



Working with Angular

Getting Started with Angular

Prerequisites

- Have some exposure to JavaScript and Web development
- Know the basics of HTML and CSS
- Have Node.js installed on your machine: <https://nodejs.org/en/>

Why Angular?

- What is Angular
- Version numbers and release schedule
- How is different from PHP, ASP and JSP

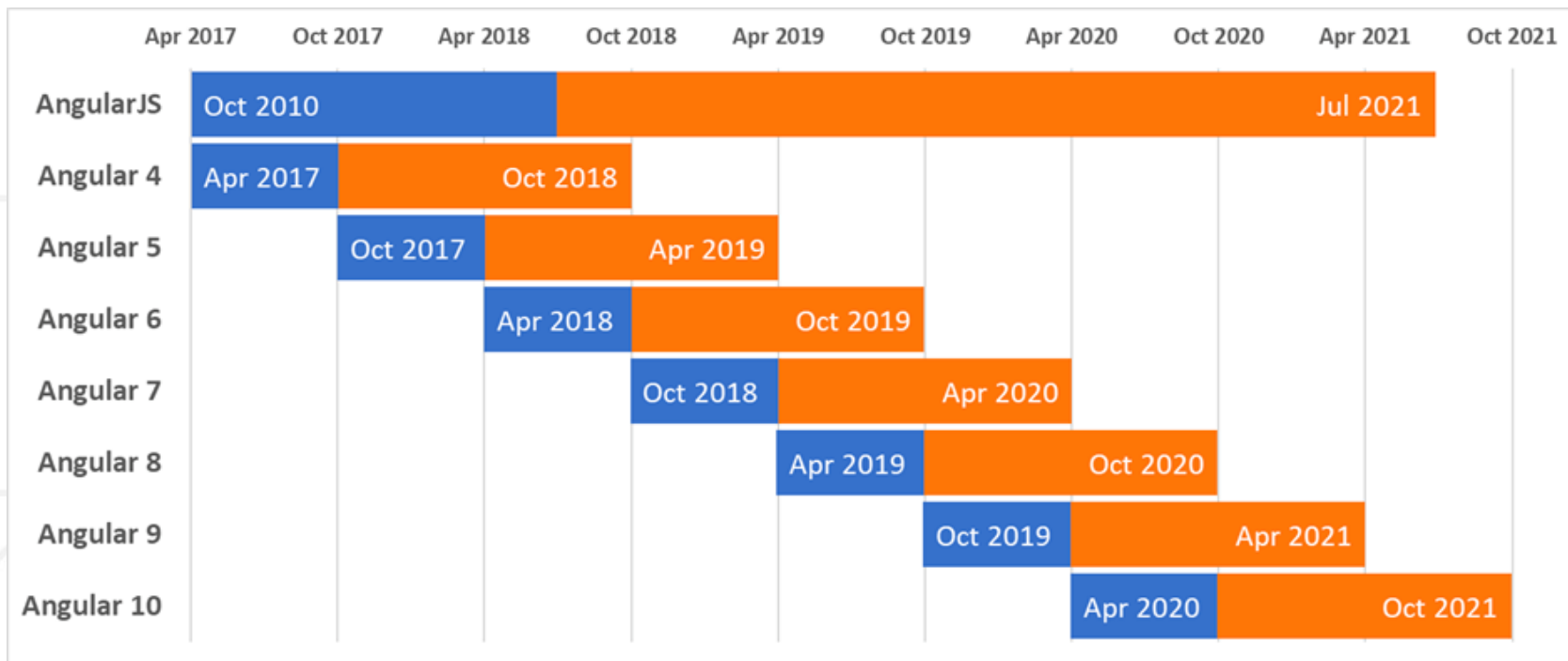


<https://angular.io/>

What is Angular?

- Open-source web application framework, maintained by Google and the community – <http://angular.io>
- Decouple DOM manipulation from application logic
- Decouple the client-side of an application from the server side
- Provide structure for the journey of building an application

Angular Roadmap



Angular Versioning and Release Schedule

ANGULAR JS = ANGULAR 1.x Versions

ANGULAR = ANGULAR 2+ Versions

Angular Versioning and Release Schedule

Version x.y.z+1

Every week (roughly)

Bug fixes only



Version x.y+1

Every month (roughly)

