



Copyright 2018-2020 Funny Ant LLC

1



# npm QuickStart

Node.js package manager

2

# Installing Dependencies

Package manager

3

## Global Installs

```
$ npm install typescript --global // -g for short
```

4

# Global Packages Location

`$ npm get prefix`

Mac:

`/Users/[username]/.npm-packages/lib/node_modules`

PC:

`%AppData%\npm\node_modules` (Windows 7)

`%USERPROFILE%\Application Data\npm\node_modules` (Windows XP)

5

# Local Installs

`$ npm init -y #creates package.json -y uses defaults`

`$ npm install typescript --save-dev #saves in package.json  
#creates node_modules directory`

`$ tsc -v #fails because can't find package`

`$ node_modules/.bin/tsc -v #outputs version info`

6

# package.json

dependencies and devDependencies

\$ npm install rxjs --save

```
{
  "name": "npm-quickstart",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {},
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "rxjs": "^5.2.0"
  }
}
```

\$ npm install typescript --save-dev

```
{
  "name": "npm-quickstart",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {},
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "rxjs": "5.2.0"
  },
  "devDependencies": {
    "typescript": "^2.2.1"
  }
}
```

## Semantic Versioning

- Major.Minor.Patch
- If you were starting with a package 1.0.4, this is how you would specify the ranges:
  - Patch releases: ~1.0.4
  - Minor releases: ^1.0.4
  - Major releases: \* or x | 1.0.4 → 2.0.0

## Sharing Dependencies

```
$ npm install //installs dependencies and devDependencies  
$ npm install --production //installs dependencies only
```

9

## Updating Dependencies

```
$ npm update lite-server -g //update a global dependency  
$ npm update rxjs --save //update a local dependency
```

10

## Uninstalling Dependencies

```
$ npm uninstall lite-server -g //uninstall a global dependency  
$ npm uninstall rxjs --save //uninstall a local dependency
```

11

## Lab

Node Package Manager (npm)

Open **TypeScriptLabManual.pdf** and follow the directions to do the following sections:

Create Project

Install TypeScript & RxJS

13

## Understanding package-lock.json

- In an ideal world, the same **package.json** should produce the exact same **node\_modules** tree, at any time
- In some cases, this is indeed true. But in many others, npm is unable to do this
- To **reliably** produce the **exact node\_modules tree**, **package-lock.json** was created.

14

## Broken package.json scenarios

- Different versions of npm (or other package managers) may have been used to install a package, each using slightly different installation algorithms.
- A new version of a direct semver-range package may have been published since the last time your packages were installed, and thus a newer version will be used.
- **A dependency of one of your dependencies may have published a new version, which will update even if you used pinned dependency specifiers (1.2.3 instead of ^1.2.3)**
- The registry you installed from is no longer available, or allows mutation of versions (unlike the primary npm registry), and a different version of a package exists under the same version number now.

15

## Using npm as a Build Tool

Build Automation | npm scripts

16

## What about Grunt and Gulp?

- npm's scripts directive can do everything that these build tools can, more succinctly, more elegantly, with less package dependencies and less maintenance overhead
- What's Wrong with Gulp and Grunt?
  1. Complexity
  2. Dependence on plugin authors
  3. Frustrating debugging
  4. Disjointed documentation

17



## Your First Script

```
$ npm init -y //creates package.json
{
  "name": "npmscriptsdemo",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "hi": "echo hello world "
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}

$ npm run-script hi //hello world
```



18

## Shortcuts

Instead of:

```
$ npm run-script [script-name]
```

You could use the shorter:

```
$ npm run [script-name]
```

Or if it's one of the npm supported scripts you can omit the run command:

```
$ npm [script-name]
```

19

## Supported Scripts

- prepublish: Run BEFORE the package is published. (Also run on local **npm install** without any arguments.)
- publish, postpublish: Run AFTER the package is published.
- preinstall: Run BEFORE the package is installed
- install, postinstall: Run AFTER the package is installed.
- preuninstall, uninstall: Run BEFORE the package is uninstalled.
- postuninstall: Run AFTER the package is uninstalled.
- preversion, version: Run BEFORE bump the package version.
- postversion: Run AFTER bump the package version.
- pretest, test, posttest: Run by the **npm test** command.
- prestop, stop, poststop: Run by the **npm stop** command.
- prestart, start, poststart: Run by the **npm start** command.
- prerestart, restart, postrestart: Run by the **npm restart** command.
  - Note: npm restart will run the stop and start scripts if no restart script is provided.

20

## Running Local Node Modules

```
$ npm init #creates package.json

$ npm install typescript --save-dev #saves in package.json
                                     #creates node_modules directory

$ tsc -v #fails because can't find package

$ node_modules/.bin/tsc -v #returns version

//package.json snippet
"scripts": {
  "tsc": "tsc"
}

$ npm run tsc -- -v #returns version but path not required in package.json
```

21

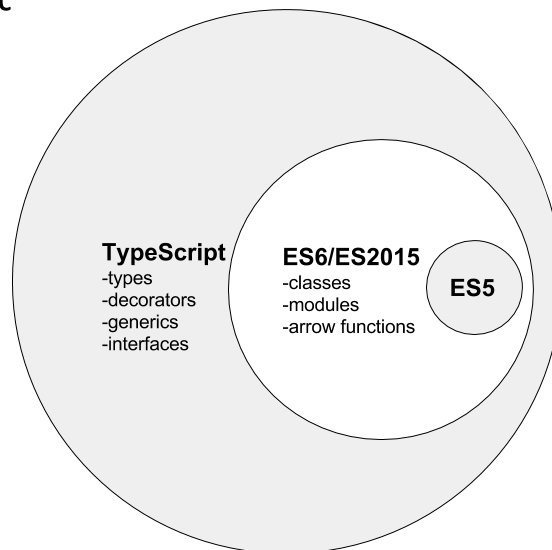


# TypeScript/ES2015

Introduction

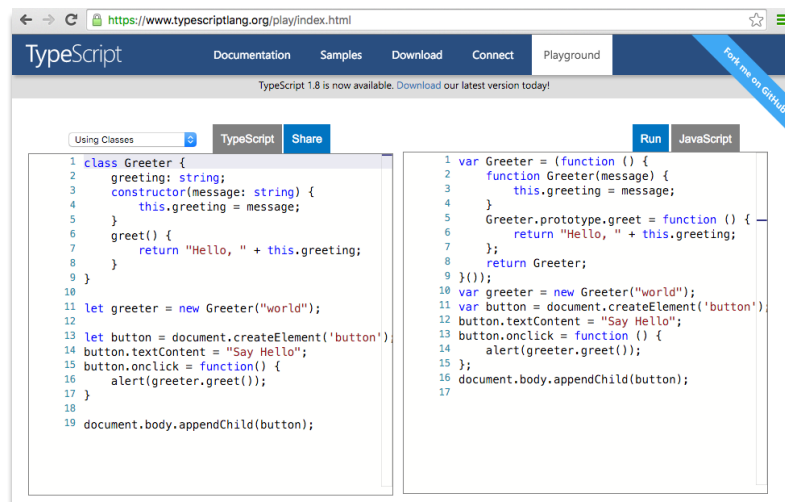
23

## TypeScript



24

# How TypeScript Works



25

# Why TypeScript?

- TypeScript is a primary language for Angular application development
  - Angular is written in TypeScript.
- Recommended by Angular core team and Google
- Types enable better tooling including
  - Refactoring
  - Navigating code
  - Code completion
- Decorators (annotations) provide an easily understandable API

26

## Who is Behind TypeScript?

### Anders Hejlsberg

Core developer of TypeScript

Microsoft technical fellow

Lead architect C#

Original author Turbo Pascal: Delphi



Image from Wikimedia Commons

27

## Installing TypeScript

### Global

```
npm install -g typescript
```

### Local

```
npm init //creates package.json
```

```
npm install typescript --save-dev //saves version to package.json
```

```
//To ensure you're getting local version run using npm scripts
```

28

## Configuring TypeScript

```
tsc --init
message TS6071: Successfully created a tsconfig.json file.

//default tsconfig.json
{
  "compilerOptions": {
    "target": "es5",
    ...
  }
}
```

29

## Compiling with TypeScript

```
tsc
//OR
tsc --watch
```

30

## JavaScript is valid TypeScript


Q: What would be the output of the following JavaScript code run through the TypeScript compiler?

```
function greeter(name) {  
    return "Hello " + name;  
}  
  
console.log(greeter("Anders"));
```

A: The same code.

31

## Type Annotations



```
function greeter(name: string) {  
    return "Hello " + name;  
}  
  
console.log(greeter("Anders"));
```

32

# Lab

Node Package Manager (npm) & TypeScript

Open **TypeScriptLabManual.pdf** and follow the directions to do the following sections:

Run TypeScript

Type Annotations

33

## ES2015 Classes

```
class Student{  
  public firstName: string;  
  middleInitial: string;  
  lastName: string;  
  
  constructor(firstName: string,  
    middleInitial: string,  
    lastName: string){  
  
    this.firstName = firstName;  
    this.middleInitial = middleInitial;  
    this.lastName = lastName;  
  }  
  
  getFullName(){  
    return this.firstName + " " + this.middleInitial + ". " + this.lastName;  
  }  
}  
  
let student = new Student("John", "D", "Rockefeller");  
console.log(student.getFullName());
```

34



## TypeScript Automatic Property Assignment

```
class Student {  
    firstName: string;  
    middleInitial: string;  
    lastName: string;  
  
    constructor(firstName, middleInitial, lastName) {  
        this.firstName = firstName;  
        this.middleInitial = middleInitial;  
        this.lastName = lastName;  
    }  
}  
-----  
class Student {  
    constructor(public firstName:string,  
                public middleInitial:string,  
                public lastName:string) {}  
}
```

Equivalent

35

## Classes & Interfaces

```
interface Person {  
    firstName: string;  
    lastName: string;  
}  
  
class Student implements Person {  
    fullName: string;  
    constructor(public firstName, public middleInitial, public lastName) {  
        this.fullName = firstName + " " + middleInitial + " " + lastName;  
    }  
}  
  
function greeter(person : Person) {  
    return "Hello, " + person.firstName + " " + person.lastName;  
}  
  
let student = new Student("Jon", "M.", "Turner");  
console.log(greeter(student));
```

Interface

Class that implements interface

Function that takes parameter of the interface type

Create instance of class and invoke method

36

## Scope: var, let, const

- var
  - Function scope
- let
  - Block scope
- const
  - Block scope
  - Cannot change or be redeclared

37

## ES2015 Arrow Functions

```
let evens = [0,2,4,6,8];  
  
//verbose  
// let odds = evens.forEach(function(v) {  
//     return v+1;  
// });  
  
//terse  
let odds = evens.forEach(v => v + 1);  
  
odds.forEach(line=> console.log(line)); //1,3,5,7,9
```

38

# Lab

Node Package Manager (npm) & TypeScript

Open **TypeScriptLabManual.pdf** and follow the directions to do the following sections:

Classes

Scope

Arrow Functions

39

## ES2015 Arrow Functions (this)

without an arrow function

```
var organization={
  name: "Avengers",
  heroes: ["Hulk", "Iron Man", "Captain America"],
  printHeroes: function() {
    var self = this;
    this.heroes.forEach(function(hero) {
      console.log(hero + " is a member of the " + self.name + ".");
    });
  };
};
organization.printHeroes()
```

Capture *this* into *self* variable

this.name is undefined so we have to use self.name


40

## ES2015 Arrow Functions (this)

with an arrow function

```
var organization={
  name: "Avengers",
  heroes: ["Hulk", "Iron Man", "Captain America"],
  printHeroes: function() {
    this.heroes
      .forEach(h => console.log(h + " is a member of the " + this.name + "."));
  }
};

organization.printHeroes();
```



- *this.name* is "Avengers"
- Arrow functions don't create a new function context

41

## TypeScript Decorators

- With the introduction of Classes in ES2015, there now exist certain scenarios that require additional features to support metadata modifying classes and class members. Aspect-Oriented Programming (AOP).
- Decorators provide a way to add both annotations and a meta-programming syntax for class declarations and members
- Decorators are a stage 1 proposal for JavaScript and are available as an experimental feature of TypeScript
  - They may change in future releases

42

## TypeScript Decorators: Configuring

```
{
  "compilerOptions": {
    "module": "commonjs",
    "target": "es5",
    "noImplicitAny": false,
    "sourceMap": false,
    "experimentalDecorators": true
  }
}
```

43

## Decorators

```
function ClassLogger(target: Function) {
  console.log("The function that created this class is: " + target);
}

@ClassLogger
class Customer {
}

var customer = new Customer();
```

44



## ES Modules

ES Modules

45

## ES Modules Explained

- An ES6 module is a file containing JS code.
  - There's no special module keyword
  - A module reads like a script except:
    - ES modules are automatically strict-mode code
    - You can use import and export in modules
    - A JavaScript file is a module if it contains the import and/or export keyword
- Modules are executed within their own scope, not in the global scope
- Note: This feature is not implemented in any browsers natively at this time. It is implemented in by module loaders.

46

## ES Modules Syntax

- The **export statement** is used to export functions, objects or primitives from a given file (or *module*)
- The **import statement** is used to import functions, objects or primitives that have been exported from an external module, another script, etc.

47

## ES Module Example

```
//my-module.ts
export function myFunction(){
  return "myFunction was run.";
}

-----

//program.ts
import {myFunction} from "./my-module";
console.log(myFunction());
```

48

## ES Module Privacy Example


```
//my-module.ts
export function myFunction(){
  return "myFunction was run.";
}

function myPrivateFunction(){
  return "myPrivateFunction was run.";
}

-----

//program.ts
import {myFunction, myPrivateFunction} from "./my-module";

//Module has no exported member myPrivateFunction.
```



49

## ES Module Exporting

export function, object, primitive, and class

```
//my-module.ts

export function myFunction(){
  return "myFunction was run.";
}

var myObject = {
  name: 'I can access myObject\'s name',
  myMethod: function(){return 'myMethod on myObject is running.';}
}
export {myObject}

export const myPrimitive = 55;

export class MyClass{
  myClassMethod(){
    return "myClassMethod on myClass is running."
  }
}
```

50



# ES Module Importing

import function, object, primitive, and class

```
//program.ts

import {myFunction, myObject, myPrimitive, MyClass} from "./my-module";

console.log(myFunction());

console.log(myObject.name);

console.log(myObject.myMethod());

console.log(myPrimitive);

let myClass = new MyClass();
console.log(myClass.myClassMethod());
```

51

# ES2015 Template literals

```
class Student{

  constructor(private firstName: string,
               private middleInitial: string,
               private lastName: string){}

  getFullName(){
    return `
      First: ${this.firstName}
      Middle: ${this.middleInitial}
      Last: ${this.lastName}`;
  }
}
```

Use backticks `` | NOT single quotes "

52

# Lab

TypeScript

Open **TypeScriptLabManual.pdf** and follow the directions to do the following sections:

Modules

Do the remaining sections in the manual if time permits

53

# Review (optional)

TypeScript

54

## TypeScript Review

- Make a quick written list of 4-6 language features that TypeScript or ES6/ES2015 adds to JavaScript (ES5).
- Draw a set diagram like the one I showed earlier to show the relationship between ES5, ES6/ES2015, and TypeScript.

55



## Angular Overview

Big Picture

56

## Why Angular

Why would we build our application front-end in JavaScript?

