Introduction To ANGULAR

Angular from scratch to creating a complete app.

When :

7th May 2018,

Instructor:

Rushi is a humble, lifelong programmer.

His interests are in Web technologies and ML

rushikesh

rushis.com

Agenda

- What is Angular
- Benefits
- Architecture
- Tooling
- Transition from ES5 -> ES6

  -> Typescript
- Angular CLI
- NgModules
- Dependency Injection
- Components
- Component Interaction
- Lifecycle Hooks
- Templates & Data Binding

Agenda

- Style Guide
- Pipes
- Attribute Directives
- Structural Directives
- Forms
- Template Forms
- Reactive Forms
- Custom Validators
- Observables & RxJS
- HttpClient
- Routing & Navigation
- Angular Compilation
- AOT Advantages
- Debugging

Prerequisites

- Good knowledge in Javascript (ES2015 and above)
- Basics of TypeScript
- Knowledge of web development will be helpful
- Knowledge of Design Patterns will be helpful
- Familiarity with AngularJS will be handy.

What is Angular

- Angular is a platform that makes it easy to build applications with the web.
- Angular combines declarative templates, dependency injection, end to end tooling, and integrated best practices to solve development challenges.
- Angular empowers developers to build applications that live on the web, mobile, or the desktop

What is Angular

- Open Source
- Backed by Google
- Large Developer Community
- Regular Updates
- World Wide ng Conference.
- Readily available open source components.

Why Angular When AngularJS Exists

- Complete Rewrite
- Address Community Concerns
- Speed & Performance
- Future proof
- Cross Platform (PWA, Native, Desktop)
- Developer Productivity
- Complete Development Experience

Angular Versioning

- Semantic Versioning
  - Angular follows Semantic versioning.
  - It is of form MAJOR.MINOR.PATCH ex: 6.0.1
  - More info about semver can be found here and here.
- Release Cadence
  - A major release every 6 months. 1-3 minor releases for each major release
  - A patch release almost every week
  - This cadence of releases gives you access to new beta features as soon as they are ready, while maintaining the stability and reliability of the platform for production users.

