Angular 4 Guideline

Installation

1. Installing Angular:
   a) Download latest nodejs from: https://nodejs.org/en/ and install it.
   b) From the cmd run the following commands:
      - For windows: **npm install -g @angular/cli**
      - For linux/mac: **sudo npm install -g @angular/cli**

2. Uninstalling Angular:
   a) From the cmd run the following commands:
      - For windows: **npm uninstall -g angular-cli @angular/cli**
      - For linux/mac: **sudo npm uninstall -g angular-cli @angular/cli**
   b) **npm cache clean**

Important References

1. https://nodejs.org/en/ (node.js download link)
2. https://developer.mozilla.org/en-US/docs/Web/API/Element (DOM property list by MDN)
3. https://developer.mozilla.org/en-US/docs/Web/Events (DOM event list by MDN)
4. https://angular.io/api?type=directive (ng directive api)
5. https://angular.io/guide/http (http docs)
6. https://angular.io/docs/ts/latest/guide/testing.html
7. https://semaphoreci.com/community/tutorials/testing-components-in-angular-2-with-jasmine
8. https://github.com/angular/angular-cli#running-unit-tests
9. http://www.typescriptlang.org/Handbook

Angular Commands

1. To create a new project called 'first-app' from cmd, cd to that directory and build it in a server use the following commands:
   - **ng new** first-app
   - cd first-app/
   - **ng serve** [to host the app in the server]

2. Install npm to load 'node_modules' folder for a project, go to that project directory from cmd and run: **npm install**

3. To create a ng component use the following commands:
   - With name xyx: **ng g c xyz** [g for generate, c for component]
   - Without test file with name xyx: **ng g c xyz --spec false**

4. To install bootstrap run: **npm install --save bootstrap**
5. To create a directive with name xyx: **ng g d xyz** [g for generate, d for directive]
6. To create a new pipe with name filter: **ng g p filter**
7. To install firebase: **npm install --save firebase**
8. Build app with AoT option (Ahead of Time): **ng build --prod –aot**
9. To install ngRx: **npm install --save @ngrx/store**
10. To install effects to handle side effects like async call: **npm install --save @ngrx/effects**
11. To install router store to listen router state change effect: **npm install --save @ngrx/router-store**
12. To install dev tools to check state change dynamically for devs [Chrome dependency: Redux DevTools]: **npm install --save @ngrx/store-devtools**
13. Run test for unit testing: **ng test**
14. To run typescript file: **tsc typing**

Starting Point of Angular

- **main.ts:** import app.modules.ts, which contains all the necessary modular imports.
- **app.modules.ts:** import app.component which includes all the components in the app folder.
- **app.component.ts:** It has the 'app-root' selector.
- **index.html:** It has <app-root>Loading...</app-root> where 'app-root' selector decides to change the Loading... part by the app.component.html file in the app folder.

Basic

Misc

1. Angular is a JS framework which changes the DOM element at run time.
2. Component is a typescript class.
3. Decorator (@) is a feature of typescript which can be imported from any library.
4. NgModule bundles all the components.

5. File extension will be bundled by the webpack in the run time.

6. Angular selector can be use like:
   - html element.
   - css attribute.
   - css class.

7. Default behaviour of view encapsulation: shadow DOM by emulation.

8. Local reference **(#)** in html file can hold html input and use its value to the typescript code.
   - Local reference can pass through method.
   - Local reference can use through **viewchild**.

9. **ng-component** is a directive can be used as component.

10. **ng-template** is an element which itself not rendered, but it determines some element need to be rendered.

11. **ngAfterViewInit(),** you can access the html element and its value after this built in angular method is called.

Data Binding

1. Data binding can be used like:
   - String interpolation: **{{}}**
   - Property binding: **[]**
     For string value, in property binding square bracket is not needed. **E.g. name="amount"**
   - Event binding: **()**
   - Two way binding = property binding + event binding, **[(ngModel)]**.
     Two way binding requires: **import { FormsModule } from '@angular/forms';**

2. Custom property/event binding can be done through selector.

3. Outside of a component, property is not public. To publicly expose it use Input ('alias name') just before the property.

4. $event is a reserved variable name, can be used in event binding.

