Angular Hands-on Workshop

Ervin Suhanko
Front End Software Engineer,
Team Lead & Mentor

Prerequisite

- Basic understanding of HTML, CSS, and JavaScript
- Basic understanding of Programming
- Familiarity with ES6 and Typescript is helpful

5th – 6th – 7th – 8th – 9th Sep'22 | 9:30 – 10:30 AM EST

Agenda

- Angular Introduction and Framework Overview
- Angular Project Structure
- Angular CLI (Command-line Interface tool)
- TypeScript Essentials
- Modules
- Components and Data flow
- Routing
- Pipes and Directives
- Reactive Forms
- RxJS Essentials
- Services and HTTP
- Tips, Tricks and Best Practices

Agenda

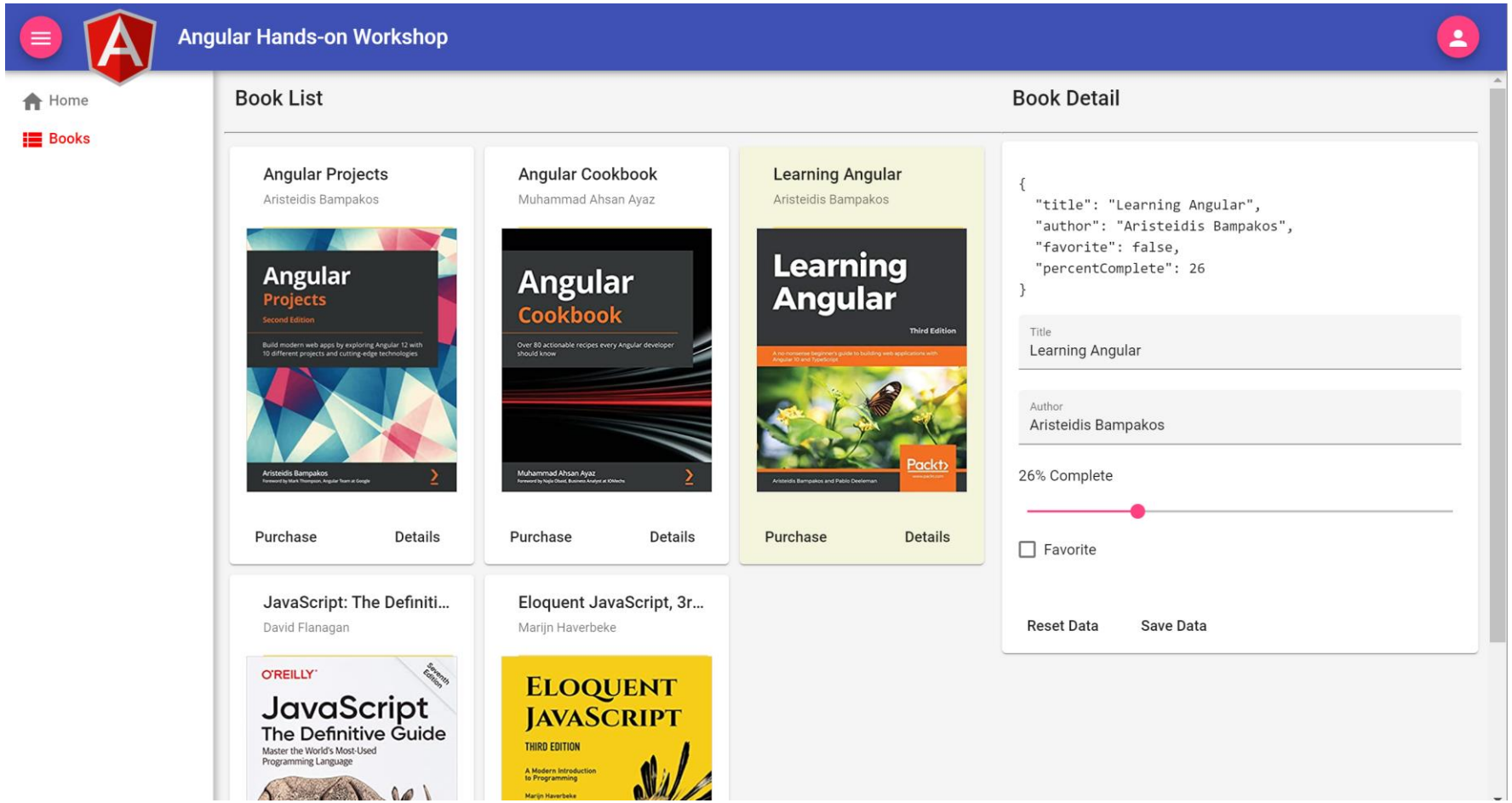
Day 1	Day 2	Day 3	Day 4	Day 5
Angular Introduction and Framework Overview	Modules	Routing	RxJS Essentials	Tips, Tricks and Best Practices
Angular CLI (Command-line Interface tool)	Components and Data flow	Pipes and Directives	Services and HTTP	
Angular Project Structure		Reactive Forms		
TypeScript Essentials				

Prerequisite

- Basic understanding of HTML, CSS and JavaScript
- Basic understanding of Programming
- Familiarity with ES6 and Typescript is helpful

Workshop demo application

https://github.com/ervinOrion/angular-hands-on-workshop



What is Angular?