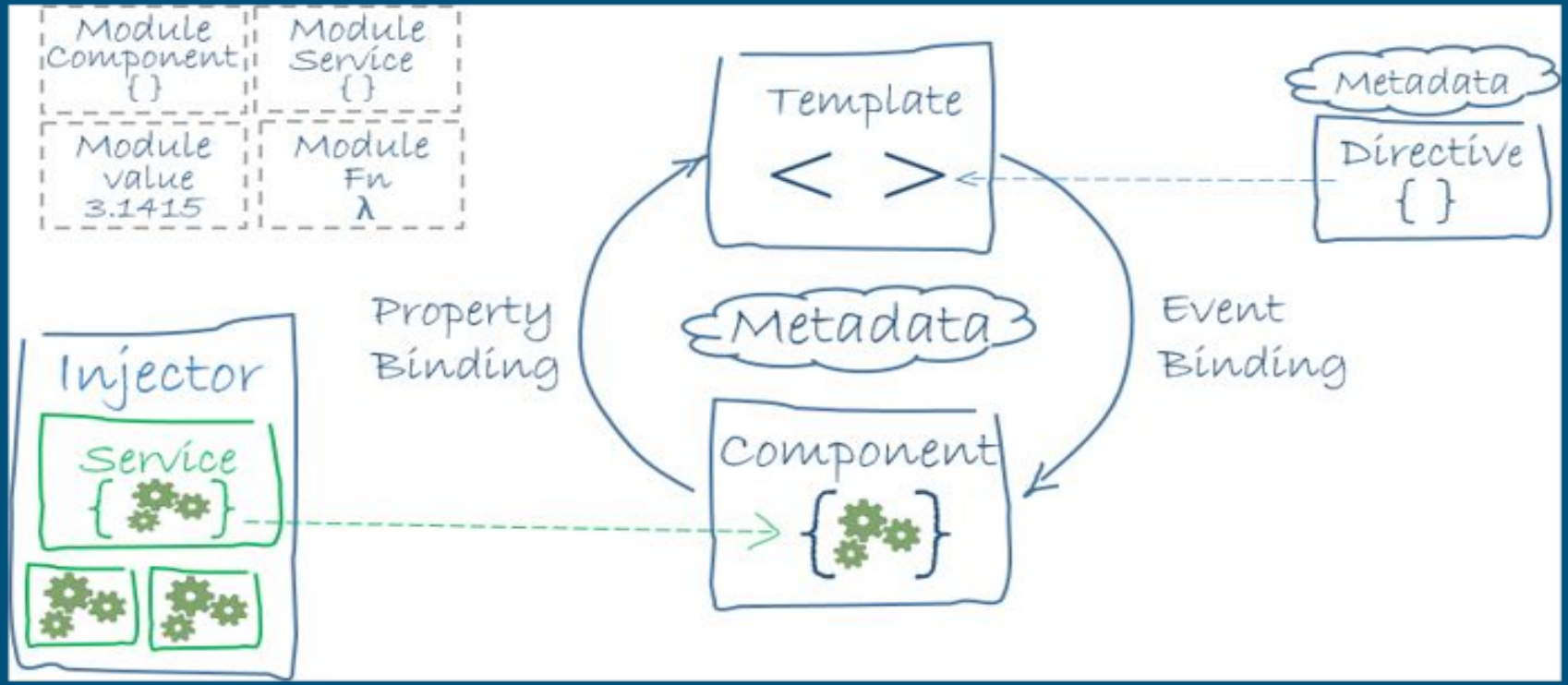
Angular Versioning

- Long Term Support (LTS)
  - All of Angular releases are supported actively for about 6 months (until the next major release), and then they are supported through long-term support (LTS) for another 12 months.
  - During the LTS period, only critical fixes and security patches will be merged and released.
  - The LTS state of one major version starts on the day of the next major release.
  - LTS status ends approximately one year later, when they release another major version.

Angular Versioning (3 is skipped)

- Router Culprit
  - Angular is being developed in a MonoRepo it means a single repo for everything. @angular/core, @angular/compiler, @angular/router etc are in the same repo and may have their own versions.
  - The advantage of MonoRepo is, you don't have to deal with the versioning of the code dependencies.
  - Now the problem is with the @angular/router which was already in a 3.X version. And that's because of some active and huge developments on the router section, like route-preload.
  - Now releasing Angular as version 3, with it's route on version 4 will create confusion. To avoid this confusion they decided to skip the version 3

Architecture - High level



Architecture - High level

- Angular app is defined by a set of NgModules.
- An app always has at least a root module that enables bootstrapping, and typically has many more feature modules.
- NgModules provide a compilation context for components.
- Components define views, which are sets of screen elements that Angular can choose among and modify according to your program logic and data.
- Every app has at least a root component.

Architecture - High level

- Components use services, which provide specific functionality not directly related to views.
- Service providers can be injected into components as dependencies, making your code modular, reusable, and efficient.
- Both components and services are simply classes, with ***decorators*** that mark their type and provide metadata that tells Angular how to use them.

Architecture - High level

- The metadata for a component class associates it with a template that defines a view. A template combines ordinary HTML with Angular directives and binding markup that allow Angular to modify the HTML before rendering it for display.
- The metadata for a service class provides the information Angular needs to make it available to components through Dependency Injection (DI).

Tooling

- VSCode (any editor/ide with appropriate plugins)
- Node
- npm
- Module loaders (SystemJS, CommonJS, AMD etc)
- Babel
- Typescript Compiler (tsc)
- WebPack

Transition From ES5 -> ES6 -> Typescript

- Live Coding
  - Start with ES5 angular.
  - Create Component
  - Create Service
  - Create Directive
  - Convert this code to modular javascript
  - Convert this code to ES6 modules
  - Convert this code to Typescript
  - Time to introduce Angular CLI

Angular CLI

- Angular CLI is a command-line interface (CLI) to automate your development workflow. It allows you to:
  - create a new Angular application
  - run a development server with LiveReload support to preview your application during development
  - add features to your existing Angular application
  - run your application's unit tests
  - run your application's end-to-end (E2E) tests
  - build your application for deployment to production.

