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

Building a Shopping Cart Application

Project 1: The objective of this project is to build a shopping cart application using TypeScript. The application should allow users to add items to their cart, update quantities, remove items, and calculate the total cost. Additionally, the application should incorporate various design patterns and advanced TypeScript features covered in the course.

Note: Simple project, Students are expected to finish within a day.

Requirements:

1. **Product and Cart Data Structures**: Define TypeScript interfaces or classes to represent products and cart items. Utilize advanced types such as generics, utility types, and mapped types to ensure type safety.

2. **Product Catalog**: Create a module that contains a collection of available products. Use decorators to add metadata or validation to the product data.

3. **Shopping Cart Management**: Implement a shopping cart module that handles adding, updating, and removing items from the cart. Use higher-order functions (e.g., map, filter, reduce) and functional programming principles (e.g., pure functions, immutability) to manipulate the cart data.

4. **Discounts and Promotions**: Create a strategy pattern to handle different discount and promotion strategies (e.g., flat rate discount, percentage discount, buy-one-get-one-free). Decorators can be used to apply discounts or promotions to specific products or cart items.

5. **Error Handling**: Implement error handling using custom error classes, error propagation, and error recovery strategies. Use the never type for exhaustiveness checking and type guards for safer type assertions.

6. **Asynchronous Operations**: Simulate asynchronous operations, such as fetching product data from an API or processing payments, using promises or async/await.

7. **Testing**: Write comprehensive tests for the application using a testing framework like Jest. Practice mocking, stubbing, and dependency injection for testing purposes.

This capstone project covers a wide range of topics from the TypeScript course, including generics, utility types, decorators, functional programming, error handling, asynchronous patterns, modules, testing, and design patterns. It allows students to apply their knowledge practically and comprehensively, fostering a deeper understanding of TypeScript's advanced features and best practices.

Building a Financial Portfolio Management System

Project 2: The objective of this project is to create a financial portfolio management system using TypeScript. The system should allow users to manage their investments, track portfolio performance, and simulate various financial scenarios. Additionally, the application should incorporate advanced TypeScript features and design patterns covered in the course.

Note: Intermediate level complexity, Students may not be able to finish within a day

Requirements:

1. **Investment Data Structures**: Define TypeScript interfaces or classes to represent different types of investments, such as stocks, bonds, mutual funds, and real estate. Utilize advanced types like generics, utility types, and mapped types to ensure type safety.

2. **Portfolio Management**: Implement a module to manage a user's investment portfolio. This module should handle adding, updating, and removing investments from the portfolio. Use higher-order functions (e.g., map, filter, reduce) and functional programming principles (e.g., pure functions, immutability) to manipulate the portfolio data.

3. **Performance Tracking**: Create a module to track the performance of individual investments and the overall portfolio. Use decorators to add metadata or validation to the performance data.

4. **Investment Strategies**: Implement a strategy pattern to handle different investment strategies (e.g., growth strategy, income strategy, balanced strategy). Decorators can be used to apply specific strategies to individual investments or the entire portfolio.

5. **Risk Analysis**: Develop a module to analyze the risk associated with different investments or the overall portfolio. Use conditional types and type guards for safer type assertions and exhaustiveness checking.

6. **Tax Calculations**: Create a module to calculate the tax implications of investment activities, such as capital gains, dividends, and interest income. Use function overloading and type aliases to handle different tax calculation scenarios.

7. **Asynchronous Operations**: Simulate asynchronous operations, such as fetching real-time market data or executing trades, using promises or async/await.

8. **Error Handling**: Implement error handling using custom error classes, error propagation, and error recovery strategies. Use the never type for exhaustiveness checking and type guards for safer type assertions.

9. **Testing**: Write comprehensive tests for the application using a testing framework like Jest. Practice mocking, stubbing, and dependency injection for testing purposes.

This capstone project covers a wide range of topics from the TypeScript course, including generics, utility types, decorators, functional programming, error handling, asynchronous patterns, modules, testing, and design patterns. It allows students to apply their knowledge practically and comprehensively, while also aligning with the financial planning, insurance, and investment domain of the company.