Why would we use a single-page application framework?

What is your current understanding?

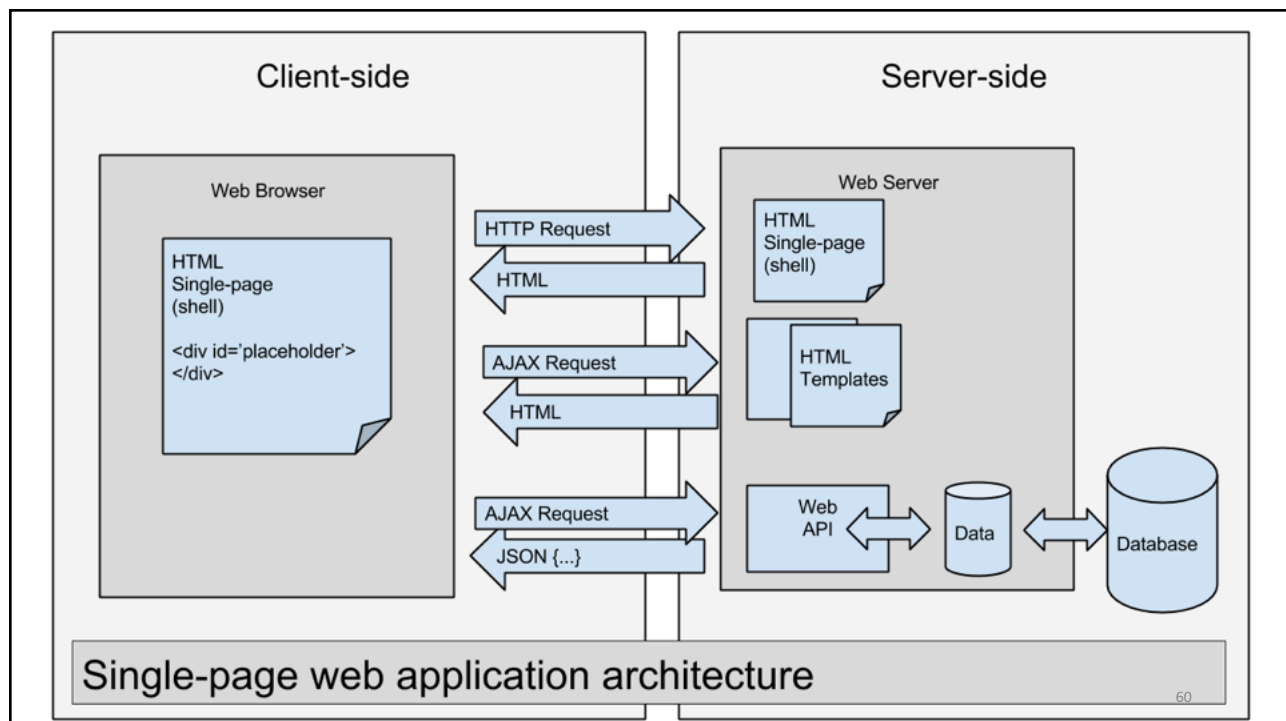
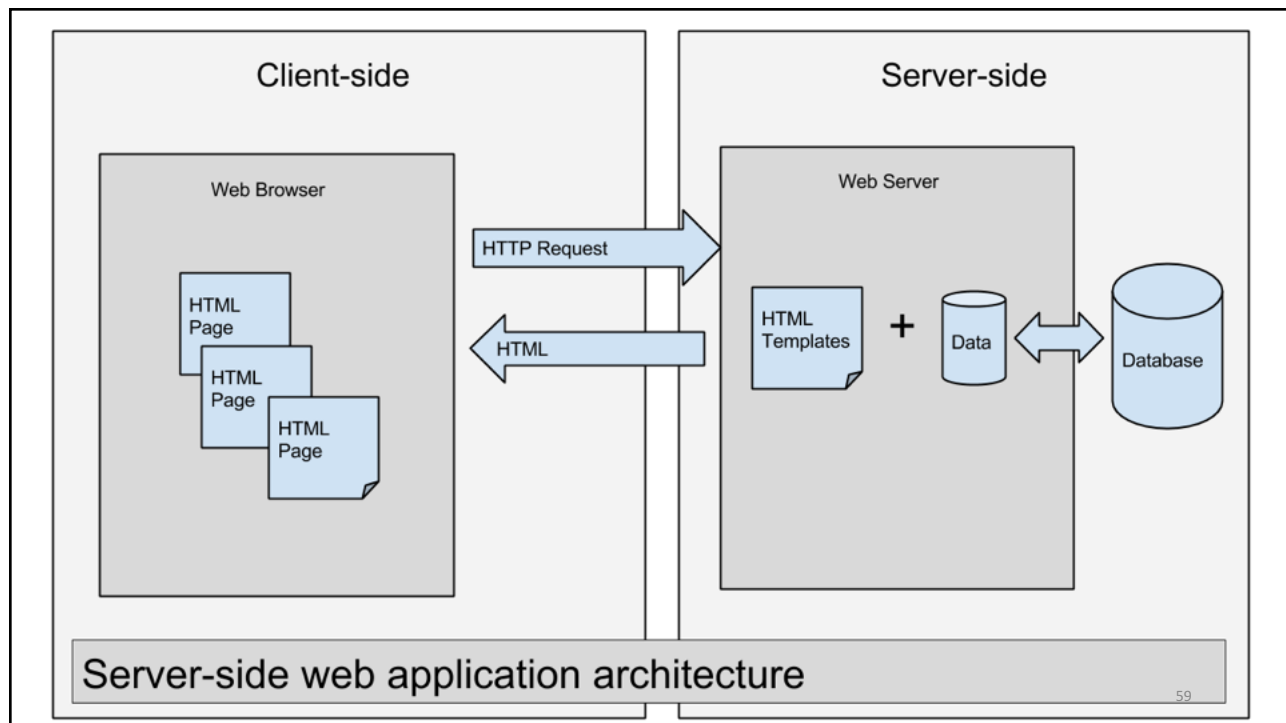


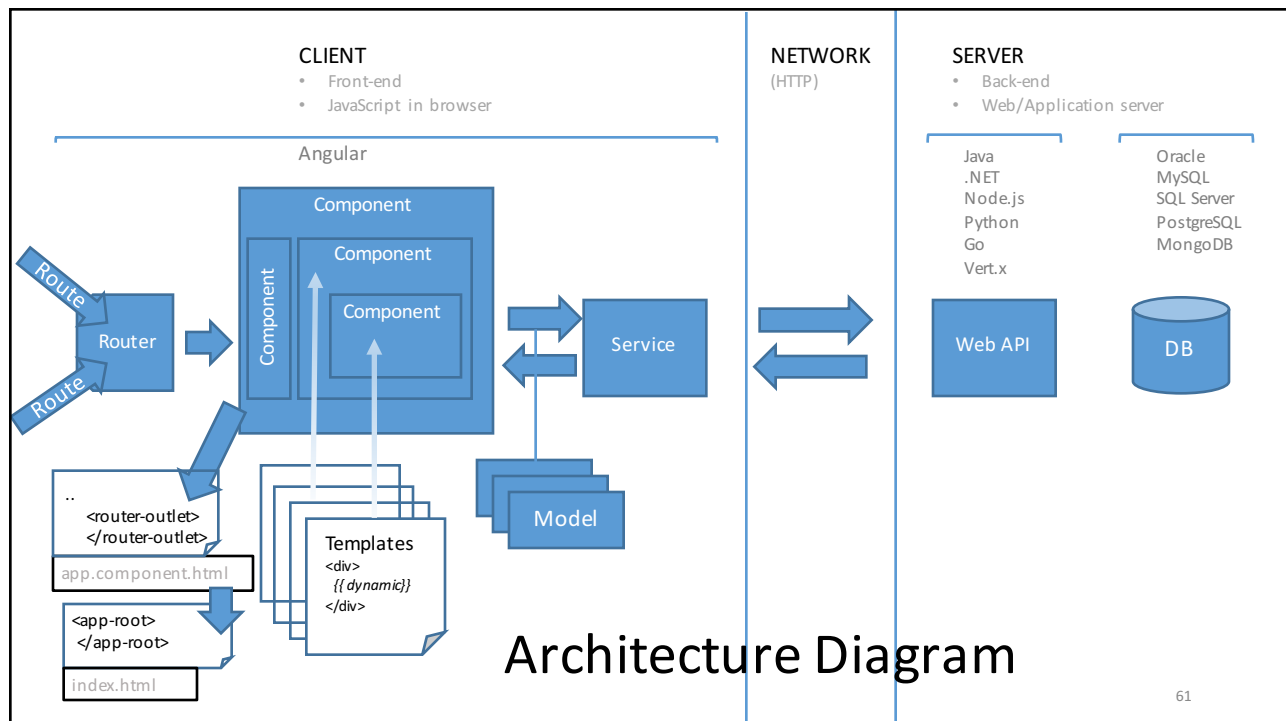
57

## Why a single-page application?

- The user experience of a desktop or native application
- The deployment story of a web application

58





## Browser Support

Browser	Supported versions
Chrome	latest
Firefox	latest
Edge	2 most recent major versions
IE	11 10 9
IE Mobile	11
Safari	2 most recent major versions
iOS	2 most recent major versions
Android	Nougat (7.0) Marshmallow (6.0) Lollipop (5.0, 5.1) KitKat (4.4)

## Angular Versions

### AngularJS

- 1.x
- angularjs.org

### Angular

- >=2.x
- angular.io

63

## Style Guide

- Follow the official Angular Style Guide
  - <https://angular.io/docs/ts/latest/guide/style-guide.html>
- Opinionated guide to Angular
  - Naming
  - Syntax
  - Conventions
  - Application structure

64

## Angular Big Picture Review

- With a partner, write down your answers to the following questions on sheet of paper in the next 5 minutes
  - What site should I visit to get the official documentation on AngularJS?
  - What site should I visit to get the official documentation on Angular?
  - In one sentence, why are so many people adopting Angular and other Single-page application frameworks (SPA) frameworks?
  - Draw a single-page application architecture diagram.
  - Draw an Angular architecture diagram.

65

## React & Angular Compared

### React

- Facebook
- Components
- Library
  - Just the View in MVC
  - Need to include other libraries
    - React Router (Routing)
    - Axios (AJAX)
- Usually ES6 (Babel compiler)
- Create React App
  - Uses Webpack
- Redux

### Angular

- Google
- Components
- Framework
  - Modular
    - Component Router
    - HttpClient
    - Forms
- Usually TypeScript (tsc compiler)
- Angular CLI
  - Uses Webpack
- Reactive Extensions for Angular (ngrx)

66



## React vs. Angular: Key Insights

- *Angular continues to put “JS” into HTML. React puts “HTML” into JS.*  
–Cory House
- Angular is a more comprehensive library while React is more of a targeted micro library.
- Because React is smaller it is:
  - Easier to understand
  - Easier to include in a project
- React is much more popular (but has existed longer)
- React is used more by design/digital/interactive agencies as well as in the Enterprise
- Angular is used more for Enterprise software particularly at larger organizations

67



## Project Setup

using the Angular CLI (Command Line Interface)

68

## Angular Style Guide: Naming Conventions

Symbol Name	File Name
<code>@Component({ ... })</code> <code>export class AppComponent { }</code>	app.component.ts
<code>@Component({ ... })</code> <code>export class HeroesComponent { }</code>	heroes.component.ts
<code>@Component({ ... })</code> <code>export class HeroListComponent { }</code>	hero-list.component.ts
<code>@Component({ ... })</code> <code>export class HeroDetailComponent { }</code>	hero-detail.component.ts
<code>@NgModule({ ... })</code> <code>export class AppModule</code>	app.module.ts
<code>@Pipe({ name: 'initCaps' })</code> <code>export class InitCapsPipe implements PipeTransform { }</code>	init-caps.pipe.ts
<code>@Injectable()</code> <code>export class UserProfileService { }</code>	user-profile.service.ts

69

## Angular CLI: Features

- Create a new Angular application
- Run a development server with LiveReload support to preview your application during development
- Scaffolds Angular application code
- Run your application's unit tests
- Run your application's end-to-end (E2E) tests
- Build your application for deployment to production

70

## Creating a New Project

- Install the Angular CLI

```
npm install -g @angular/cli
```

- Create a new project

```
ng new my-project --routing  
cd my-project
```

71

## Running Your Project

```
ng serve --open //runs dev server at http://localhost:4200/ and will automatically reload  
or  
ng serve -o
```

72

## Angular CLI: Generating Code

- Generate a module

```
ng generate module projects --routing
```

- Generate a component

```
ng g component projects/project-list
```

*g is short-hand for generate*

- CLI generates files relative to the **app** folder
- Generated code follows the official **Angular Style Guide**

73

## Customizing the Angular CLI

- Use **angular.json** to customize
- Common CLI customizations including
  - Asset Configuration
  - CSS Preprocessors
  - Global Libraries
  - Global Scripts
  - Angular Material
  - Bootstrap
  - Font Awesome
- Are documented here
  - <https://github.com/angular/angular-cli/wiki/stories>

74

# Bootstrapping an Angular Application

1. Create a root component
  - a. By convention named AppComponent
  - b. By convention the file is named app.component.ts
  - c. Add selector (custom tag) to index.html
2. Create a root module
  - a. By convention named AppModule
  - b. By convention the file is named app.module.ts
  - c. Declare and bootstrap the root component
  - d. Import the BrowserModule from Angular
3. Bootstrap the root module
  - a) By convention the file is named main.ts
  - b) platformBrowserDynamic
    - i. platformBrowser means it will run in a web browser (not a native mobile app etc...)
    - ii. Dynamic means the template will be compiled in the browser (not on the server)

