

FULL STACK

Working with Angular application



You Already Know

Course:

Angular



Recap

- Build Angular components
 - Angular CLI
 - Nested components
 - Lifecycle of Angular components
- Understand Bootstrap
 - Creating a responsive web application
- Explain binding and events
 - Template model
 - Built-in directives
 - Basics of webpack and SystemJS



Recap

- Explain dependency injection and services
 - Dependency injection API
 - Creating a service
- Work with directives, pipes, and forms
 - Working with Angular directives
 - Working with custom pipes
 - Working with Angular forms
- Test an Angular app
 - Testing Angular class
 - Testing service and DOM



A Day in the Life of a MEAN Stack Developer

In this sprint, Joe has to develop the front-end of the application using an Angular framework, which includes signup and login functionalities. He has been chosen to lead this project and complete the initial part of the application.

He agrees to complete the task as early as possible, as the release of the application is in the next week.

In this lesson, we will learn how to solve this real-world scenario to help Joe complete his task effectively and quickly.



Learning Objectives

By the end of this lesson, you will be able to:

- 🕒 List the importance of an Angular framework
- 🕒 Build an Angular application with customized requirements
- 🕒 Create responsive forms for user input and validation
- 🕒 Implement routing mechanism in your Angular application
- 🕒 Upgrade an AngularJS project to Angular 2+



FULL STACK

Upgrading from AngularJS

Introduction to AngularJS

- AngularJS was created by Misko Hevery.
- The first version of AngularJS was launched in the year 2009.
- Features of AngularJS:
 - It is a JavaScript MVW framework.
 - It supports HTML extension by tags, attributes, and expressions.
 - It supports Event Handling.
 - It also supports Data Binding.
 - It has a rich library of packages that support and implement built-in templates and Routing.
 - It has form validations and animations.
 - It supports reactive and responsive web applications.



AngularJS vs Angular

AngularJS



- Uses JavaScript as base language to build single-page application
- No longer used for development
- MVC-based architecture
- Doesn't support mobile compatibility
- ES5, ES6, and Dart-based coding
- Developed on the controllers, which aren't used anymore
- Client-side development
- ng-app and angular bootstrap function are used to initialize

Angular

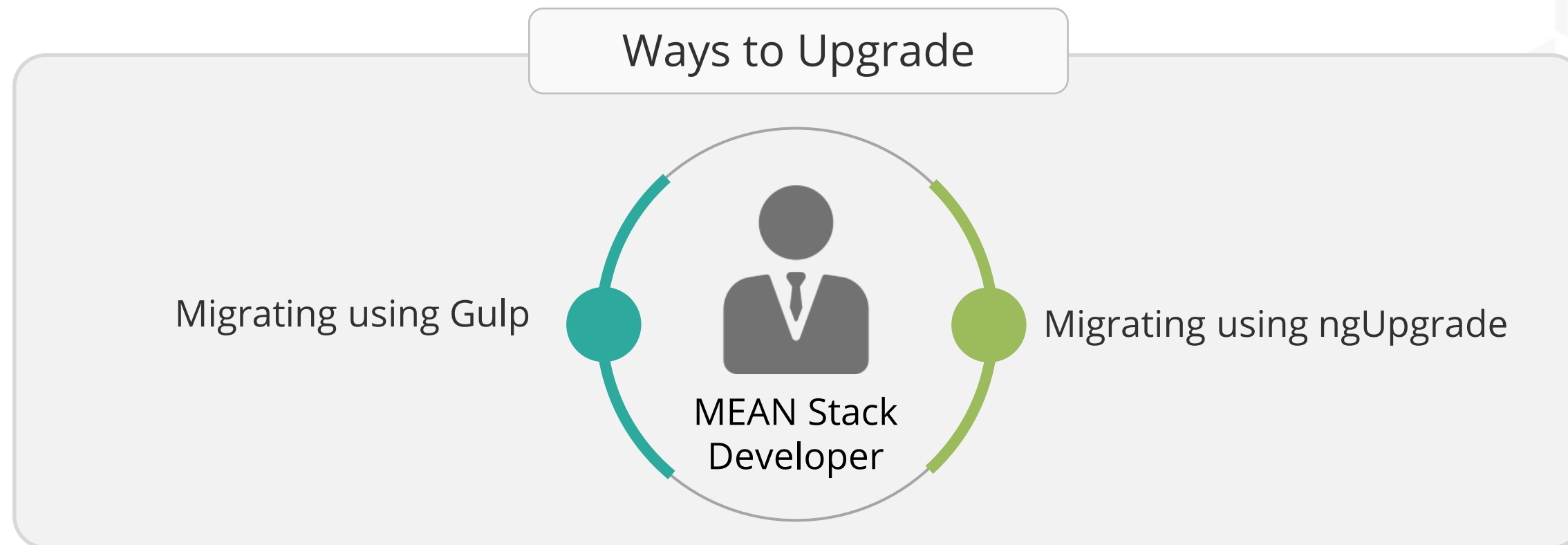


- Complete rewrite of AngularJS version
- Updated versions are regularly released because of Semantic Versioning
- Component-based development
- Based on service/controller architecture
- Mobile-oriented framework
- ES5, ES6, and Typescript are used to write code
- The class is the only method to define services
- Runs on client-side and server-side

Migrating from AngularJS

Upgrading or migrating from AngularJS to Angular is a difficult task.

- It is important to upgrade from your current Angular to the latest version keeping the business needs in mind.



Migrate Using ngUpgrade

Ingredients to keep ready before migration begins:



The diagram features a vertical stack of five colored rectangular boxes on the left, each representing an ingredient. From top to bottom, the colors are dark grey, blue, red, orange, and teal. To the right of each box is a light-colored rectangular area containing descriptive text. A thin vertical line runs to the left of the boxes, and a circular outline is positioned at the top left of the stack.

Code

Make sure your current application is feature-oriented and has one feature per file

Typescript

It is important to install and set up typescript

Modular

Webpack is widely used

Angular 2+

All the controllers are replaced with components in Angular 2+ versions

Set up ngUpgrade

Setting up ngUpgrade helps run both apps alongside each other and necessary changes can be made in your application

FULL STACK

Bootstrapping Angular application

Bootstrapping Angular application



Duration: 30 min.

Problem Statement:

Create a responsive navbar using bootstrap and angular

ASSISTED PRACTICE

Assisted Practice: Guidelines to Demonstrate Bootstrap Angular application

1. Configure the Angular application and install bootstrap.
2. Create a header component.
3. Create a navigation bar using bootstrap classes.
4. Push the code to the GitHub repositories.



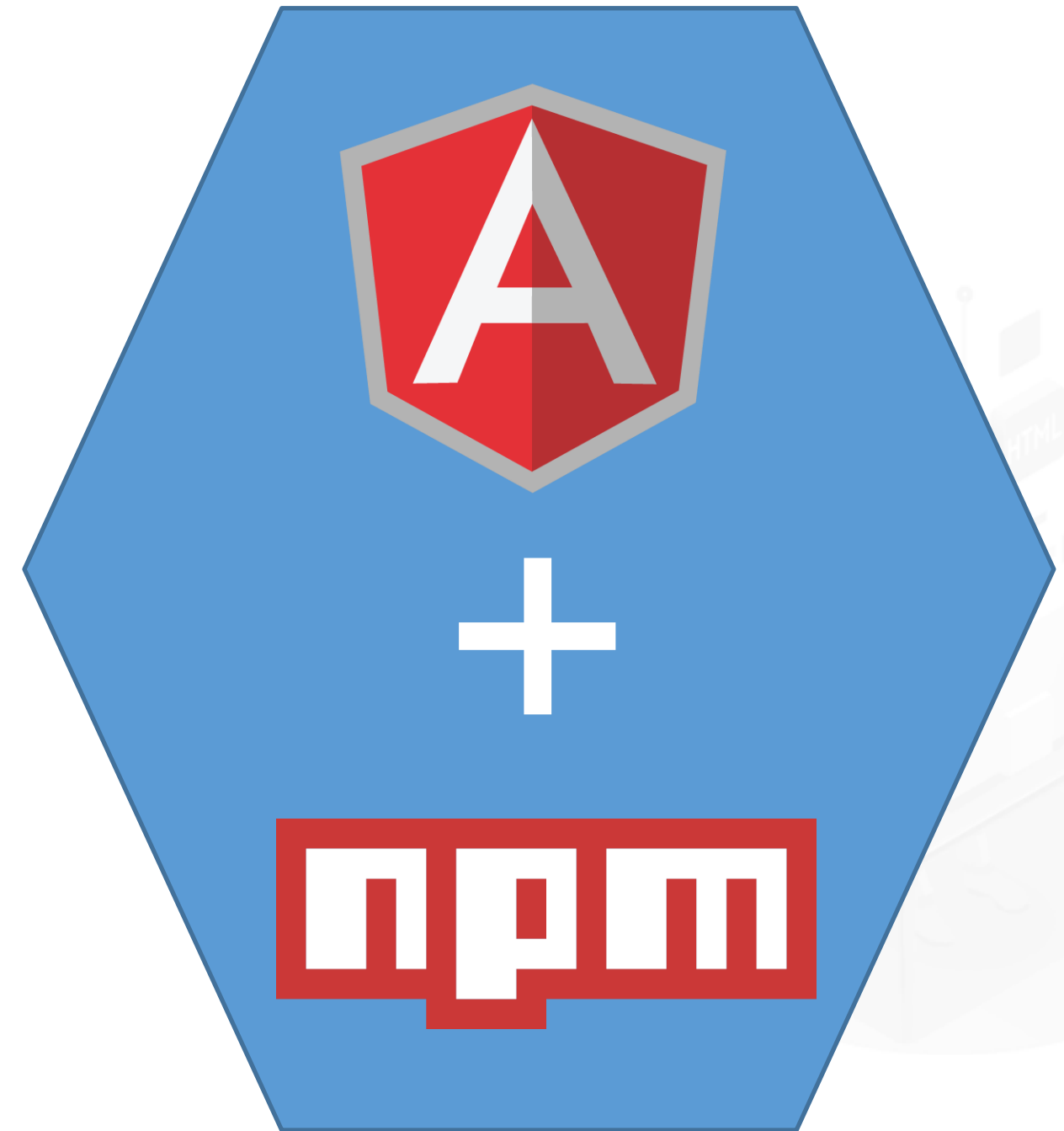
FULL STACK

Creating Libraries

Creating Libraries

To create general solution for a domain, developers can build solutions using Angular libraries, which can be shared and published with **npm** packages.

