Advanced TypeScript

Contents

[Step 1: Initialize a TypeScript Project 1](#_Toc185710591)

[Step 2: Install Additional Dependencies 1](#_Toc185710592)

[Step 3: Project Structure 2](#_Toc185710593)

[Step 4: Implementing the Code 2](#_Toc185710594)

[Step 5: Compile and Run 8](#_Toc185710595)

[Step-by-Step Guide to Set Up Webpack 8](#_Toc185710596)

[1. Install Webpack and Webpack CLI: 8](#_Toc185710597)

[2. Install Additional Dependencies: 8](#_Toc185710598)

[3. **Update** tsconfig.json: 8](#_Toc185710599)

[4. **Create** webpack.config.js: 9](#_Toc185710600)

[5. **Create** src/index.html: 10](#_Toc185710601)

[6. **Create** src/index.ts: 10](#_Toc185710602)

[7. **Update** package.json Scripts: 11](#_Toc185710603)

[8. **Running** the Project 11](#_Toc185710604)

# Step 1: Initialize a TypeScript Project

First, initialize a new Node.js project and install TypeScript:

mkdir advanced-typescript

cd advanced-typescript

npm init -y

npm install typescript --save-dev

npx tsc --init

# Step 2: Install Additional Dependencies

We'll also install some useful packages:

npm install ts-toolbelt utility-types

# Step 3: Project Structure

Create the following project structure:

advanced-typescript/

├── src/

│ ├── interfaces.ts

│ ├── types.ts

│ ├── guards.ts

│ ├── utils.ts

│ ├── classes.ts

│ └── functions.ts

├── tsconfig.json

└── package.json

# Step 4: Implementing the Code

**src/interfaces.ts**

// Advanced TypeScript Interfaces

// Demonstrates advanced features like optional properties, readonly properties, and method signatures.

interface User {

readonly id: number; // Readonly property

name: string;

age?: number; // Optional property

greet: (message: string) => void; // Method signature

}

const user: User = {

id: 1,

name: 'Alice',

greet: (message: string) => console.log(message)

};

// Differentiating Object Literals and Type Literals

// An object literal has its own properties and methods.

const objectLiteral = {

id: 2,

name: 'Bob',

greet: (message: string) => console.log(message)

};

// A type literal uses an interface to define the shape.

const typeLiteral: User = {

id: 3,

name: 'Charlie',

greet: (message: string) => console.log(message)

};

user.greet('Hello, Alice!');

objectLiteral.greet('Hello, Bob!');

typeLiteral.greet('Hello, Charlie!');

**src/types.ts**

// Advanced TypeScript Types

// Demonstrates using string and number literals as types.

type Status = 'success' | 'error'; // String literal types

type Quantity = 1 | 2 | 3 | 4 | 5; // Number literal types

const customStatus: Status = 'success'; // Renamed to avoid conflict

const quantity: Quantity = 3;

// Type Aliases, Unions, and Intersections

type StringOrNumber = string | number; // Union type

type Name = { name: string };

type Age = { age: number };

type Person = Name & Age; // Intersection type

const person: Person = { name: 'David', age: 30 };

**src/guards.ts**

// Type Manipulation and Guards

// Smart Casting and Type Guards

function isString(value: unknown): value is string {

return typeof value === 'string';

}

function example(value: string | number) {

if (isString(value)) {

console.log('String:', value); // TypeScript now knows value is a string

} else {

console.log('Number:', value); // TypeScript now knows value is a number

}

}

// Defining Mapped Types and the infer Keyword

type CustomReadonly<T> = {

readonly [P in keyof T]: T[P];

};

type ExtractFunctionReturnType<T> = T extends (...args: any[]) => infer R

? R

: never;

function add(a: number, b: number): number {

return a + b;

}

type AddReturnType = ExtractFunctionReturnType<typeof add>; // number

**src/utils.ts**

// Utility Types and Programming

// Utility Types and Type Programming

import { O } from 'ts-toolbelt';

import { Assign } from 'utility-types';

interface A { a: number; }

interface B { b: string; }

type Merged = O.Merge<A, B>; // { a: number; b: string; }

type PartialA = Partial<A>; // { a?: number; }

type ReadonlyA = Readonly<A>; // { readonly a: number; }

// Utilizing ts-toolbelt and utility-types

const merged: Merged = { a: 1, b: 'merged' };

const partialA: PartialA = {};

const readonlyA: ReadonlyA = { a: 5 };