Angular CLI

- Before you can use Angular CLI, you must have Node.js 6.9.0 and npm 3.0.0 or higher installed on your system.
- Install using: **`npm install -g @angular/cli`**
- **`ng init`**: create a new application in the current directory
- **`ng new`**: create a new directory and run ng init inside the new directory.
- Stories describing how to do more with the CLI

Angular CLI

- Running the command **`ng new my-app`** will do the following:
  - a new directory my-app is created
  - all source files and directories for your new Angular application are created based on the name you specified (my-app) and best-practices from the official Angular Style Guide
  - npm dependencies are installed
  - TypeScript is configured for you
  - the Karma unit test runner is configured for you
  - the Protractor end-to-end test framework is configured for you
  - environment files with default settings are created.

Angular CLI

- **ng serve**: to start the built-in development server on default port
- **ng test**: to run all unit tests
- **ng e2e**: to run all E2E tests
- **ng build --target=production**: To build and bundle your application for deployment
- **ng eject**: If, at any given time, you wish to manually configure Webpack and you no longer want to use Angular CLI with your Angular application

Angular CLI

- You can use the **`ng generate`** command to add features to your existing application, **g is the alias for generate**
  - `ng g [class|cl] my-new-class`: add a class to your application
  - `ng g [component|c] my-new-component:` add a component to your application
  - `ng g [directive|d] my-new-directive`: add a directive to your application
  - `ng g [enum|e] my-new-enum`: add an enum to your application
  - `ng g [module|m] my-new-module`: add a module to your application
  - `ng g [pipe|p] my-new-pipe`: add a pipe to your application
  - `ng g [service|s] my-new-service`: add a service to your application

NgModules

- Angular is a platform and framework for building client applications in HTML and TypeScript.
- Angular is itself written in ***TypeScript***.
- It implements core and optional functionality as a set of TypeScript libraries that you import into your apps.
- The basic building blocks of an Angular application are ***NgModules***, which provide a compilation context for ***components***.

NgModules

- Organizing your code into distinct functional modules helps in managing development of complex applications, and in designing for reusability. I
- This technique lets you take advantage of lazy-loading—that is, loading modules on demand—in order to minimize the amount of code that needs to be loaded at startup.

Dependency Injection (DI)

- DI is a pattern in which a class receives its dependencies from external sources rather than creating them itself.
- An Angular injector is responsible for creating service instances and injecting them into classes.
- You rarely create an Angular injector yourself.
- Angular creates injectors for you as it executes the app, starting with the root injector that it creates during the bootstrap process.

Dependency Injection (DI)

- Angular doesn't automatically know how you want to create instances of your services or the injector to create your service.
- You must configure it by specifying providers for every service.
- Providers tell the injector how to create the service. Without a provider, the injector would not know that it is responsible for injecting the service nor be able to create the service.
- The @Injectable decorator identifies services and other classes that are intended to be injected.

Dependency Injection (DI)

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

- **providedIn** tells Angular that the root injector is responsible for creating an instance of the service (by invoking its constructor) and making it available across the application.
- The CLI sets up this kind of a provider automatically for you when generating a new service.

Dependency Injection (DI)

```
@Injectable({
  providedIn: ModuleName,
})
```

- Users can explicitly opt-in to using the service, or the service can be provided in a lazily-loaded context. In this case, the provider should be associated with a specific @NgModule class, and will be used by whichever injector includes that module.
-

Dependency Injection (DI)

- Angular module providers (**`@NgModule.providers`**) are registered with the application's root injector.
- Angular injects the corresponding services in any class it creates.
- Once created, a service instance lives for the life of the app and Angular injects this one service instance in every class that needs it.
- When you register providers in the @Injectable decorator of the service itself, optimization tools such as those used by the CLI's

Dependency Injection (DI)

- When you register providers in the @Injectable decorator of the service itself, optimization tools such as those used by the CLI's production builds can perform tree shaking, which removes services that aren't used by your app.
- Tree shaking results in smaller bundle sizes.
- Angular module providers are registered with the root injector unless the module is lazy loaded.