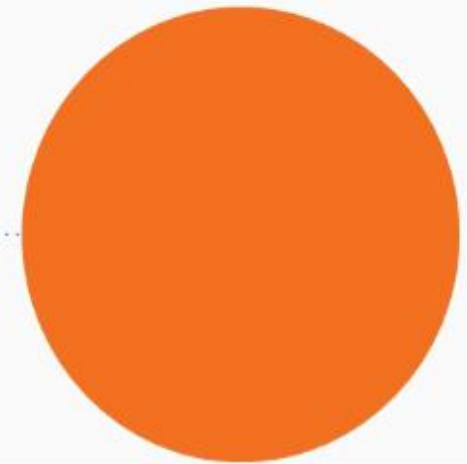
Non-breaking features added



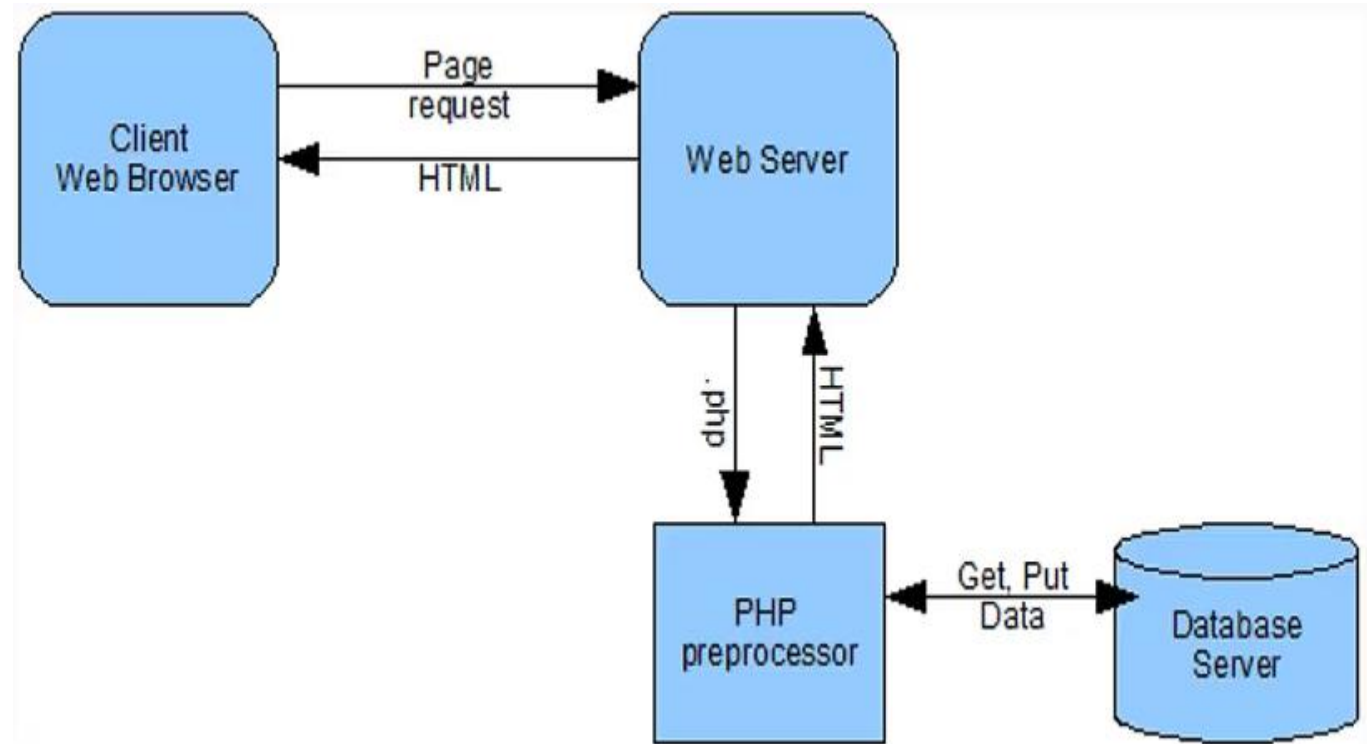
Version x+1

Every six months

Possibly with breaking changes

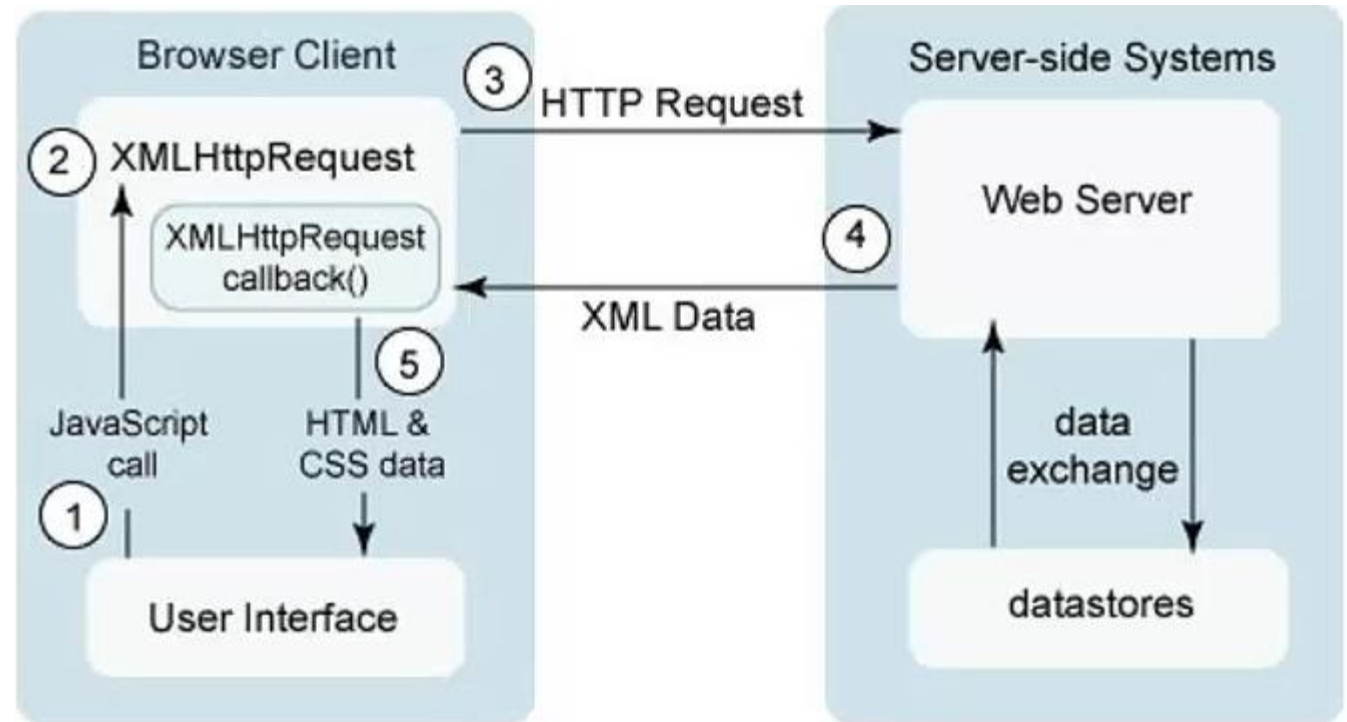


- In the past, all of the front-end code (HTML, JS and CSS) was generated from the back-end.
- User interactions with the webpage often required a full-page refresh.



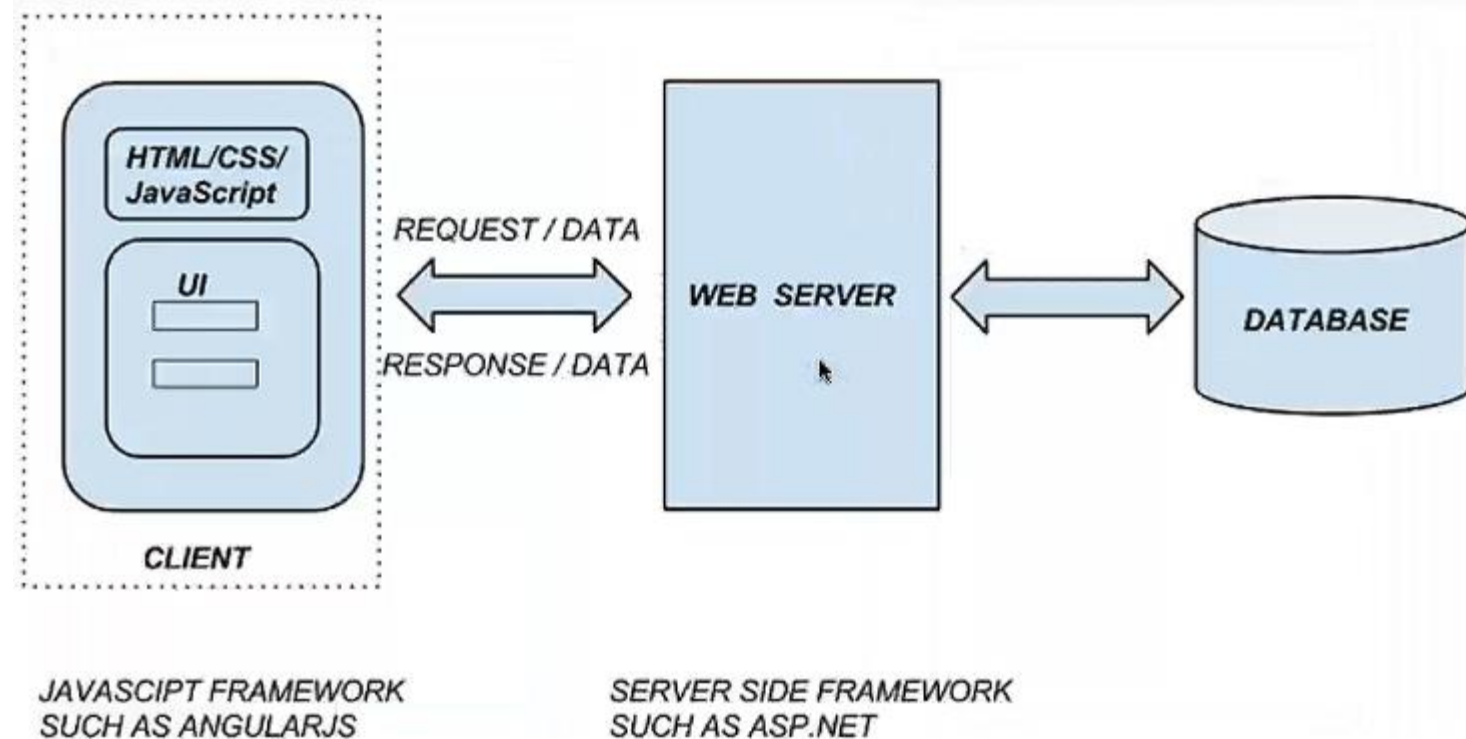
Ajax

- Then came into play AJAX and JQuery.
- The main idea was to load content asynchronously in the background to refresh portions of webpage.



Angular

- With Angular, the front-end code is now independent from the back-end.
- The web server becomes a web-service that outputs JSON data, not dynamic HTML or CSS.



Different from AngularJS or React?

- Why is Angular a revolution
- What about AngularJS or 1.x
- How is different from React

Angular is pretty much a Revolution

- New languages: Typescript, ES6 and Dart
- New tools: module managers, RxJS and transpiler
- New template syntax
- New concepts: components, pipes and so on
- New tooling: Angular CLI – command line interface

What about AngularJS or 1.x ?

- AngularJS or 1.x is still maintained as long as it is more popular than Angular or Angular 2+
- AngularJS is now six years old, which is very old in today's web dev world
- Most of Angular new features are included in 1.x to facilitate migration from AngularJS to Angular

How is Different from React?

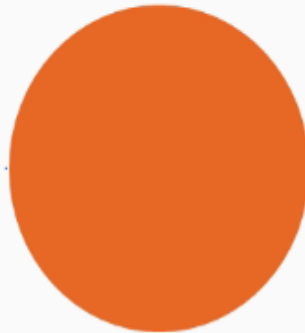
React

JavaScript library to
build web components
~130 KB



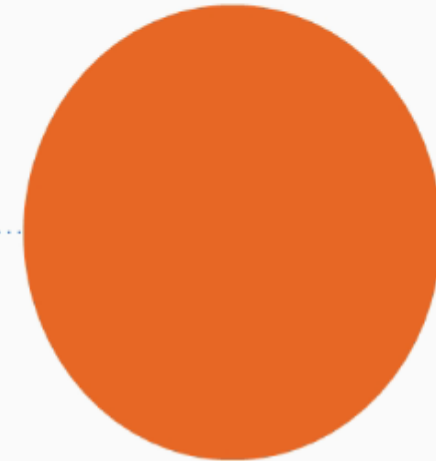
AngularJS

Web development framework
JavaScript only - Many APIs
~150 KB



Angular

Web development framework
JavaScript, TypeScript, or Dart
Many more APIs and patterns
~500 KB



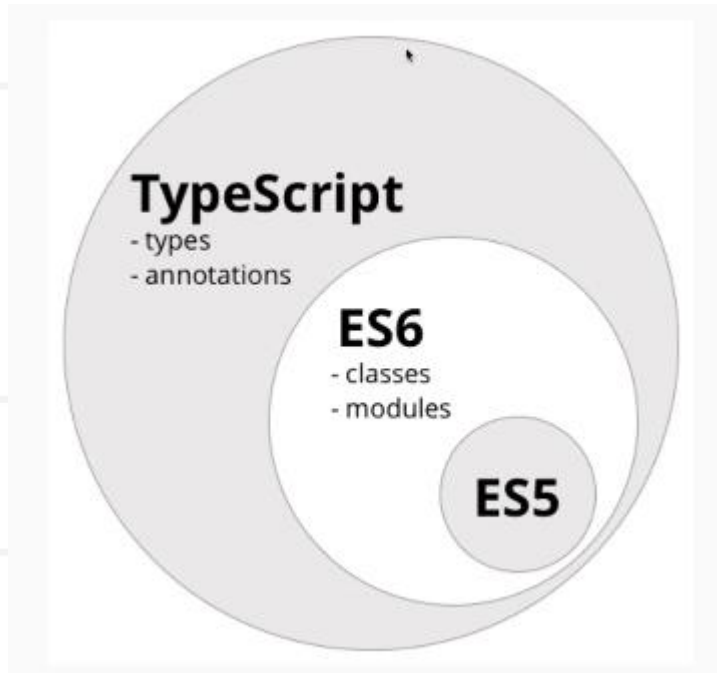
Why TypeScript and What is it?

