JavaScript, TypeScript and ReactJS Training

This Training class will provide an introduction to the benefits of the React JavaScript framework, along with detailed training on JavaScript and TypeScript, so course participants can start to develop applications quickly using the framework.

Objectives

In this ReactJS training class, attendees will learn how to:

- Understand and practice JavaScript thoroughly
- Understand how to to use TypeScript effectively
- Understand the programming model provided by the React framework
- Define React components
- Use the React framework to handle events and stateful data

Topics

- JavaScript fundamentals
- JS Operators
- Control flow
- Arrays
- Functions
- Objects
- Prototypes
- Patterns to create objects
- Error handling
- ES6 features
- ES6 modules
- ES6 classes
- ES6 promises
- Introduction to TypeScript
- Basics of variables
- Comments
- Enum
- Generic types
- Functions
- Mapped types
- Objects

- Index signature
- Variables - advanced
- Exceptions
- Alias
- Type checking
- Iterators
- Manipulating objects and arrays
- Sharing code
- React Overview
- Basic Components and JSX
- React Functional Component Concepts
- React Router
- State Management for React
- Using React Hooks
- Creating Custom React Hooks
- Building React Apps with Mobx
- MobX applications - principles and concepts
- Connecting MobX to React
- Unit Testing React with React Testing Library

Prerequisites

- Attendees for this ReactJS training should have some prior understanding of web development, HTML, AJAX, JavaScript and ES6 or higher.

Duration

- 10 Days

Detailed course outline

Day 1:

JavaScript Fundamentals

- Basic Syntax
- 'use strict': The Modern Mode
- Variables
- Data types
- Primitive vs. Reference Values
- User Interaction: alert, prompt, confirm
- Numbers
- Strings
- boolean vs. Boolean

JS Operators

- Operators and their Precedence
- Comparison Operators
- Logical Operators
- Conditional (ternary) Operator

Control flow

- if/else statements
- switch...case Statement
- while/ do...while Statement
- for Statement
- break/ continue/ labeled Statement

Arrays

- Array
- Stack and Queue
- Shallow vs. Deep copy
- splice(): Delete, Insert, and Replace
- slice(): Copying a portion of an Array
- indexOf(), lastIndexOf() & find(), findIndex() & filter(): Finding/ Filtering
- concat() & reverse()
- map(): Transforming & forEach(): Iterate
- sort(): Sorting Elements
- reduce() / reduceRight(): Reducing an Array into a Value

Functions

- Functions as Objects
- apply(), call() & bind()
- Recursion
- Closure
- JavaScript Timers

Day 2:

Objects

- Object's Properties & their various Attributes
- Object to Primitive Conversions

Prototypes

- Demystifying prototypes
- Prototypes Methods - I
- Prototypes Methods - II
- Prototypal Inheritance
- Demystifying 'this'

Patterns to create objects

- Factory Pattern
- Constructor Pattern

Error Handling

- Handling errors using try/catch/finally
- Error types
- Operator: throw

Day 3:

ES6 features

- 'let': Block-Scoped Variables
- Hoisting: 'var' vs. 'let'
- 'const': Declaring Constants
- Arrow Functions: this, argument, new, prototype
- Default Function Parameters
- Rest Parameters
- Spread operator
- 'for...of': New Loop Statement
- Object Literal Syntax Extensions
- Octal and Binary Literals
- Template Literals
- Destructuring: Array
- Destructuring: Object

ES6 modules

- Import & Export: In Depth
- Default Export & limitations

ES6 classes

- Class Fundamentals
- Getter/ Setter and Static methods
- Inheritance via extends & super
- Extending Built-in Types
- new.target Metaproperty

ES6 promises

- Promises: Life-Cycle
- Promises: Create, Resolve & Reject
- Chaining Promises
- Using async/await

Day 4:

Introduction to TypeScript

- What is TypeScript?
- TypeScript Philosophy
- Why Use TypeScript?
- Using TypeScript