- An Angular library is an Angular project that differs from an app in that it cannot run on its own. A library must be imported and used in an app.
- Libraries are majorly used to extend the functionality of basic Angular project operations. For example, **ng add @angular/forms** adds premade reactive forms to your application instead of coding them line-by-line.
- To integrate reusable libraries into the application, one has to install the **npm** packages and import the relative classes or functionality wherever required.



Creating Libraries

One can create libraries to minimize the coding effort and optimize the code easily.



Custom libraries can be both application-specific or general in nature to facilitate other angular applications as well.

It helps developers understand the code and rework on the application when required.



Creating Libraries



Refactoring parts of a library: Components and pipes designed in library should be stateless. Also make use of **@inputs** and **@outputs** to expose the interactions between components. Services should declare their own providers. Moreover, check all the internal dependencies and **RouterModule**.



Reusable codes and schematics: Custom library can include elements like components, services, and other artifacts. It also includes the schematic information on the steps to incorporate the library directly to the code, which can be a **CLI** command that pulls the library and installs it into your application.



Publishing and Linking libraries: Includes the schematics to define how to publish your library as an **npm** package. Make use of **npm link** to avoid reinstalling the library after every build.

Creating Libraries



Duration: 30 min.

Problem Statement:

Create your own angular library and import it in your application.

ASSISTED PRACTICE

Assisted Practice: Guidelines to Create Libraries

1. Install Node.js.
1. Install Visual Studio Code.
1. Install Angular CLI.
1. Create a new Angular project.
1. Create a custom library.
1. Create a component in the library.
1. Export the library and inline component to the main application.
1. Push the code to GitHub repositories.



FULL STACK

Data Interactions

Property Binding

Every DOM property can be written via special attributes on HTML elements using square brackets []. An HTML attribute can start with anything.

Angular maintains the properties and attributes in sync when you use them, so if you are familiar with Angular 1.x directives, such as ng-hide, you can work directly with the hidden property and ng-hide is not required.

```
<div ng-hide="isHidden">Hidden element or not?</div>
```

The most common property binding sets an element property to a component property value.

```
>_ training@localhost:~
```

```
<img [src]="heroImageUrl">
```

```
<button [disabled]="isUnchanged">Cancel is disabled</button>
```

Event Binding

The binding directives you've met so far allow data to flow in one direction: from a component to an element.

Users don't just stare at the screen. They enter text into input boxes. They pick items from lists. They click buttons. Such user actions may result in a flow of data in the opposite direction: from an element to a component.

The only way to know about a user action is to listen for certain events, such as keystrokes, mouse movements, clicks, and touches. You declare your interest in user actions through Angular event binding.

Event binding syntax consists of a target event name within parentheses on the left of an equal sign and a quoted template statement on the right.

```
<button (click)="onSave()">Save</button>
```

Two-Way Binding

You often want to both display a data property and update that property when the user makes changes.

On the element side, it takes a combination of setting a specific element property and listening for an element change event.

Angular offers a special two-way data binding syntax for this purpose, [(x)]. The [(x)] syntax combines the brackets of property binding, [x], with the parentheses of event binding, (x).

The two-way binding syntax is just syntactic sugar for property binding and event binding. Angular desugars the SizeComponent binding into this:

```
<my-sizer [size]="fontSizePx" (sizeChange)="fontSizePx=$event"></my-sizer>
```


Data Interactions



Duration: 40 min.

Problem Statement:

Create an Angular project to demonstrate two-way binding.

ASSISTED PRACTICE

Assisted Practice: Guidelines to Demonstrate Data Interactions

1. Configure the Angular application.
2. Responding to user inputs and clicks.
3. Add bootstrap to the Angular application.
4. Push the code to GitHub repositories.



FULL STACK

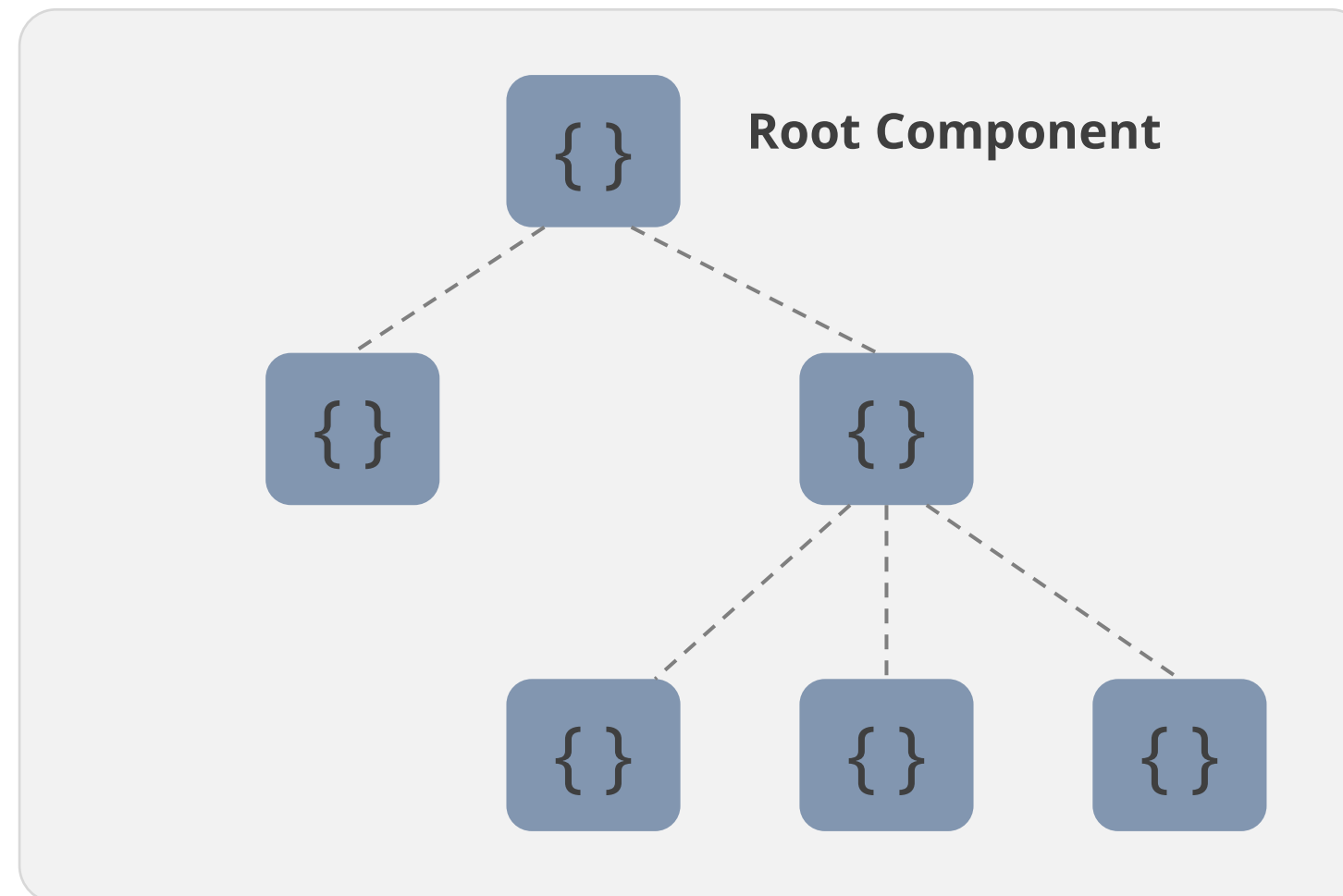
Nesting Components

Components

{ } Components in an Angular application encapsulate the template, data, and behavior of view.

Components are also known as View Components.