- What is TypeScript
- Why use it
- How is different from JavaScript



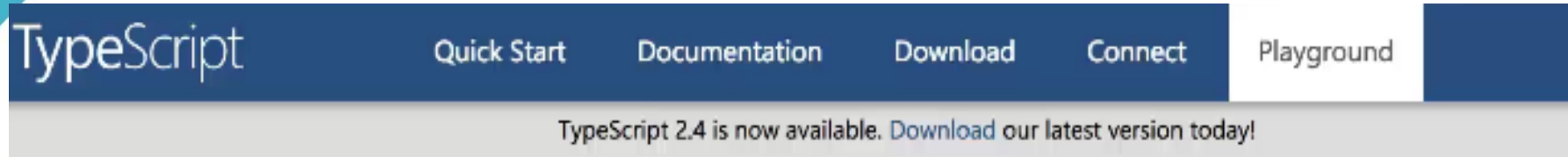
<http://www.typescriptlang.org/>

What is TypeScript?



- TypeScript is strict superset of ES6 – tomorrow' JavaScript
Which is the next iteration of ES5 – today's JavaScript
- More familiar to back-end developers as it feels like Java, C# or .Net
- Browser independent as Typescript compiles down to ESx anyway
- As a result, you write your code once and it works everywhere
- You're free to use the features you want – and JavaScript is valid TypeScript

What is TypeScript?



Using Inheritance TypeScript Share Options Run JavaScript

```
1 class Animal {
2     constructor(public name: string) { }
3     move(distanceInMeters: number = 0) {
4         console.log(`${this.name} moved ${dis
5     }
6 }
7
8 class Snake extends Animal {
9     constructor(name: string) { super(name);
10    move(distanceInMeters = 5) {
11        console.log("Slithering...");
12        super.move(distanceInMeters);
13    }
14 }
15
16 class Horse extends Animal {
17     constructor(name: string) { super(name);
```

```
1 var __extends = (this && this.__extends) || (
2     var extendStatics = Object.setPrototypeOf
3         ({ __proto__: [] } instanceof Array &
4         function (d, b) { for (var p in b) if
5     return function (d, b) {
6         extendStatics(d, b);
7         function __() { this.constructor = d;
8         d.prototype = b === null ? Object.cre
9     };
10 }());
11 var Animal = /** @class */ (function () {
12     function Animal(name) {
13         this.name = name;
14     }
15     Animal.prototype.move = function (distance
16         if (distanceInMeters === void 0) { di
17         console.log(this.name + " moved " + d
```

Building Blocks of an Angular Application

Topics

- All You Need to Know about Angular Modules
- Introducing Components, Pipes, Directives, and Services
- Bootstrapping an Angular Application
- Using Angular CLI to Create Our Application

Know about Angular Modules

Angular Modules

- What are modules
- How modules configure an application
- Lazy-loading code using modules

Module

Components

HTML template

Pipes

Your own
classes

Services

Directives

What are Modules?

- A module is a package of TypeScript objects
- Every Angular application has at least one module
- The main module describes the structure of the application
- It configures dependency injection as well as some specific module configuration
- We can define modules for specific features that can be shared between different applications

Lazy-loading

- A way to load modules on-demand, as they are needed

Typical App Module

```
@NgModule({
  declarations: [
    AppComponent, HelloComponent, BindingsComponent,
    comm.CommunicationComponent, comm.CommunicationComponent2, comm.HelloWorld, comm.HelloWorld2,
    HidingComponent, HttpComponent, ModelComponent,
    NgifComponent, NgswitchComponent, FormsComponent, RouterComponent, JasmineTestComponent,
    PromiseExampleComponent, ObservableExampleComponent, LoginComponent, NgforComponent, ListPostsCompon
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpClientModule,
    RouterModule //, routing
    //,ReactiveFormsModule
  ],
  providers: [AuthGuard, LoginService, PostsService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Components, Pipes, Directives, & Services

- What are Components
- What are Pipes
- What are Directives
- What are Services

What are Components

- A component is a reusable piece of HTML bundled with its TypeScript controller
- Creating a component is like creating your own HTML element
- 80% + of the Angular code you are going to write will be components
- An Angular app is tree of components

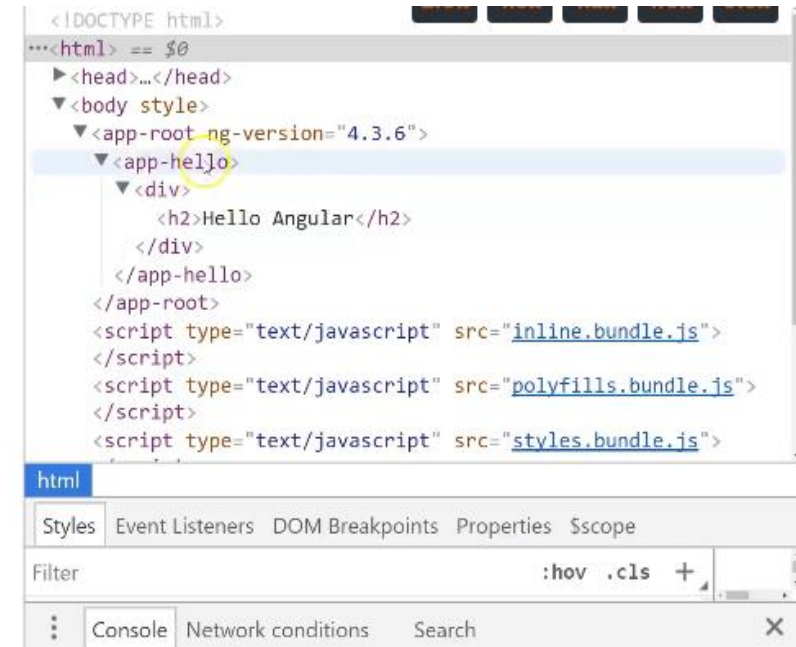
```
import { Component } from '@angular/core';

@Component({
  selector: 'app-hello',
  template: `
    <div>
      <h2>Hello {{name}}</h2>
    </div>
  `
})
export class HelloComponent {

  name : string;

  constructor() {
    this.name = 'Angular';
  }
}
```

Hello Angular



Services, Pipes and Directives

- Services are used to handle the business logic of the app
- Pipes are used to format data
- Directives manipulate the DOM by creating elements, hiding them and so on
- Components are specific directives that have a HTML template

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { Post } from './post';

@Injectable()
export class PostsService {

  constructor(private http: HttpClient) { }

  getPosts() : Observable<Post> {
    return this.http.get<Post[]>(url: "http://localhost:8080/posts.json")
      .switchMap(postsData =>
        Observable.interval(2000)
          .take(postsData.length)
          .map(index => postsData[index])
      );
  }

  getPostsAsPromise() : PromiseLike<Post[]> {
    return this.http.get<Post[]>(url: "http://localhost:8080/posts.json").
```

API List

TYPE: P Pipe

STATUS: All

 Filter

common

- P AsyncPipe

P I18nSelectPipe

P CurrencyPipe

P SlicePipe
- P DatePipe

P JsonPipe

P DecimalPipe

P UpperCasePipe
- P I18nPluralPipe

P LowerCasePipe

P PercentPipe

P TitleCasePipe

DatePipePIPE

| | |
|-------------|--|
| npm Package | @angular/common |
| Module | <pre>import { DatePipe } from '@angular/common';</pre> |
| Source | common/src/pipes/date_pipe.ts |
| NgModule | <pre>CommonModule</pre> |

Typical Directives



ngIf

The most common usage of the `ngIf` directive is to conditionally show the inline template as seen in this example:

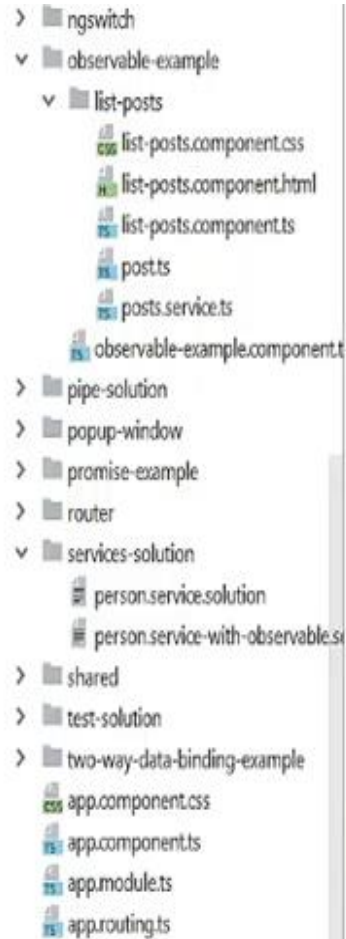
```
1. @Component({
2.   selector: 'ng-if-simple',
3.   template: `
4.     <button (click)="show = !show">{{show ? 'hide' : 'show'}}</button>
5.     show = {{show}}
6.     <br>
7.     <div *ngIf="show">Text to show</div>
8.   `
9. })
10. class NgIfSimple {
11.   show: boolean = true;
12. }
```



Bootstrapping an Angular Application

- Files involved in bootstrapping an Angular App
 - Role of index.html
 - Role of main.ts
 - Role of app.module.ts
-
- The main module file app.module.ts defines the structure of the application
 - main.ts is the file use to bootstrap and application by specifying which module it should load and how it should be loaded
 - This allows Angular to know which HTML root element to look for in index.html to load the entire components tree

Typical main.ts



```
1 import './polyfills.ts';
2
3 import { platformBrowserDynamic } from '@angular/platform-browser-dy
4 import { enableProdMode } from '@angular/core';
5 import { environment } from './environments/environment';
6 import { AppModule } from './app/';
7
8 if (environment.production) {
9   enableProdMode();
10 }
11
12 platformBrowserDynamic().bootstrapModule(AppModule);
13
```

Typical app.module.ts

File Explorer view showing project structure:

- post.ts
- posts.service.ts
- observable-example.component.ts
- pipe-solution
- popup-window
- promise-example
- router
- services-solution
 - person.service.solution
 - person.service-with-observable.s
- shared
- test-solution
- two-way-data-binding-example
- app.component.css
- app.components.ts
- app.module.ts**
- app.routing.ts
- index.ts
- person.interface.ts
- rxjs-operators.ts
- environments
 - favicon.ico
 - index.html
 - main.ts
 - polyfills.ts
 - styles.css
 - test.ts
 - tsconfig.app.json
 - tsconfig.spec.json
 - typings.d.ts
- angular-cli.json
- .editorconfig

```
34 @NgModule({
35   declarations: [
36     AppComponent, HelloComponent, BindingsComponent,
37     comm.CommunicationComponent, comm.CommunicationComponent2, comm.Hell
38     HidingComponent, HttpComponent, ModelComponent,
39     NgifComponent, NgswitchComponent, FormsComponent, RouterComponent, J
40     PromiseExampleComponent, ObservableExampleComponent, LoginComponent,
41   ],
42   imports: [
43     BrowserModule,
44     FormsModule,
45     HttpClientModule,
46     RouterModule //, routing
47     //,ReactiveFormsModule
48   ],
49   providers: [AuthGuard, LoginService, PostsService],
50   bootstrap: [AppComponent]
51 })
52
53 export class AppModule { }
```


Hello Angular

```
<!DOCTYPE html>
<html>
  <head>...</head>
  <body style>
    <app-root ng-version="4.3.6">
      <app-hello>
        <div> == $0
          <h2>Hello Angular</h2>
        </div>
      </app-hello>
    </app-root>
    <script type="text/javascript" src="inline.bundle.js">
    </script>
    <script type="text/javascript" src="polyfills.bundle.js">
    </script>
    <script type="text/javascript" src="styles.bundle.js">
    </script>
  </body>
</html>
```

html body app-root app-hello div

Styles Event Listeners DOM Breakpoints Properties \$scope

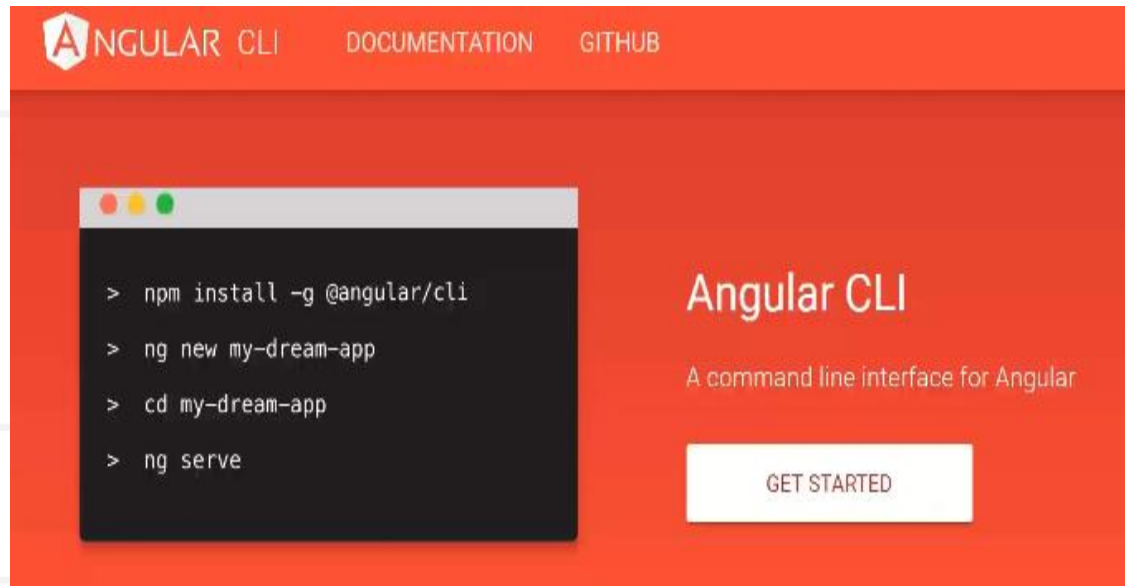
Filter :hov .cls +

Console Network conditions Search

Using Angular CLI to Create Our Application

- What is Angular CLI and how to use it
- Creating projects and files with Angular CLI
- Using CLI for development purposes
- Using CLI to build our application

What is Angular CLI



- Angular projects are complex, with many dependencies and a lot of set-up
- As a result, the Angular team decided to provide a command line tool called Angular CLI to help – <https://cli.angular.io/>
- CLI creates the entire structure of your project by following the best practices from the Angular team
- It also allows you to generate new components, pipes and so forth from simple commands

Angular CLI

- Check Version:
`ng --version`
- To update the CLI version or Reinstall
`npm uninstall -g angular-cli`
`npm cache clean`
`npm install -g angular-cli`

Create a Start Project

1. Open a terminal window in VS Code.
2. Navigate to a suitable folder—for example, Documents.
3. Enter the following command, which will create a new Angular app in a folder called start and will create lots of files:
`ng new start-app`
4. Navigate to the start folder.
`cd start-app`
5. Enter the following command to start the app:
`ng serve`
6. Open your web browser and browse to localhost:4200. You should see the text “welcome to app!”

What's in the Root Folder?

| File or Folder | What It Is |
|---------------------------|--|
| e2e | Folder for testing files (testing, Karma, and Protractor) |
| node_modules | Folder for project node dependencies |
| src | Folder for project source code |
| .editorConfig | Editor configuration file |
| .gitignore | Git ignore file |
| angular-cli.json | CLI configuration file. You change your CLI options in this file |
| karma-conf.json | Karma configuration file (testing, Karma, and Protractor) |
| package.json | Node dependencies configuration file |
| protractor-conf.js | Protractor configuration file (testing, Karma, and Protractor) |
| README.md | Readme informational file, contains information on CLI commands |
| tslint.json | Lint configuration file |

What's in the Source Folder?

| File or Folder | What It Is |
|----------------------|---|
| app | Folder for your application source code files, currently contains source code for an application component (more on this later) |
| assets | Folder for your application image and CSS files |
| environments | Folder for configuration files for environments—for example, configurations for development and production |
| favicon.ico | Application icon |
| index.html | The HTML page for the Angular single page application |
| main.ts | Code to start the application (more on this later) |
| styles.css | Global style definitions |
| test.ts | Code to run the application tests |
| tsconfig.json | Typescript/compiler configuration file |

Modify the Start Project

1. Open a terminal window.
2. Navigate to the start-app folder and ensure that the ng server command is running.
3. Edit the file `src/app/app.component.ts` by changing it to the following:

```
title = 'app works! and has been modified....';
```



Compile Errors

Runtime Errors



File Watcher and Web Server

ng serve running, this watches our files (performing a compile and redeploy when necessary) and runs a local web server on localhost:4200. When you change something and click Save, the watcher does the following:

- Creates a Webpack build, including transpilation to compatible JavaScript and bundling code (more on Webpack later in this book)
- Generates a new index.html file, adding script references as required to reference the JavaScript files bundled by Webpack
- Performs a new deployment onto the local web server
- Refreshes the web page

Bootstrapping

Bootstrapping usually refers to a self-starting process that's supposed to proceed without external input. In this case, it refers to how an Angular application starts up.