75

## Demos

Instructor Demonstration

Lab 1: Creating a New Project

Lab 2: Running Your Project

Lab 3: Style: Using a CSS Framework

76

## Labs

Lab 1: Creating a New Project

Lab 2: Running Your Project

Lab 3: Style: Using a CSS Framework

Attendees Hands-On

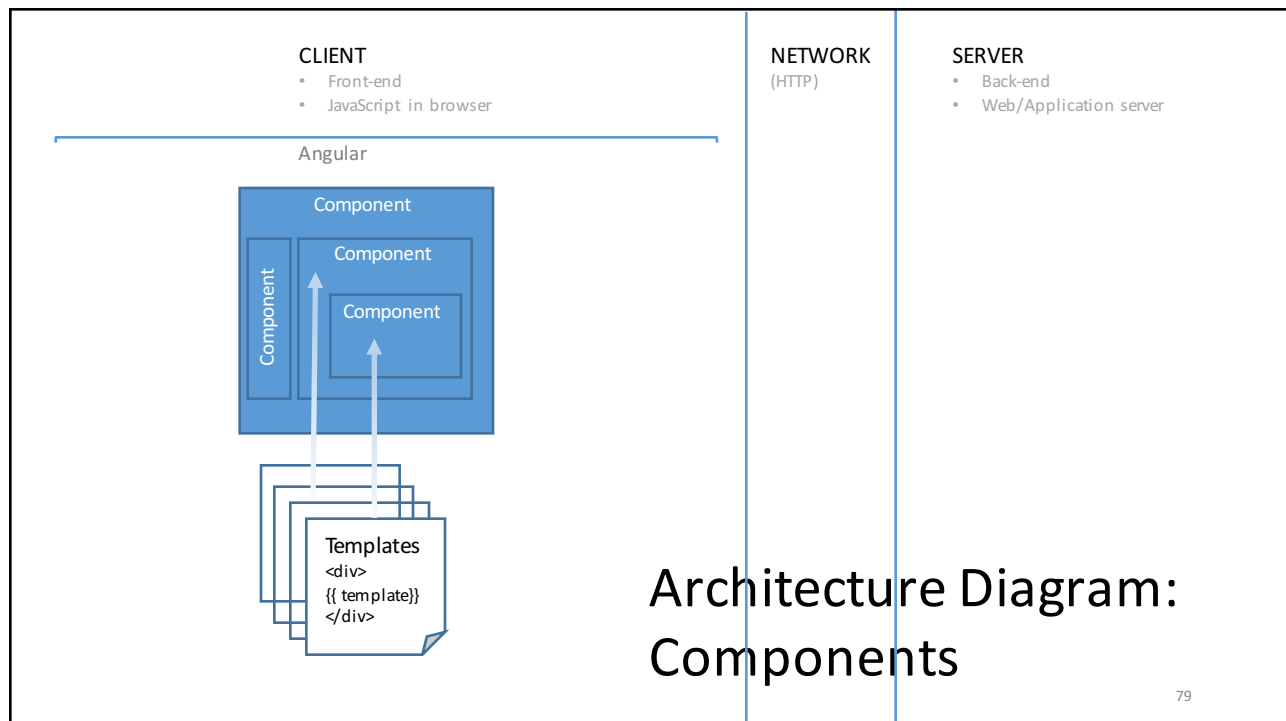
77



## Components

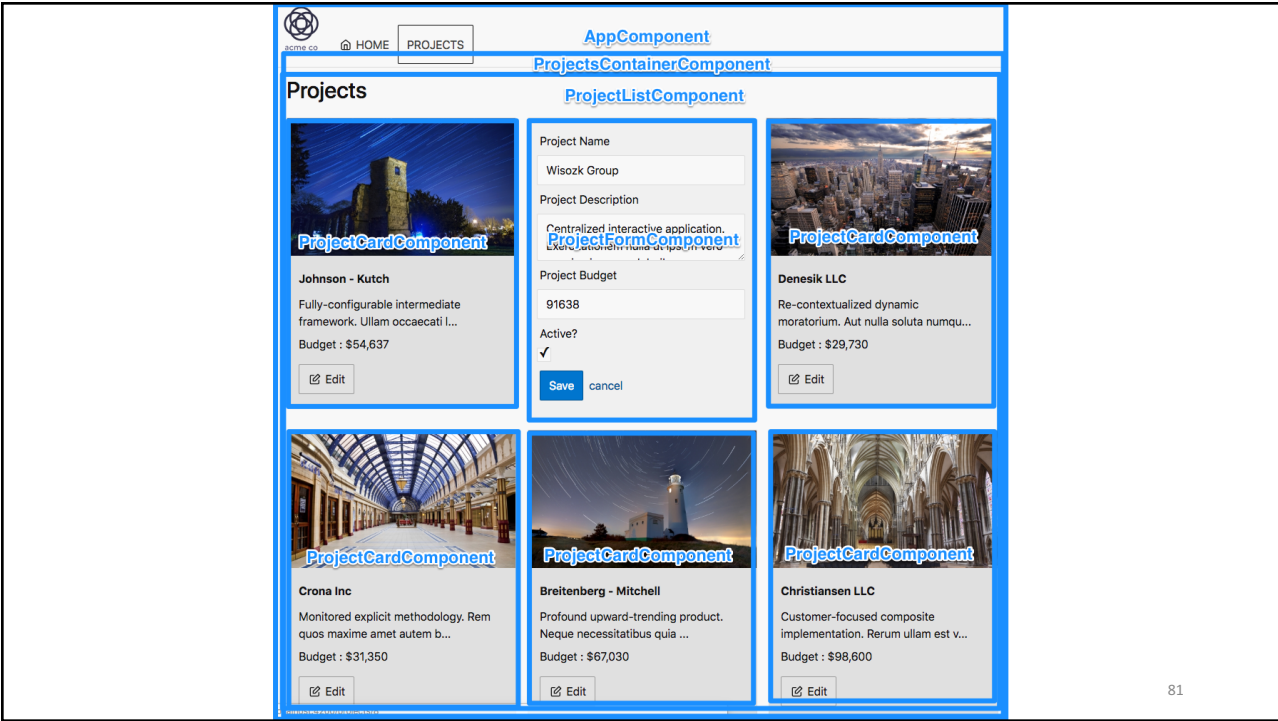
Angular

78

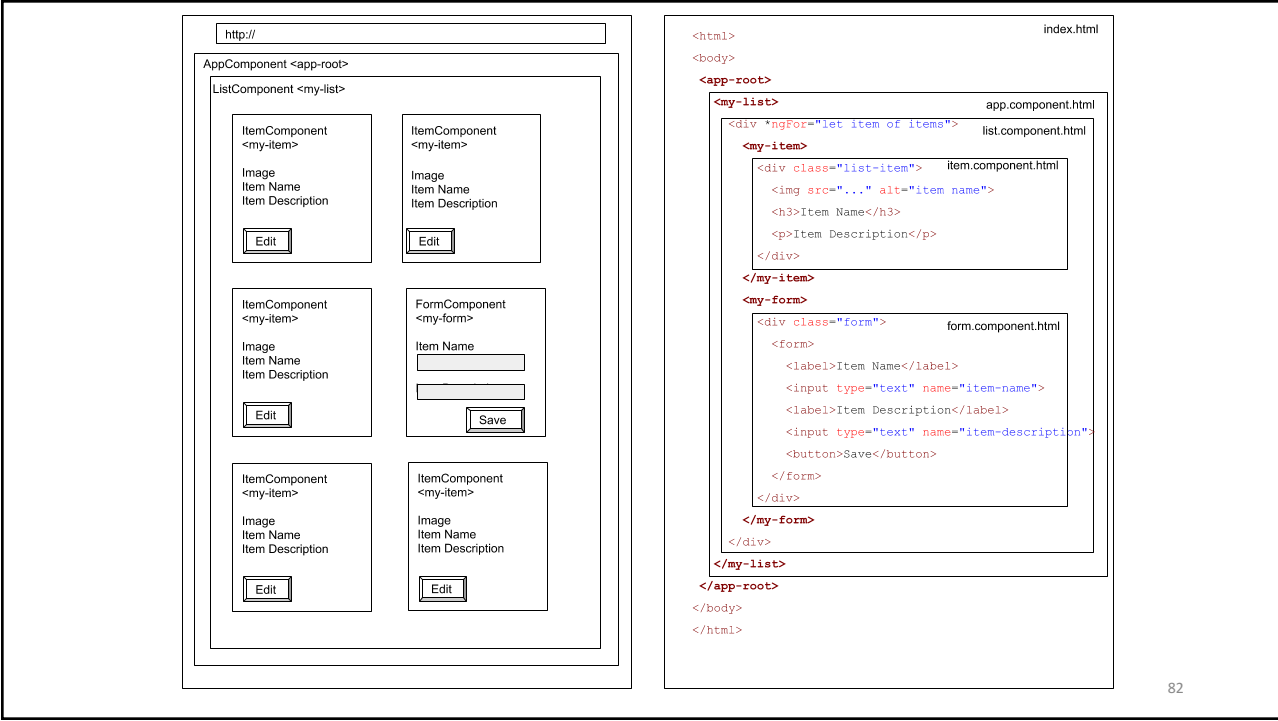


## What is a Component?

- A **Component** controls a patch of screen real estate (view)
- Examples of views controlled by components
  - the shell at the application root with navigation links
  - the list of projects
  - the project form
- The view is defined using HTML in a template
- Component's application logic - what it does to support the view
  - defined in a class
  - the class interacts with the view through an API of properties and methods



81



82



# Demo: Component Basics

Instructor Only Demonstration

`code\demos\component-first`

`code\demos\component-nesting`

83



## Modules

Angular

*NgModule*

85

## ES Modules vs Angular Modules

### JavaScript (ES) Modules

- ES6 modules represent a single file
- JavaScript modules are needed to:
  - to structure our applications (we cannot use a single file)
  - to avoid leaking code to the global namespace and thus to avoid naming collisions
  - to encapsulate code; to hide implementation details and control what gets exposed to the “outside”
  - to manage dependencies
  - to reuse code

### Angular Modules

- Angular Modules are an Angular specific construct used to
- Logically group different Angular artifacts such as components, services, pipes, and directive
- Provide metadata to the Angular compiler which in turn can better “reason about our application” structure and thus introduce optimizations
- Lazy load code

86

## Angular Module (NgModule)

- Organizes Angular code
- Logically groups different Angular framework artifacts such as components, services, pipes, and directive
- Similar to packages in Java
- Similar to namespaces in .NET
- Except Angular Modules are not organizing language constructs, but instead framework constructs

87

# Declarations

```
@NgModule({  
  declarations: [  
    ProjectsContainerComponent,  
    ProjectListComponent,  
    ProjectCardComponent,  
    ProjectFormComponent,  
    ValidationErrorsComponent,  
    TruncateStringPipe  
  ]  
})  
export class ProjectsModule {}
```



If **used** in the **template** of any component listed in this module then they must be listed in **declarations**.

Angular has it's own HTML compiler. It turns Angular HTML templates into JavaScript code that generates dynamic HTML.

The compiler looks for Angular components, directives, and pipes in a template and associates them with your code.

88

## Demo: Module Declarations

Instructor Only Demonstration

code\demos\module-declarations

89

## Feature

- Chunk of functionality that delivers business value
- Realized by some number of user stories
- Often the same as a:
  - Table in the database
  - Entity in your domain model

90

## Feature Modules

- Feature modules are NgModules for the purpose of organizing code
- You can organize code relevant for a specific feature
- Helps with collaboration between developers and teams, separating directives, and managing the size of the root module
- A feature module is an organizational best practice, as opposed to a concept of the core Angular API
- A feature module delivers a cohesive set of functionality focused on a specific application need such as a user workflow, routing, or forms
- Collaborates with the root module and with other modules through the services it provides and the components, directives, and pipes that it share

91

## Feature Module Example

```
@NgModule({
  imports: [
    ReactiveFormsModule,
    CommonModule,
    SharedModule,
    ProjectsRoutingModule
  ],
  declarations: [
    ProjectsContainerComponent,
    ProjectListComponent,
    ProjectCardComponent,
    ProjectFormComponent,
    ValidationErrorsComponent
  ],
  providers: [ProjectService]
})
export class ProjectsModule {}
```

Imports are always modules and are always named with a Module suffix.

These modules can be parts of the Angular framework, your app's reusable code, or your app's other features or routing modules.

If used in the template of any component listed in this module then they must be listed in declarations.

Services can be registered in providers.

92

## Application Structure

- LIFT
  - Locate code quickly
  - Identify the code at a glance
  - Keep the **Flattest** structure you can
  - Try to be DRY
    - Avoid being so DRY that you sacrifice readability
- *Folders-by-feature*
  - Do create folders named for the feature area they represent.

```
<project root>
├── src
│   ├── app
│   │   ├── core
│   │   │   ├── core.module.ts
│   │   │   ├── exception.service.ts/spec.ts
│   │   │   └── user-profile.service.ts/spec.ts
│   │   ├── heroes
│   │   │   ├── hero
│   │   │   │   ├── hero.component.ts/html/css/spec.ts
│   │   │   │   ├── hero-list
│   │   │   │   │   ├── hero-list.component.ts/html/css/spec.ts
│   │   │   │   └── shared
│   │   │   │       ├── hero-button.component.ts/html/css/spec.ts
│   │   │   │       ├── hero.model.ts
│   │   │   │       ├── hero.service.ts/spec.ts
│   │   │   │       ├── heroes.component.ts/html/css/spec.ts
│   │   │   │       ├── heroes.module.ts
│   │   │   │       └── heroes-routing.module.ts
│   │   │   └── shared
│   │   │       ├── shared.module.ts
│   │   │       ├── init-caps.pipe.ts/spec.ts
│   │   │       ├── text-filter.component.ts/spec.ts
│   │   │       └── text-filter.service.ts/spec.ts
│   │   ├── villains
│   │   │   ├── villain
│   │   │   │   ├── ...
│   │   │   ├── villain-list
│   │   │   │   ├── ...
│   │   │   ├── shared
│   │   │   │   ├── ...
│   │   │   ├── villains.component.ts/html/css/spec.ts
│   │   │   ├── villains.module.ts
│   │   │   └── villains-routing.module.ts
│   │   ├── app.component.ts/html/css/spec.ts
│   │   ├── app.module.ts
│   │   └── app-routing.module.ts
│   ├── main.ts
│   ├── index.html
│   └── ...
└── node_modules/...
```

93

# Demo: Module Imports & Exports

Instructor Only Demonstration

`code\demos\module-imports-exports`

94

## Lab

Lab 4: Your First Component

Attendees Hands-On

95

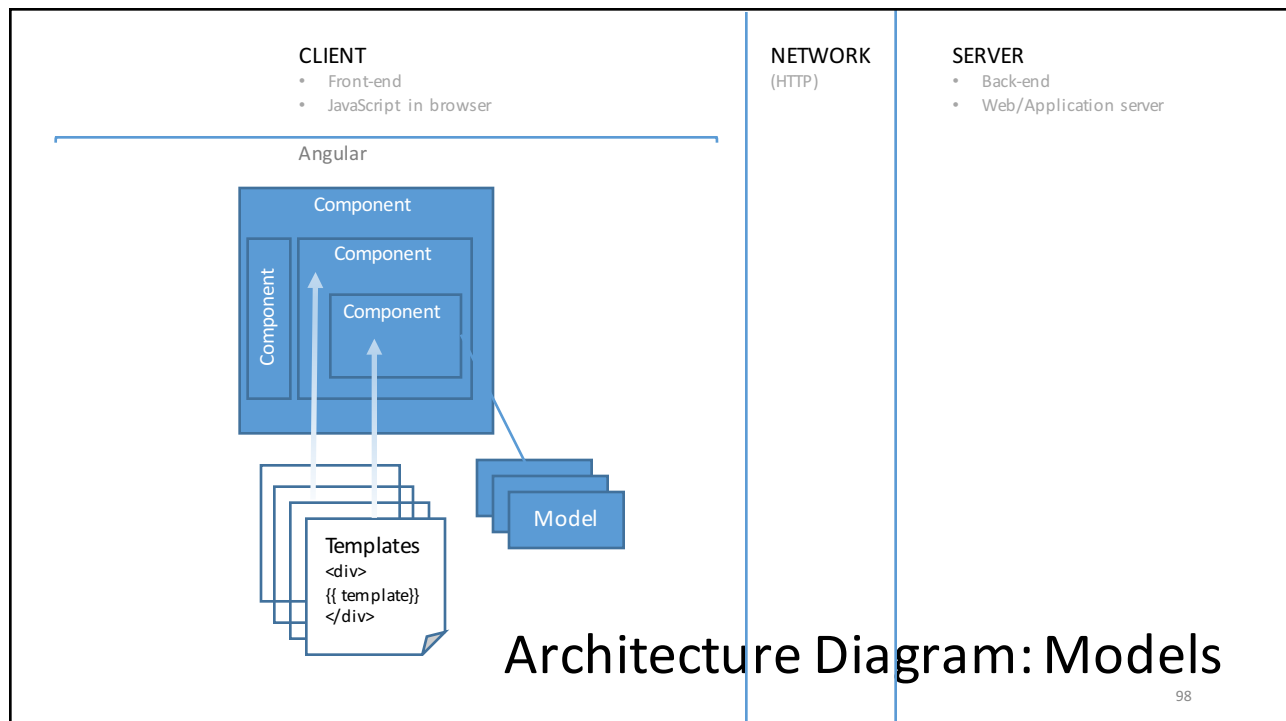
## Root Module vs Feature Modules

- Every Angular app has at least one module, the *root module*, conventionally named AppModule.
- While the *root module* may be the only module in a small application, most apps have many more *feature modules*.
  - A feature module is a cohesive block of code dedicated to an application domain, a workflow, or a closely related set of capabilities.
- An Angular module, whether a *root* or *feature*, is a class with an @NgModule decorator.