// ts-toolbelt example

type RequiredAndReadonly = O.Required<O.Readonly<A>>; // { readonly a: number; }

// utility-types example

type Combined = Assign<{ c: boolean }, { a: number; b: string }>; // { c: boolean; a: number; b: string }

const combined: Combined = { c: true, a: 10, b: 'combined' };

**src/classes.ts**

// Advanced Class and Symbol Usage

// Applying Class Decorators

function sealed(constructor: Function) {

Object.seal(constructor);

Object.seal(constructor.prototype);

}

@sealed

class Greeter {

message: string;

constructor(message: string) {

this.message = message;

}

greet() {

console.log(this.message);

}

}

const greeter = new Greeter('Hello, world!');

greeter.greet();

// Understanding Symbols

const sym1 = Symbol('key1');

const sym2 = Symbol('key2');

const symbolObject = {

[sym1]: 'value1',

[sym2]: 'value2'

};

console.log(symbolObject[sym1]); // value1

console.log(symbolObject[sym2]); // value2

**src/functions.ts**

// Advanced TypeScript Functions

// Working with Generator Functions

function\* idGenerator() {

let id = 1;

while (true) {

yield id++;

}

}

const gen = idGenerator();

console.log(gen.next().value); // 1

console.log(gen.next().value); // 2

// Writing Asynchronous TypeScript

async function fetchData(url: string): Promise<any> {

const response = await fetch(url);

const data = await response.json();

return data;

}

fetchData('https://jsonplaceholder.typicode.com/todos/1')

.then(data => console.log(data))

.catch(error => console.error(error));

# Step 5: Compile and Run

Compile the TypeScript code:

**npx tsc**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Step-by-Step Guide to Set Up Webpack

## Install Webpack and Webpack CLI:

npm install webpack webpack-cli webpack-dev-server --save-dev

## Install Additional Dependencies:

npm install ts-loader html-webpack-plugin --save-dev

## **Update** tsconfig.json:

Make sure tsconfig.json is configured correctly for module resolution:

{

"compilerOptions": {

"outDir": "./dist",

"module": "commonjs",

"target": "es5",

"sourceMap": true,

"strict": true,

"esModuleInterop": true,

"moduleResolution": "node"

},

"include": ["src"]

}

## **Create** webpack.config.js:

Create a Webpack configuration file in the root directory:

const path = require('path');

const HtmlWebpackPlugin = require('html-webpack-plugin');

module.exports = {

entry: './src/index.ts',

module: {

rules: [

{

test: /\.ts$/,

use: 'ts-loader',

exclude: /node\_modules/,

},

],

},

resolve: {

extensions: ['.ts', '.js'],

},

output: {

filename: 'bundle.js',

path: path.resolve(\_\_dirname, 'dist'),

},

plugins: [

new HtmlWebpackPlugin({

template: 'src/index.html',

}),

],

devServer: {

contentBase: path.join(\_\_dirname, 'dist'),

compress: true,

port: 9000,

},

};

## **Create** src/index.html:

Create an HTML file to serve the bundled JavaScript:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Advanced TypeScript</title>

</head>

<body>

<script src="bundle.js"></script>

</body>

</html>

## **Create** src/index.ts:

Create an entry point TypeScript file to import other modules:

import './interfaces';

import './types';

import './guards';

import './utils';

import './classes';

import './functions';

console.log('Advanced TypeScript project initialized!');

## **Update** package.json Scripts:

Add a start script to package.json:

"scripts": {

"start": "webpack serve",

"build": "webpack"

}

## **Running** the Project

To run the project, simply use the following command:

**npm start**

Webpack will bundle your project and serve it at http://localhost:9000.

* **Webpack Configuration**: This config file sets up Webpack to bundle TypeScript files and use ts-loader for compiling TypeScript to JavaScript. It also uses HtmlWebpackPlugin to generate an HTML file that includes the bundled JavaScript.
* **Development Server**: webpack-dev-server serves the bundled files from the dist directory, enabling live-reloading during development.
* **Entry Point**: src/index.ts acts as the entry point, importing all other modules so they get included in the bundle.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*