A JavaScript **framework** for building single page client-side application on any scale using HTML, CSS and Typescript

Why Angular?

- Angular is one of the most popular web frameworks, a modern developer's platform.
- Easy to learn
- Structured Releases
- Project Architecture and Maintenance
- Supported by Google Typescript
- Performance Across Platforms

- Modules
- Components
- Templates
- MetaData
- Data-Binding
- Directives
- Pipes
- Services
- Dependency Injection
- Routes

- Modules
- Components
- Templates
- MetaData
- Data-Binding
- Directives
- Pipes
- Services
- Dependency Injection
- Routes

Modules

Components

Templates

MetaData

Data-Binding

Directives

Pipes

Services

Dependency Injection

Routes

Modules

Directives

Components

Pipes

Templates

Services

MetaData

Dependency Injection

Data-Binding

Routes

Modules

Components

Templates

MetaData

Data-Binding

Directives

Pipes

Services

Dependency Injection

Routes

What We Need?

- IDE Editor – Visual Studio (VS) Code
  https://code.visualstudio.com/

What We Need?

Why Visual Studio (VS) Code?

- Runs in Linux, Windows, and OS X
- Has numerous features that support
  - TypeScript
  - Auto-completion
  - Intellisense
  - Syntax checking
  - Refactorings
- It's FREE

What We Need?

✓IDE Editor – Visual Studio (VS) Code
https://code.visualstudio.com/

- Node package manager (npm)
https://nodejs.org/en/download/

What We Need?

Why npm?

- Installs libraries, packages, and applications
- Executes scripts

What We Need?

- ✓IDE Editor – Visual Studio (VS) Code
  https://code.visualstudio.com/

- ✓Node package manager (npm)
  https://nodejs.org/en/download/

What Else Do We Need?

- Angular ☺ https://angular.io/
- Angular CLI https://angular.io/cli
- TypeScript https://www.typescriptlang.org/
- Testing tools, linters, …

Angular CLI

The Angular CLI is a command-line interface tool that you use to initialize, develop, scaffold, and maintain Angular applications directly from a command shell.

Install: npm install -g @angular/cli

Angular CLI Useful Commands

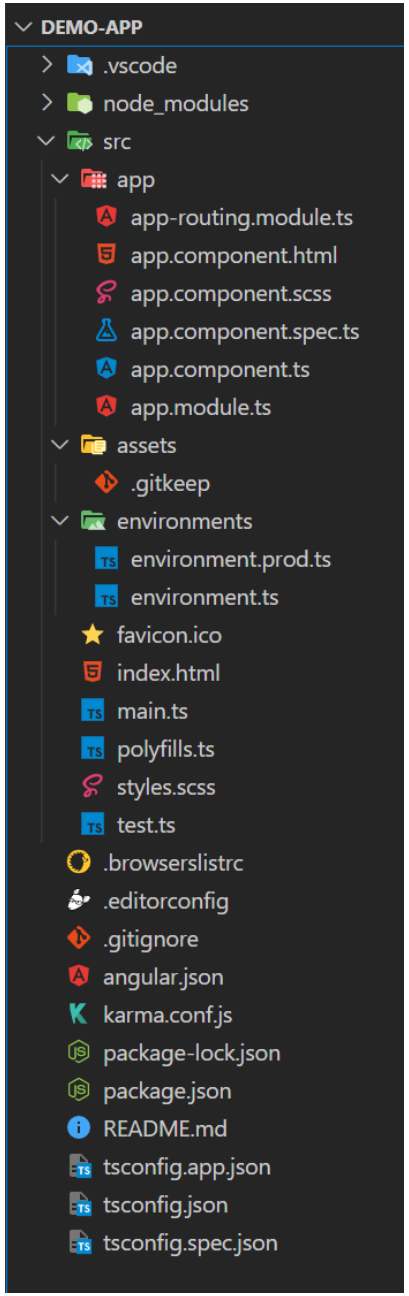
- ng new app-name (create a new project)
- ng generate component <name>
- ng generate service <name>
- ng generate module <name>

- ng serve (build and serve a project)
- ng lint (lint your project)
- ng test (run unit tests in a project)
- ng build (build a project to output directory)

Demo

Create a project with Angular CLI

Angular Project Structure



Typescript

- TypeScript is the programming language we use when building Angular applications
- It’s an Open-Source Language
- It’s a Superset of JavaScript
- Transpiles to plain JavaScript
- Strongly typed
- Class-based object-orientation

Typescript

- Basic Types
- Array and Object Types
- Interfaces
- Enums

TypeScript Introduction

TypeScript Introduction

What is TypeScript?