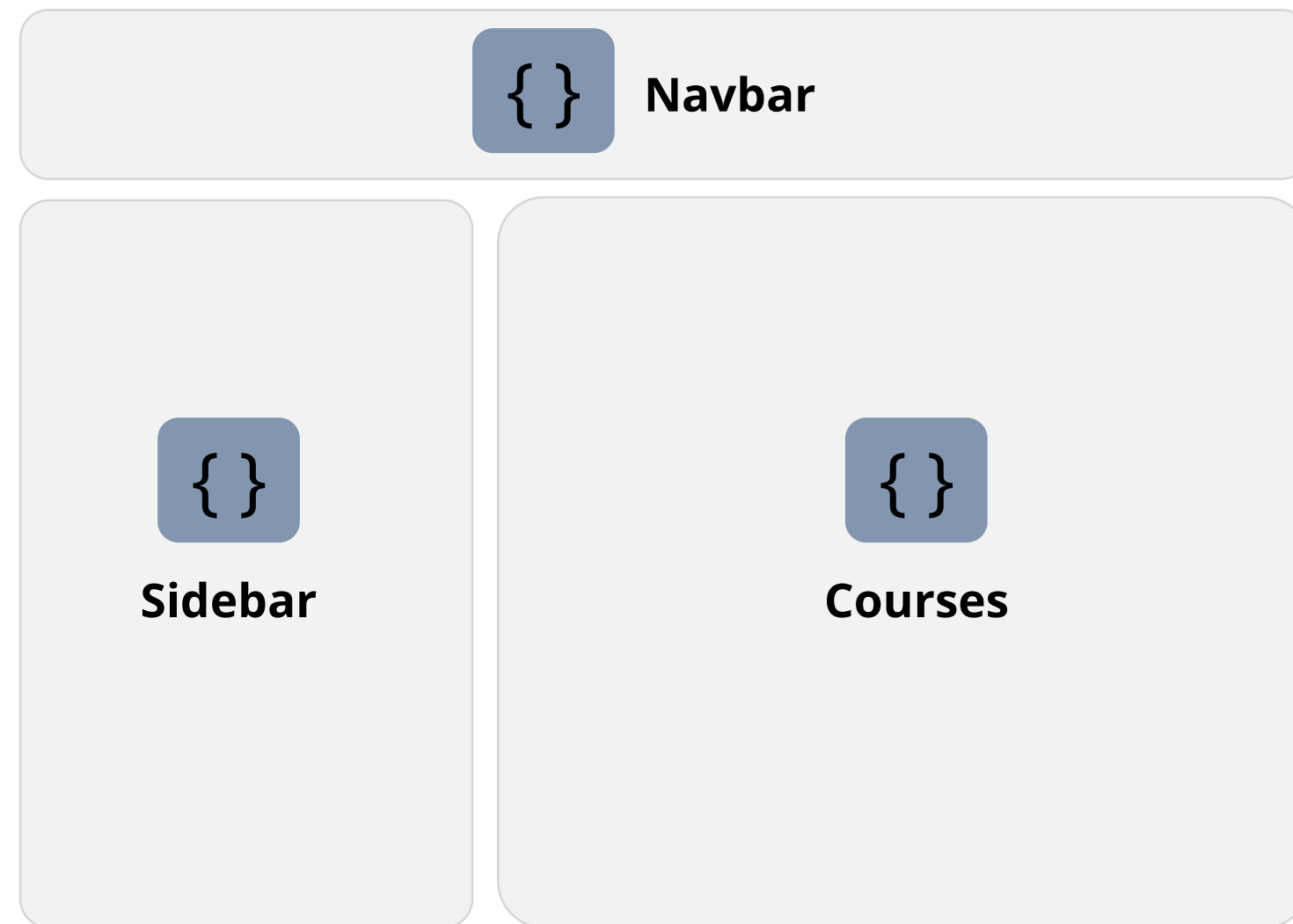
Every app has at least one component called the Root Component.



Components

However, in real world an application encapsulates many components.

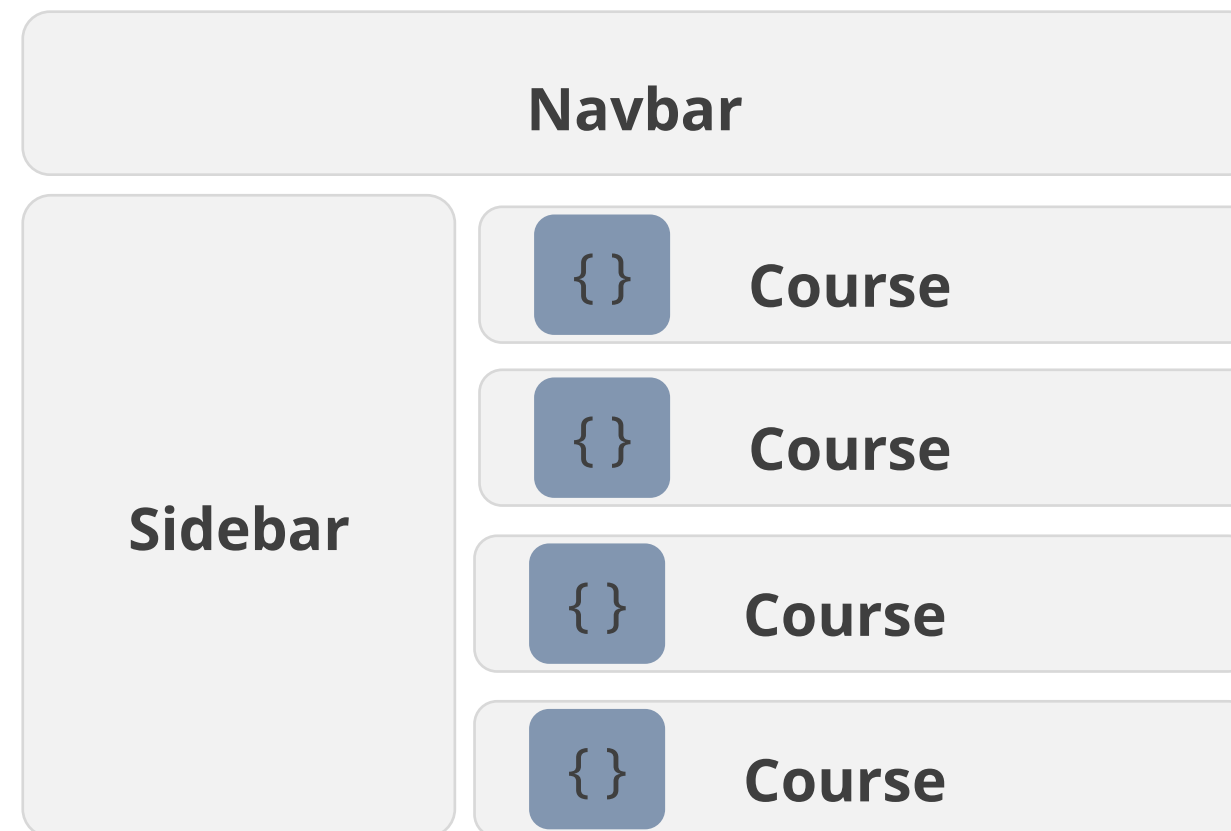
Here is an example:



Nested Components

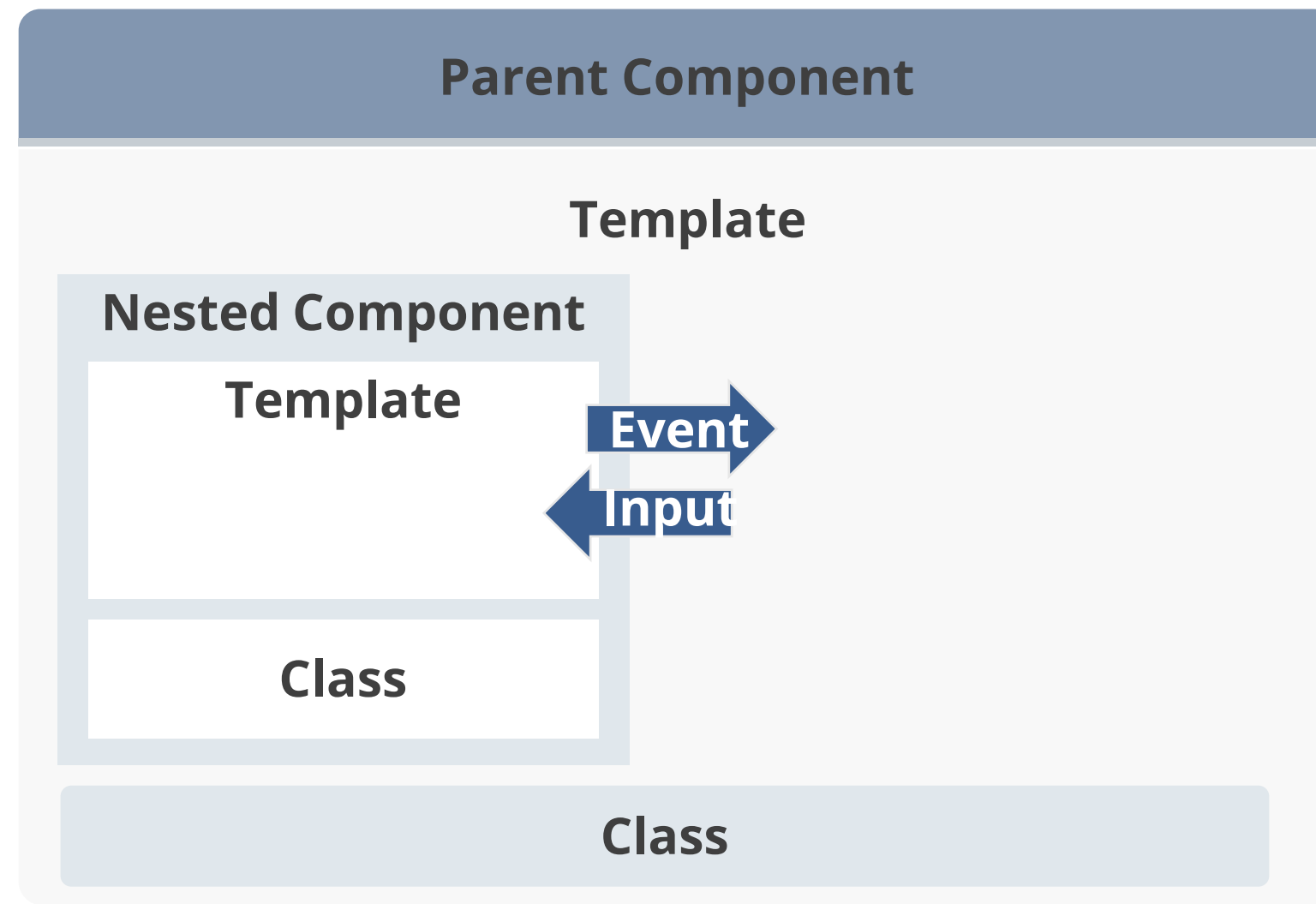
- Each component has a template for the view and data or logic behind the view.
- Components can also contain other components called nested components.