96

## Models

97



## Models

are just classes

```
export class Project {  
  constructor(  
    public id: number,  
    public name: string,  
    public description: string,  
    public imageUrl: string,  
    public contractTypeId: number,  
    public contractSignedOn: Date,  
    public budget: number,  
    public isActive: boolean,  
  ) {}  
}
```



# Models

## constructor overloads

```
export class Project {
  id: number;
  name: string;
  description: string;
  isActive: boolean;
  contractSignedOn: Date;
  budget: number;

  constructor(obj?: any) {
    this.id = (obj && obj.id) || null;
    this.name = (obj && obj.name) || null;
    this.description = (obj && obj.description) || null;
    this.contractTypeId = (obj && obj.contractTypeId) || null;
    this.isActive = (obj && obj.isActive) || false;
    this.contractSignedOn = (obj && obj.contractSignedOn) || new Date();
    this.budget = (obj && obj.budget) || 0;
  }
}

let project = new Project({ name: 'Acme Website Redesign', budget: 30000 });
```

100

# JSON Pipe

```
@Component({
  selector: 'json-pipe',
  template: `<div>

    <p>Without JSON pipe:</p>
    <pre>{{object}}</pre>

    <p>With JSON pipe (no pre tag):</p>
    <p>{{object | json}}</p>

    <p>With JSON pipe (and pre tag):</p>
    <pre>{{object | json}}</pre>

  </div>`,
})
export class JsonPipeComponent {
  ...
}
```

101

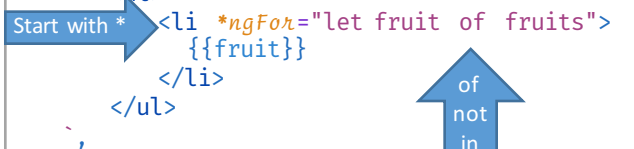
## Angular CLI: Generate Class

```
ng g class projects/shared/project.model
```

102

## ngFor

```
@Component({
  selector: 'app-ng-for-demo',
  template: `
    <ul>
      <li *ngFor="let fruit of fruits">
        {{fruit}}
      </li>
    </ul>
  `
})
export class NgForComponent {
  fruits = ['Apple', 'Orange', 'Plum'];
}
```



103

# Demo: ngFor

Instructor Only Demonstration  
`code\demos\ngFor`

104



# Data Binding

Angular

105

# Data Binding

Four forms (types)

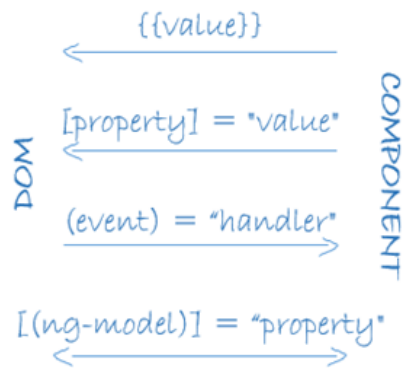


Image from angular.io

- **Interpolation**
- Property binding
- Event binding
- Two-way data binding

106

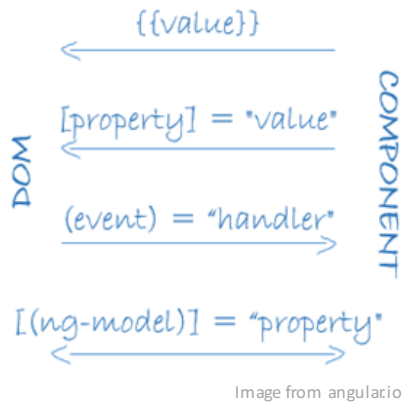
## Interpolation

```
@Component({
  selector: 'app-root',
  template: `
    <h2>{{image.name}}</h2>
    <p>{{image.path}} </p>
  `,
  styles: []
})
export class AppComponent {
  image = {
    path: '../assets/angular_solidBlack.png',
    name: 'Angular Logo'
  };
}
```

107

# Data Binding

Four forms (types)



- Interpolation
- **Property binding**
- Event binding
- Two-way data binding

108

## Property Binding

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

@Component({
  selector: 'app-root',
  template: `
    <img [src]="image.path" [title]="image.name"
        [alt]="image.name">
  `,
})
export class AppComponent {
  image = {
    path: '../assets/angularlogo.png',
    name: 'Angular Logo',
  };
}
```

109

# Demo: Data Binding

## Interpolation & Property Binding

Instructor Only Demonstration

code\demos\interpolation

code\demos\property-binding

110

## Input Property

- @Input
  - Decorator that marks a class field as an input property
- Property Binding to a Component Property

```
@Component({
  selector: 'app-root',
  template: `
    <app-fruit-list [fruits]="data"></app-fruit-list>
  `,
  styles: []
})
export class AppComponent {
  data: string[] = ['Apple', 'Orange', 'Plum'];
}
```

111

# Demo: Input Property

Instructor Only Demonstration

`code\demos\input-property`

112

## Labs

Lab 5: Creating Data Structures (Models)

Lab 6: Passing Data into a Component

Lab 7: Looping Over Data

Attendees Hands-On

113

# Data Binding

Four forms (types)

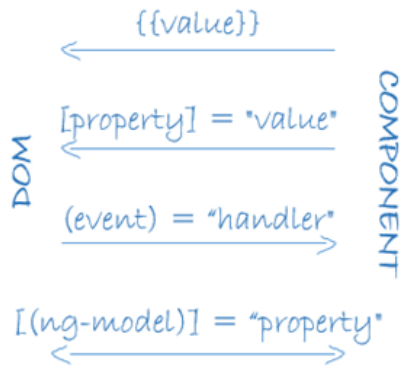


Image from angular.io

- Interpolation
- Property binding
- **Event binding**
- Two-way data binding

114

# Event Binding

```
@Component({
  selector: 'app-event-binding-demo',
  template: `
    <a href="/event-binding" (click)="onClick($event)">Click Me!</a>
    <p [innerText]="message"></p>
  `,
})
export class EventBindingComponent {
  message = '';

  onClick(event) {
    event.preventDefault();
    this.message = 'clicked';
  }
}
```

**Annotations:**

- \$(event)** is template variable available in Angular
- Can use any standard browser event.  
[https://developer.mozilla.org/en-US/docs/Web/Events#Standard\\_events](https://developer.mozilla.org/en-US/docs/Web/Events#Standard_events)
- Prevents the default browser behavior for that element.  
<https://developer.mozilla.org/en-US/docs/Web/API/Event/preventDefault>

115



# Demo: Event Binding

Instructor Only Demonstration

`code\demos\event-binding`

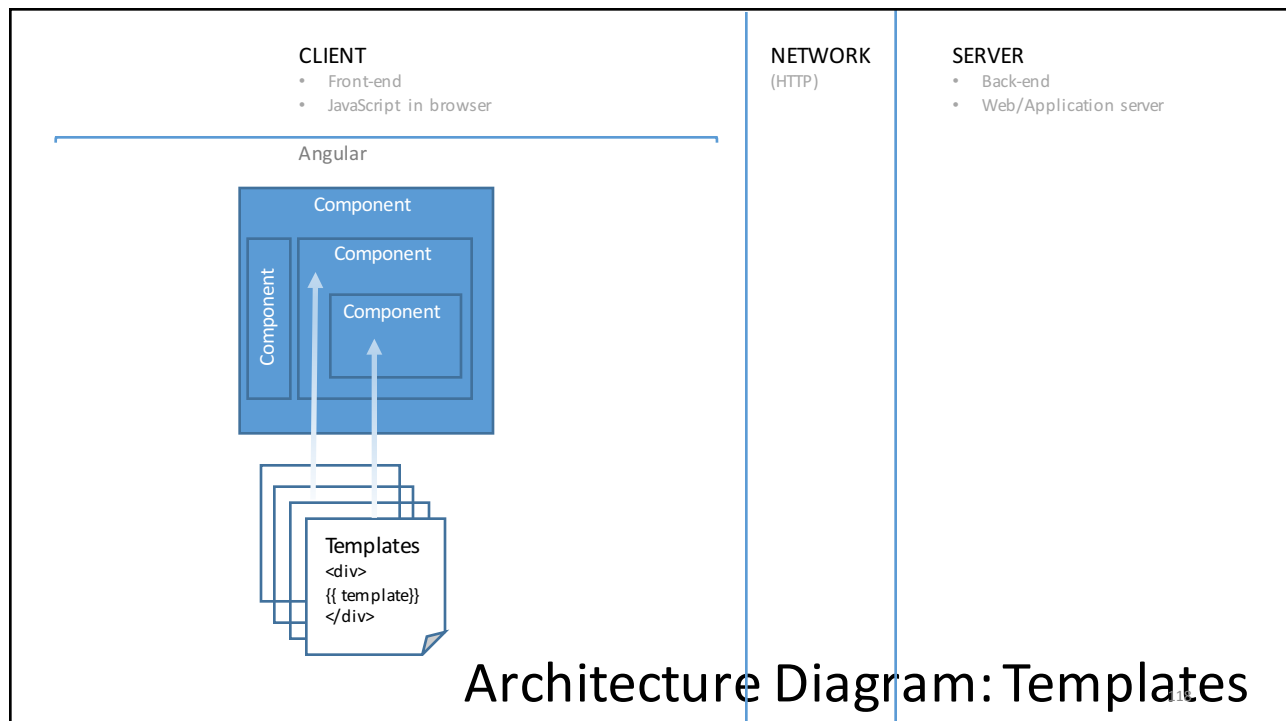
116



## Pipes

Angular

117



## What are Pipes?

- A pipe takes in data as input and transforms it to a desired output
- Commonly used to format data
- Get their name from the pipe operator “|”
- Can format various types: String, Number, Data, Array

## Using Pipes

- Pipes can be used
  - In HTML templates
  - In JavaScript (often in components)
- Pipes can be
  - Built-in to Angular
  - Custom pipes created by application developers
    - Custom pipes can be used to implement any type of formatting

120

## Changes from AngularJS

- Filters in AngularJS are Pipes in Angular
- The “filter” filter is not built-in Angular
  - Was used to filter out arrays based on a search term
- The “orderBy” filter is not built-in to Angular
  - Was used to sort array elements for display
- “filter” and “orderBy”
  - Prone to breaking if code was minified
  - Could lead to performance issues
  - Replacements can be implemented as a custom filter

121

## Using a Built-in Pipe

```
<h4>{{project.name | uppercase}}</h4>
```

122

## Common Built-In Pipes

Class	Name
PercentPipe	percent
UpperCasePipe	uppercase
LowerCasePipe	lowercase
TitleCasePipe	titlecase
DatePipe	date
DecimalPipe	decimal
CurrencyPipe	currency

123

## Pipe Syntax

- Syntax

```
{{data-expression | pipe-name [:pipe-parameter: pipe-parameter-2...]}}
```

- Example

```
releaseDate: Date(1977, 5, 25);
```

```
{{releaseDate | date: "M-dd-yyyy"}}
```

124

## Chaining Pipes

- Chained pipes are executed from left to right

```
{{ releaseDate | date | uppercase }}
```
- The *releaseDate* property is converted to a date string
- The date string is then converted to uppercase

125

## Decimal Pipe

- Takes a number as input
- And a *digitInfo* string defining the number format  
{minIntegerDigits}.{minFractionDigits}-{maxFractionDigits}
- Example
  - Usage: {{47.243 | number: "3.2-4"}}
  - Output: 047.243

126

## CurrencyPipe

- Use [currency](#) to format a number as currency.
- Takes a number and up to three parameters
  - *currencyCode* is the [ISO 4217](#) currency code, such as USD for the US dollar and EUR for the euro.
  - *display* indicates whether to use the currency symbol or code.
    - code: use code (e.g. USD).
    - symbol(default): use symbol (e.g. \$).
    - symbol-narrow: some countries have two symbols for their currency, one regular and one narrow (e.g. the canadian dollar CAD has the symbol CA\$ and the symbol-narrow \$).
    - boolean (deprecated from v5): true for symbol and false for code. If there is no narrow symbol for the chosen currency, the regular symbol will be used.
  - *digitInfo* defining the number format:
    - "{minIntegerDigits}.{minFractionDigits}-{maxFractionDigits}"
  - Example:
    - Usage: {{47.341 | currency: 'USD': '2.1-2'}}
    - Output: \$47.34

127

# Demo: Input Property

Instructor Only Demonstration

code\demos\pipes

128

## Using Pipes in JavaScript

- Syntax
  - pipeClass.transform(value, param1, param2, etc);
- In the module:
  - `import { DatePipe } from '@angular/common';`
  - `@NgModule({`
    - `providers: [..., DatePipe]``})`
- In the component:
  - `constructor(private datePipe: DatePipe){}`

---

```
releaseDate: Date = new Date(1975, 5, 25);  
releaseDateFmt: string = this.datePipe.transform(this.releaseDate, "M-dd-yyyy")
```

129

# How to Structure an Application

## 1. Components

- If a component gets too complex split it into smaller components

## 2. After you create more components, more questions arise

- What types of components are there?
- How should components interact?
- Should I inject services into any component?
- How do I make my components reusable across views?

130

# Component Architecture

## Smart/Container Components

- Are concerned with *how things work*
- *Sets data* into child component input properties
- *Receives events* by subscribing to children
- *Loads and modifies data* via calls to an API
- Also know as *container* components or *controller* components

## Presentation Components

- Are concerned with *how things look*
- *Receive data* via input properties from parent
- *Send events* with information to their parent
- Don't specify how the data is loaded or changed
- Also know as *pure* components or *dumb* components

131



## When to Create Another Component

- Is it possible for your code chunk to be reused?
  - If yes, construction of a new component seems like a great idea.
  - Even if the reuse is within a single component.
- Is your code quite complex?
  - If yes maybe its good idea to split in separate components in order to make your code more readable and maintainable.

132

## Custom Events in a Component

- @Output
  - Decorator that marks a class property as sending a custom output event
- EventEmitter
  - Class used in directives and components to emit custom events synchronously or asynchronously, and register handlers for those events by subscribing to an instance

134

# Demo: Output Events

Instructor Only Demonstration

`code\demos\output-events`

135

## Component Styles

- Angular applications are styled with regular CSS
- Angular has the ability to bundle *component styles* with our components
  - enables a more modular design than regular stylesheets

136

# Component Styles

demos/components/styling-external.ts

External Styles