- 1.The web browser opens the file index.html by default.
- 2.The browser loads the script files on the end. This includes main.bundle.js, which is a transpiled version of the typescript file main.ts. This is our main app entry point.
- 3.Main.bundle.js loads some modules then calls the following Angular system code:

```
platformBrowserSpecific().bootstrapModule(AppModule)
```

4.AppModule is loaded—it's the root Angular module used to bootstrap the application. This is an Angular module, not a JavaScript module they're different things. If you look at AppModule.ts you'll see that it contains the following line to tell the module to bootstrap with the AppComponent::

```
@NgModule({  
  ...  
  bootstrap: [AppComponent]  
})
```

5.The AppModule bootstraps with the AppComponent , injecting the component into the space between the start and end tags app-root:

```
<app-root>Loading...</app-root>
```

Useful CLI Options

- **--flat** :Generates a cli project with a flat file structure, not generating each component in its own directory.
- **--inline-template** :Generates components with inline templates. Component template markup will be generated within the component rather than in a separate file.
- **--inline-style**: Generates components with inline styles (more on those later). Component styles will be generated within the component rather than in a separate file.
- **--spec false**: Generates component without the unit testing 'spec' files that are normally generated for you by default.
- **--minimal** :Generates a minimal cli project with inline templates, styles and without tests.

The package.json File

Node is designed to be run from the command line within a project folder. It allows developers to store information pertinent to their project in a package.json file, which should reside in the root folder of your project. This file specifies many useful things about your project:

- The name and version of your project.
- What Node modules your project depends on (and what versions of these modules you need).
- What Node modules are required for your project in production.
- What Node modules are required for your project in development (that is, not needed for production).

Updating package.json

- By using node commands (on the command line) that install/update/delete Node modules and update this file.
- By editing this file yourself. Then you run Node commands to install/update/delete Node modules to match this file.

To install the dependencies outlined in the package.json file, enter the following on the command line in the root folder of your project:

```
npm install
```

The Folder `node_modules`

When you install a Node module , it's downloaded and placed into the subfolder `node_modules` within your project folder.

Sometimes it takes a long time for npm to download and install the project Node modules.

There are two different ways of installing modules into Node. You can run the command `npm install` specifying the module (to install it) or you can edit the `package.json` file and then run npm install.

Installing The Latest Angular

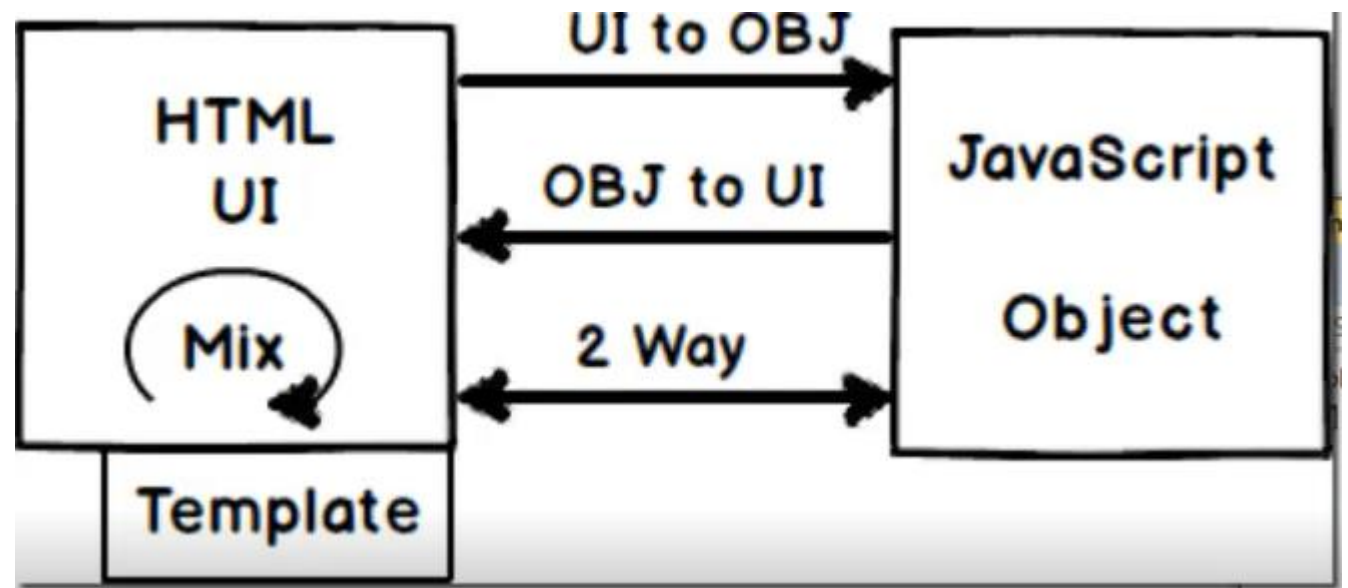
- `npm install -g @angular/cli`
- `ng new my-dream-app`
- `cd my-dream-app`
- `ng serve`

Data Bindings



Data Bindings

1. One Way
 1. UI to Object
()
 2. Object to UI
[]
2. Two way [()]
3. Interpolation { }
4. Template




Angular Components, Directives and Pipes

Topics

- What are components
- Templates and Expressions
- Different Types of Data Bindings
- Most Important Angular Directives
- Most Important Angular Pipes



Angular Components

- What are components
 - Example of Angular components
 - Let's create a component and use it
- 

What are components

- Components are HTML elements that we create.
- An Angular application is a tree of components.

```
<my-app>
  <header title="A title"></header>
  <menu [items]="items"></menu>
  <data-view></data-view>
</my-app>
```

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

@Component({
  selector: 'app-hello',
  template: `
    <div>
      <h2>Hello {{name}}</h2>
    </div>
  `
})
export class HelloComponent {

  name: string;

  constructor() {
    this.name = 'Angular';
  }
}
```

To create a component

```
$ ng generate component Another
```

```
installing component
```

```
create src/app/another/another.component.css  
create src/app/another/another.component.html  
create src/app/another/another.component.spec.ts  
create src/app/another/another.component.ts  
update src/app/app.module.ts
```

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

```
@Component({  
  selector: 'app-another',  
  templateUrl: './another.component.html',  
  styleUrls: ['./another.component.css']  
})
```

```
export class AnotherComponent implements OnInit {
```

```
  constructor() { }
```

```
  ngOnInit() {  
    ;  
  }  
}
```

another works!

```
▼ <body>
```

```
▼ <app-root ng-version="4.3.4">
```

```
▼ <app-another _ngghost-c0>
```

```
  <p _ngcontent-c0>  
    another works!  
  </p>
```

```
  </app-another>
```

```
</app-root>
```

```
<script type="text/javascript">
```

html

Styles Computed Event Listeners >>

Filter :hov .cls +

⋮ Console Sensors X

Templates and Expressions

- What expression can do
- How to customize HTML templates
- CSS scoping in components

Templates and Expressions

What expression can do `{{syntax}}`

- Expressions binds data from your model to your HTML template
 - When the data changes, the HTML gets automatically refreshed
- Expressions run in the context of your component
 - Any method or property of your component class can be used in an expression
- Expressions can use regular JavaScript features like appending text or defining conditions using the ternary operator – condition ? Then : else

Typical Expressions

```
@Component({
  selector: 'app-hello',
  template: `
    <div>
      <h2>Hello {{firstName}} {{name}}</h2>
    </div>
  `
})
export class HelloComponent {

  name : string;
  firstName : string;

  constructor() {
    this.name = 'Angular';
    this.firstName = 'John';
    setTimeout(callback: () => this.name = 'A new value', ms: 5000)
  }
}
```

Hello John Angular

Hello John A new value

```
...<html> == $0
  ▶ <head>...</head>
  ▶ <body>...</body>
  </html>
```



Typical HTML Templates

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

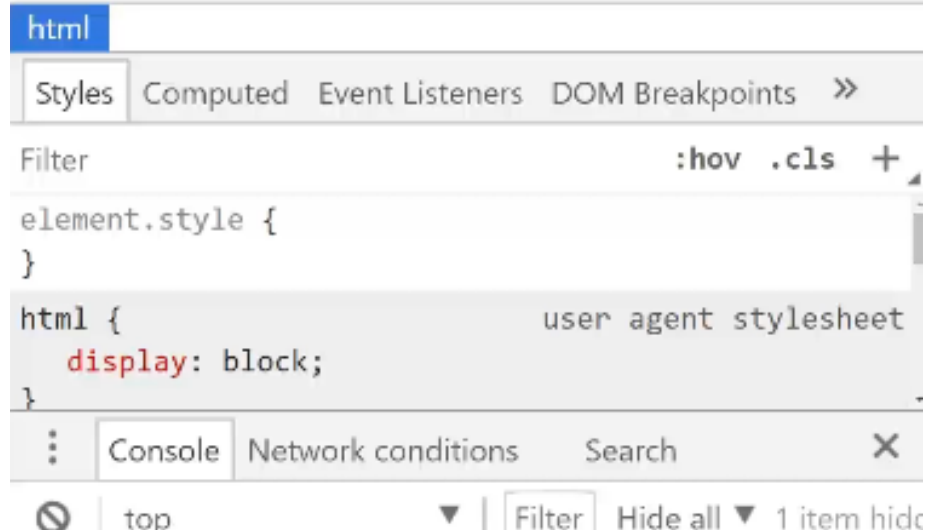
@Component({
  selector: 'app-hello',
  templateUrl: 'template.html'
})
export class HelloComponent {

  name : string;
  firstName : string;

  constructor() {
    this.name = 'Angular';
    this.firstName = 'John';
  }
}
```