Directive

1. Directives are instruction in the DOM. Directives are two types as follows:
   a) Structural directive. E.g. ***ngIf, *ngFor, *ngSwitch**
      - *ngIf add/remove DOM element
        Behind the scene: *ngIf is equivalent to <ng-template [ngIf]></ng-template>
      - *ngFor iterate all elements. e.g: *ngFor="let item of items"

b) Attribute directive. E.g. **ngStyle, ngClass**
   - ngStyle takes key-value pair. key = backgroundcolour, value = colour (set a background colour)
   - ngClass takes key-value pair. key = class name, value = add/remove class (add/remove a css class)

2. Creating custom directive:
   a) Renderer2 for styling is a better approach to style DOM element where in some case DOM element are not accessible.
   b) @HostBinding is a decorator can be used to create custom directive to set DOM element style property. This is a replacement of Renderer2.
   @HostListener is used to react user events for this custom directive.

Service

1. Service is a normal typescript class to hold the common functions among different component.

2. If you want to get a single instance of a Service then remove it from provider, if multiple instance is required then add it in the provider.

3. Injecting a service into another service require to use **@Injectable()** metadata.

4. Cross component communication through service can be done using event emitter.

5. Event Emitter allow you to handle your own event outside of the component.

Route

1. Router needs a path and an action after selecting the path. Action can be a component.

2. **<router-outlet>** simply marks the place to inform the angular to load the component of the currently selected route.

3. **routerLink** is used to route the path as per the configured route without doing page refresh. This will prevent to loss state.

4. Root component always stays at **http://localhost:4200/**
   So the sub-component can be used using both:
   - relative path (./server or server)
   - absolute path: (/server)

5. **routerLinkActive** makes a tab active.

6. **Router.Navigate()** method can route a path programmatically.
   Unlike routerLink, Navigate method does not know on which router it currently on, it knows only which component it is set on.

7. **user/:id** by this URL means anything after **:** will be generated dynamically.

8. **route.snapshot.param** can be used first time when the component is loaded, otherwise **route.params.subscribe** should be used since it checks whether the param of the url changed asynchronously.

9. Child/nested route can be done using children property.

10. **redirectTo** is used to re-direct the current route.

11. Guard takes array whereas resolver takes JavaScript object.

12. The semantics of Angular dictate that you use promises as a sort of 'call back handle' do something asynchronous in a service, return a promise and when the asynchronous work is done, the promise's then function is triggered.

13. Resolver is used to load asynchronous data. A resolve contains one or more promises that must resolve successfully before the route will change.

14. Use # in the URL for older browser support: "localhost:4200/#/users", allows web server to parse the path before #.

15. Path after # will be taken care by client side angular. This will prevent the file not found 404 error if the server does not have the index.html page.

Observer

1. Observable is used to handle asynchronous event data.

2. Create your own observable by importing Observable from **'rxjs/Obseravle'** package using **Observable.Create()** method.

3. **Observable.Interval()** can be used for infinite event.

4. In general in the subscription method expects 3 methods for:
   - Handle data
   - Handle error
   - Handle completion

5. Destroy Observable by calling unsubscribe method in **ngOnDestroy()**, to prevent memory leak while switching pages.
   reactivex.io site is for Observable details which is very useful to use:
   - object
   - operator

Forms

1. **ngModel** is used to register or indicate angular that form is a type of JavaScript representational control. It is also very helpful for default value setup, property binding and two way binding.

2. **ngSubmit** takes control over the default JavaScript submit button of a form.

3. **ngForm** is to set in a local form control reference to get the access of the form object created by angular.

4. **ngModuleGroup** is used to validate set of form controls as a group.

5. To change values of a form control from code use below:
   - **setValue()**: it updates all values of the form by the JavaScript object created inside the code.
   - **patchValue():** it updates specific item value of the form by the JavaScript object created inside the code.

6. What need to be imported in the app.module.ts:
   - **FormsModule** need to be imported for **Template Driven Approach**.
   - **ReactiveFormsModule** need to be imported for **Reactive Approach**.

7. In the Reactive Approach:
   - **FormGroup** and **FormControl** are used to create form dynamically from the typescript code.
   - **formGroupName, formControlName, formArrayName** are used to connect the input to these type of controls.

8. FormGroup is a JavaScript object which take key-value pair to create form.

9. FormArray holds an array of form controls.

Pipe

1. Pipe transform output into template.

2. Pipe can be parameterized with a colon. E.g. **{{ server-date | date: 'fullDate' }}**

3. While applying multiple pipe, order need to be maintained, because pipe applied **from left to right**.

4. To create custom pipe implement an interface **'PipeTransform'** with method **transform()** to your custom pipe class. For this add the custom pipe class to the app.module under declarations. Also add the pipe decorator with a pipe name before the custom pipe class.