```
@Component({
  selector: 'styling-external',
  template: '<h1>Styling Components: External</h1>',
  styleUrls: ['./styling-external.css'],
})
export class StylingExternalComponent {}
```

Reference external style sheet.

demos/components/styling-external.css

```
h1 {
  color: rgb(255, 165, 0);
}
```

Use CSS to style your template.

By default these styles will be local to this component and not affect the rest of the page (ViewEncapsulation.Emulated).

137

# Component Styles

View Encapsulation

- Component CSS styles are *encapsulated* into the component's own view and do not affect the rest of the application (when using the default)
- We can control how this encapsulation happens on a *per component* basis by setting the *view encapsulation mode* in the component metadata
- There are three modes to choose from
  - Native
  - Emulated
  - None

138

# Component Styles

## Emulated View Encapsulation

<pre>▼&lt;app-project-card _ngcontent-c2 _ngghost-c3 ng-reflect-project="[object Object]"&gt;   ▼&lt;div _ngcontent-c3 class="card"&gt;     &lt;img _ngcontent-c3 src="assets/placeimg_500_300_arch4.jpg" alt="Dusty Epsilon"&gt;     ▼&lt;section _ngcontent-c3 class="section dark"&gt;       ▼&lt;h5 _ngcontent-c3 class="strong"&gt; == \$0         &lt;strong _ngcontent-c3&gt;Dusty Epsilon&lt;/strong&gt;       &lt;/h5&gt;       ▶&lt;p _ngcontent-c3&gt;...&lt;/p&gt;       &lt;p _ngcontent-c3&gt;Budget : \$42,400&lt;/p&gt;       ▶&lt;button _ngcontent-c3 class="bordered"&gt;...&lt;/button&gt;     &lt;/section&gt;   &lt;/div&gt; &lt;/app-project-card&gt;</pre>	<table><tr><th>Filter</th></tr><tr><td>element.style { }</td></tr><tr><td>h5[_ngcontent-c3] {   color: ■ #2552b5; }</td></tr><tr><td>h5 {   font-size: 1rem; }</td></tr><tr><td>h1, h2, h3, h4, h5, h6 {  <input checked="" type="checkbox"/> line-height: 1.2;  <input checked="" type="checkbox"/> margin: calc(1.5 * var</td></tr></table>	Filter	element.style { }	h5[_ngcontent-c3] { color: ■ #2552b5; }	h5 { font-size: 1rem; }	h1, h2, h3, h4, h5, h6 { <input checked="" type="checkbox"/> line-height: 1.2; <input checked="" type="checkbox"/> margin: calc(1.5 * var
Filter						
element.style { }						
h5[_ngcontent-c3] { color: ■ #2552b5; }						
h5 { font-size: 1rem; }						
h1, h2, h3, h4, h5, h6 { <input checked="" type="checkbox"/> line-height: 1.2; <input checked="" type="checkbox"/> margin: calc(1.5 * var						

139

# Demo: Input Property

Instructor Only Demonstration

code\demos\component-styles

140

# Labs

Lab 8: Formatting Data for Display

Lab 9: More Reusable Components

Lab 10: Responding to an Event

Lab 11: Create a Form to Edit Your Data

Lab 12: Communicating from Child to Parent Component

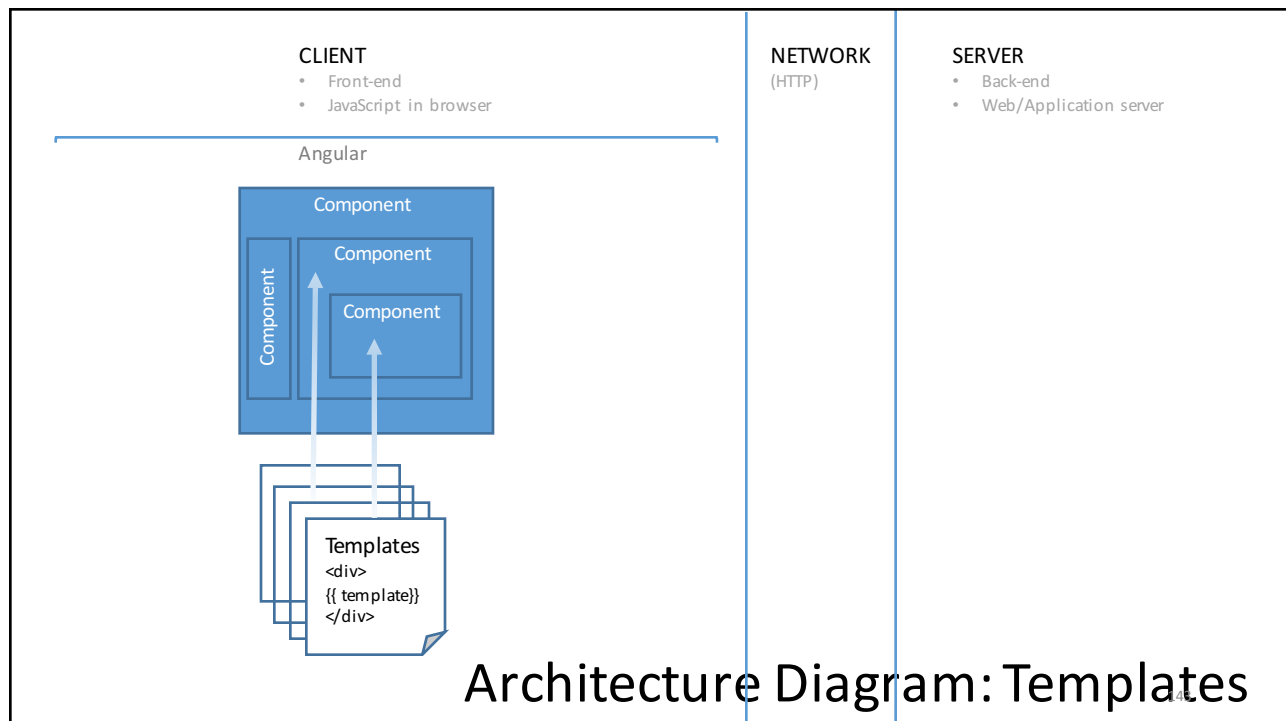
Attendees Hands-On Together

141

# Directives

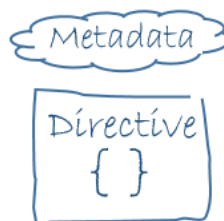
Built-In to Angular

142



## What is a Directive?

- Angular transforms the DOM according to the instructions given by **directives**
- A directive is a class with directive metadata.
  - In TypeScript, we apply the `@Directive` decorator to attach metadata to the class



## Kinds of Directives

- Component
  - A **component** is a *directive-with-a-template*
- Structural
  - **Structural** directives alter layout by adding, removing, and replacing elements in DOM
- Attribute
  - **Attribute** directives alter the appearance or behavior of an existing element

145

## Structural Directives

- **Structural** directives alter layout by adding, removing, and replacing elements in DOM
  - ngIf
  - ngFor
  - ngSwitch

146

## ngIf

- Takes a boolean and makes an entire chunk of DOM appear or disappear

```
<p *ngIf="condition">
  condition is true and ngIf is true.
</p>

<p *ngIf="!condition">
  condition is false and ngIf is false.
</p>
```

147

## Demo: ngIf

Instructor Only Demonstration

code\demos\ngIf

148



## Labs

Lab 13: Hiding and Showing Components

Lab 14: Preventing a Page Refresh

Lab 15: More Component Communication (*optional, do only if you are finishing labs early and want extra practice, similar to Lab 12*)

Attendees Hands-On Together

149

## Remove or Hide

[ngIf](#) or [hidden](#)

```
<p *ngIf="condition">
  Element to be added or removed
</p>
<button class="btn btn-warning" (click)="toggleIf()">add | remove</button>
```

```
<p [hidden]="!isVisible" >
  Element to show or hide using CSS
</p>
<button class="btn btn-warning" (click)="toggleVisibility()">
  show | hide
</button>
```

150

## Hide or Remove

`hidden` or `ngIf`

### Hide: using `[hidden]`

The component's behavior continues  
Remains attached to its DOM element  
Continues to listen to events  
Angular keeps checking for changes that could affect data bindings  
The component — and all of its descendent components — tie up resources  
Performance and memory burden can be substantial  
Showing again is quick

### Remove: using `ngIf`

The component's behavior stops  
DOM element is removed  
Stops listening to events  
Angular stops checking for changes in data bindings  
The component — and all of its descendent components — are cleaned up  
Performance and memory burden are significantly reduced  
Showing again can be slow

151

## Remove or Hide Heuristic

`hidden` or `ngIf`

- In general, it is best to **use `ngIf`** to remove unwanted components rather than hide them
- The *more complicated* the element is, the *more likely* this will be the *right choice*

152

## Understand How Structural Directives Work

```
<!-- Examples (A) and (B) are the same -->
```

```
<!-- (A) *ngIf paragraph -->
```

```
<p *ngIf="condition">  
  Our heroes are true!  
</p>
```

```
<!-- (B) [ngIf] with template -->
```

```
<ng-template [ngIf]="condition">  
  <p>  
    Our heroes are true!  
  </p>  
</ng-template>
```

153

## NgSwitch

```
@Component({  
  selector: 'my-app',  
  template:  
    <div class="container">  
      <button (click)="value=1">select - 1</button>  
      <button (click)="value=2">select - 2</button>  
      <button (click)="value=3">select - 3</button>  
      <h5>You selected : {{value}}</h5>  
  
      <hr>  
      <div [ngSwitch]="value">  
  
        <div *ngSwitchCase="1">1. Template - <b>{{value}}</b> </div>  
        <div *ngSwitchCase="2">2. Template - <b>{{value}}</b> </div>  
        <div *ngSwitchCase="3">3. Template - <b>{{value}}</b> </div>  
        <div *ngSwitchDefault>Default Template</div>  
  
      </div>  
    </div>  
  </div>  
})  
export class AppComponent {  
  value: number;  
}
```

154

# Demo: ngSwitch

Instructor Only Demonstration

`code\demos\ngSwitch`

155



## Forms

Reactive Forms

156

## Benefits of Forms

- Forms are the mainstay of business applications
  - Login
  - Place an Order
  - Book a Flight
  - Schedule a Meeting

157

## Form Strategies

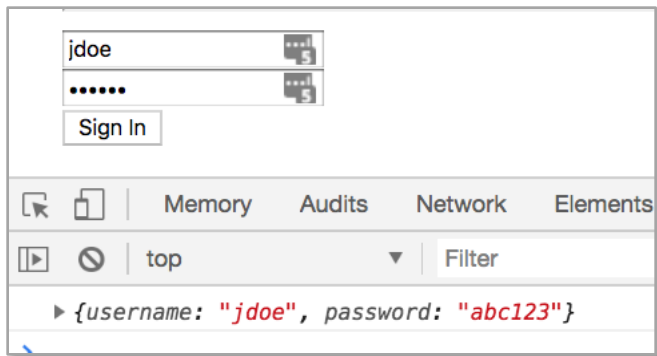
### **Template-driven**

- AngularJS style
- FormsModule
- Use the core ng prefix
- Declarative in the template

### **Model-driven (new)**

- Reactive programming
- ReactiveFormsModule
- Use the form prefix
- Configured in component code
- Enables dynamic forms
- Facilitates unit testing

158



# Demo: Reactive Forms Binding

Instructor Only Demonstration

code\demos\reactive-forms-binding

160

## Labs

Lab 16: Forms | Binding

161



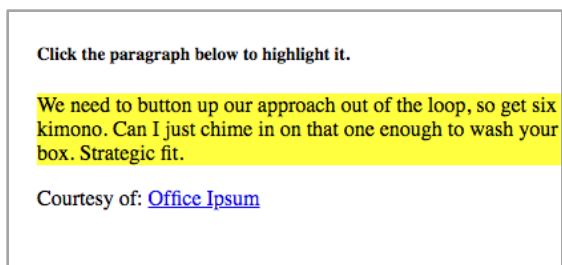
A screenshot of a web form with two input fields: 'username' and 'password'. The 'username' field is empty and has a red error message 'Username is required.' below it. The 'password' field is filled with a blue highlight. Below the password field is a 'Sign In' button.

## Demo: Reactive Forms Validation

Instructor Only Demonstration

code\demos\reactive-forms-validation

162



Click the paragraph below to highlight it.

We need to button up our approach out of the loop, so get six kimono. Can I just chime in on that one enough to wash your box. Strategic fit.

Courtesy of: [Office Ipsum](#)

Dynamically adds or removes CSS classes.

## Demo: ngClass

Instructor Only Demonstration

code\demos\ngClass

163

## Labs

Lab 16: Forms | Binding

Lab 17: Forms | Saving

Lab 18: Forms | Validation

Attendees Hands-On

Lab 19: Forms | Refactor (Instructor Walkthrough)

164

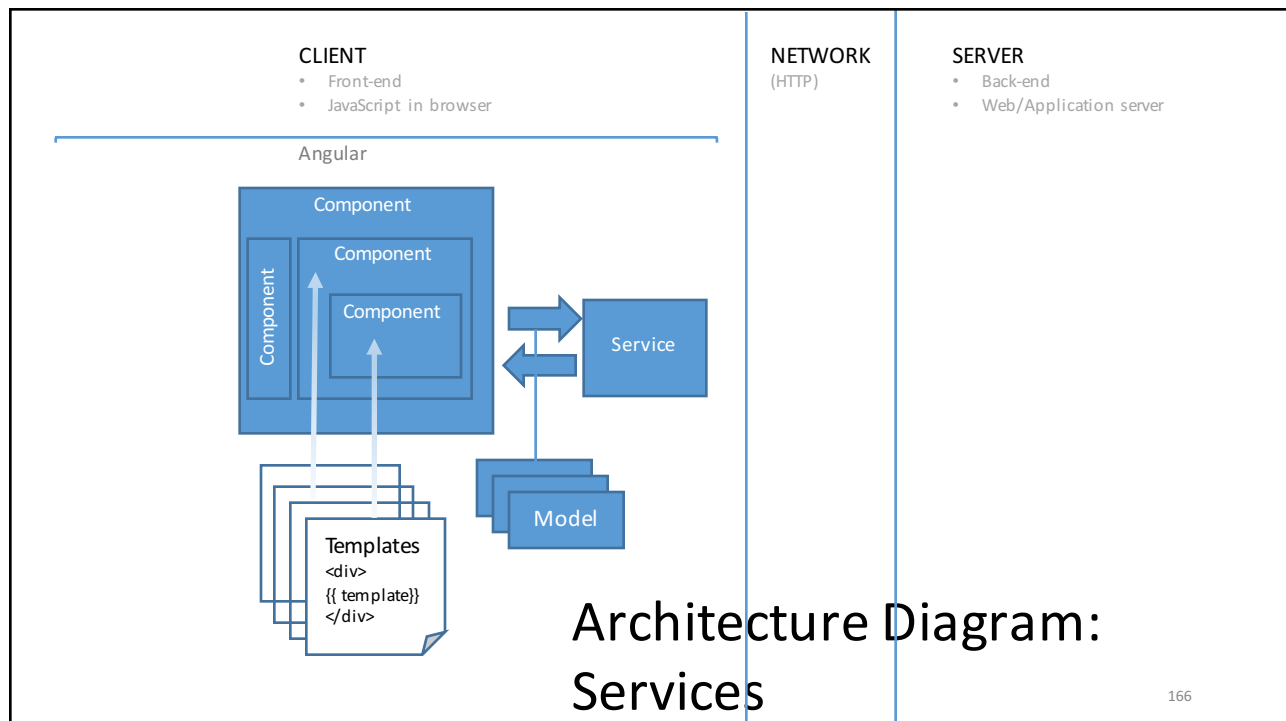


## Services

Angular

165





## What is a Service?

- *Service* is a broad category, almost anything can be a service
- A service is typically a class with a narrow, well-defined purpose
- It should do something specific and do it well

## Service Compared to Component

- By separating a component's view-related functionality from other kinds of processing, you can make your component classes lean and efficient.
- Ideally, a component's job is to enable the user experience and nothing more.
- A component can delegate certain tasks to services, such as fetching data from the server, validating user input, or logging directly to the console. By defining such processing tasks in an *injectable service class*, you make those tasks available to any component.

168

## Data Service

project.service.ts