Hello John Angular

```
...<html> == $0
  ▶ <head>...</head>
  ▶ <body>...</body>
  </html>
```



Typical CSS Templates

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

```
@Component({  
  selector: 'app-hello',  
  templateUrl: './template.html',  
  styleUrls: ['./style.css']  
})
```

```
export class HelloComponent {
```

```
  name : string;  
  firstName : string;
```

```
  constructor() {  
    this.name = 'Angular';  
    this.firstName = 'John';  
  }  
}
```

Hello John Angular

```
<html>  
  <head>...</head>  
  <body>  
    <app-root ng-version="4.3.6">  
      <app-hello _ngghost-c0>  
        <div _ngcontent-c0>  
          <h2 _ngcontent-c0>Hello John Angular  
          </h2> == $0  
        </div>  
      </app-hello>  
    </body>  
  </html>
```

html body app-root app-hello div h2

Styles Computed Event Listeners DOM Breakpoints >>

| Filter | :hov .cls + |
|---------------------|-----------------------|
| h2[_ngcontent-c0] { | <style>...</style> |
| color: red; | |
| } | |
| h2 { | user agent stylesheet |
| display: block; | |

Templates and Expression

Templates and Expression make our HTML code dynamic

- Expressions are JavaScript that runs on every time our data model gets updated.
- Templates use expressions to make the HTML dynamic.
- Both HTML and CSS are scoped at the component level.

Different Types of Data Bindings

- One-way data bindings with [] and {{ }}
 - One-way data bindings with ()
 - Two-way data bindings with [(ngModel)]
-
- Expressions {{ }} and bindings with [] are one-way data bindings
 - Event bindings with () are one-way data bindings
 - [(ngModel)] works both ways

One-way data-bindings

One-way data-bindings with [] and {{ }}

- Square brackets [] can be used to bind a component property to an HTML property
 - For instance:

```
<input type="text" [value]="name" />
```

- Expressions with {{ }} work the same way
 - The Main difference is that expressions can be used everywhere, not only on HTML properties:

```
<div>{{name}}</div>
```

- Parentheses () can be used to register an event listener
 - For instance:

```
<button (click)="alertName()">Click me</button>
```

- Parentheses define a data-binding from the HTML to your component
- Square brackets and expressions define a data-binding from your component to the HTML
- That's why all of them are one-way data bindings


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

```
@Component({
```

```
  selector: 'app-bindings',
```

```
  templateUrl: 'bindings.component.html'
```

```
})
```

```
export class BindingsComponent {
```

```
  name : string = "A Test";
```

```
  alertName() : void {
```

```
    alert(`${this.name} was clicked`);
```

```
  }
```

```
}
```

Most Important Angular Directives

- What are directives
- *ngFor and *ngIf
- *ngSwitch
- *ngClass

ngFor Directive

*ngFor Directive

- ngFor is a structural directive
- Equivalent of AngularJS ng-repeat
- A loop to iterate through data model items:

```
<ul>
  <li *ngFor="let person of persons">
    {{ getDisplayName(person) }}
  </li>
</ul>
```

ngIf Directive

ngIf Directive

- ngIf is structural directive
- Equivalent of AngularJS ng-if
- A way to display an HTML element based on a condition:

```
<span *ngIf="person.firstName.length > 3">  
  {{ getFullName(person) }}  
</span>
```

ngSwitch directive

ngSwitch directive

- Switch-case implementation for Angular
- Allows to display content conditionally:

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

@Component({
  selector: 'app-ngswitch',
  template: `
    <div [ngSwitch]="page">
      <p *ngSwitchCase="1">Viewing content of first page</p>
      <p *ngSwitchCase="2">Viewing content of second page</p>
      <p *ngSwitchCase="3">Viewing content of third page</p>
      <p *ngSwitchDefault>No page selected</p>
    </div>
    <div>
      <button (click)="page = 1">Page 1</button>
      <button (click)="page = 2">Page 2</button>
      <button (click)="page = 3">Page 3</button>
    </div>
  `
})
export class NgswitchComponent {
```

No page selected

Page 1 Page 2 Page 3

Viewing content of first page

Page 1 Page 2 Page 3

Viewing content of second page

Page 1 Page 2 Page 3

Viewing content of third page

Page 1 Page 2 Page 3


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

```
@Component({
```

```
  selector: 'app-ngswitch',
```

```
  template: `
```

```
    <div [ngSwitch]="page">
```

```
      <p *ngSwitchCase="1">Viewing content of first page</p>
```

```
      <app-hello *ngSwitchCase="2">Viewing content of second page</app-hello>
```

```
      <p *ngSwitchCase="3">Viewing content of third page</p>
```

```
      <p *ngSwitchDefault>No page selected</p>
```

```
    </div>
```

```
    <div>
```

```
      <button (click)="page = 1">Page 1</button>
```

```
      <button (click)="page = 2">Page 2</button>
```

```
      <button (click)="page = 3">Page 3</button>
```

```
    </div>
```

```
  })
```

```
export class NgswitchComponent {
```

Hello Angular

Page 1

Page 2

Page 3

Directives

Directives are a way to manipulate the DOM to make it dynamic

- `*ngIf` can remove HTML elements conditionally
- `*ngFor` can repeat elements `ngSwitch` is similar to `*ngIf` with multiple conditions

Most Important Angular Pipes

- What are pipes
- String pipes
- Number pipes
- Data pipe
- How to create our own pipes

What are pipes

- Pipes are used to format data for display
- Equivalent of AngularJS filters
- Pipes can be chained
- We can write our own pipes

String pipes

String pipes

- Uppercase, lowercase and titlecase are pipes used to format strings
- There are also pipes for internationalization (i18n)
- Here is an example of how to use uppercase:

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

@Component({
  selector: 'app-ngfor',
  template: `
    <ul>
      <li *ngFor="let person of persons">
        {{ getDisplayName(person) | uppercase | lowercase }}
      </li>
    </ul>`
})
export class NgforComponent {

  public persons = [
    {
      "_id": "58c8684e9e416245fffb60088",
      "name": "Edith Eaton",
      "company": "DADABASE",
      "email": "editheaton@dadabase.com",
      "phone": "+1 (854) 486-2088",
      "address": "280 Florence Avenue, Salix, Alaska, 3036",
    }
  ]
}
```



API List

TYPE: **P** Pipe

STATUS: All

 Filter

common

P AsyncPipe

P I18nSelectPipe

P CurrencyPipe

P SlicePipe

P DatePipe

P JsonPipe

P DecimalPipe

P UpperCasePipe

P I18nPluralPipe

P LowerCasePipe

P PercentPipe

P TitleCasePipe

Number Pipes

Number Pipes

- Number, percent and currency are pipes use to format numbers
- These pipes take parameters to be customized

```
@Component({
  selector: 'currency-pipe',
  template: `<div>
    <p>A: {{a | currency:'USD':false}}</p>
    <p>B: {{b | currency:'USD':true:'4.2-2'}}</p>
  </div>`
})
export class CurrencyPipeComponent {
  a: number = 0.259;
  b: number = 1.3495;
```

Date pipe

Date pipe

- Allows to format dates with some predefined or custom formats:

```
<ul>
  <li *ngFor="let person of persons">
    {{ person.birthDate | date: 'shortDate' }}
  </li>
</ul>
```

- `expression` is a date object or a number (milliseconds since UTC epoch) or an ISO string (<https://www.w3.org/TR/NOTE-datetime>).
- `format` indicates which date/time components to include. The format can be predefined as shown below or custom as shown in the table.
 - `'medium'` : equivalent to `'yMMMdjms'` (e.g. Sep 3, 2010, 12:05:08 PM for en-US)
 - `'short'` : equivalent to `'yMdjM'` (e.g. 9/3/2010, 12:05 PM for en-US)
 - `'fullDate'` : equivalent to `'yMMMMEEEEd'` (e.g. Friday, September 3, 2010 for en-US)
 - `'longDate'` : equivalent to `'yMMMMd'` (e.g. September 3, 2010 for en-US)
 - `'mediumDate'` : equivalent to `'yMMMd'` (e.g. Sep 3, 2010 for en-US)
 - `'shortDate'` : equivalent to `'yMd'` (e.g. 9/3/2010 for en-US)
 - `'mediumTime'` : equivalent to `'jms'` (e.g. 12:05:08 PM for en-US)
 - `'shortTime'` : equivalent to `'jm'` (e.g. 12:05 PM for en-US)