5. By default pipe does not re-calculate filter if data changes. It will have performance issue. But you can enable the feature by adding **'pure: false'** decorative.

HTTP & New HTTP Client

1. For http: post, put, get request all actions need to be done through observable, unless it will not perform the http request.
2. In the response, map can be used to process data in a centralized file.

3. HTTP event types:
   - 0: send event.
   - 1: upload progress event.
   - 3: download progress event.

4. In the put/get request you can add:
   - Observe as an option which can take event or body.
   - Headers as an option which can be used as http header, like in an external api which requires authorization token.
   - Param as an option which can query param like auth.

5. **HttpRequest()** is an another option to create any http call which has a great feature to track the current request progress.

6. Based on the uploaded/downloaded event type you can use loaded/total to show the progress bar in the UI.

7. **Intercept** is a special option where you can implement a common task. You need to do every time when http call is made. E.g. **adding token for authorization.**

Angular Module

1. Typescript and webpack need import. Angular module does not require this.

2. **Declaration, Imports, Providers** are the properties of Angular Module.

3. In the app/root module only you can add **forRoot**, in other module you can add **forChild.**

4. Selector vs Routing in case of module:
   - For the selector you have to declare in the module where you plan to use the selector.
   - For routing, the component you tried to load is declared somewhere, before you try to access this route.

5. Lazy loading enhance the performance of the app. It loads all the components and modules associated with the feature, when the feature route is visited by the user. In lazy loading this routing is done dynamically by using **loadChildren**.

6. Core module only imported by app module.

7. When a module is used in two different module then use it in both imports and exports property of the module.

8. Preload all the lazy loading modules after the app loaded. Use **PreloadAllModules**.

9. Protecting Lazy Loaded Routes with canLoad.

10. What if you want to use route protection (**canActivate** to be precise) on lazily loaded routes:

    You can add canActivate to the lazy loaded routes but that of course means, that you might load code which in the end can't get accessed anyways. It would be better to check that BEFORE loading the code.

11. You can enforce this behaviour by adding the canLoad guard to the route which points to the lazily loaded module:
    { path: 'recipes', loadChildren: './recipes/recipes.module#RecipesModule', canLoad: [AuthGuard] }

NgRx (state management, replacement of services)

1. Use **NgRx** where any state which spans in multiple component.

2. NgRx is the replacement of services (subscribe to an event) to manage application state in a better way.

3. A reducer function will be triggered whenever an action is dispatched. It cannot handle async reply.

4. An action may or may not require payload. Payload is the set of parameters those actually used in a normal function.

5. **...variable_name** is way in typescript to update the old variable value by the new value, which is a **spread operator**.

6. **@Effects** expects to get an observable. Converts promise into an observable.

7. **mergemap** offers to merge multiple observable into one.

8. NgRx select slice or dispatch actions. No need of emitting event and subscribe it.

Angular Universal (Server Side Rendering)

Lazy load will not be pre-rendered in server, it will still be pre-rendered in the client end.

Animations

1. Animation is movement/journey from one state to another state.

2. **Transition** is the main method for animation.

New Features in Angular 4

1. else condition in the html element

2. new Renderer2

3. email validator

4. @angular/animations way of importing changed.

Custom Setup

Set target to es5 to support in all browser.

