

# Scouting App Training

Team 294

Introduction to Angular and Web Service Programming

Angular Basics

# Angular Documentation

<https://angular.io/docs>

Version 10.2

The screenshot shows the Angular Documentation website for version 10.2. The browser address bar displays `angular.io/docs`. The website has a blue header with the Angular logo and navigation links: **FEATURES**, **DOCS** (active), **RESOURCES**, **EVENTS**, and **BLOG**. A search bar is located on the right side of the header. On the left, a sidebar lists the documentation structure: **INTRODUCTION**, **GETTING STARTED** (with sub-links: Try it, Setup), **MAIN CONCEPTS** (with sub-links: Components, Templates, Directives, Dependency Injection), **BUILT-IN FEATURES**, **BEST PRACTICES**, **ANGULAR TOOLS**, **TUTORIALS**, **RELEASE INFORMATION**, and **REFERENCE**. At the bottom of the sidebar, a button indicates the current version: **stable (v10.2.0)**. The main content area is titled **Introduction to the Angular Docs** and includes a brief overview of Angular as a framework for single-page apps. Below the text are three cards: **Get Started** (Local setup), **Learn and Explore** (Introduction to Angular concepts), and **Take a Look** (Try it now). Further down is a **Hello World** section with a link to the Tour of Heroes tutorial. The **Assumptions** section states that the docs assume familiarity with HTML, CSS, JavaScript, and the latest standards, as well as classes, modules, TypeScript, types, and decorators.

angular.io/docs

ANGULAR FEATURES DOCS RESOURCES EVENTS BLOG

Search

INTRODUCTION

GETTING STARTED

- Try it
- Setup

MAIN CONCEPTS

- Components
- Templates
- Directives
- Dependency Injection

BUILT-IN FEATURES

BEST PRACTICES

ANGULAR TOOLS

TUTORIALS

RELEASE INFORMATION

REFERENCE

stable (v10.2.0)

## Introduction to the Angular Docs

Angular is an application design framework and development platform for creating efficient and sophisticated single-page apps.

These Angular docs help you learn and use the Angular framework and development platform, from your first application to optimizing complex single-page apps for enterprises. Tutorials and guides include downloadable examples to accelerate your projects.

### Get Started

Set up your local environment for development with the Angular CLI.

Local setup

### Learn and Explore

Learn about the fundamental design concepts and architecture of Angular apps.

Introduction to Angular concepts

### Take a Look

Examine and work with a small ready-made Angular app, without any setup.

Try it now

### Hello World

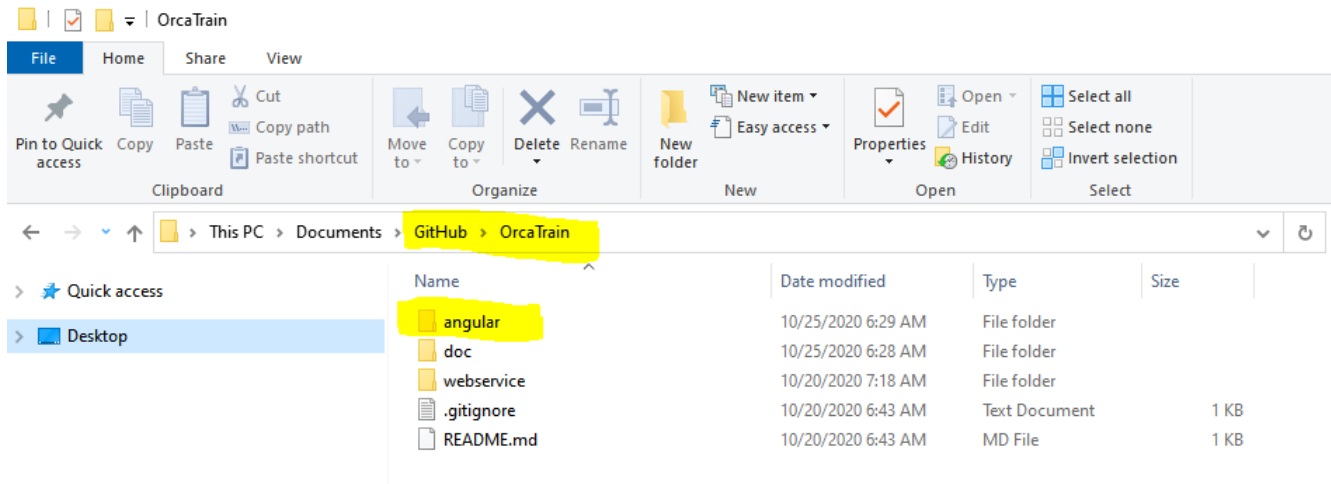
Work through a full tutorial to create your first app.

Tour of Heroes tutorial

## Assumptions

These docs assume that you are already familiar with [HTML](#), [CSS](#), [JavaScript](#), and some of the tools from the [latest standards](#), such as [classes](#) and [modules](#). The code samples are written using [TypeScript](#). Most Angular code can be written with just the latest JavaScript, using [types](#) for dependency injection, and using [decorators](#) for metadata.

