Angular Introduction

Downloads: http://bit.ly/centric-ng1

Agenda

Datum	Onderwerp
1-2-2017	TypeScript Introduction
15-2-2017	Advanced TypeScript
1-3-2017	Angular Introduction
15-3-2017	Angular Building Blocks
29-3-2017	Components
12-4-2017	Ajax
26-4-2017	Data Entry
10-5-2017	Single Page Applications

Downloads: http://bit.ly/centric-ng1

What are we going to cover?

Understand what Angular is and how to use it.

- The major Angular building blocks
- Group functionality into modules
- Writing Angular components
- Create templates
- Use data binding expressions

What is Angular?

Angular is a development platform for building mobile and desktop web applications.

Open source and originally developed at Google.

- Distributed under the MIT license

Important Angular features

Command line interface (CLI)

Angular prefers TypeScript

Component based architecture

Dynamic templates

Data binding

Dependency injection

Angular ❤ TypeScript

Angular has a preference for TypeScript.

- Angular itself is written in TypeScript
- You can write your application code in ECMAScript or Dart if you prefer

Angular CLI

The Angular Command Line Interface (CLI) makes it easy to get started with a new Angular project.

- The CLI use Webpack to bundle and server your code

Angular CLI commands

The CLI supports many common actions

- Create a new project with:
  ng new <project name>
- Generate a new component with:
  ng generate component <component name>
- Run a development server with:
  ng serve
- Run unit or end to end tests with:
  ng test or **ng e2e**
- Many more

Angular style guide

An opinionated guide to Angular syntax, conventions, and application structure.

- Describes good and bad practices when building Angular applications

Major building blocks

Modules

Components

Templates

Directives

Service

Modules

Each Angular application must have a root module.

A module is a class with the **@NgModule()** decorator.

- The object passed in configures the module

Browser based applications will need to import the standard **BrowserModule** in their root module.

A minimal main module

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [AppComponent],
  imports: [BrowserModule],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Just In Time compilation

During development Just-in-time (JiT) compilation
is often used.

- This is loaded using the **platformBrowserDynamic()** service
- And is returned as a **PlatformRef** instance

Bootstrapping the application

Use the **PlatformRef** instance to bootstrap the main application module.

- The **selector** of the components in the modules bootstrap collection will be queried in the HTML page
- An instance of each component will be injected

Bootstrapping the application example

```
import './polyfills.ts';
import { platformBrowserDynamic }
    from '@angular/platform-browser-dynamic';
import { AppModule } from './app/';

platformBrowserDynamic()
  .bootstrapModule(AppModule);
```

Components

A component is a fundamental Angular concept that manages part of the rendered markup.

It is a class with the **@Component()** decorator.

- Each component must have a **selector** and a **template**

Hello world component

```
import { Component } from '@angular/core';

@Component({
  selector: ‘hello-world',
  template: 'Hello Angular world!'
})
export class HelloWorld {
  name = 'Angular';
  now = new Date();
}
```

Templates

Templates are used to render a components markup.

They can be created as an inline string or a separate file.

- Use the templateUrl with a relative path

Template syntax

Interpolation is used to inject the result of template expressions into the generated markup.

- Template expressions are most ECMAScript that have no side effects or use the global namespace

The expression context is the component instance.

- This component instance is the source of binding values

Template syntax example

```
<h2>
  Hello {{ name }} world!
</h2>
<p>
  The current time is: {{ now.toLocaleTimeString() }}
</p>
```

Data binding syntax

Angular uses properties for data binding expressions.

Property bindings can be either input or output bindings.

- [] is used for input bindings
- () Is used for output or event bindings

[()] is used to simulate two way data binding.

Data binding syntax example

```
<p>
  <input type="text" [(ngModel)]="name" />
  <button (click)="greet()">Greet</button>
</p>
```

Directives

Angular ships with a number of useful directives.

NgIf, NgFor and NgSwitch are template directives.

- Template directives need to be prefixed with an asterisk *

NgClass and NgStyle are normal directives.

- Use them with standard property binding [].

Directives example

```
<ul>
  <li *ngFor="let movie of movies">{{movie.title}}</li>
</ul>
```

Dependency injection

Dependency injection (DI) is used to decouple different parts of the application.

- The different dependencies are automatically inserted at runtime
- When unit-testing, fake dependencies can be used instead

Conclusion

Angular is a complete framework for building applications.

- It isn’t just for browser based applications

Angular applications have a components base architecture.

- Where each component has a template as user interface

With many supporting types like directives and services.

- Modules and dependency injection tie everything together