Dependency Injection (DI)

- A component's providers (**`@Component.providers`**) are registered with each component instance's own injector.
- Angular can only inject the corresponding services in that component instance or one of its descendant component instances. Angular cannot inject the same service instance anywhere else.
- Each new instance of the component gets its own instance of the service and, when the component instance is destroyed, so is that service instance.

Components

- Components define views, which are sets of screen elements that Angular can choose among and modify according to your program logic and data.
- Every app has at least a root component.
- Components use services, which provide specific functionality not directly related to views.
- Service providers can be injected into components as dependencies, making your code modular, reusable, and efficient.

Components

- The @Component decorator identifies the class immediately below it as a component, and provides the template and related component-specific metadata.
- You define a component's view with its companion template.
- A template is a form of HTML that tells Angular how to render the component.
- Views are typically arranged hierarchically.

Component Interaction: Child to Parent

- The parent Component can pass value to child component by assigning the value using property binding.
- The @Input decorator identifies the child Component property.
- The @Input decorator also takes a parameter to alias the property.
- We can use an input property setter to intercept and act upon a value from the parent.
-

Component Interaction: Child to Parent

- We can detect and act upon changes to input property values with the ngOnChanges() method of the OnChanges lifecycle hook interface.
-

Component Interaction: Parent to Child

- The child component exposes an **EventEmitter** property with which it emits events when something happens. The parent binds to that event property and reacts to those events.
- The child's EventEmitter property is an output property, typically adorned with an **`@Output`** decorator.
- The parent Component binds an event handler that responds to the child event payload.
-

Component Interaction: Parent to Child

- A parent component cannot use data binding to read child properties or invoke child methods. You can do both by creating a **template reference variable** for the child element and then reference that variable within the parent template
- **Caveat**: The local variable approach is simple and easy. But it is limited because the parent-child wiring must be done entirely within the parent template. The parent component itself has no access to the child.

Component Interaction: Parent to Child

- **Caveat:** You can't use the local variable technique if an instance of the parent component class must read or write child component values or must call child component methods.
- **Solution**: When the parent component class requires that kind of access, inject the child component into the parent as a ViewChild.
-

LifeCycle Hooks

- A component has a lifecycle managed by Angular.
- Angular creates it, renders it, creates and renders its children, checks it when its data-bound properties change, and destroys it before removing it from the DOM.
- Angular offers lifecycle hooks that provide visibility into these key life moments and the ability to act when they occur.
- This can be done by implementing one or more of the lifecycle hook interfaces in the Angular core library.

LifeCycle Sequence

- After creating a component/directive by calling its constructor, Angular calls the lifecycle hook methods in the following sequence at specific moments:
  - ngOnChanges()
  - ngOnInit()
  - ngDoCheck()
  - ngAfterContentInit()
  - ngAfterContentChecked()
  - ngAfterViewInit()
  - ngAfterViewChecked()
  - ngOnDestroy()

Templates & Views

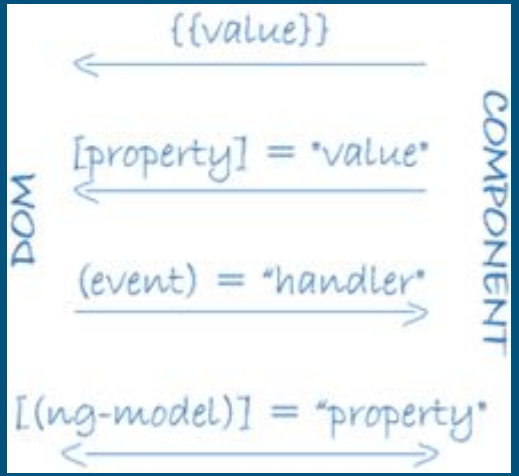
- Views allow you to modify or show and hide entire UI sections or pages as a unit.
- The template immediately associated with a component defines that component's host view.
- The component can also define a view hierarchy, which contains embedded views, hosted by other components.
- A view hierarchy can include views from components in the same NgModule or views from components that are defined in different NgModules.