In the diagram, courses are components.



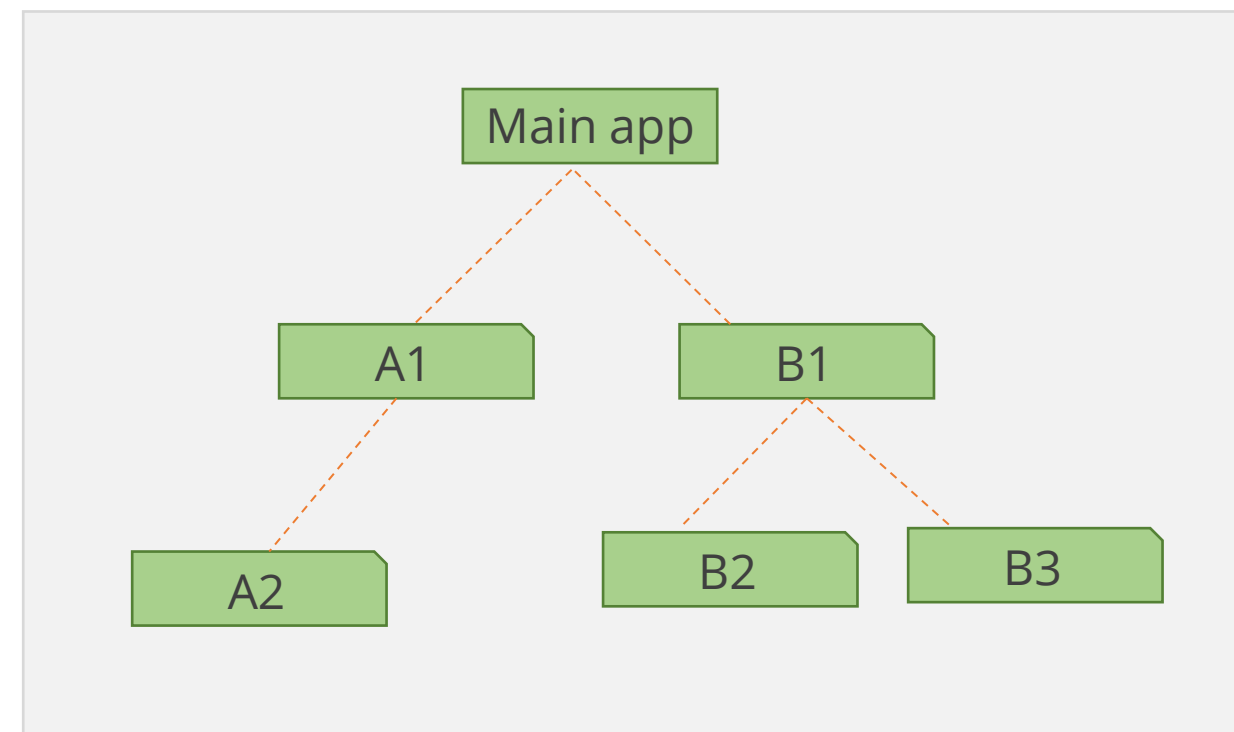
Deeper Nesting

The diagram shows the communication between deeply nested components in Angular.



Deeper Nesting

As you write a complex app and reuse more components, the components get smaller and more nested as part of the process. Hence, it is important to understand how nested components work.



Angular Component Lifecycle

constructor

ngOnChanges

ngOnInit

ngDoCheck

ngAfterContentInit

ngAfterContentChecked

ngAfterViewInit

ngAfterViewChecked

ngOnDestroy

A component has a lifecycle managed by Angular.

- Angular creates the component and renders it, creates and renders its children, checks when its data-bound properties change, and destroys it before removing it from the DOM.
- Angular offers lifecycle hooks that provide visibility to the key lifecycle moments of the component. It also provides the ability to act when these moments occur.
- A directive has the same set of lifecycle hooks, excluding the hooks that are specific to component content and views.

FULL STACK

Component-Level Interactions

Component-Level Interactions

- Component-level interactions can be done between two components of similar hierarchy or between a parent and a child.
- Data sharing between Angular components:
 1. Parent to child: via Input
 2. Child to parent: via Output() and EventEmitter
 3. Child to parent: via ViewChild
 4. Unrelated components: via a Service
- This is an important concept that one should practice as it facilitates large amount of Angular functionalities.



@Input and @Output

@Input and @Output are mechanisms to receive and send data from one component to another.

```
@Component({
  Selector: 'app-items'
  ....
})

export class addTask{

  @Input() item_name
  @Output() onNameChanger = new
    EventEmitter()

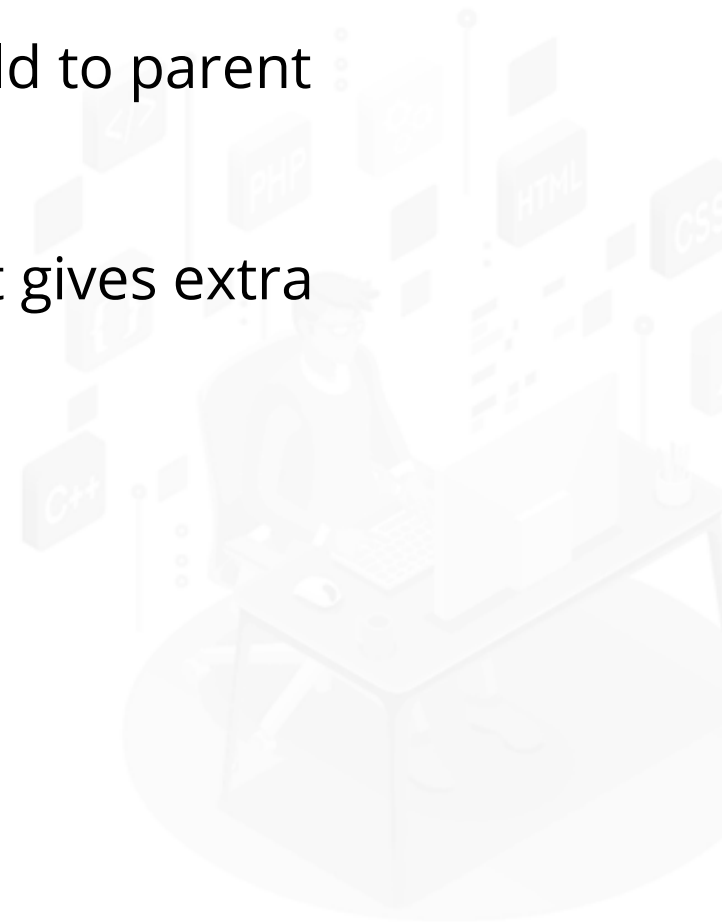
}
```

Example:

- The **@Input** decorator indicates that data will be received from a component. The received data will be stored in **item_name**.
- The **@Output** decorator indicates that data will be sent to another component via **onNameChanger** property.

EventEmitter and ViewChild

- EventEmitter is something which has the capability to propagate the event from child component to parent component
- EventEmitter and @Output complement each other to navigate the requests from child to parent component.
- ViewChild is needed mainly for transferring data from child to parent component as it gives extra controls for parent component to access child events.
- Use the below code to import ViewChild:
import { Component, OnInit, ViewChild } from '@angular/core';
- Supply the child component as view child using below codes:
@ViewChild(EcchildComponent)
private counterComponent: EcchildComponent;



Component Level Interaction



Duration: 40 min.

Problem Statement:

Create an Angular project to transfer data between parent and child components.

ASSISTED PRACTICE

Assisted Practice: Guidelines to Demonstrate Component Level Interaction

1. Create Angular app.
1. Create parent and child components.
1. Define @input and @output operations.
1. Push the code to GitHub repositories.