The Basics of Variables

- Declaring a Variable
- Declaring Types in Untyped Code
- Hoisting Variables
- TypeScript Scope is JavaScript Scope
- Switch Scope
- The Multiple Methods of Declaring a String
- String-Tagged Templates
- What is a Number in TypeScript?
- Booleans, Functions, and Objects

- Avoiding `any` at Any Time Possible
- Mutable and Immutable Arrays
- Undefined Versus Null
- Returning nothing with Void
- The Primitive Type never
- Unknown: A Better any
- Literal Type to Narrow Primitive Type
- Symbol and Unique Symbol
- Casting to Change Type

Comment

- TypeScript's Comments are like JavaScript's with One Exception

Enum

- Enum With and Without Values
- Accessing Enum Values
- Speeding Up Enum
- Merging and Adding Functionality to Enum

Generic Type

- Generic
- Generic and Classes
- Generic Constraint
- Generic with Construction Functions
- Generic Outside Class
- Generic Comparison
- Generic Inference
- Generic Default
- Generic and keyof

Functions

- Definition
- Named and Anonymous Functions

- Function and Inference Variables
- Generic Return Type, Optional Parameter and Default Value
- Functions in Classes
- Function Relationship with "this"
- Function and Inference Return Types
- Overload Functions to Enrich your Definition
- String Literal and Overload Function
- Types of Function Headers

Mapped Type

- Definition and Usages
- Immutable Data with Readonly
- Partial
- Nullable
- Pick
- Omit
- Record
- Extract
- Exclude
- ReturnType
- Custom Mapped Type

Objects

- Introduction to TypeScript's Many Objects
- The Curly Braces Object
- New Object
- Lowercase vs UpperCase Object

Index Signature

- Definitions and Usages
- String or Number Indexes
- Members of the Same Type
- Keys with Constants and Symbols

Variables Advanced

- Intersecting with Types, Interfaces, and Generics
- Literal Type, Narrowing, and Const
- Union with Types and Tagged Union
- Const Assertion for Literal Values
- Tuple For Type and Length Arrays
- Casting to Change Type
- keyof to Validate a Member's Name
- On How TypeScript Handles Variance
- How to Narrow a Type with the in Operator
- What is a Conditional Type?
- TypeScript Inference
- Set and Dictionary

Day 5:

Exception

- Creating an Exception
- Catching Synchronous Exceptions
- Catching Asynchronous Exceptions
- Assertion Functions

Alias

- Aliases with the Structural Behavior of TypeScript
- Aliases with Type
- Aliases with Generic Types and Recursivity
- The Differences between Type Aliases and Interfaces
- Branded Alias

Type Checking

- Comparing Variables
- Type Checking with typeof

- Type Checking with instanceof
- Type Checking and Interface with a Discriminator
- Type Checking with Intersections
- Type Checking an Interface with Custom User-Defined Type Guard
- Optional Chaining and Optional Element Access
- Nullish Coalescing
- Assertion Functions

Iterators

- Iterating an Object's Keys with For-In
- Iterating an Object with Standard For/While
- Iterating and the Asynchronous Loop

Manipulating Objects and Array

- Typing an Array
- Array with a Skipped Value
- Destructuring an Array
- Destructuring an Object
- The Spread Operator and Arrays
- The Spread Operator and Objects
- The Bang Operator

Sharing Code

- Namespace
- Module
- Default Module
- Lazy Loading Module
- Import Shortcuts
- Definition Files and Global Definition Files
- Definition File Locations

Day 6:

React Overview

- What is React?
- What's in a Name?
- React Component Model
- What React Is Not
- What You Will Not Find in React
- Motivation for Creating React
- A React JavaScript Example
- One-Way Data Flow
- JSX
- A JSX Example
- The Virtual (Mock) DOM
- Only Sub-components that Actually Change are Re-Rendered
- create-react-app