```
import { Injectable } from '@angular/core';
import { Observable, of } from 'rxjs';
import { Project } from './project.model';
import { PROJECTS } from './mock-projects';
```

```
@Injectable({
  providedIn: 'root'
})
```

```
export class ProjectService {
```

```
  constructor() { }
```

```
  list(): Observable<Project[]> {
    return of(PROJECTS);
  }
```

```
}
```

Provides this service in the root module's injector.  
We cover dependency injection and injectors in the next section.

The RxJS `of` method creates an **Observable** which (like a Promise) allows us to easily work with Asynchronous operations like AJAX calls

169

## Using a Data Service in a Component

projects-container.component.ts

```
export class ProjectsContainerComponent implements OnInit {  
  projects: Project[] = PROJECTS;  
  
  constructor(private projectService: ProjectService) {}  
  
  ngOnInit() {  
    this.projectService.list()  
      .subscribe(data => {  
        this.projects = data;  
      });  
  }  
}
```

OnInit is an interface that has one method `ngOnInit` which is a component lifecycle event

Tell Angular you need an instance of the service and the framework will **inject** it into the constructor.

The `list` method returns an RxJS **Observable**. We **subscribe** to the **Observable** and when it has data it will call the arrow function we passed as an argument.

170

## Demo: Services

Instructor Only Demonstration

code\demos\services

171

# Labs

Lab 20: Services

Attendees Hands-On

172



# Dependency Injection

Angular

173

# Dependency Injection

## explained

- Imagine you are not allowed to use the new keyword and create instances of other objects (dependencies) you need when writing code
- Dependency Injection is a practice where objects are designed in a manner where they receive instances of the objects from other pieces of code, instead of constructing them internally
- This means that any object implementing the interface which is required by the object can be substituted in without changing the code, which simplifies testing, and improves decoupling

174

# Dependency Injection

## example

```
//no DI
class CustomersComponent{
    private customerService = new CustomerDataAccessService ();
}

//using DI
class CustomersComponent{
    private customerService : CustomerDataAccessService;

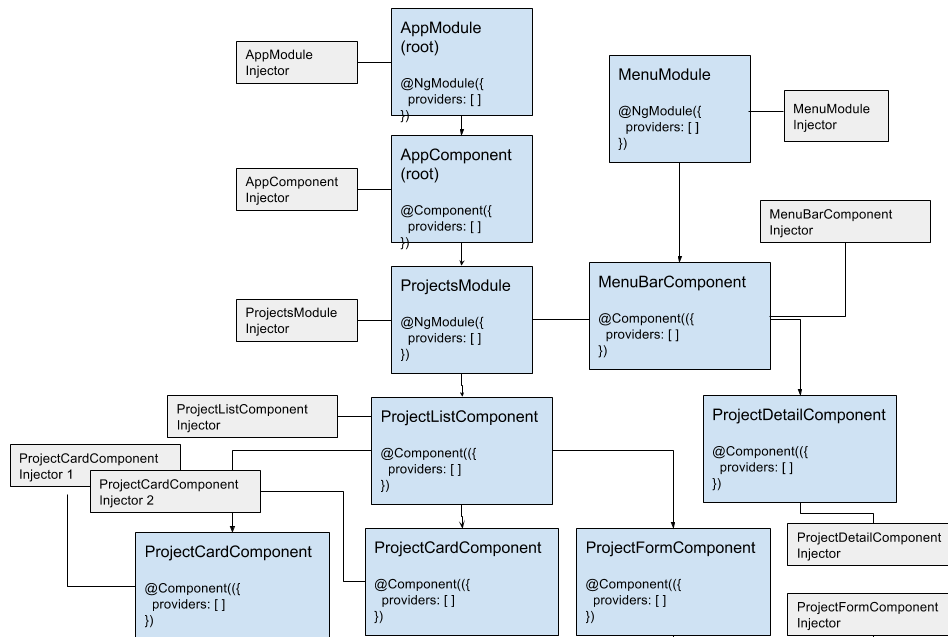
    constructor(customerService : CustomerDataAccessService){
        this.customerService = customerService;
    }
}
```

175

# Dependency Injection Frameworks

- Java
  - Spring, Guice
- .NET
  - Ninject, Structure Map, Unity, Spring.NET, .NET Core (built-in)
- Angular
  - Dependency Injection is built-in

176



Hierarchical Dependency Injectors

177

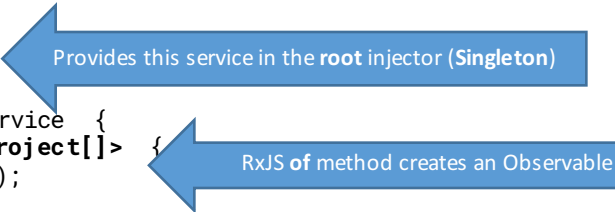
## Providing Services in the Root Module Injector

[project.service.ts](#)

```
import { Injectable } from '@angular/core';
import { Observable, of } from 'rxjs';

import { Project } from './project.model';
import { PROJECTS } from './mock-projects';

@Injectable({
  providedIn: 'root'
})
export class ProjectService {
  list(): Observable<Project[]> {
    return of(PROJECTS);
  }
}
```



178


## Providing Services in a Feature Module Injector

[project.service.ts](#)

```
import { Injectable } from '@angular/core';
import { Observable, of } from 'rxjs';

import { Project } from './project.model';
import { PROJECTS } from './mock-projects';

@Injectable({
  providedIn: ProjectsModule
})
export class ProjectService {
  list(): Observable<Project[]> {
    return of(PROJECTS);
  }
}
```



179

## Service Registration Best Practices

- Provide services in the Service itself (providedIn)
- Set **providedIn** to
  - **root** if you want the service to be a **Singleton**
  - **Feature Module** (*ProjectsModule*) if you want the service to be **lazy-loaded**
    - **Lazy Loading** requires additional steps which are covered in the Routing section of the course

180

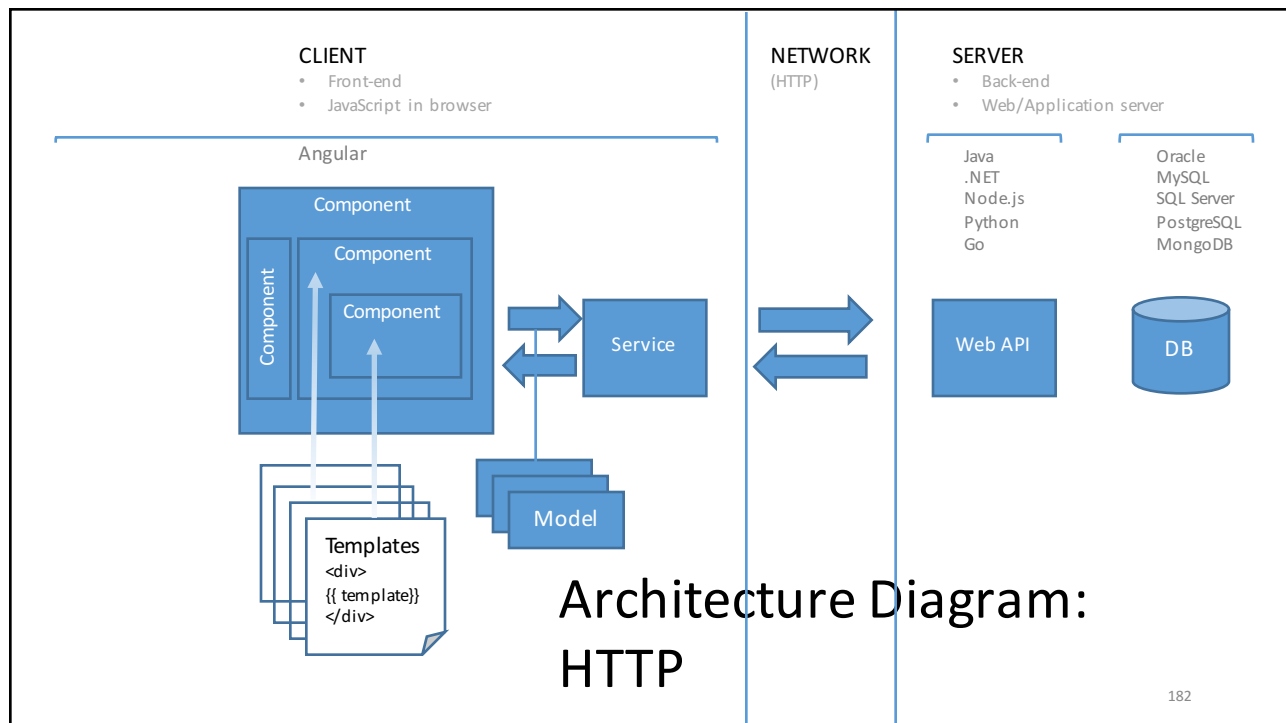


# Http

Angular

181





## Labs

Lab 21: Setup Backend REST API (attendees with instructor)

Rest Review (instructor only demonstration) (optional)

## HttpClient Overview

- Import HttpClientModule
- HttpClient in Services
- Observables in Components

184

## Import HttpClientModule

```
import { HttpClientModule } from '@angular/common/http';

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

185

## HttpClient in Services

```
import { HttpClient } from '@angular/common/http';
import { environment } from '../../environments/environment';

...
export class ProjectService {
  private projectsUrl = environment.backendUrl + '/projects/';

  constructor(private http: HttpClient) {}

  list(): Observable<Project[]> {
    return this.http.get<Project[]>(this.projectsUrl);
  }
}
```



186

## Observables in Components

```
...
export class ProjectsContainerComponent implements OnInit {
  projects: Project[];
  errorMessage: string;

  constructor(private projectService: ProjectService) {}
  ngOnInit() {
    this.projectService.list().subscribe(
      data => {
        this.projects = data;
      }
    );
  }
  ...
}
```

187

# Labs

Lab 22: HTTP GET

188

# Demos: Http Get

Instructor Only Demonstration

`code\demos\http-get`

189

## Error Handling in Services

```
import { Observable, of, throwError } from 'rxjs';
import { catchError } from 'rxjs/operators';
import { HttpClient, HttpResponse } from '@angular/common/http';

...
export class ProjectService {
  ...
  list(): Observable<Project[]> {
    return this.http.get<Project[]>(this.projectsUrl).pipe(
      catchError((error: HttpResponse) => {
        console.log(error);
        return throwError('An error occurred loading the projects.');
      })
    );
  }
}
```

190

## Error Handling in Components

```
...
export class ProjectsContainerComponent implements OnInit {
  projects: Project[];
  errorMessage: string;

  constructor(private projectService: ProjectService) {}
  ngOnInit() {
    this.projectService.list().subscribe(
      data => {
        this.projects = data;
      },
      error => {
        this.errorMessage = error;
      }
    );
  }
  ...
}
```

191

# Demos: Http Error Handling

Instructor Only Demonstration

`code\demos\http-error-handling`

192

## Labs

Lab 23: HTTP Error Handling

193

## HttpClient PUT

```
...
import { HttpClient, HttpResponse, HttpHeaders } from '@angular/common/http';

const httpOptions = {
  headers: new HttpHeaders({ 'Content-Type': 'application/json' })
};

export class ProjectService {
  ...
  put(project: Project): Observable<Project> {
    const url = this.projectsUrl + project.id;
    return this.http.put<Project>(url, project, httpOptions).pipe(
      catchError((error: HttpResponse) => {
        console.log(error);
        return throwError('An error occurred updating the projects.');
```

194

## Labs

Lab 24: HTTP PUT

Lab 25: Showing a Loading Indicator (optional)