Typescript Basic

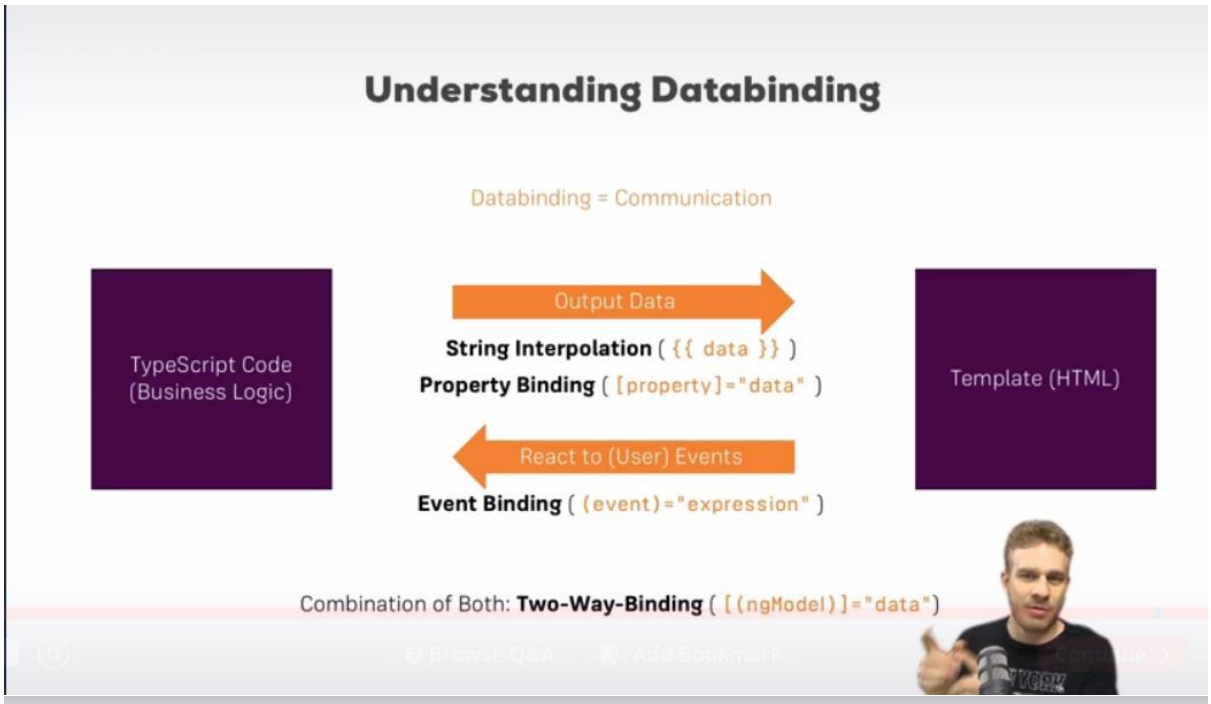
What is TypeScript?

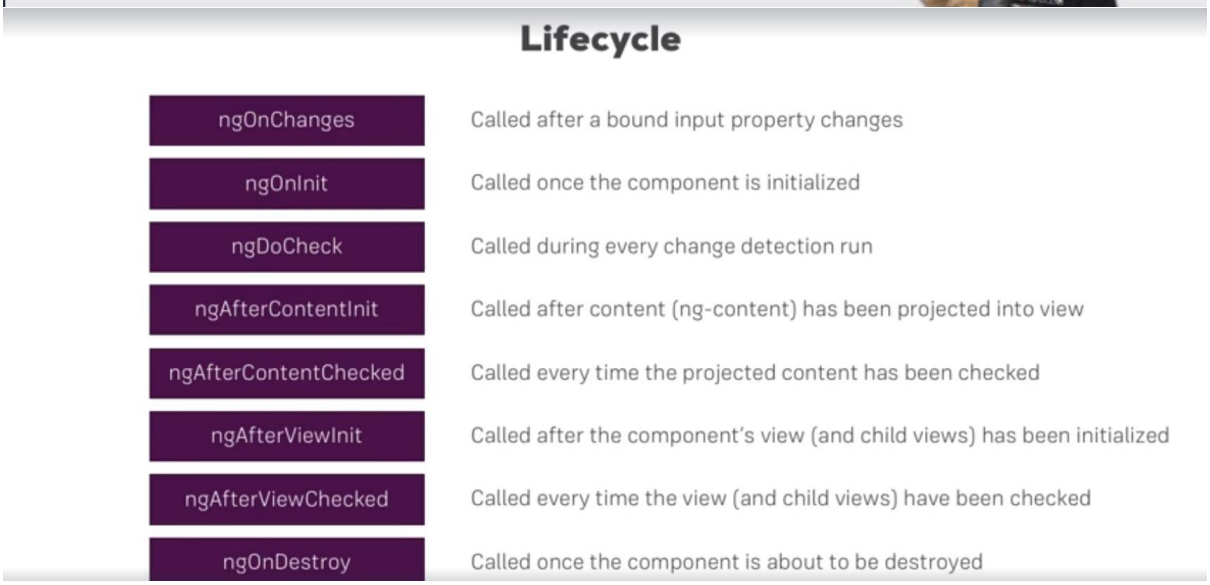
TypeScript is a superset to JavaScript, which means that it compiles into pure JavaScript in the end. Why do we use it then?

- First, it provides 'strong typing' (that's where the name comes from). This means that we can (and should) assign types to our variables and class members. These types won't compile to JavaScript (as JS does not know types) but we will get compilation errors if we assign wrong types or make any other type-related errors. This is a HUGE help in the daily development work and should not be underestimated!