# VS Code



Open the angular folder in VS Code

Run the app from a command line window from the angular directory  
ng serve

<http://localhost:4200>

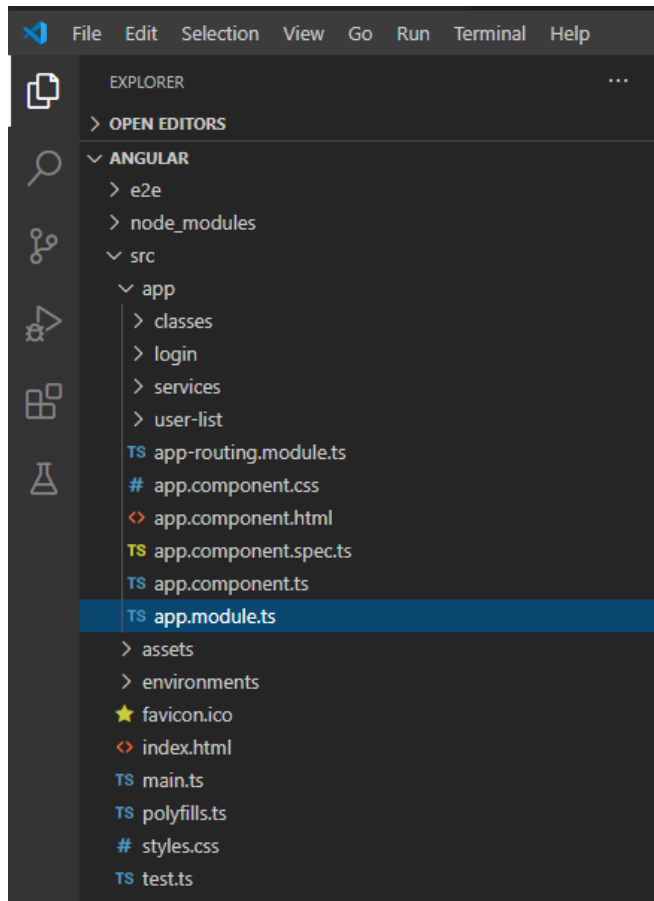
# Angular Project

Classes – hold the data

Services – connect to web services

Components – user interface

- Login
- User List



# Component

Create using Angular CLI

- ng generate component login

Each component has 4 associated files

- .html
- .css
- .ts
- .spec.ts

```
✓ src
  ✓ app
    > classes
    ✓ login
      # login.component.css
      <> login.component.html
      TS login.component.spec.ts
      TS login.component.ts
```

# HTML Template

Standard HTML with Angular extensions

Template statements allow you to respond to user events

Line 17 tells Angular to call the login method whenever the user clicks the login button

```
<> login.component.html X
src > app > login > <> login.component.html > ...
1  <div class="container">
2
3      <h1>Login</h1>
4
5      <table>
6          <tr>
7              <th>Username</th>
8              <td><input type="text"/></td>
9          </tr>
10         <tr>
11             <th>Password</th>
12             <td><input type="password"/></td>
13         </tr>
14     </table>
15
16     <div>
17         <button (click)="login()" >Login</button>
18     </div>
19
20 </div>
```

# TypeScript

Each component has a TypeScript file associated with it where you put any component specific code.

Line 23 defines the login function that was referenced in the template.

Code that is shared between components should be in a service.

Use the router to navigate to different pages

```
TS login.component.ts X
src > app > login > TS login.component.ts > ...
1  import { Component, OnInit } from '@angular/core';
2  import { Router } from '@angular/router';
3  import { User } from '../classes/user';
4  import { AppService } from '../services/app.service';
5
6  @Component({
7    selector: 'app-login',
8    templateUrl: './login.component.html',
9    styleUrls: ['./login.component.css']
10 })
11 export class LoginComponent implements OnInit {
12
13   constructor(
14     private appService: AppService,
15     private router: Router
16   ) { }
17
18   user: User = new User();
19
20   ngOnInit(): void {
21   }
22
23   login() {
24     this.router.navigate(['/user-list']);
25   }
26
27 }
28
```

# Structural Directives

Angular provides built-in directives to bring your html to life and do things like branching and looping

Line 8 defines a loop that will print a new row for every user in the userList array

Line 9 will print the value contained in the associated variable

```
<> user-list.component.html X
src > app > user-list > <> user-list.component.html > ...
1  <div>
2    <h1>Users</h1>
3
4  <table>
5    <tr>
6      <th>Name</th>
7    </tr>
8    <tr *ngFor="let user of this.appService.userList">
9      <td>{{ user.username }}</td>
10   </tr>
11 </table>
12
13 </div>
```



# Classes

Angular applications use classes and interfaces to provide structure to data and allow for type checking

Angular applications are designed for the web and are best written using a functional programming style (rather than object-oriented)

Classes should only contain data and behavior should be modeled in functions

```
TS user.ts  X
src > app > classes > TS user.ts > User
1  export class User {
2      public username: string;
3      public password: string;
4      public token: string;
5  }
```

# Services

Services contain shared logic that is available to all components

Angular uses dependency injection to give each component a reference to the service

Services should hold data that is shared across the application

Services should contain the logic to get the data from a web service

```
TS app.service.ts X
src > app > services > TS app.service.ts > AppService
1  import { Observable } from 'rxjs';
2  import { HttpClient, HttpHeaders, HttpResponse } from '@angular/common/http';
3  import { User } from '../classes/user';
4  import { Injectable } from '@angular/core';
5  import { throwError } from 'rxjs';
6  import { catchError, retry } from 'rxjs/operators';
7
8  @Injectable({providedIn: 'root'})
9  export class AppService {
10     public user: User;
11     public userList: User[];
12
13     constructor(private http: HttpClient) {
14         this.userList = [
15             {"username": "paul", "password": "abc", "token": "testtoken1"},
16             {"username": "michael", "password": "def", "token": "testtoken2"},
17             {"username": "bryan", "password": "ghi", "token": "testtoken3"}
18         ];
19     }
20 }
```