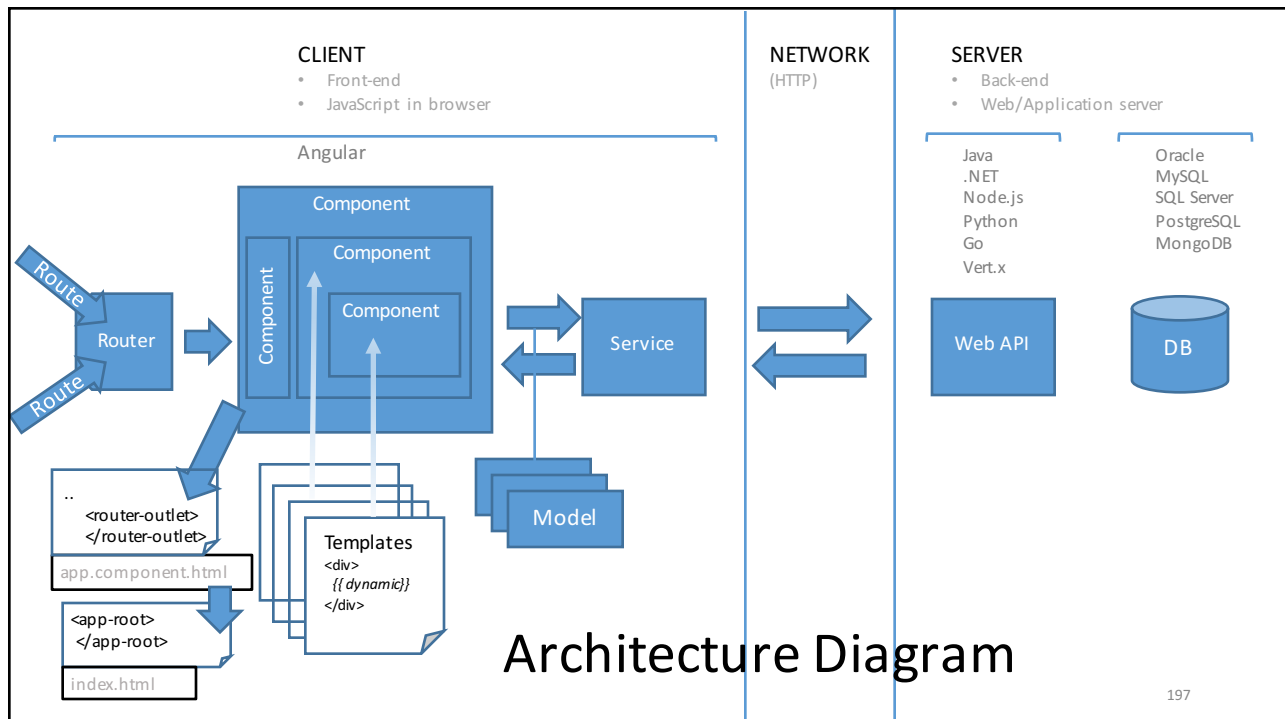
195



# Routing & Navigation

Angular

196



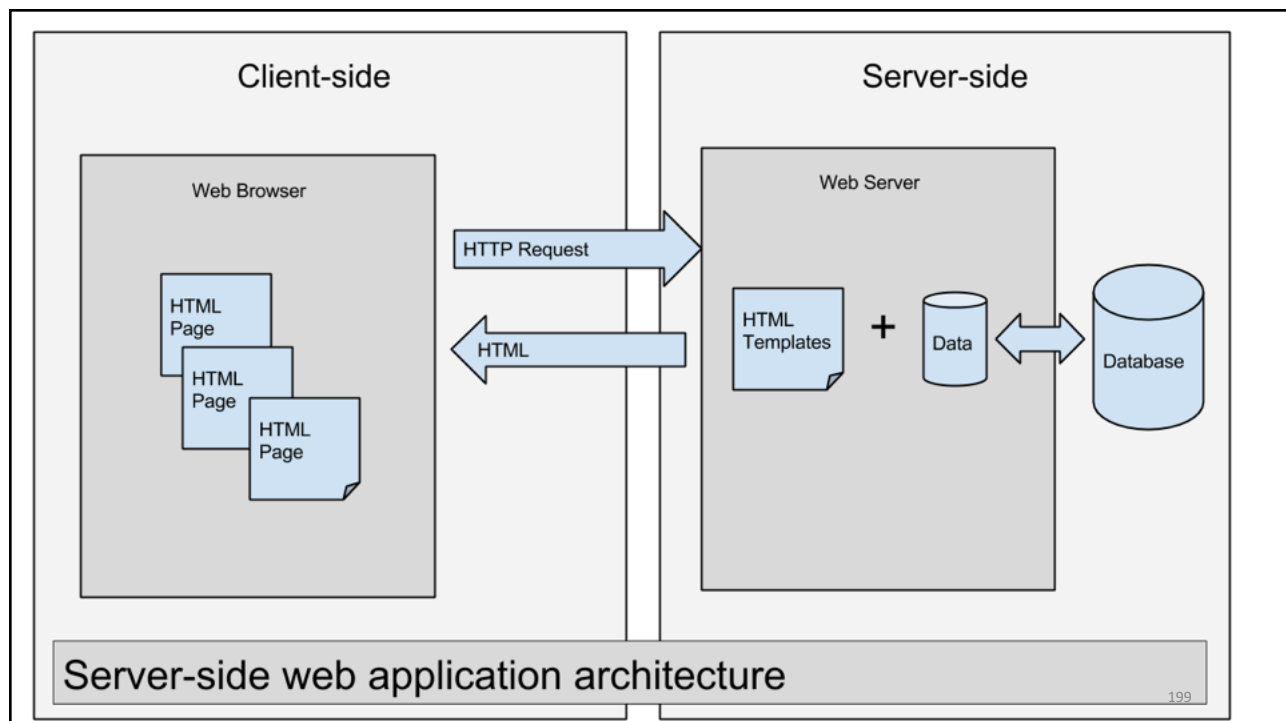
197



## Router Overview

- A router watches the browser's url for changes and runs the corresponding code for that url (route).
- JavaScript applications commonly break the back button in the browser. A router can fix this problem.
- Before HTML5 there was no way to write to the browser's history via JavaScript
- The HTML5 history API (also know as pushState/replaceState) enables JavaScript code to add or modify history entries
- Before HTML5 history, JavaScript applications used the **fragment** identifier introduced by a hash mark # and is the optional last part of a **URL** for a document. It is typically used to identify a portion of that document (hyperlink bookmarks).

198

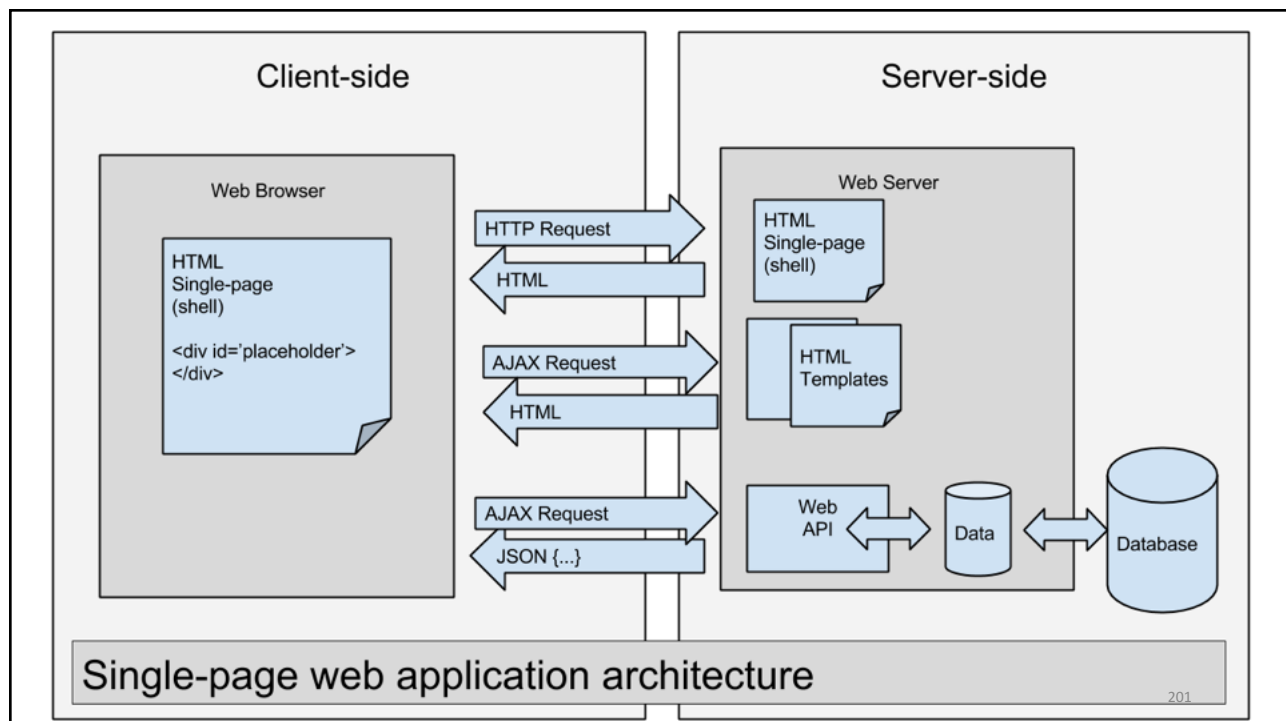


# Component Router

## Angular

- Navigates you through components and their corresponding client-side views without a page reload
- Bind the router to links on a page and it navigates to the appropriate application view when they are clicked
- Also can navigate imperatively when the user clicks a button, selects from a drop box, or in response to some other stimulus
- Logs activity in the browser's history journal so the back and forward buttons work as expected

200



## Routing Summary

- The application is provided with a configured router.
- The component has a RouterOutlet where it can display views produced by the router.
- It has RouterLinks that users can click to navigate via the router.

202

## Router Configuration

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { HomeComponent } from './home/home.component.ts'

const routes: Routes = [
  { path: 'home', component: HomeComponent }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Create an array of **Route** objects.  
Configure the **routes** in the app.

Create a **RouterModule** configured with the application routes.

The Angular CLI generates all this code except the Route objects.

We will learn more about modules next.

203

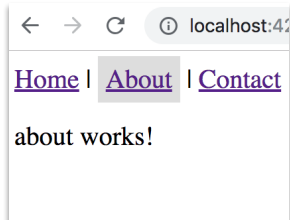
## Router Outlet

```
<div class="container">  
  <router-outlet>  
  </router-outlet>  
</div>
```



Placeholder where the component configured in the **Route** object gets rendered

204



## Demo: Routing Basics

Instructor Only Demonstration

code\demos\routing-basics

205

# Labs

## Lab 26: Router Navigation

206

# Navigating

- Using a routerLink directive
  - Route parameters
  - Query parameters
- Navigating with Code
  - Router
  - Route parameters
  - Query parameters
- Retrieving Parameters
  - ActivatedRoute
  - Synchronous
  - Asynchronous

207

## Route Parameters or Query Parameters?

- There is no hard-and-fast rule. In general,
  - *prefer a route parameter when*
    - the value is required.
    - the value is necessary to distinguish one route path from another.
  - *prefer a query parameter when*
    - the value is optional.
    - the value is complex and/or multi-variate.

208

## Matrix URL Notation

- The query string parameters are not separated by "?" and "&". They are **separated by semicolons (;)**
- This is *matrix URL* notation — something we may not have seen before.
- *Matrix URL* notation is an idea first floated in a [1996 proposal](#) by the founder of the web, Tim Berners-Lee.

209



# Demo: Routing Navigation

Instructor Only Demonstration  
code\demos\routing-navigation

210

## Labs

Lab 27: Router Parameters

211



# Build & Deploy

Using Angular CLI (Command-Line Interface)

212

## Angular CLI: Builds

- Builds your application for deployment to production
  - Combining, minifying
  - Tree-shaking
  - Ahead-of-time compilation
  - Remove Decorators

213



## CLI: Builds

- The **build** command sends output to the **dist** directory.
- **Copy** the contents of the **dist** directory to your web server to deploy.
- Run a development build  
`ng build --dev`  
Note: running just “ng build” defaults to a “dev” build
- Run a production build  
`ng build --prod`
- Passing the dev or prod flags effectively sets other flags
  - These flags are “meta” flags

214

## Demo

Angular CLI: Builds

Instructor

215

## Dev vs Prod Builds

Flag	--dev	--prod
--aot	false	true
--environment	dev	prod
--output-hashing	media	all
--sourcemaps	true	false
--extract-css	false	true
--named-chunks	true	false
--build-optimizer	false	true with AOT and Angular 5

216

## Bundles

- runtime.bundle.js is webpack
- polyfills.bundle.js includes core-js and zone.js
- styles.bundle.js | .css includes all the CSS component styles as well as global styles in styles.css combined into one file
- vendor.bundle.js includes all the angular framework and rxjs
  - *when using build optimizer the vendor bundle is combined into the main bundle*
- main.bundle.js includes all your application code

```
chunk {0} runtime.ec2944dd8b20ec099bf3.js (runtime) 1.44 kB [entry] [rendered]
chunk {1} main.a51dad06404438ada278.js (main) 273 kB [initial] [rendered]
chunk {2} polyfills.f6ae3e8b63939c618130.js (polyfills) 59.6 kB [initial] [rendered]
chunk {3} styles.3ff695c00d717f2d2a11.css (styles) 0 bytes [initial] [rendered]
```

217

## Bundle Sizes

	dev (kB)	prod (kB)
<b>runtime.bundle.js</b>	5.83	1.45
<b>main.bundle.js</b>	27.90	239.00
<b>polyfills.bundle.js</b>	202.00	61.30
<b>styles.bundle.js   css*</b>	11.40	0.08
<b>vendor.bundle.js</b>	2,730.00	**
<b>total</b>	2,977.13	301.83

\* Styles produce js file in dev mode but a css file in prod.

218

## Ahead-of-Time (AOT) Compiler

- Angular offers two ways to compile your application:
  - *Just-in-Time* (JIT), which compiles your app in the browser at runtime
  - *Ahead-of-Time* (AOT), which compiles your app at build time.
- Converts your Angular HTML and TypeScript code into efficient JavaScript code during the build phase *before* the browser downloads and runs that code.
- The size of the Angular framework downloaded in the browser decreases in size by over 50%. Angular's template compiler code does not need to be sent to the browser because the template compilation has been done ahead-of-time.

219

## Build Optimizer

- Build-Optimizer (PurifyPlugin) is a Webpack plugin created by the Angular team, specifically to optimize Webpack builds beyond what Webpack can do on its own.
- Optimizations
  - Removal of Angular decorators from AoT builds
  - adding `/*@__PURE__*/` annotations to transpiled/downleveled TypeScript classes. The point of this is to make it easier for minifiers like Uglify to remove unused code.
  - Full list of optimizations is available here:
    - <https://www.npmjs.com/package/@angular-devkit/build-optimizer>

220

## Build Optimizer and Vendor Chunk

- When using Build Optimizer the vendor chunk will be disabled by default. You can override this with `--vendor-chunk=true`.
- Total bundle sizes with Build Optimizer are smaller if there is no separate vendor chunk because having vendor code in the same chunk as app code makes it possible for Uglify to remove more unused code.

221

# Labs

Lab 29: Build & Deploy

222

# Appendices

The remainder of the topics in this manual are not always able to be covered during the introductory course. Instructors can choose to include them as time allows or questions arise. Students can use them as additional information to go deeper on topics particularly around the setup, build, and deployment of an Angular project.

223

## Custom Pipe Example

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

@Pipe({
  name: 'charlength'
})
export class CharacterLengthPipe implements PipeTransform{

  transform(value:string, length:number){
    return value.substring(0, length);
  }
}
```

224

## Labs

Lab 28: Custom Pipe

225



# Past, Present & Future

Features in Different Angular Versions

226

## Angular 4

- Released 2017-03-23 (code named invisible-makeover)
- View Engine generates smaller code
- Enhanced `*ngIf` syntax
- Animation code now in own packages
- TypeScript 2.1
  - Improved compiler speed
  - Compliant with `StrictNullChecks`
- Universal
  - Run your Angular code on a server, not just browser
    - Faster initial render
    - Search Engine Optimization benefits for public sites
- Flat ESModules
  - More efficient tree shaking
- HttpClient (introduced in 4.3)

227

## Angular 5

- Released 2017-11-01 (code named pentagonal-donut)
- Performance (Angular CLI v1.5)
  - Build optimizer
    - Webpack Plugin: `require('@angular-devkit/build-optimizer').PurifyPlugin;`
    - Additional optimizations (better tree shaking)
  - Angular Compiler
    - Faster builds and rebuilds, uses TypeScript transforms
  - Intl and Reflect polyfills no longer needed
  - Can configure (ts.config) `preserveWhitespaces: false` in template
- Features
  - `updateOn blur or submit`
  - Router new lifecycle events
  - Package to facilitate adding service worker to apps (Progressive Web Apps)
- Upgrade tool
  - <https://angular-update-guide.firebaseio.com/>

228

## Angular 6

- Released May 3rd
- Aligning library releases (version numbers)
- Angular Elements
- Extending Angular CLI with Schematics
- RxJS 6 imports
- Service Providers
  - Instead of a module referencing services
  - Services reference modules

229



## Aligning Library Releases

	Today	With v6
Angular	5.2.10	6.0
Material	5.2.4	6.0
CLI	1.7	6.0

230

## Angular Elements

Wrap Angular Component as a  
Custom Element (aka DOM Element)

231

## Angular Elements: How it Works

### You

- Write components
- Wrap as Custom Element

### They

- Import script
- Use component

232

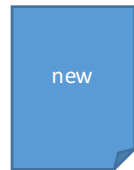
## Use Cases

- Use Angular component in another **Angular app**
- Use Angular components with **other libraries/frameworks**
  - Like a simple jQuery app or VueJS app
- Use Angular components in **any app** written with **any technology**

233

# Extending Angular CLI with Schematics

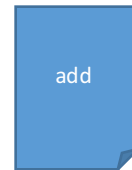
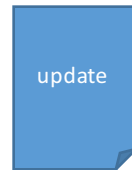
ng [command]



project



component  
module  
service  
pipe  
...



234

## ng update

- Update
  - Npm dependencies
  - Code
  - V6: RxJS, Material

235

## ng add

- Add capabilities
- Instant code and library scaffolding
- Supported by:
  - Angular Material (Ex. ng add @angular/material)
  - Angular Elements
  - Progressive Web App
  - Ng-bootstrap
  - Clarity
  - NativeScript
- Also, per-library 'generate' schematics

236

## RxJS 5 Imports

Non-Exhaustive List

rxjs rxjs/Rx rxjs/Observable rxjs/Subject rxjs/BehaviorSubject rxjs/ReplaySubject rxjs/observable/interval rxjs/observable/timer rxjs/observable/from rxjs/observable/of rxjs/observable/fromEvent rxjs/observable/defer rxjs/observable/if rxjs/observable/throw rxjs/observable/empty rxjs/observable/never rxjs/observable/using rxjs/observable/generate	rxjs/operators/audit rxjs/operators/auditTime rxjs/operators/buffer rxjs/operators/bufferCount rxjs/operators/bufferTime rxjs/operators/bufferToggle rxjs/operators/bufferWhen rxjs/operators/catch rxjs/operators/combineAll rxjs/operators/combineLatest rxjs/operators/concat rxjs/operators/concatAll rxjs/operators/concatMap rxjs/operators/concatMapTo rxjs/operators/count rxjs/operators/debounce rxjs/operators/debounceTime rxjs/operators/defaultIfEmpty	rxjs/operators/audit rxjs/operators/auditTime rxjs/operators/buffer rxjs/operators/bufferCount rxjs/operators/bufferTime rxjs/operators/bufferToggle rxjs/operators/bufferWhen rxjs/operators/catch rxjs/operators/combineAll rxjs/operators/combineLatest rxjs/operators/concat rxjs/operators/concatAll rxjs/operators/concatMap rxjs/operators/concatMapTo rxjs/operators/count rxjs/operators/debounce rxjs/operators/debounceTime rxjs/operators/defaultIfEmpty	There are a LOT
---	---	---	-----------------

237

## RxJS 6 Imports

- rxjs: Contains creation methods, types, schedulers, and utilities