- Second, TypeScript introduces some nice features, JS does not have out of the box (at least in the ES5 specification). This includes classes ('class' keyword), interfaces, generics and modules. Being able to use these constructs makes our code cleaner, easier to read and helps us avoid nasty errors. Especially in combination with the strong typing we are really able to write high quality code and track down errors quickly.

Images


Understanding Databinding
Databinding = Communication
TypeScript Code (Business Logic)
Output Data
String Interpolation ({{ data }})
Property Binding ([property]="data")
React to (User) Events
Event Binding ((event)="expression")
Template (HTML)
Combination of Both: Two-Way-Binding ([(ngModel)]="data")


Lifecycle
ngOnChanges
Called after a bound input property changes
ngOnInit
Called once the component is initialized
ngDoCheck
Called during every change detection run
ngAfterContentInit
Called after content (ng-content) has been projected into view
ngAfterContentChecked
Called every time the projected content has been checked
ngAfterViewInit
Called after the component's view (and child views) has been initialized
ngAfterViewChecked
Called every time the view (and child views) have been checked
ngOnDestroy
Called once the component is about to be destroyed

le Introduction
Lecture 81
vailable

Attribute vs Structural

Attribute Directives	Structural Directives
Look like a normal HTML Attribute (possibly with databinding or event binding)	Look like a normal HTML Attribute but have a leading * (for desugaring)
Only affect/ change the element they are added to	Affect a whole area in the DOM (elements get added/ removed)

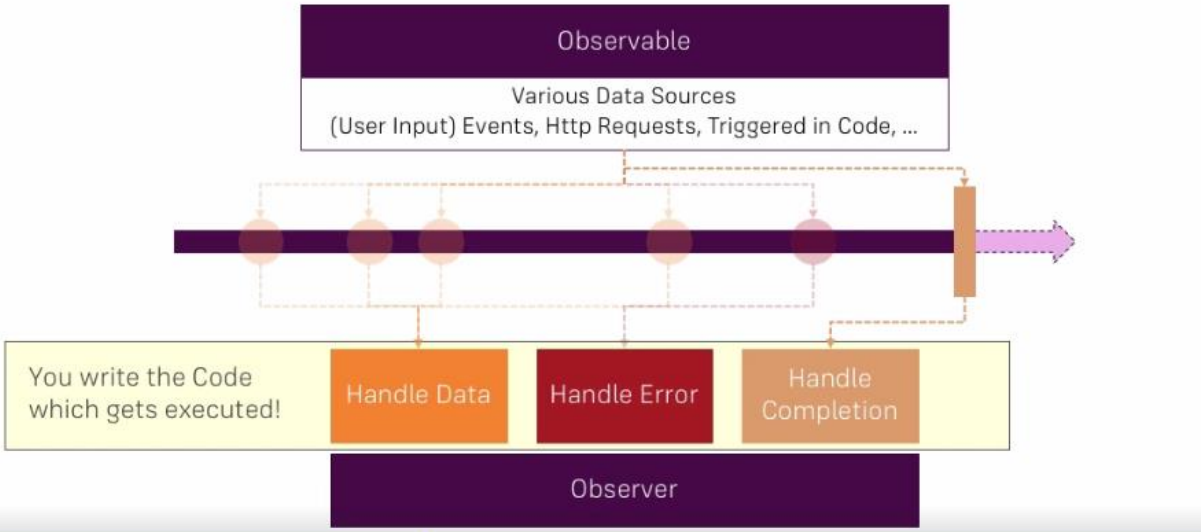
Understanding the Hierarchical Injector
Section 9, Lecture 99

Hierarchical Injector

AppModule	Same Instance of Service is available **Application-wide**
AppComponent	Same Instance of Service is available for **all Components** (but **not for other Services**)
Any other Component	Same Instance of Service is available for **the Component and all its child components**

duction
57

What is an Observable?