Templates & Views

- A template looks like regular HTML, except that it also contains Angular template syntax.
- Angular template syntax alters the HTML based on your app's logic and the state of app and DOM data.
- Templates can use
  - data binding to coordinate the app and DOM data.
  - pipes to transform data before it is displayed
  - directives to apply app logic to what gets displayed.

Data binding

- The following diagram shows the four forms of data binding markup. Each form has a direction—to the DOM, from the DOM, or in both directions.



Data binding

- Angular processes all data bindings once per JavaScript event cycle, from the root of the application component tree through all child components.
- Data binding plays an important role in communication between a template and its component
- It is also important for communication between parent and child components.

Style Guide

- Programming style is a set of rules or guidelines used when writing the source code for a computer program.
- It is often claimed that following a particular programming style will help programmers read and understand source code conforming to the style, and help to avoid introducing errors.
- The programming style used in a particular program may be derived from the coding conventions of a company or other computing organization.

Style Guide

- Programming styles are often designed for a specific programming language/framework.
- Angular style guide presents preferred conventions and, as importantly, explains why. More In-depth info here.
- Apply the **single responsibility principle (SRP)** to all components, services, and other symbols. This helps make the app cleaner, easier to read and maintain, and more testable.
- Separate file names with dots and dashes.

Style Guide

File Name	Content
app.module.ts	@NgModule({ ... }) export class AppModule
app.component.ts	@Component({ ... }) export class AppComponent { }
movie-detail.component.ts	@Component({ ... }) export class MovieDetailComponent { }

Style Guide

File Name	Content
init-caps.pipe.ts	@Pipe({ name: 'initCaps' }) export class InitCapsPipe implements PipeTransform { }
user-profile.service.ts	@Injectable() export class UserProfileService { }
validation.directive.ts	@Directive({ ... }) export class ValidationDirective { }

Style Guide

- use consistent names for all assets named after what they represent.
- put bootstrapping and platform logic for the app in a file named main.ts.
- Use lower camel case for naming the selectors of directives.
- use a hyphenated, lowercase element selector value (e.g. admin-users).
- use a custom prefix for the selector of directives.
-

Style Guide

- Consider giving components an element selector, as opposed to attribute or class selectors.
- extract templates and styles into a separate file, when more than 3 lines.
- Talk to the server through a service
- Implement the lifecycle hook interfaces, doing so will flag spelling and syntax mistakes.
- codelyzer tool enforces set of tslint rules for static code analysis of Angular TypeScript projects.

Pipes

- Angular pipes let you declare display-value transformations in your template HTML.
- A class with the ***@Pipe*** decorator defines a function that transforms input values to output values for display in a view.
- Angular defines various pipes, such as the ***date*** pipe and ***currency*** pipe
- To specify a value transformation in an HTML template, use the pipe operator |

Pipes

- You can chain pipes, sending the output of one pipe function to be transformed by another pipe function.
- A pipe can also take arguments that control how it performs its transformation.
- The arguments are given with a : operator
- Ex: `<p>The date is {{today | date:'fullDate'}}</p>`

Directives

- A directive is a class with a @Directive decorator.
- A component is technically a directive - but components are so distinctive and central to Angular applications that Angular defines the @Component decorator, which extends the @Directive decorator with template-oriented features.
- There are two kinds of directives besides components: structural and attribute directives.

Directives

- Just as for components, the metadata for a directive associates the class with a selector that you use to insert it into HTML.
- In templates, directives typically appear within an element tag as attributes, either by name or as the target of an assignment or a binding.

Structural Directives

- Structural directives alter layout by adding, removing, and replacing elements in DOM.
- *ngFor is an iterative; it tells Angular to generate the tags to the size of the list.
- *ngIf is a conditional; it includes the element on the DOM if expression evaluates to true.
- Ex:

```
<li *ngFor="let hero of heroes"></li>

<app-hero-detail *ngIf="selectedHero"></app-hero-detail>
```

Structural Directives

- You can have only one structural directive per host element.
- The Angular **<ng-container>** is a grouping element that doesn't interfere with styles or layout because Angular doesn't put it in the DOM.
- The <ng-container> is a syntax element recognized by the Angular parser. It's not a directive, component, class, or interface. It's more like the curly braces in a JavaScript if-block
- ~~Custom structural directive.~~

Attribute Directives

- Attribute directives alter the appearance or behavior of an existing element.
- In templates they look like regular HTML attributes, hence the name.
- Angular has pre-defined directives that either alter the layout structure (for example, ngSwitch) or modify aspects of DOM elements and components (for example, ngModel, ngStyle and ngClass).