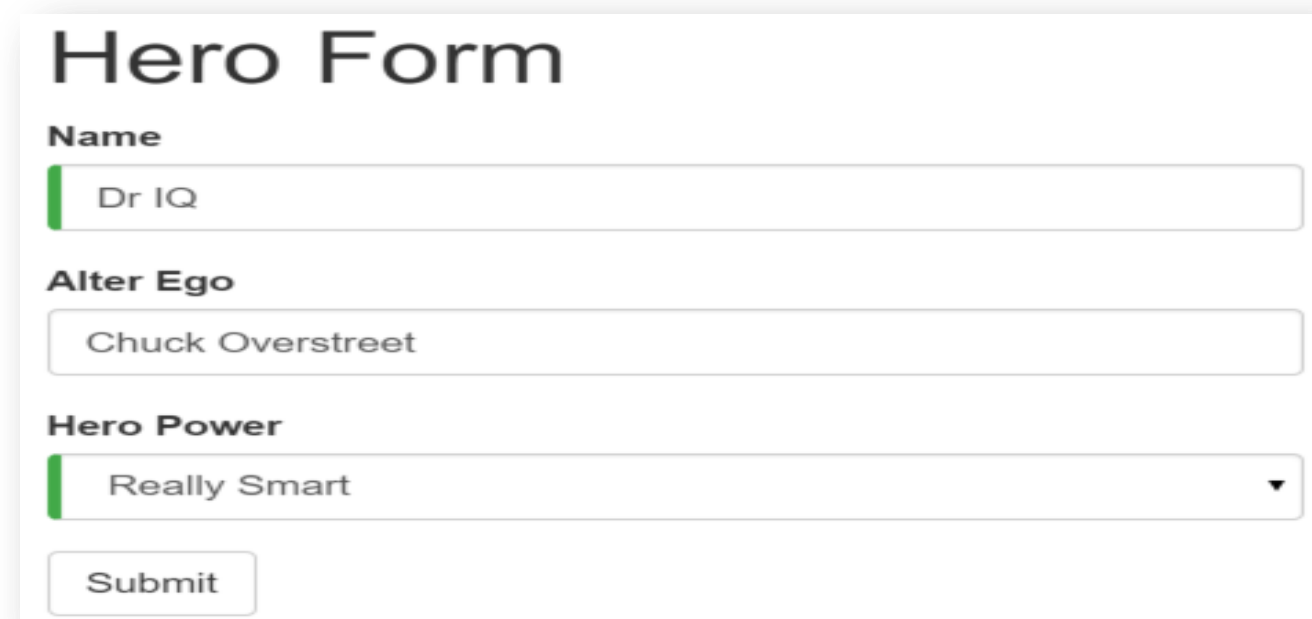


FULL STACK

Forms and Validations

Angular Form Benefits

A form creates a cohesive, effective, and compelling data entry experience.
An Angular form coordinates a set of data-bound user controls, tracks changes, validates input, and presents errors.



Hero Form

Name

Alter Ego

Hero Power

Forms are probably the most crucial aspect of your web application.
You can often get events by clicking on links or from the movement of the mouse, but it's through forms that you get the majority of your rich data input from users.



Angular Form Tools

Angular has these tools to help with the challenges:

- **Form Controls**

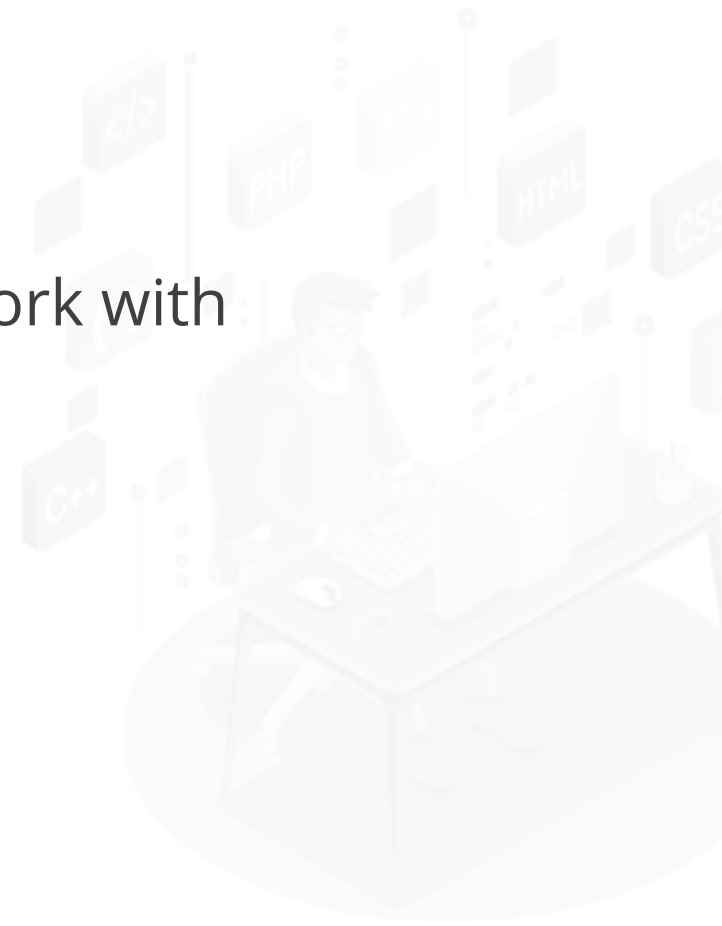
Encapsulate the inputs in your forms and give objects to them to work with

- **Validators**

Give you the ability to validate inputs any way you like

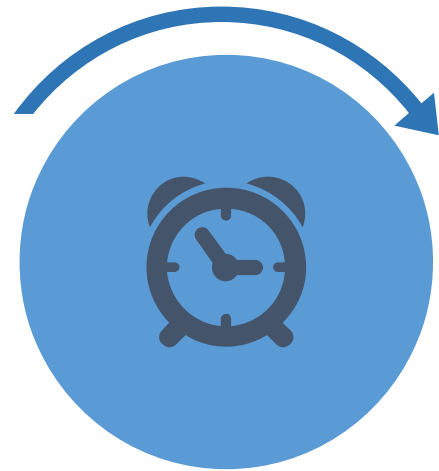
- **Observers**

Allow you to watch your form for changes and respond accordingly

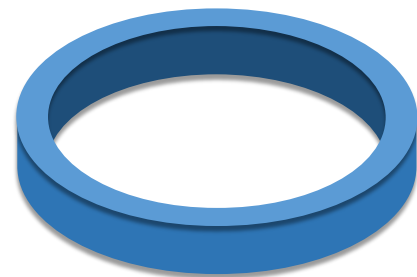


Types of Angular Forms

Reactive and template-driven forms process and manage form data differently. Each offers different advantages.



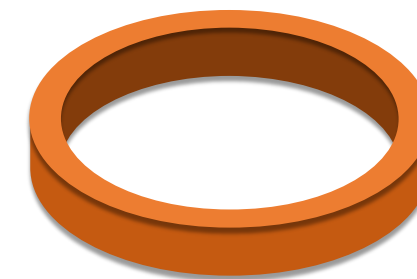
Reactive
Forms



Reactive forms are more robust, scalable, reusable, and testable



Template-driven
forms



Template-driven forms are used for adding basic forms in your app, such as a signup form

Template-Driven Approach

Angular uses built-in directives to build forms such as **ngModel**, **ngModelGroup**, and **ngForm** available in **FormsModule** module.

With template-driven forms, you can essentially leave a component class empty until you need to read or write values (such as submitting and setting initial or future data).

>_ training@localhost:~

```
Import {component} from '@angular/core' 1;

@Component ({
  selector: 'signup - form',
  template: `
    <form novalidate>...</form>
  `
})
export class signupFormComponent {
  constructor()_ {}
}
```

Model-Driven Approach

A lot of features and concerns were improved in Angular, such as communication between components, code reuse, separation of concerns, and unit tests.

To promote some of those benefits in the context of forms, Angular uses a model-driven or reactive technique of development.

Model-Driven Approach: Front-End

```
training@localhost:~  
  
<form [ngFormModel]='form' (ngSubmit)="onSubmit() ">  
  <div class="form-group" [ngclass]="{'has-error': !name.valid}">  
    <label for="name">Name</label>  
    <input type="text" class="form-control" id="name"  
      ngcontrol='name'  
      #name="ngForm">  
    <p *ngIf="!name.valid" class="help-block">  
      Name is required  
    </p>  
  </div>  
  <button type="submit" class="btn btn-primary"  
    [disabled]="!form.valid">Add user</button>  
</form>
```

Model-Driven Approach: Component

```
training@localhost:~  
  
import {Component} from 'angular2/core';  
import {Control, ControlGroup, Validators} from 'angular2/common';  
  
@Component({  
  selector: 'form',  
  templateUrl: './dev/shared/form.component.html'  
})  
Export class FormComponent {  
  form = new ControlGroup({  
    name: new Control('', Validators.required);  
  });  
  
  onSubmit() {  
    console.log(this.form.value);  
  }  
}
```



Angular Validation

Angular comes with two types of validation:

Built-in form validation

Custom form validation



Built-In Form Validation

Angular provides three out-of-the-box validators that can either be applied using the control class or using HTML properties.

For example: `<input required>` will automatically apply the required validator

- Required
- minLength
- maxLength

apply them to the Control using the second parameter.

```
this.name = new Control('', Validators.minLength(4));
```

Custom Form Validation

You can also write your own custom validators. Here is an example of a validator that checks if the first character is not a number:

```
Interface ValidationResult {  
  [key:string] :boolean;  
}  
  
class UsernameValidator {  
  static startWithNumber (control: Control) :ValidationResult {  
    if ( control.value ! "" && !isNaN  
(control.value!.charAt(0)) ) {  
    }  
    return null;  
  
  }  
}
```

Reactive Form Validation



Duration: 60 min.

Problem Statement:

You are given a project to create a reactive-driven form with appropriate validators.

ASSISTED PRACTICE

Assisted Practice: Guidelines to Demonstrate Form Validations

1. Configure the Angular application.
2. Build a front-end form for sign-up.
3. Complete the functionality by implementing the validators.
4. Push the code to GitHub repositories.