Two Approaches

Template-Driven

Angular infers the Form Object from the DOM

Reactive

Form is created programmatically and synchronized with the DOM

What are Pipes?

```
username = 'Max'
```

```
<p>{{ username }}</p>
```

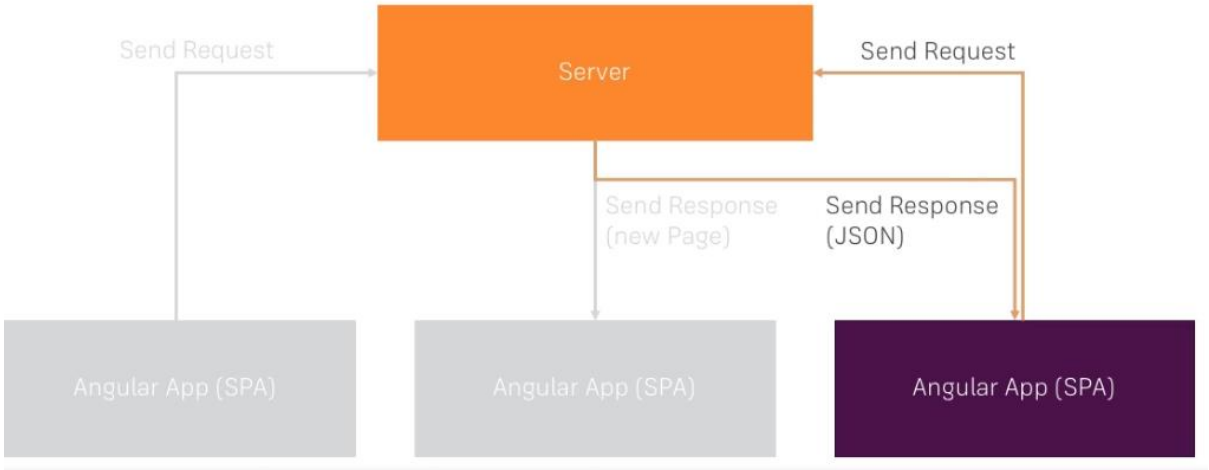
Max

```
<p>{{ username | uppercase }}</p>
```

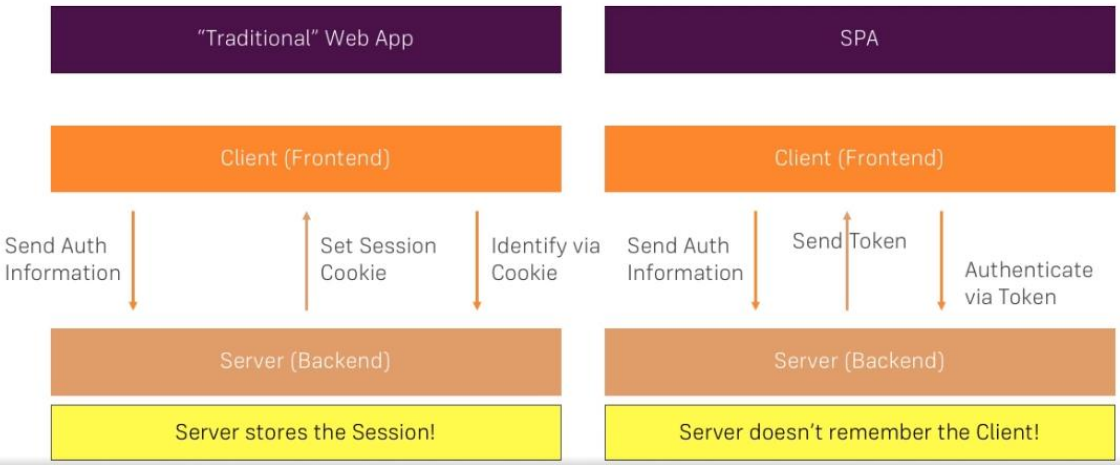
MAX

ture 231

How to access Servers in Angular Apps

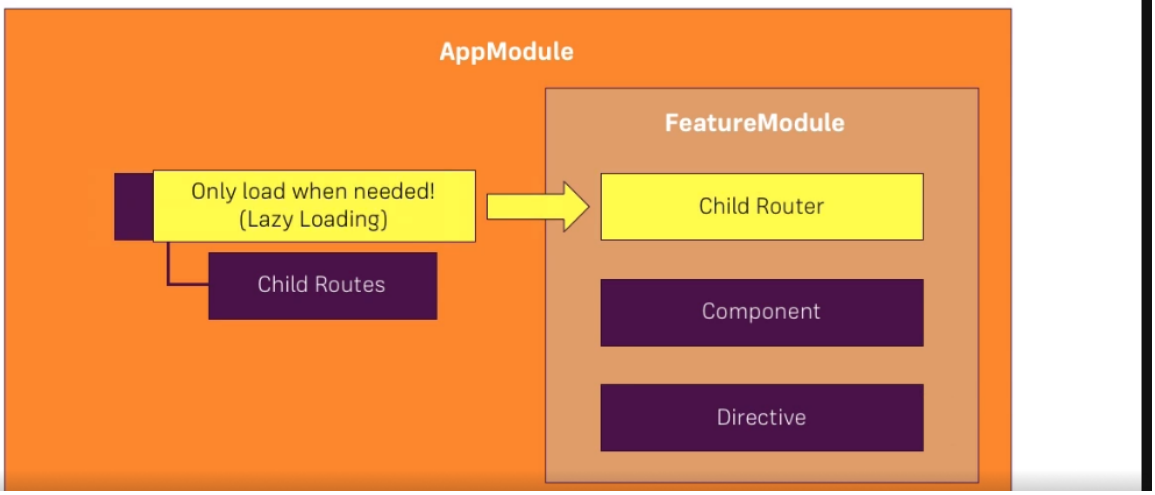


How does Authentication work?