Angular Forms

- The goals of a form is
  - to be able to synchronize the data in the view with data in the controller.
  - to perform validations
  - notify about errors
  - handle other events like cancel, reset etc.
- Angular provides two approaches to building forms: reactive forms and template forms.
- Template forms are named primarily because the form controls are defined in the template of the component.

Angular Template Forms

- With template forms, we describe the forms primarily using the NgModel directive, which is used to define the form structure.
- import FormsModule
- Angular automatically creates and attaches an **`NgForm`** directive to the `<form>` tag.
- Declare a template variable for the form.
  - Ex: `<form #heroForm="ngForm">`
- Each input element has a name property that is required by Angular forms to register the control with the form .

Angular Template Forms

- The NgForm directive supplements the form element with additional features.
  - It holds the controls you created for the elements with an ngModel directive and name attribute
  - monitors their properties, including their validity.
  - It also has its own valid property which is true only if every contained control is valid.
- Angular creates FormControl instances and registers them with an NgForm directive that Angular attached to the <form> tag.

Angular Template Forms

- Each FormControl is registered under the name you assigned to the name attribute.
- `{{diagnostic}}` makes the raw model visible while testing the form to always reflect the updated model when you edit the form.
- NgModel directive doesn't just track state; it updates the control with special Angular CSS classes that reflect the state. You can leverage those class names to change the appearance of the control.

Angular Template Forms

State	Class if true	Class if false
The control has been visited.	ng-touched	ng-untouched
The control's value has changed.	ng-dirty	ng-pristine
The control's value is valid.	ng-valid	ng-invalid

Angular Template Forms

- bind the form's ngSubmit event property to the form component's onSubmit() method
  - EX: `<form (ngSubmit)="onSubmit()" #heroForm="ngForm">`
-

Angular Reactive Forms

- Model-driven forms enable us to test our forms without being required to rely on end-to-end tests.
- That's why model-driven forms are created imperatively, so they end up as actually properties on our components, which makes them easier to test.
- FormGroup always represents a set of FormControls.
- FormBuilder is like a factory that creates FormGroups and FormControl's

Angular Reactive Forms

- Import FormBuilder
- Inject it into AppComponent
- Create the form model using FormBuilder.group() in ngOnInit() lifecycle.
- A FormControl takes a value as first, a synchronous validator as second and an asynchronous validator as third parameter.

Observables

- Observables provide support for passing messages between publishers and subscribers in your application.
- Observables offer significant benefits over other techniques for event handling, asynchronous programming, and handling multiple values.
- Observables are declarative—that is, you define a function for publishing values, but it is not executed until a consumer subscribes to it.

Observables

- The subscribed consumer receives notifications until the function completes, or until they unsubscribe.
- An observable can deliver multiple values of any type—literals, messages, or events, depending on the context.
- The API for receiving values is the same whether the values are delivered synchronously or asynchronously.

Observables

- Because setup and teardown logic are both handled by the observable, your application code only needs to worry about subscribing to consume values, and when done, unsubscribing.
- Whether the stream was keystrokes, an HTTP response, or an interval timer, the interface for listening to values and stopping listening is the same.
- Because of these advantages, observables are used extensively within Angular, and are recommended for app development.

Observables

- A publisher creates an Observable instance that defines a subscriber function.
- Subscriber function is executed when a consumer calls the subscribe() method.
- The subscriber function defines how to obtain or generate values or messages to be published.
- The subscribe() call returns a Subscription object that has an unsubscribe() method, which you call to stop receiving notifications.

HttpClient

- The HttpClient offers a simplified API for Angular applications that rests on the XMLHttpRequest interface exposed by browsers.
- Additional benefits of HttpClient include
  - testability features,
  - typed request and response objects,
  - request and response interception,
  - Observable apis,
  - and streamlined error handling.