Basic Components and JSX

- What is JSX?
- JSX Transpilation to React Code Example
- Running the Transpiled Code
- Babel
- Playing Around in CodePen
- React Components
- Creating a Functional Component Example
- Component Names Must Be Capitalized
- Components vs Elements
- Elements Are Immutable
- Properties
- Property Naming Convention
- Properties Default to 'True'
- Spread Attributes (an ES6 Feature)
- Expressions

- Fragments

React Functional Component Concepts

- Functional Components
- Nesting JSX Elements
- Example of JSX Nesting
- Comments in JSX Code
- Setting CSS Styles Using Classes
- Setting CSS Styles Directly
- JSX Escapes Values
- Working with Lists of Items
- Keys in Lists
- Example List With Key
- State
- Types of State Data
- State Hierarchy
- Lifting State Up
- Props vs. State
- Pass Down a Function
- Immutability
- Immutability – Why?
- Virtual DOM and State
- Setting state
- Updating Input fields
- Passing Props to Components
- Passing Functions to Components
- Event Handling
- Event Handler Example
- Event Binding - DOs
- Event Binding – Don'ts
- Passing Parameters to Event Handlers
- Component Life-cycle
- Life-cycle in Functional Components

Day 7:

React Router

- Routing and Navigation
- react-router
- Creating a react-router based project
- A Basic Routed Component
- Router vs. BrowserRouter
- The Route component
- Redirect Route
- Navigating with
- Navigating with
- Route Parameters
- Retrieving Route Parameters
- QueryString Parameters
- Using Router with Redux

State Management for React

- React State Basics – Props and State
- Props
- State in Class Based Components
- Managing State with Hooks in Functional Components
- The Problem with Props and State
- Redux State Library
- Redux Advantages
- Redux Disadvantages
- Basic Rules for State Management
- Types of State
- Data State
- Communication State
- Control State
- Session State
- Location State

- Location State Side Effects

Day 8:

Using React Hooks

- Functional Component Shortcomings
- Hooks Overview
- Hook Rules
- React Linter Example
- Functional Component Props
- The useState Hook
- Functional Component using the useState hook
- useState with Multiple Variables
- useState can also be used with Objects
- The useEffect Hook
- useEffect Hook Example
- Using useEffect Hook to Load Data
- Restricting when useEffect is Called
- The useContext Hook
- Additional Hooks
- The useReducer Hook
- An Example Reducer Function
- Calling and Using useReducer
- The useMemo Hook
- useMemo Example
- The useCallback Hook
- useCallback Example
- The useRef Hook
- Using useRef to Hold Values
- The useImperativeHandle Hook
- useImperativeHandle Hook Example
- The useLayoutEffect Hook

Creating Custom React Hooks

- Custom Hooks
- Custom Message Hook
- Using the Custom Message Hook
- A Custom useList Hook
- Using the useList Custom Hook
- The built-in useDebugValue Hook
- Viewing the Effect of the useDebugValue Hook

Day 9:

Building React Apps with Mobx

- Introduction to MobX
- Observable (State)
- Actions
- Reactions
- Computations

MobX applications - principles and concept

- What MobX Reacts To
- Writing our own MobX
- MobX Applications Mindset
- Store Types
- How To Structure Your Stores
- How To Model Your Data
- mobx-react And mobx-react-lite

Connecting MobX to React

- MobX React API - observer And useObserver And <Observer />
- Connecting Our Stores To React
- When To Use MobX State VS React State

Day 10:

Unit Testing React with React Testing Library

- React Testing Framework
- Features
- Snapshot Testing
- Code Coverage
- Interactive Mode
- Projects created with create-react-app
- Default App Component Test
- Unit Tests
- Anatomy of a Unit Test
- Common Matchers
- Combining Tests
- Running Tests
- Testing Promise based async code with 'done'
- Setup and Teardown
- react-testing-library
- A Simple Component Test
- A Simple Snapshot Test
- Running and Updating SnapShot Tests
- Building Component Tests
- Calling Render
- Render Properties
- Simulating Events
- Testing Results
- Using Query Functions
- Text Matching
- Counter Component
- counter-test.js