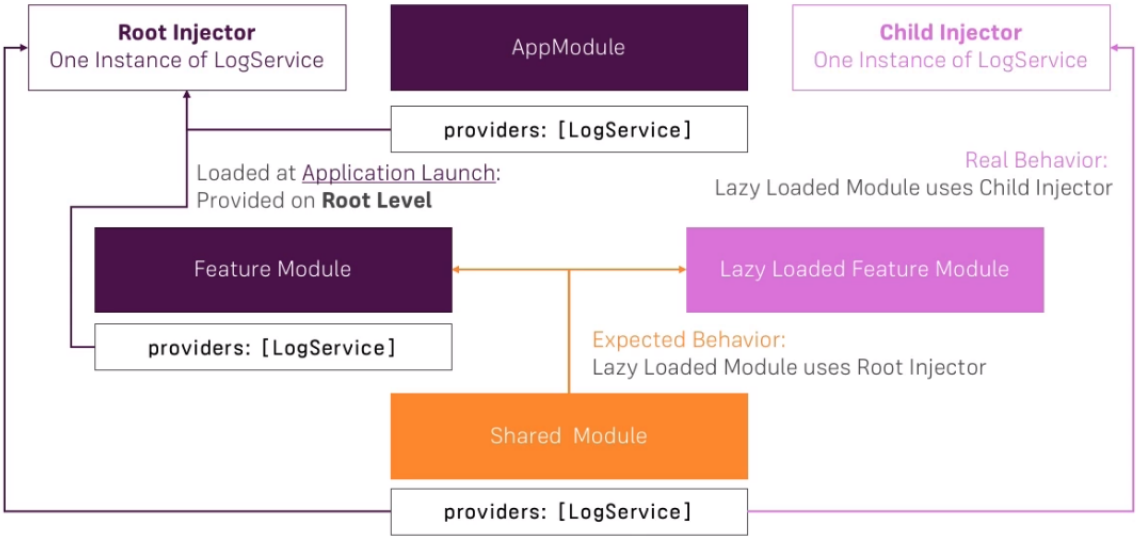


72

Modules and Routing (Lazy Loading)