HttpClient

- To use angular HttpClient, you need to import the HttpClientModule.
  - Ex: `import { HttpClientModule } from '@angular/common/http';`
- It is a best practice to separate presentation of data from data access by encapsulating data access in a separate service and delegating to that service in the component.
- The HttpClient captures all errors in its HttpErrorResponse

HttpClient

- Error inspection, interpretation, and resolution is something you want to do in the service, not in the component.
- An HttpClient method does not begin its HTTP request until you call subscribe() on the observable returned by that method. This is true for all HttpClient methods.
- One can create interceptors that can inspect and transform HTTP requests to/from the server.

Routing & Navigation

- The Angular Router enables navigation from one view to the next as users perform application tasks.
- Router logs activity in the browser's history journal so the back and forward buttons work as well.
- A routed Angular application has one singleton instance of the **`Router`** service. When the browser's URL changes, that router looks for a corresponding Route from which it can determine the component to display.

Routing & Navigation

- A router has no routes until you configure it.
- Most routing applications should add a **<base>** element to the index.html as the first child in the <head> tag to tell the router how to compose navigation URLs.
  - Ex: `<base href="/">`
- Import RouterModule from router module.
  - Ex: `import { RouterModule, Routes } from '@angular/router';`
- Each Route maps a URL path to a component.

Routing & Navigation

- There are no leading slashes in the path.
- The router parses and builds the final URL for you, allowing you to use both relative and absolute paths when navigating between application views.
- The router uses a first-match wins strategy when matching routes, so more specific routes should be placed above less specific routes. Order of routes matters.

Angular compilation

- 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.
- JIT compilation is the default when you run the build-only or the build-and-serve-locally CLI commands **ng build** **ng serve**
- For AOT compilation, append the --aot flags to the build-only or the build-and-serve-locally CLI commands: **ng build --aot** **ng serve --aot**
- The **--prod** meta-flag compiles with AOT by default.

AOT Advantages

- **Faster rendering:** With AOT, the browser downloads a pre-compiled version of the application. The browser loads executable code so it can render the application immediately, without waiting to compile the app first.
- **Fewer asynchronous requests**: The compiler inlines external HTML templates and CSS style sheets within the application JavaScript, eliminating separate ajax requests for those source files.

AOT Advantages

- **Smaller Angular framework download size:** There's no need to download the Angular compiler if the app is already compiled. The compiler is roughly half of Angular itself, so omitting it dramatically reduces the application payload.
- **Detect template errors earlier:** The AOT compiler detects and reports template binding errors during the build step before users can see them.

AOT Advantages

- **Better security:** AOT compiles HTML templates and components into JavaScript files long before they are served to the client. With no templates to read and no risky client-side HTML or JavaScript evaluation, there are fewer opportunities for injection attacks.
- **Tree Shaking:** is a process in which unused module exports are removed. It works on both source and library modules.

Debugging

- **`ng.probe($0)`** $0 is a global variable the Chrome DevTools make available, whose value is the most recently selected element. You can use this for $1, $2, $3 & $4 to inspect the previous four DOM elements.
- **`ng.probe($0).componentInstance`** gets the reference to the instance of our component class
- **`ng.probe($0)`** will only give us the necessary information when the debug mode is enabled and will not work on Production where debug mode is usually disabled.

Debugging

- Angular has a built-in profiler which can be used to profile your Angular application.
- Currently it offers only one kind of profiling i.e timeChangeDetection.
- **`ng.profiler.timeChangeDetection()`** timeChangeDetection performs change detection on the application and prints how long a round of change detection takes for the current state of UI. The recommended value for this is to be less than 3 milliseconds.

Debugging

- Augury is a Chrome extension that provides a visual representation of the Angular components on a page, their dependencies, router information, whether change detection is used for the component, etc.
- Augury can provide information only in development mode and not in production mode

Resources & Bibliography

- Angular Docs
- Angular CLI
- Codelyzer
- Augury

Thank You

"I'm not a great programmer; I'm just a good programmer with great habits."

— Kent Beck