- **TypeScript is a so-called "superset"** to JavaScript. It's a programming language which builds up on JavaScript. It extends JavaScript.
- TypeScript is a typed language, where we can specify the type of the variables, function parameters and object properties.
- TypeScript perform **static checking,** it’s a static type checker

TypeScript Basics

Type Annotation

```
let myVar: type = value
```

- We can declare the type of a variable using a type annotation
- After a variable name, we write a colon and then a type

Basic Types

- **numbers**
- **strings**
- **booleans**
- **null**
- **undefined**
- **symbols**

TypeScript Introduction

Basic Types

number	string	Boolean
`let age: number;` `age = 12;`	`let userName: string;` `userName = 'Max';`	`let isInstructor: boolean;` `isInstructor = true;`

```
let age: number
Type 'string' is not assignable to type 'number'. ts(2322)
View Problem (⌥F8)    No quick fixes available
age = '12';
```

TypeScript Introduction

Array and Object Types

Array

```
let hobbies: string[];

hobbies = ['Sports', 'Cooking']
```

```
let hobbies: string[];

hobbies = ['Sports', 'Cooking', 12];
```

Type 'number' is not assignable to type 'string'. ts(2322)

View Problem (⌥F8) No quick fixes available

Object

```
let person: any
let person;

person = {
  name: 'Max',
  age: 32
};
```

```
let person: {
  name: string;
  age: number;
};
```

Combine Array and Object

```
let people: {
  name: string;
  age: number;
}[];
```

Type Inference

- By default, TypeScript tries to infer as many types as possible
- Types are inferred by TypeScript compiler when:
  - Variables are initialized
  - Function return types are determined

TypeScript Basics

Union Types

```
let course: string | number

course = 12341;
```

TypeScript Basics

Type Aliases

```
type Person = {
  name: string;
  age: number;
};
```

```
let person: Person;
```

```
let people: Person[];
```

TypeScript Basics

Functions & Function Types

```
function add(a: number, b: number): number
  return a + b;
}
```

```
function add(a: number, b: number): number
function add(a: number, b: number) {
  return a + b;
}
```

```
function print(): void (+1 overload)
function print(value: any) {
  console.log(value);
}
```

TypeScript Basics

Generic Types

```typescript
function insertAtBeginning(array: any[], value: any) {
  const newArray = [value, ...array];
  return newArray;
}

const demoArray = [1, 2, 3];

const updatedArray = insertAtBeginning(demoArray, -1); // [-1, 1, 2, 3]

updatedArray[0].split('');
```

```typescript
function insertAtBeginning<T>(array: T[], value: T) {
  const newArray = [value, ...array];
  return newArray;
}

const demoArray = [1, 2, 3];

const updatedArray = insertAtBeginning(demoArray, -1); // [-1, 1, 2, 3]

updatedArray[0].split('');
```

TypeScript Basics

Classes

```typescript
class Student {
  firstName: string;
  lastName: string;
  age: number;
  private courses: string[];

  constructor(first: string, last: string, age: number, courses: string[]) {
    this.firstName = first;
    this.lastName = last;
    this.age = age;
    this.courses = courses;
  }

  enrol(courseName: string) {
    this.courses.push(courseName);
  }

  listCourses() {
    return this.courses.slice();
  }
}
```

```typescript
class Student {
  // firstName: string;
  // lastName: string;
  // age: number;
  // private courses: string[];

  constructor(
    public firstName: string,
    public lastName: string,
    public age: number,
    private courses: string[]
  ) {}

  enrol(courseName: string) {
    this.courses.push(courseName);
  }

  listCourses() {
    return this.courses.slice();
  }
}
```

TypeScript Basics

Interfaces

```
interface Human {
  firstName: string;
  age: number;

  greet: () => void;
}
```

```
class Instructor implements Human {
  firstName: string;
  age: number;
  greet() {
    console.log('Hello!!!!');
  }
}
```

TypeScript Basics

Thank You