Date Pipe

Date Pipe

- Angular CLI is here to help:
 - ng generate pipe myCustomPipe
- CLI generates the default structure for the pipe
 - All we have to do is implement the transforms() method:

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'myCustom'
})
export class MyCustomPipe implements PipeTransform {

  transform(value: any, args?: any): any {
    return null;
  }

}
```


Forms

Angular 4 has a new Forms module that makes it easier to do the following:

- Create forms dynamically
- Validate input with common validators (required)
- Validate input with custom validators

Form Model Objects

Below supports both ways of writing forms:

- FormGroup
- FormControl

Stores state information for the form, including the following:

- Values for all the controls inside the form
- Groups of fields in the form
- Fields in the form
- Validators

Form Group

Stores the value and validity state of a group of FormControl instances:

- Values for all the controls inside the form group

Stores the value and validity state of an individual control—for instance a listbox:

- Value
- Validation state
- Status (for example, disabled)

Supports both methods of writing forms: template and Reactive. When you have form validation, you need to highlight invalid data when it occurs.

| Style | Description |
|--------------|---|
| ng-touched | Style applied if control has lost focus |
| ng-untouched | Style applied if control hasn't lost focus yet |
| ng-valid | Style applied if control passes validation |
| ng-invalid | Style applied if control doesn't pass validation |
| ng-dirty | Style applied if user has already interacted with the control |
| ng-pristine | Style applied if user hasn't interacted with the control yet |

Topics

- Component Communication in Angular
- What are Services?
- Understanding Dependency Injection in Angular
- Using the HttpClient to Interact with the Server

Why would components communicate

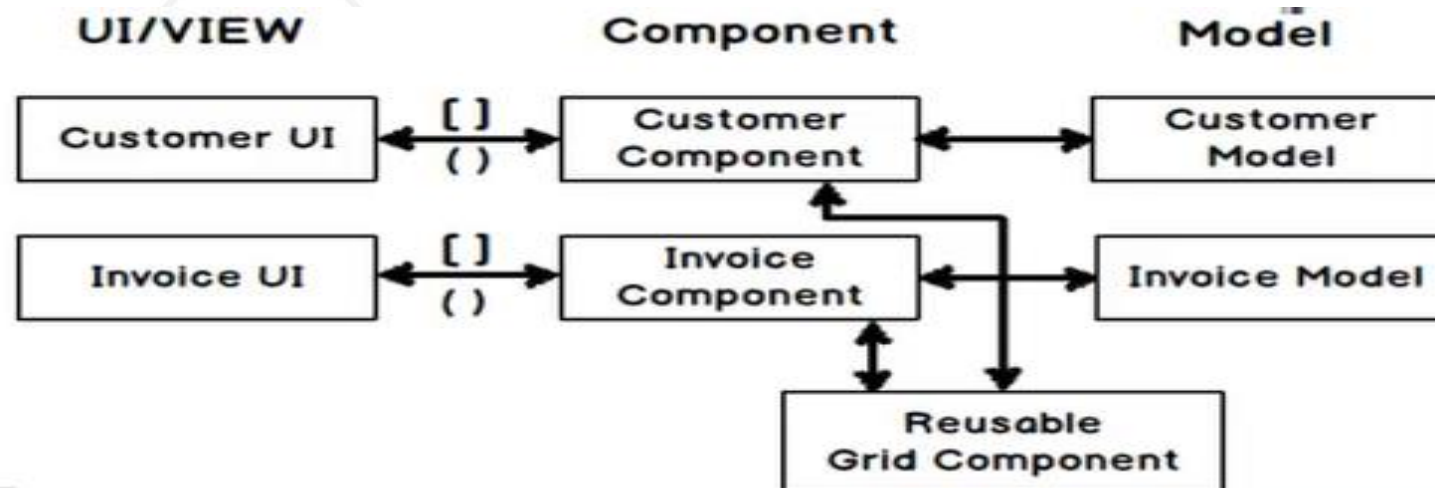
Why would components communicate

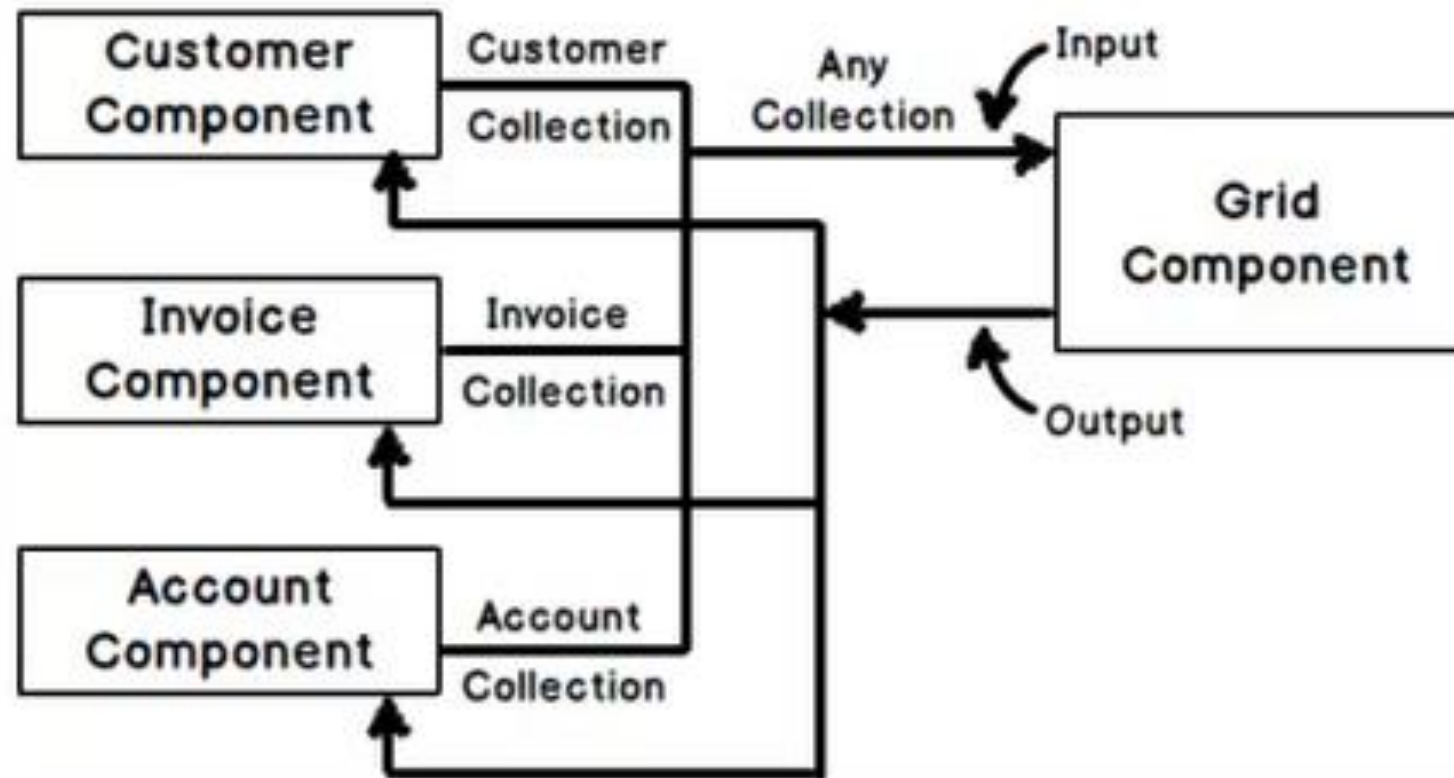
- To share some data – username, session and so on
- To react to events triggered by other components when user clicks here, do something elsewhere
- To avoid retrieving the same data from the server over and over again

Using @Input and @Output

Using @Input and @Output

- An Angular app is a tree of components that can interact with one another
- @Input() and @Output() are decorators made to achieve that
- @Input() is a way to pass data to a component
- @Output() is a way to emit data from a component through events






```
@Component({
  selector: 'app-popup-window',
  templateUrl: './popup-window.component.html',
  styleUrls: ['./popup-window.component.css']
})
export class PopupWindowComponent implements OnInit {

  @Input()
  isOpen = false;

  @Input()
  title = "Title";

  @Output()
  onClose = new EventEmitter<string>();

  ngOnInit() {
    console.log('NG ON INIT');
  }
}
```

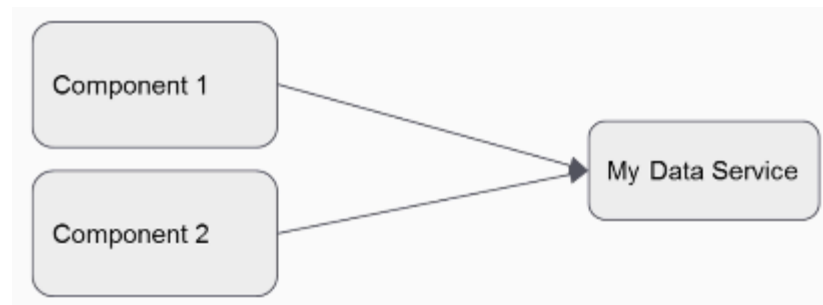
Components

Components can use services or specific decorators to communicate and to share data

- Services are the best option:
- Very flexible and loosely coupled.
- @Input and @Output can be used to have two-way communication between parent/child or siblings.
- @ViewChild can inject a reference to a child component.

Using services for component communication

- One way to achieve data-sharing between components is to use services
- Services are singletons so these two components would access the same data once loaded



What are Services

- What are Services
- The @Injectable decorator
- The providers array
- Role of constructors in dependency injection

What are Services

- Services are meant to handle application logic – fetch data, update data and so on
- They are used to interact with servers
- Unlike components, they don't have a HTML template
- They are classes that can be injected

Create Services using CLI

CLI Supports to create services using below command:

`ng generate service my-new-service`

Note: there will be a warning while creating a service that the service needs to be added in the Provider's Array of the module.

The @Injectable Decorator

The @Injectable Decorator

- The @Injectable decorator turns a class into an injectable services.