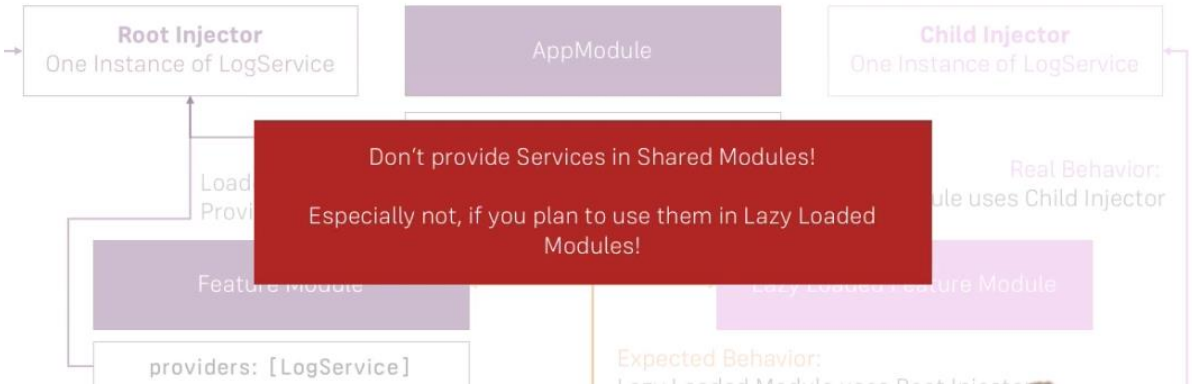


Modules and Service Injection

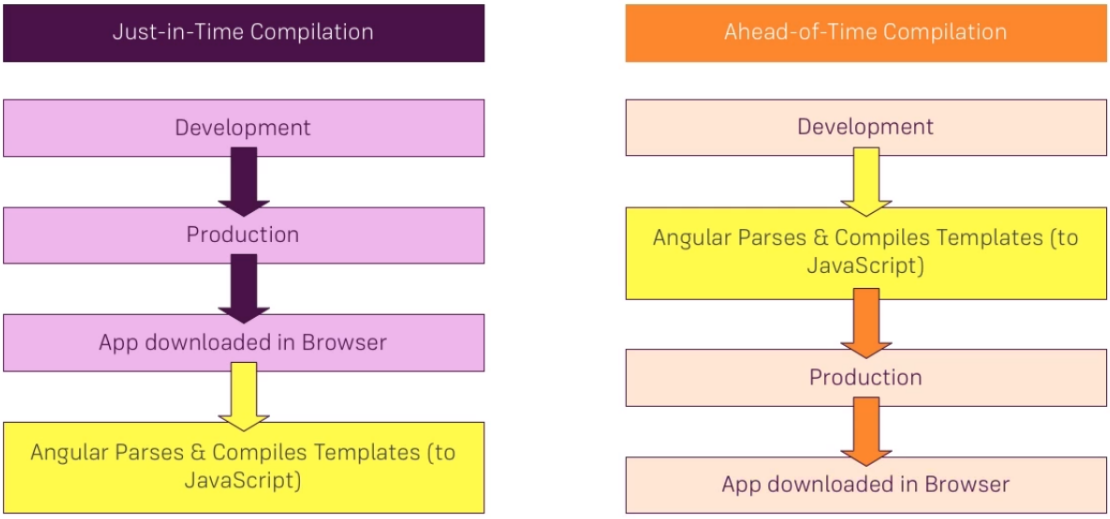


cture 275

Modules and Service Injection



Ahead-of-Time Compilation



Advantages of AoT Compilation

Faster Startup sind Parsing and Compilation doesn't happen in Browser

Templates get checked during Development

Smaller File Size as unused Features can be stripped out and the Compiler itself isn't shipped

Lecture 284

Deployment Steps & Things to Keep in Mind

Build your App for Production

Consider AoT Compilation

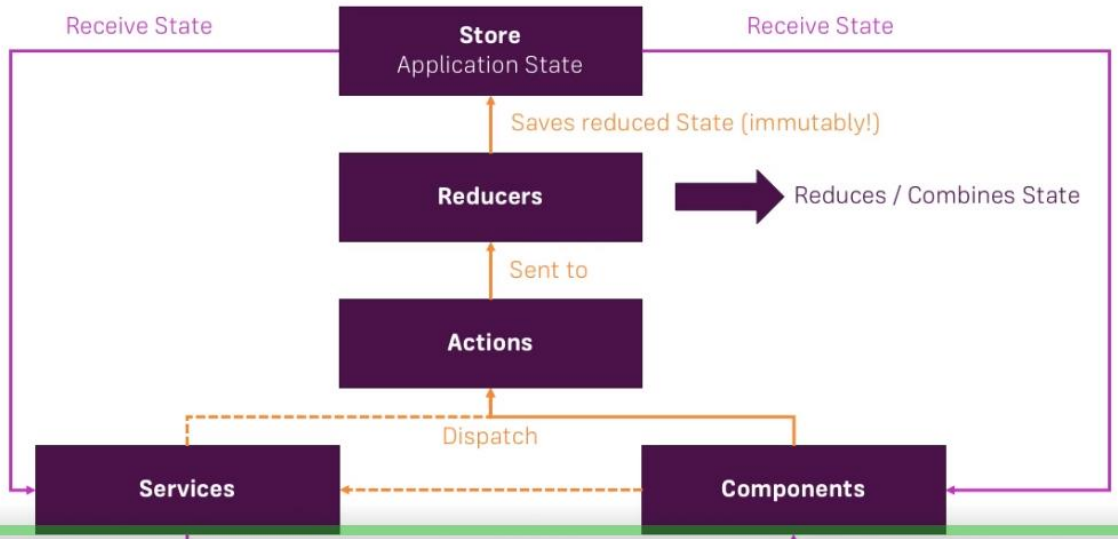
Set the correct <base> element

For example.com/my-app you should have <base href="/my-app/">

Make sure your Server ALWAYS returns index.html

Routes are registered in Angular App, so the server won't know your routes! Return index.html in case of 404 errors!

Redux to the Rescue



Why Unit Tests?

Guard against Breaking Changes

Analyze Code Behavior (Expected and Unexpected)

Reveal Design Mistakes