Strategy Pattern

- Understanding the Strategy pattern and its use cases
- Implementing a Strategy pattern in TypeScript to encapsulate algorithms and behaviors
- Applying the Strategy pattern to enable runtime algorithm selection and flexibility
- Exercises:

...1 Sorting Strategy
...2 Shipping Strategy

9.4 Applying principles of SOLID in TypeScript

- Single Responsibility Principle (SRP)
- Open/Closed Principle (OCP)
- Liskov Substitution Principle (LSP)
- Interface Segregation Principle (ISP)
- Dependency Inversion Principle (DIP)
- Exercises:

...1 User Service - SRP
...2 File Manager – SRP
...3 Shape – OCP
...4 Report Generator – OCP
...5 Animal – LSP
...6 Payment Gateway – LSP
...7 All-In-One Printer and Scanner – ISP
...8 Authenticator – ISP
...9 User Manager – DIP
...10 Notification Manager - DIP

10 Error Handling and Asynchronous Patterns

10.1 Advanced error-handling techniques

- Error propagation and handling
- Custom error classes
- Error recovery strategies
- Error boundaries in frameworks
- Exercises:

...1 Division By Zero with TCB
...2 Error handling in a Web Server using TCB
...3 Creating and Throwing Custom Error
...4 File Processing Application Error Handling
...5 Fetch Data with a Recovery Mechanism
...6 Process Payment with Retry Mechanism
...7 Error Boundaries in React

10.2 Patterns for asynchronous programming

- Introduction to Callback
- What is a Pyramid of Doom?
- What is a Promise?
- Understanding Resolve and Reject
- Promise Factory Pattern
- Promise Concurrency
- Introduction to async/await
- When do you use async and await
- Exercises:

...1 Callback
...2 Order and deliver the book using Promises
...3 Gasoline purchase using Promise Chain
...4 Purchase Products using Promise Factory
...5 Get user data using async/await
...6 Fetch data using async/await

11 Advanced Module Systems and Bundling

11.1 Deep dive into ES6 modules and namespaces

- ES6 modules
- Namespaces
- Module resolution
- Ambient modules
- Exercises:

...1 Create a module and import that into another one
...2 Create auth.ts, data.ts, ui.ts and import them in app.ts
...3 Create a namespace UI in ui.ts and use that in app.ts
...4 Using a moment.js ambient module in a typescript project

11.2 Optimizing bundling and code splitting in TypeScript

- Bundling tools and techniques
- Code splitting
- Dynamic module loading
- When not to use dynamic load module loading?
- Tree shaking and dead code elimination

12 Advanced Testing Strategies

12.1 Comprehensive testing of TypeScript code

- Unit testing (Jest)
- Mocking and stubbing
- Coverage analysis
- Dependency injection for testing
- Mocking external services and APIs