```
import {  
  Observable,  
  Subject,  
  pipe,  
  of,  
  from,  
  interval,  
  merge,  
  fromEvent,  
} from 'rxjs';
```

- rxjs/operators: Contains all pipeable operators

```
import { map, filter, scan } from 'rxjs/operators';
```

238

## Upgrading to Angular 7

- If already on Angular 6 and RxJS 6
  - ng update @angular/cli @angular/core
- If using Angular Material also run
  - ng update @angular/material

239

# Angular 7

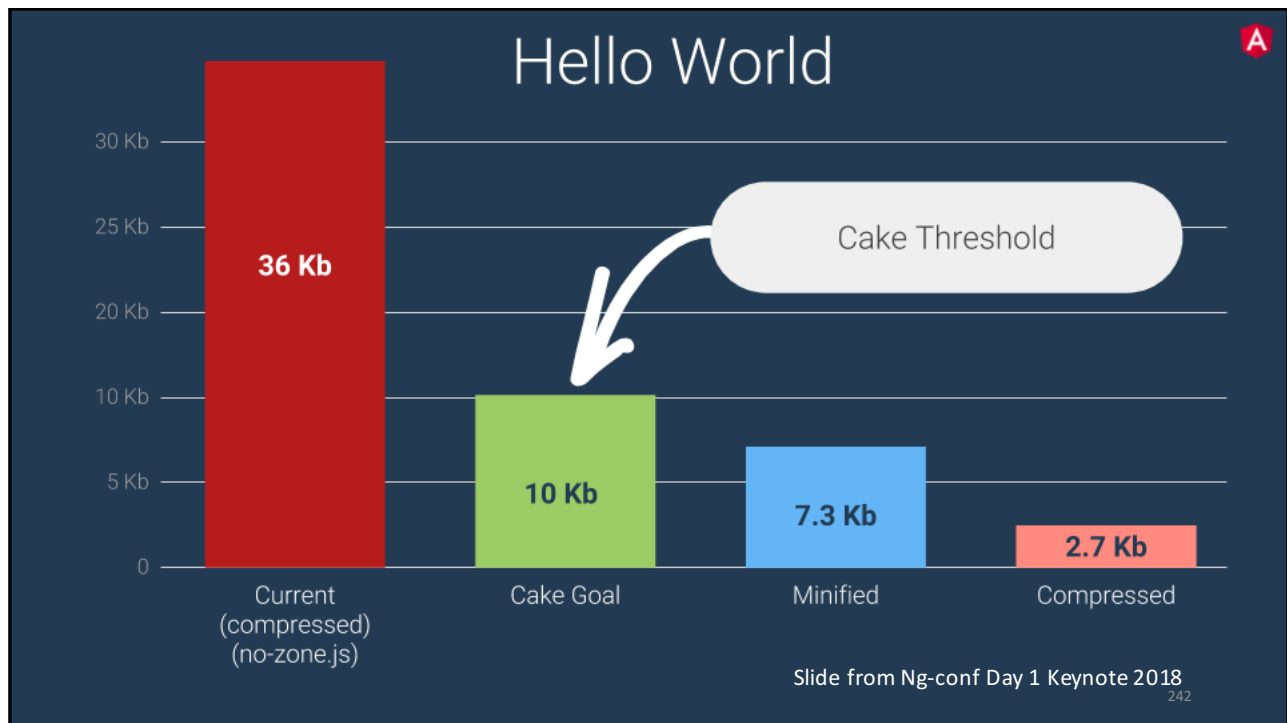
- Angular CLI prompts
- Angular Material CDK
  - ScrollingModule\Virtual Scroll Viewport
  - DragDropModule
- Application Performance
  - Reflect-metadata polyfill not included in production builds
  - Angular CLI Budget Bundles
- Updated Dependencies
  - TypeScript 3.1
  - RxJS 6.3
  - Node 10 support
- Angular Elements now supports content projection using web standards (slots) for custom elements

240

# Ivy

- Next generation Angular Renderer
- Backward compatible
  - No changes required for existing apps
- Benefits
  - Faster
    - Improve the speed of rendering
  - Smaller
    - Optimizes the size of the final package
  - Simpler
    - Human-readable code
    - Easy debugging (stack trace heaven)

241



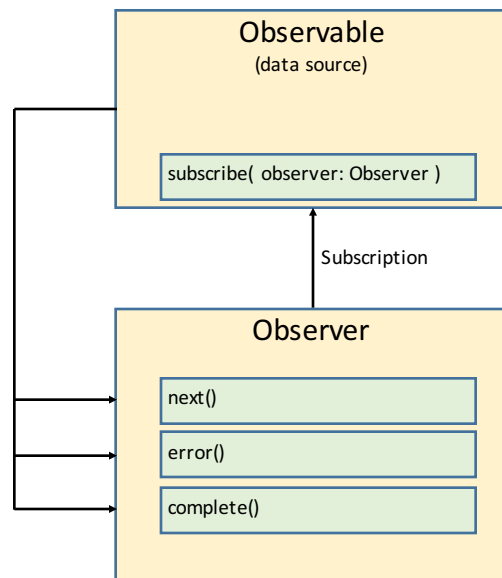
RxJS

# RxJS

- **Reactive Extensions for JavaScript**

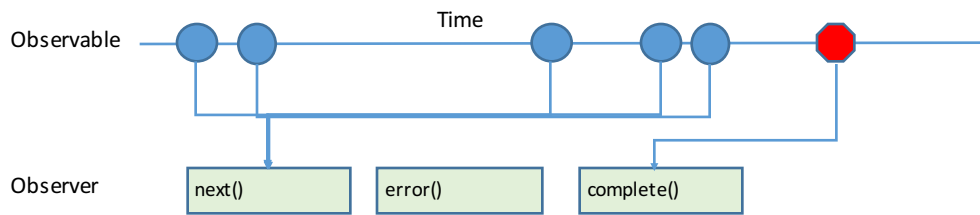
- RxJS is a library for composing asynchronous and event-based programs by using observable sequences.
- It provides one core type, the [Observable](#), satellite types (Observer, Schedulers, Subjects) and operators inspired by [Array#extras](#)(map, filter, reduce, every, etc) to allow handling asynchronous events as collections
- Using RxJS, developers represent asynchronous data streams with Observables and query asynchronous data streams using the many operators (functions) provided

244



245





246

## Observable

- A representation of any set of values over any amount of time
- The most basic building block of RxJS
- Represents a data source that streams values over time
- Observables are lazy push collections of multiple values

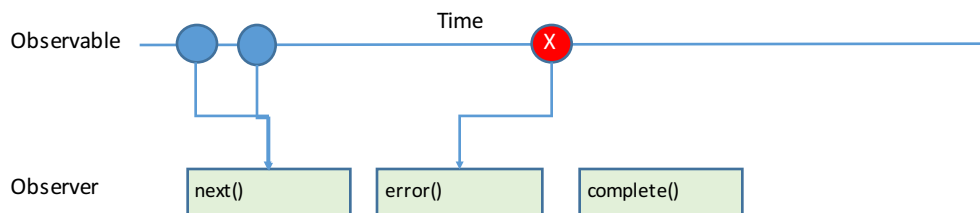
	Single	Multiple
Pull	<a href="#">Function</a>	<a href="#">Iterator</a>
Push	<a href="#">Promise</a>	<a href="#">Observable</a>

247

## Observable Creation Functions

- Usually used to create Observables from scratch
- Pure functions attached to the Observable class in RxJS  $\leq 5.4$
- Stand-alone functions in RxJS  $\geq 5.5$  to RxJS  $< 6$ 
  - `import { of } from 'rxjs/observable/of';`
- RxJS  $\geq 6$ 
  - `import { of } from 'rxjs';`

248



249

## Creating Observables

```
of(1,2,3)
.subscribe(x=> console.log(x)); // 1, 2, 3

from([1,2,3])
.subscribe(x=> console.log(x)); // 1, 2, 3

create(internalObserver => {
  internalObserver.next('a');
  internalObserver.next('b');
})
.subscribe(x => console.log(x)); //a, b
```

250

## Creating Observables

```
let button = document.querySelector("button");
fromEvent(button, "click")
.subscribe(x => console.log(x));

let input = document.querySelector("input");
fromEvent(input, "keyup")
.subscribe((x: Event) => console.log(x.target.value));
```

251

## Demo: Observables

Instructor Only

`code/demos/rxjs-observables`

252

## Observer

- A collection of callbacks that knows how to listen to values delivered by the Observable
- An Observer is a consumer of values delivered by an Observable
- Observers are simply a set of callbacks, one for each type of notification delivered by the Observable:
  - next
  - error
  - complete

253

## Observer Example

```
//Observer
let observer: Observer<any> = {
  next: x => console.log(x),
  complete: () => console.log('completed'),
  error: x => console.log(x)
}

of(1,2,3)
.subscribe(observer); // 1, 2, 3, completed
```

254

## Demo: Observers

Instructor Only

[code/demos/rxjs-observers](#)

[code/demos/rxjs-subscriptions](#)

255

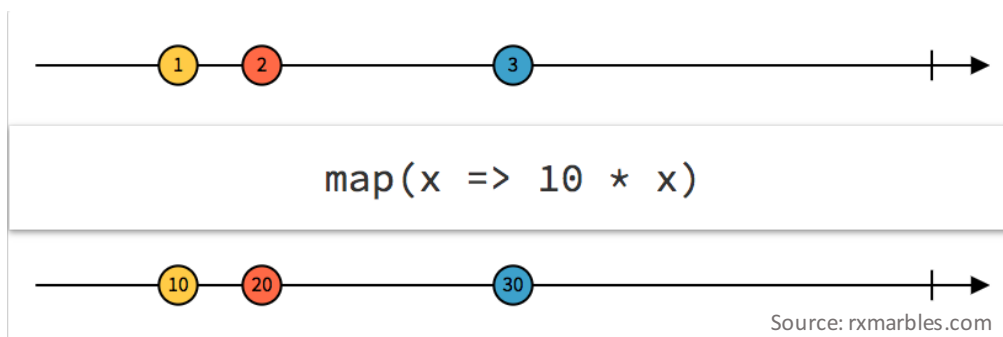
# Operators

- Methods on Observable instances in RxJS <=5.4
- Stand-alone functions in RxJS >=5.5
  - import { map, filter } from 'rxjs/operators';
  - Use the this keyword to infer what is the input Observable

256

## map Operator

```
//map returns same number of items as source  
of(1, 2, 3)  
  .pipe(map(x => x * 10))  
  .subscribe(x => console.log(x)); //10, 20, 30
```



257

## switchMap Operator

```
let obs1$ = of(1, 2, 3);  
let obs2$ = of('a', 'b');  
  
obs1$.pipe(switchMap(() => obs2$)).subscribe(observer);  
// a, b, a, b, a, b, completed
```

---

```
let obs1$ = of(1, 2, 3);  
let obs2$ = of('a', 'b');  
  
obs1$.pipe(switchMap(() => obs2$, (n, 1) => n + 1)).subscribe(observer);  
// 1a, 1b, 2a, 2b, 3a, 3b
```

258

## Demo: Operators

Instructor Only

[code/demos/rxjs-operators](#)

259

# Subject

- A Subject is like an Observable, but can multicast to many Observers
- Subjects are like EventEmitters: they maintain a registry of many listeners
- Plain Observables are unicast (each subscribed Observer owns an independent execution of the Observable)

260

## Subject Example

```
var subject = new Subject();

subject.subscribe({
  next: (v) => console.log('observerA: ' + v)
});

subject.subscribe({
  next: (v) => console.log('observerB: ' + v)
});

subject.next(1);
subject.next(2);

// observerA: 1
// observerB: 1
// observerA: 2
// observerB: 2
```

261



## Practical Application of RxJS

```
this.items = this.searchTermStream.pipe(  
  debounceTime(300),  
  distinctUntilChanged(),  
  switchMap((term: string) => this.wikipediaService.search(term))  
);
```

---

- `debounceTime` waits for the user to stop typing for at least 300 milliseconds
- `distinctUntilChanged` ensures that the service is called only when the new search term is different from the previous search term
- `switchMap` calls the `WikipediaService` with a fresh, debounced search term and coordinates the stream(s) of service response

263

## Demo: Practical Application of RxJS

Instructor Only

`code/demos/rxjs-practical`

264

## EventEmitter or Observable

- Summary
  - Use EventEmitter in Components
  - Use some form of an Observable (Observable, Subject, BehaviorSubject) in Services
- Explanation
  - Do NOT count on EventEmitter continuing to be an Observable
  - Do NOT count on those Observable operators being there in the future
  - Only call event.emit()... Don't defeat angular's abstraction

265

## Observables and Reactive Programming

### In Angular

- We can structure our application to use Observables as the backbone of our data architecture
- Using Observables to structure our data is called Reactive Programming
- Reactive programming is programming with asynchronous data streams
- Observables are the main data structure we use to implement Reactive Programming.

266

## Promises vs. Observables

### Promises

- Returns single value
- Not cancellable
- Standard as of ES 2015

### Observables

- Returns multiple values over time
- Cancellable
- Retry
- Supports map, filter, reduce and similar operators
- Proposed feature for ES 2016
  - Angular uses Reactive Extensions (RxJS)

267