```
import { Injectable } from '@angular/core';  
  
@Injectable()  
export class DataService {
```

The Providers Array

The Providers Array

- Any class marked as injectable is candidate for dependency injection
- In order for dependency injection to be possible, that class has to be added to the providers array of any injector:

```
@NgModule({  
  declarations: [...],  
  imports: [...],  
  providers: [  
    AuthGuard, LoginService, PostsService, DataService  
  ],  
  bootstrap: [AppComponent]  
})  
export class AppModule { }
```


Services

Services are classes that can be injected in any other Angular object

- Business logic is meant to be handled in services
- Any class decorated with `@injectable` is a service
- The array of providers lists classes that can be injected

What is dependency injection

What is dependency injection

- It's a technique where an object supplies the dependencies of another object
- Dependency injection is one form of inversion of control
- Main goal – decouple objects from their dependencies so that dependencies can easily get swapped at runtime
- Useful for testing, mocking and stubbing

In most web applications, users navigate from one page to the next as they perform application tasks. Users can navigate in these ways:

- Entering a URL in the address bar
- Following links, clicking buttons, and so on
- Going backward or forward in the browser history

The router can interpret a browser URL as an instruction to navigate to a component and pass optional parameters (which contain information) to the component to give it contextual information and help it decide which specific content to present or what it needs to do.

Routers

| Object | Type | Description |
|---------------------|-----------|---|
| RouterModule | Module | A separate Angular module that provides the necessary service providers and directives for navigating through application views. |
| Router | | Displays the application component for the active URL. Manages navigation from one component to the next. |
| RouterOutlet | Directive | The directive (<router-outlet>) that marks where the router displays a view. |
| RouterLink | Directive | The directive for binding a clickable HTML element to a route. Clicking an element with a RouterLink directive that's bound to a string or a link parameters array triggers a navigation. |
| | | |

Routers

| Object | Type | Description |
|-----------------------|------|---|
| ActivatedRoute | | A service that's provided to each route component that contains route-specific information such as route parameters, static data, resolve data, global query params, and the global fragment. |
| RouterState | | The current state of the router including a tree of the currently activated routes together with convenience methods for traversing the route tree. |

Configuring the Component Router

- It requires a routing table which maps a URL to a component (app.routing.ts)

```
const appRoutes: Routes = [  
  {  
    path: 'if', component: NgifComponent  
  },  
  {  
    path: '', component: HelloComponent  
  },  
  {  
    path: 'http', component: HttpComponent  
  }  
];  
export const routing = RouterModule.forRoot(appRoutes);
```

Defining a router-outlet

- <router-outlet> is the place where routed components are loaded in the HTML template

```
<h1>Component Router</h1>
<ul>
  <li><a routerLink="/" >Hello World example</a></li>
  <li><a routerLink="/if" >ngIf example</a></li>
  <li><a routerLink="/http">HTTP example</a></li>
</ul>

<hr>
<router-outlet></router-outlet>
```

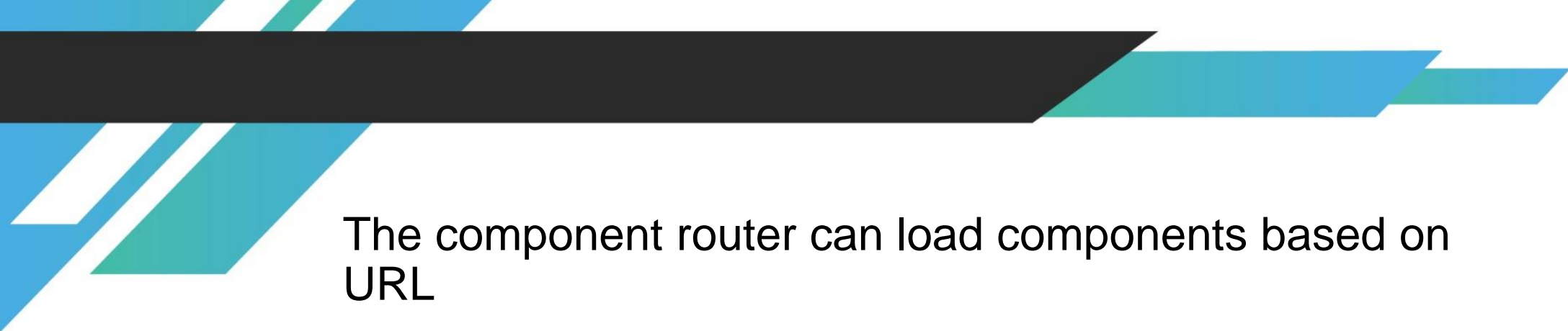
Parameters

- Parameters have to be named in the router config:


```
{  
  path: 'details/:id', component: DetailsComponent  
},
```

- Then parameters can be retrieved by injecting the `ActivatedRoute` service and subscribing to its `paramMap`:

```
constructor(public activatedRoute : ActivatedRoute) {  
  activatedRoute.paramMap  
    .subscribe((params: ParamMap) =>  
      this.id = params.get('id'))  
}
```

The component router can load components based on URL

- We have to define a routing table
 - `<router-outlet>` indicates where routed component get loaded in the HTML
 - Routes can have parameters and we can subscribe to parameter updates
- 

Template Driven Forms in Angular

- Overview of forms in Angular
- Template driven forms
- Properties added by Angular to help with validation
- Example of field validation in a form

Overview of Forms in Angular

Overview of Forms in Angular

- Angular provides tools to automate form management:
 - Data bindings
 - Form submission
 - Form validation
 - Error handling
- There are two different ways to handle forms:
 - Template-driven
 - Model-driven or Event-driven or Reactive Forms

Reactive Forms in Angular

- The FormBuilder object to create a reactive form
- How to use form validation properties in our template
- Example of field validation in a reactive form
- Writing our own custom validation function

The FormBuilder object

The FormBuilder object to create a reactive form

- The set-up of the form happens in TypeScript instead of in HTML template, using a FormBuilder object:

```
export class FormsComponent {  
  
  registerForm: FormGroup;  
  
  constructor(private formBuilder: FormBuilder) {  
    this.registerForm = this.formBuilder.group({  
      firstname: ['', Validators.required],  
      lastname: '',  
      street: '',  
      zip: '',  
      city: ''  
    })  
  }  
}
```

How to use form validation properties in our template

- We bind the formGroup property to our registerForm property, which returns the form into a reactive form:

```
<form [formGroup]="registerForm" (ngSubmit)="logForm()">
  <label>Firstname:</label>
  <input type="text" name="firstname" formControlName="firstname" >
  ...
  <button type="submit">Submit</button>
</form>
```

- Then we can access every element by name from the controls property as follow: registerForm.controls.firstname

Example of Reactive Form

```
<label>Firstname:</label>
<input type="text" name="firstname" formControlName="firstname" />
<br/>
<label>Lastname:</label>
<input type="text" name="lastname" formControlName="lastname" />
<br/>
<label>Street:</label>
<input type="text" name="street" formControlName="street" />
<br/>
<label>Zip:</label>
<input type="text" name="zip" formControlName="zip" />
<br/>
<label>City:</label>
<input type="text" name="city" formControlName="city" />
<br/>
<button type="submit">Submit</button>
</form>
```

Firstname:

Lastname:

Street:

Zip:

City:

Submit

Example of Custom Validator

```
export class FormsComponent {  
  
  registerForm: FormGroup;  
  
  constructor(private formBuilder: FormBuilder) {  
    this.registerForm = this.formBuilder.group({  
      firstname: ['', Validators.required],  
      lastname: '',  
      street: '',  
      zip: ['', Validators.required, Validators.pattern("[0-9]"),  
      city: ''  
    });  
  }  
}
```

Firstname:

Lastname:

Street:

Zip:

Zipcode must have 5 digits

City:

Thank You !

