Bright Star Institute

302.335.8868

admissions@brightstarinstitute.com

Developing Web Applications Using Angular

Duration: 90 hours

Prerequisites: Knowledge of HTML, CSS and JavaScript equivalent to attending the **Website Development with HTML5, CSS and Bootstrap** and **JavaScript Programming** courses. Knowledge of jQuery is helpful, but not required.

Description: This hands on programming course provides an in-depth introduction to the Angular Framework for the purpose of web application development. Attendees will learn the fundamental skills necessary to build Web Applications using Angular and the MVVM (Model-View-ViewModel) design pattern. Topics include using TypeScript and ECMAScript to create object-oriented Angular applications, extending HTML by creating reusable UI components, implementing data-binding, designing and using custom structural and attribute directives, as well as creating and using Angular pipes for formatting and transforming data in the UI. Students will explore creating UX's (User Experiences) by designing Web animations and implementing both template-driven and reactive style forms. Students will learn to use Angular routing to create SPA's (Single Page Applications). The course includes coverage of using DI (Dependency Injection) and Angular services to provide business and data-access logic to the application, both locally as well as communicating with RESTful web services to provide CRUD database operations.

Comprehensive hands on exercises are integrated throughout the course to reinforce learning and develop real competency.

Course Overview

Overview of Angular and the MVVM Design Pattern

- Features and Benefits of Angular
- Angular Architecture
- MVVM Design Pattern Overview
- Downloading Angular
- Choosing an IDE

Using Visual Studio Code

- Downloading and Installing Visual Studio Code
- Generating Angular Projects with the CLI (Angular Command Line Interface)
- Angular Project Structure, Files and Configuration
- Debugging Applications in VS Code and the

- Creating a Simple Application with Angular
  - Browser
  - Using the Terminal Window

Working with ECMAScript

- Enhancements to Legacy JavaScript
- New `let` to Declare Variables
- Block Scoping
- Using `for of` Loops
- Literals and Strings
  - Extended Literal Support
  - Template Literals
  - Object Literal Changes
- Function Enhancements
  - Default Parameters
  - Rest and Spread Operators
  - Arrow Functions/Lambdas

Working with TypeScript

- Types
  - Working with Built-In Types
  - Custom Types
- Setting Up and Using Node.js
- Transpiling TypeScript into JavaScript
- TypeScript Compiler Configuration
- TypeScript Functions
  - Default and Optional Parameters
  - Function Overloading
  - Parameter Types and Return Types

Angular Modules

- Using Modules to Create an Application
- Default Modules
- Bootstrapping an Application
- Exporting Classes, Functions and Values
- Limiting Scope
- Grouping Modules
- Specifying Module Dependencies
- Organizing Code Files
- Module Testing
- Best Practices

Angular Components

- Component LifeCycle
- Component Templates to Define Views
- Using Decorators to Define MetaData
- Styling
  - Per-Component Styling
  - Defining Global Styles in `angular.json`
  - Adding Bootstrap Framework to an Angular App
- View Encapsulation
  - `ViewEncapsulation.ShadowDOM`
  - `ViewEncapsulation.Emulated`

Component Templates and Data Binding

- Basic Data Binding Concepts
- Interpolation
- One-Way Property Binding
- Two-Way Property Binding
- Event Binding
- Custom Binding
  - Exposing Properties and Events to Parent Controls
  - Custom Property Binding
  - Custom Event Binding

        - `ViewEncapsulation.None`
    - Referencing DOM Elements with `ElementRef`
    - Lifecycle Hooks
        - `OnInit`, `OnDestroy`, `OnChanges`, `DoCheck`
        - `AfterContentInit`, `AfterViewInit`
        - `AfterContentChecked`, `AfterViewChecked`
    - Change Detection
    - Passing Data to Components

Angular Directives

- Built-In Directives
    - `NgIf`, `NgFor`, `NgClass`, `NgStyle`, `NgSwitch`, etc.
- Building Custom Directives
    - Using the Renderer2 Service
    - `ElementRef`
    - Attribute Directives
    - Structural Directives

Defining and Consuming Services

- Dependency Injection
    - Registering Providers with the Injector
    - Changes to Injection introduced in Angular 6
- Creating a Service
- Consuming a Service

Working with Pipes

- Text Casing Pipes
- Formatting Numbers and Dates
- Internationalization and Cultures
- Restricting Data Collections with `slice`
- Custom Pipes
    - Implementing the PipeTransform Interface
    - Pure vs Impure Pipes
    - Passing Parameters to Pipes

Working with Web Services

- Using `HttpClient`
- Importing the HTTP Module
- Creating Requests
- Processing Responses
- Web API
- Using PostMan to Test the Server-Side Service
- Interacting with a RESTFul Service
    - `POST` Requests
    - `DELETE` Requests
    - `PUT` Requests
    - `HEAD` Requests
- Dealing with CORS (Cross Origin Resource Sharing)

Asynchronous Programming in Angular

- Reactive Programming Model
- The RxJs Library
    - Observables
    - Observers
    - Subjects
    - Subscriptions
    - Operators
- Using Operators
    - Creation Operators
    - Filtering of Data Operators
    - Conversion of Data Operators
    - Math and Aggregate Operators
    - Utility Operators
    - Pipeable Operators
- Using the `async` Pipe
- Changes to RxJs in Version 6

Angular Routing and Navigation

- Overview of Routing
- Implementing Single Page Applications (SPAs)
- Location Strategies
- Client-Side vs Server-Side Routing
- Working with the Component Router
- Adding Router Imports
- Performing Router Configuration
- Using Router State
- Redirects
- Routing Parameters
- Router Lifecycle Events
- Nesting Routes
- Routing Guards
    - Defining a Guard
    - Registering Guards
    - Securing Routes
    - `CanActivate` Guard
    - `CanActivateChild` Guard
    - `CanDeactivate` Guard
    - `CanLoad` Guard

Angular Forms

- The `ngNativeValidate` Directive
- HTML `novalidate` Attribute
- Template Driven Forms
    - `ngForm` and `ngModel` in Forms
    - Input and Output Properties
- Reactive-Driven/Model-Driven Forms
    - `FormGroup`
    - `FormControl`
    - `FormArray`
    - `Validators` Class
- Using the `FormBuilder` Factory
- Working with Form State
- Client-Side Forms Validation

Unit Testing and TDD with Angular

- TDD vs End-to-End Testing
- Jasmine Testing Framework
    - Defining Expectations
- Running Tests in Karma
- Using the Angular Unit Test Framework
    - Fulfilling Dependencies
    - Mocking Out Data
    - Creating Testing Fixtures
- Testing Services and HTTP
- Using Test-Doubles (Mocks, Stubs and Spies)
- Testing Components
- Testing Forms

The Angular Animation System

- The Web Animations API
- States and Transitions
- Entering and Leaving
- Animating Properties
- KeyFrames
- Parallel Animation Groups

Building and Deploying Applications

- Developing a Deployment Strategy
- Managing Dependencies
- Tree Shaking
- Transpiling
- Linting