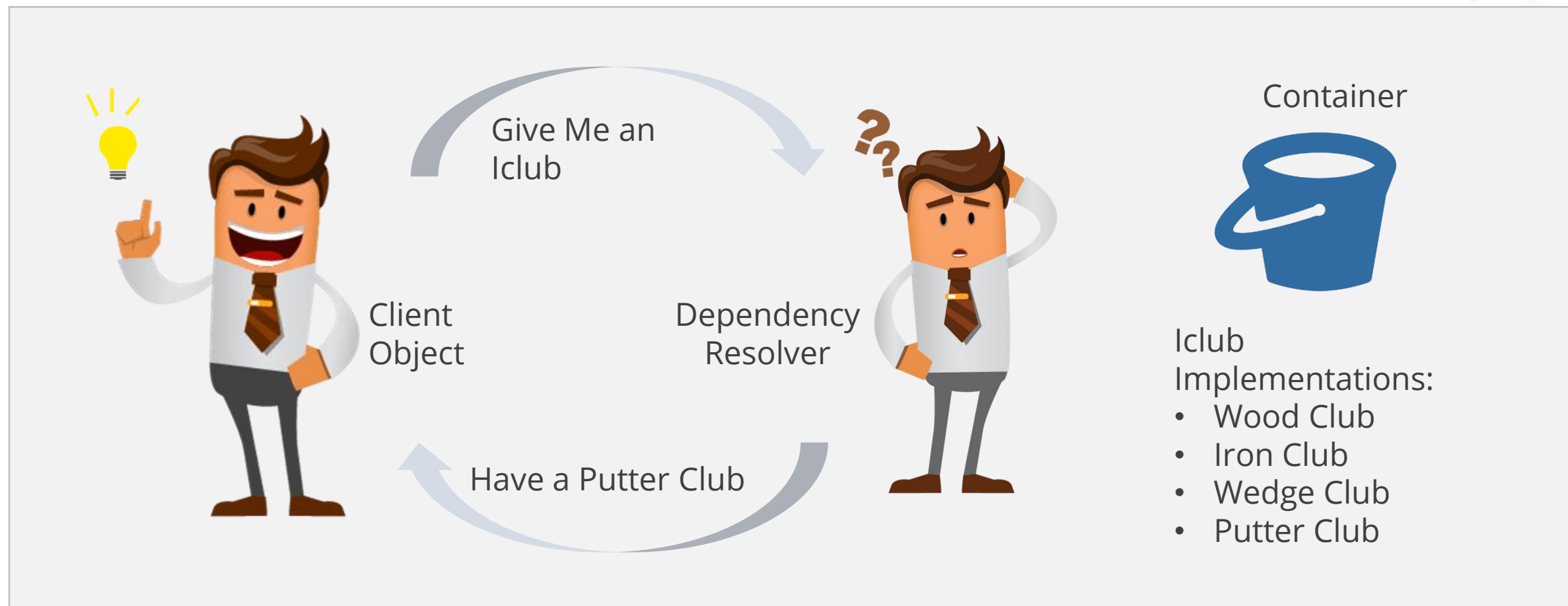


FULL STACK

Services and Injectables

Features of Dependency Injection

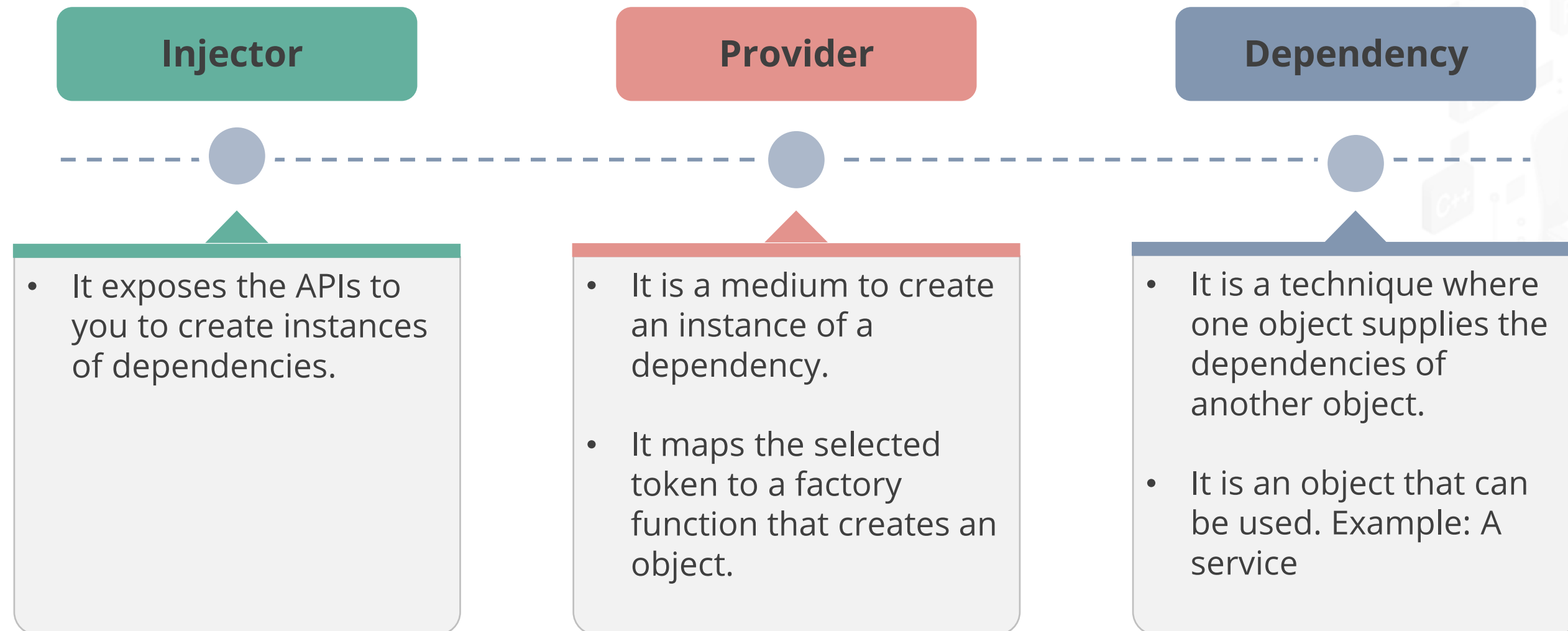
- An injector provides a mechanism to create a service instance, and to create the instance, it needs a provider.
- Provider is a medium for creating a service.
- Providers can be registered along with the injectors.



Importance of Dependency Injection API

Angular has its own dependency injection framework that can also be used as a standalone module by other applications and frameworks.

DI in Angular consists of:



Creation of Service

Import the Angular Injectable function and apply that function as an @Injectable() decorator.

A terminal window with a title bar showing 'training@localhost:~'. The window contains the following TypeScript code:

```
>_ training@localhost:~  
  
import { Injectable } from '@angular/core';  
  
@Injectable()  
export class HeroService {  
}
```

TypeScript looks at the @Injectable() decorator and emits metadata about the service.

Metadata in Angular may need to inject other dependencies into this service.

Injecting a Service

Step 1: Import the service you want to use so that you can refer to it in the code.

> training@localhost:~

```
import { HeroService } from './hero.service';
```

Injecting a Service

Step 2: Inject the HeroService.

Note: You can create a new instance of the HeroService with *new*:

```
training@localhost:~  
  
heroService = new HeroService(); // don't do this
```

The one line of code (with new) is replaced with two lines:

- Add a constructor that also defines a private property.
- Add to the component provider's metadata.

```
training@localhost:~  
  
constructor(private heroService: HeroService) { }
```

Injecting a Service

Step 3: Add the **providers** array property to the bottom of the component metadata in the @Component call to teach the injector the process to make a HeroService.

A terminal window with a light gray title bar containing a prompt icon and the text 'training@localhost:~'. The main area is dark gray with white text. The text 'providers: [HeroService]' is shown, indicating the addition of the HeroService to the providers array in the component metadata.

```
>_ training@localhost:~  
  
providers: [HeroService]
```

FULL STACK

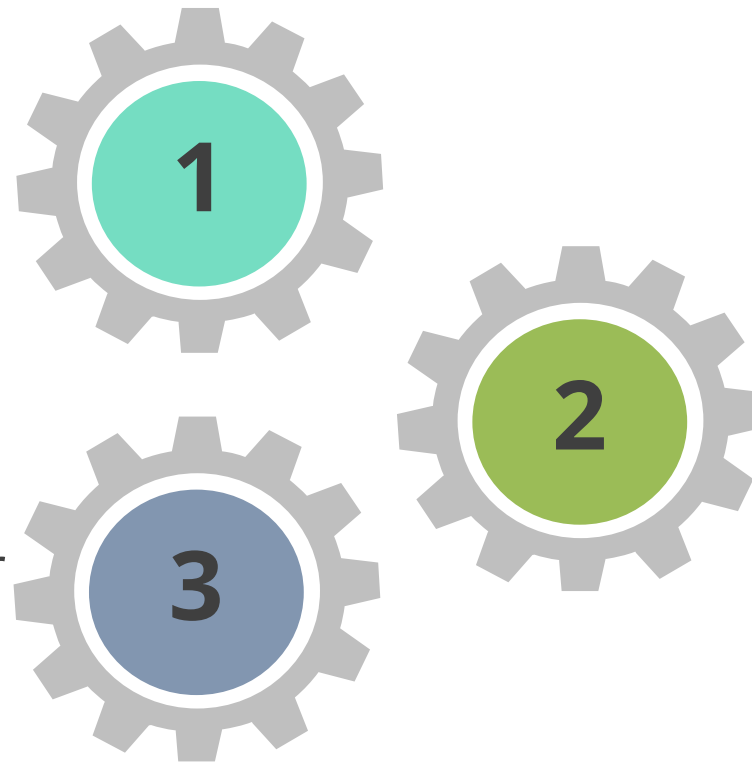
Routing Mechanisms

Router Concepts

Main components used to configure routing:

Routers define the routes that your application supports

routerLink directive is used for navigation



router-outlet is where the page content goes



Defining Routes

Route is a mapping file where you can map components to their target URL:

```
training@localhost:~  
  
Const routes = [  
  {path: '', redirectTo: 'home'}, //Index router  
  {path: 'home', component: HomeComponent},  
  {path: 'guards', component: GuardsComponent},  
  {path: 'resolver', component: ResolverComponent},  
  {path: 'preview', component: PreviewComponent},  
  {path: '**', component: E404Component //Fallback router  
];
```



Registering Root Router

This is where the Root Router gets registered:

```
>_ training@localhost:~  
  
import {RouterModule} from '@angular/router';  
Const routes = [  
.....  
];  
@NgModule({  
.....  
  imports: [  
    BrowserModule,  
    RouterModule.forRoot(routes)  
  ],  
  bootstrap: [AppComponent]  
})  
Export class AppModule{  
}
```



Navigating to a Link

routerLink Directive:

- routerLink directive is used to bind a clickable HTML element to a route.
- Click on an element with a routerLink directive that is bound to a string or a link array which triggers a navigation.

```
training@localhost:~  
  
<a [routerLink]="[ '/home' ]">Home</a>  
<a [routerLink]="[ '/guards' ]">Guards</a>  
<a [routerLink]="[ '/resolver' ]">Resolver</a>  
<a [routerLink]="[ '/preview' ]">Preview</a>
```

Rendering the Page

Router Outlet Directive:

- Marks where the router displays a view
- Acts as a placeholder that Angular dynamically fills based on the current router state

A terminal window with a light gray title bar and three window control buttons (red, orange, green) on the right. The title bar contains a prompt icon and the text 'training@localhost:~'. The main area of the terminal is dark gray and contains the text '<router-outlet></router-outlet>' in white.

```
>_ training@localhost:~  
  
<router-outlet></router-outlet>
```

Child Routes

Sometimes, when routes are accessible and viewed only within other routes, it is more relevant to create them as child routes.

Example :

- There are **tabbed navigation** sections in the product details page that present the product overview by default.
- When the user clicks on the **Technical Specs** tab, the section shows the specs instead.

Declaration of Child Route

Example :

- In case the user clicks on the link of ID 3, show the user details page with the **overview**: <localhost:3000/user-details/3/overview>
- When the user clicks on **user skills**: <localhost:3000/user-details/3/skills>
- Here, overview and skills are child routes of user-details/:id. They are reachable only within user details

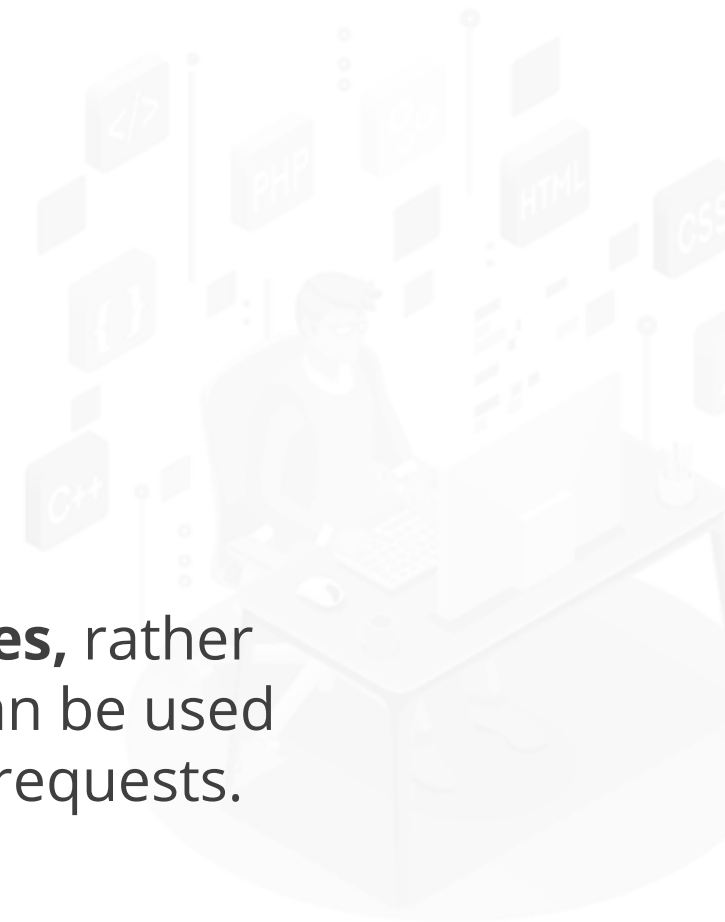
Angular HTTP and Observables

Later, callbacks were considered a bad practice, and **promises** had to be used.

Earlier, **callbacks** were preferred for handling XHR.



Now, **observables**, rather than promises, can be used to handle HTTP requests.



Using HTTP in Angular

HTTP is a separate model in Angular available under the **@angular/common/http** package.
To use HTTP in Angular:

01

Import the @angular/common/http package and inject it into the component or service

02

Use **get** method to send an HTTP request and subscribe to the response asynchronously

03

When the response arrives, map it to the desired object and display the result



Setting Up HTTP in Angular

To use the new HTTP module in the components, you have to import HTTP. Then, inject it via dependency injection in the constructor.

Here is an example:

>_ training@localhost:~

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

@Injectable()
export class MyComponent {
  constructor(private http: Http) { }
}
```

Routing Mechanisms



Duration: 20 min.

Problem Statement:

You are given a project to implement the routing mechanism in your Angular application.

ASSISTED PRACTICE

Assisted Practice: Guidelines to Demonstrate Routing Mechanisms

1. Configure the Angular application.
2. Change the source code to implement the routers.
3. Complete the functionality by implementing validators.
4. Push the code to GitHub repositories.



FULL STACK

Configuration Management

Workspace Configuration Files



When you run **ng new appName**, new Angular application is created.



This fundamental structure of the application contains various configuration files in the root folder.



It contains all the testing tool files, application config files, and folders for **node_modules**.



It creates a welcome page application and can be easily customized.

Workspace Configuration Files



It contains **src** folder that has code for your root component.



It also has an **index.html file** in root folder that contains meta data and call-selector for your application's root component.



A workspace in Angular shares a CLI configuration context.



An Angular workspace is capable of holding more than one Angular application.

Workspace Configuration Files

Workspace Configuration Files	Purpose
.editorconfig	Important file for code editors
.gitignore	Untracked files that Git should ignore
README.md	Basic documentation
angular.json	CLI configuration defaults for all projects in the workspace, including configuration options for build, serve, and test tools that the CLI uses
package.json	npm package dependencies that are available to all projects in the workspace
package-lock.json	Provides version information for all packages installed into node_modules by npm client
src/	Source code files for root-level application project
node_modules/	Provides npm packages to the entire workspace
tsconfig.json	Default TypeScript configuration for projects in the workspace
tslint.json	Default TSLint configuration for projects in the workspace

CSS

JS

HTML

Python

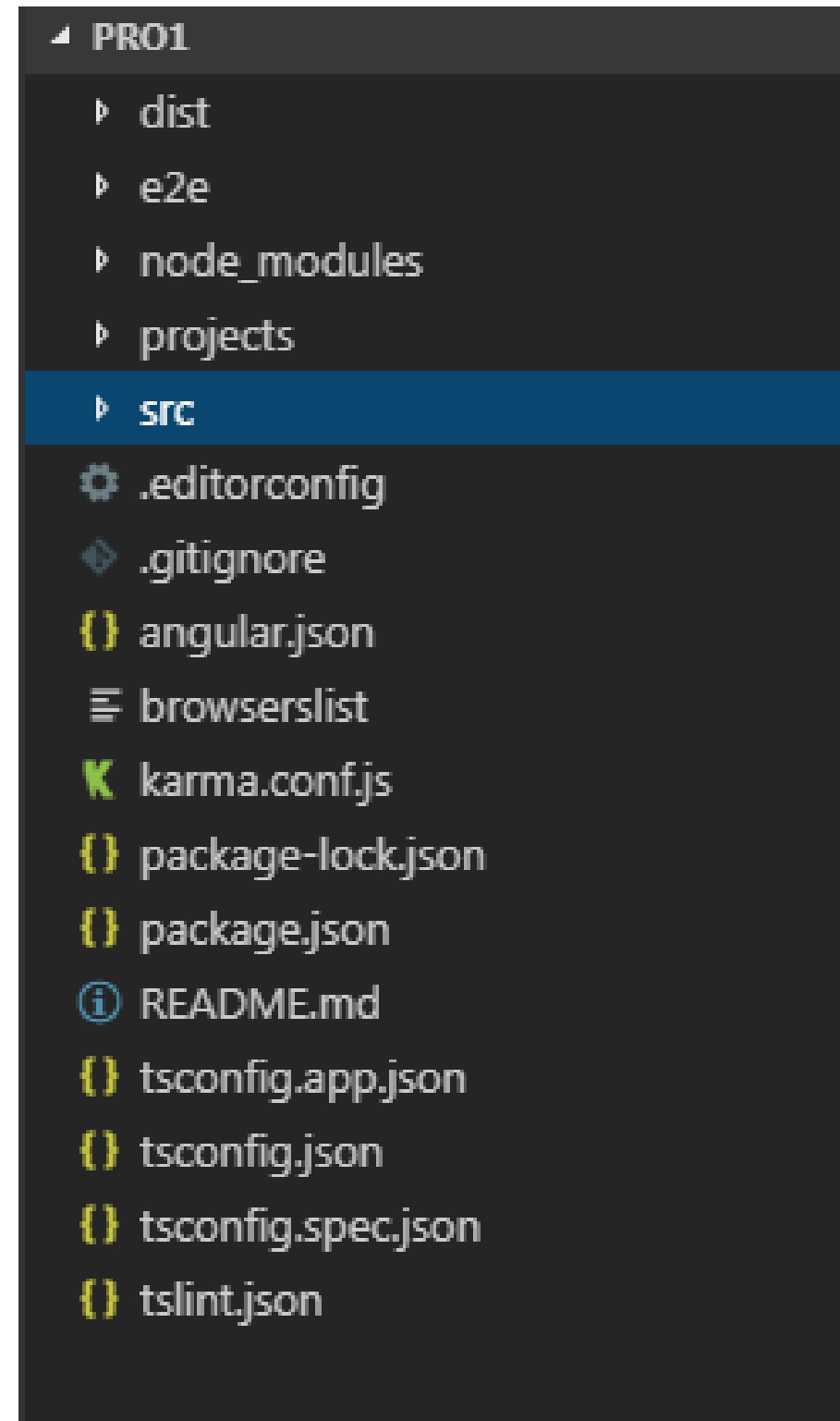
Java

PHP

Application Project Files

ng new myapp creates new application with the root folder named as **myapp**.

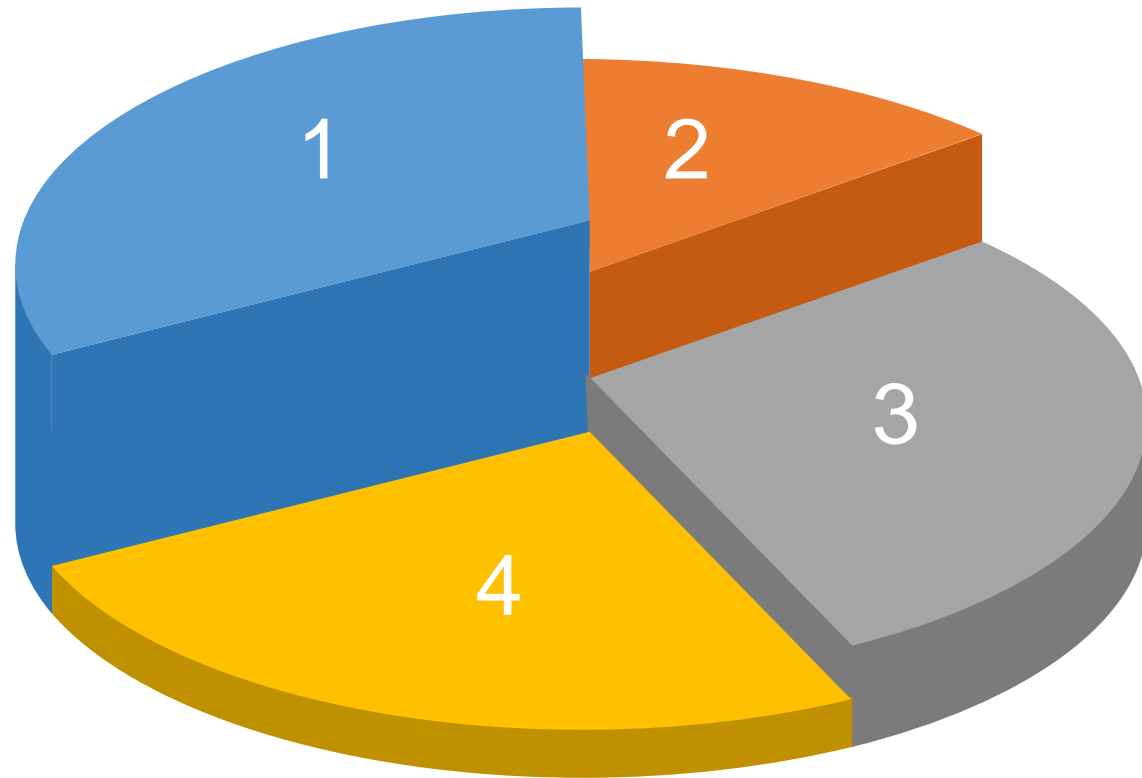
- It generates new app skeleton in the **src** folder.
- It contains source code file for root component with view template.
- It also includes project-specific Typescript configuration files inherited from workspace-wide tsconfig.json, and project-specific TSLint configuration files.
- **e2e/** folder contains source code files for a set of end-to-end tests.



Application Project Files

App Support Files	Purpose
app/	Contains the component files in which your application logic and data are defined
assets/	Contains image and other asset files to be copied as-is when you build your application.
environments/	Contains build configuration options for particular target environments
favicon.ico	An icon to use for this application in the bookmark bar
index.html	The main HTML page that is served when someone visits your site
main.ts	The main entry point for your application. Compiles the application with the JIT compiler and bootstraps the application's root module (appModule) to run in the browser
polyfills.ts	Provides polyfill scripts for browser support
styles.sass	Lists CSS files that supply styles for a project
test.ts	The main entry point for your unit tests, with some Angular-specific configuration

Multiple Projects in Angular



- An Angular workspace can have more than one angular project.
- You need to provide another command to the **new command** as shown below:
ng new my-workspace --createapplication="false"
- To create an application inside your application or libraries, which are small applications, use below command:
ng generate application my-app
ng generate library my-lib
- Both application and libraries are created in your major application and you can export them to your parent component.

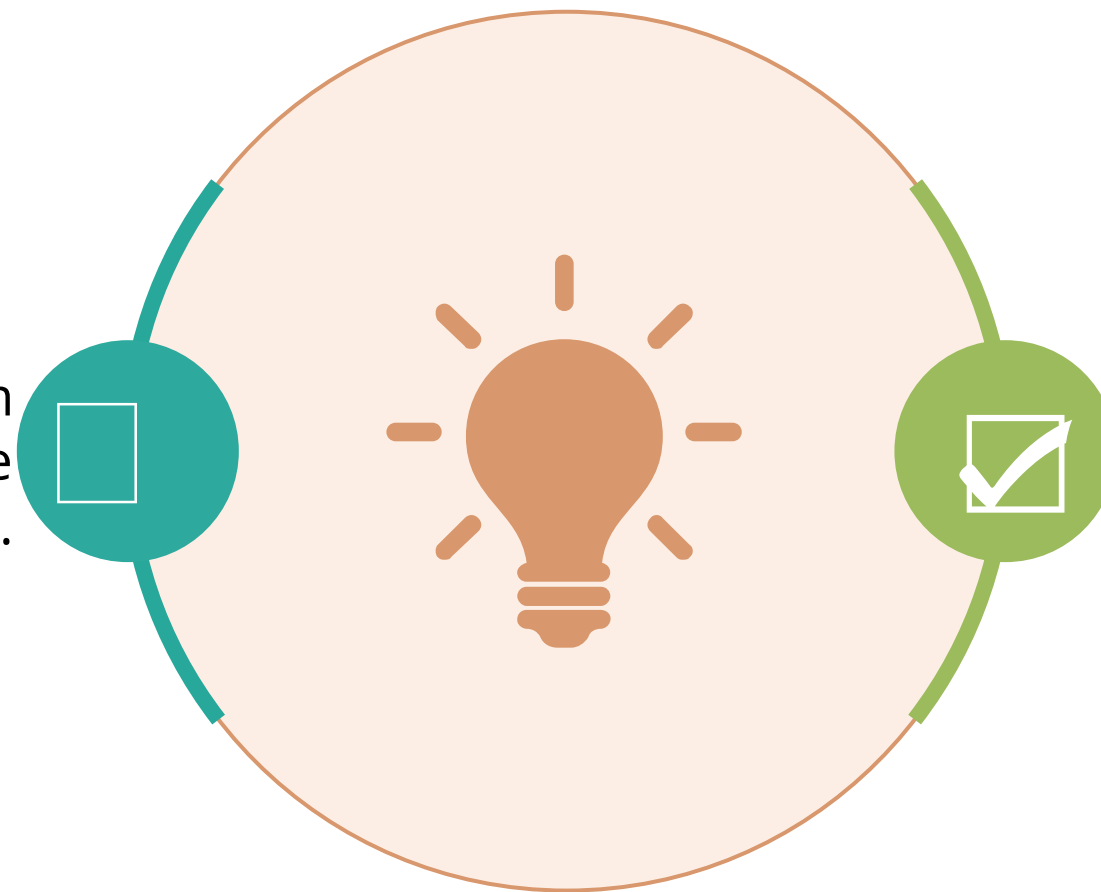
FULL STACK

npm Dependencies and Browser Support

npm Dependencies

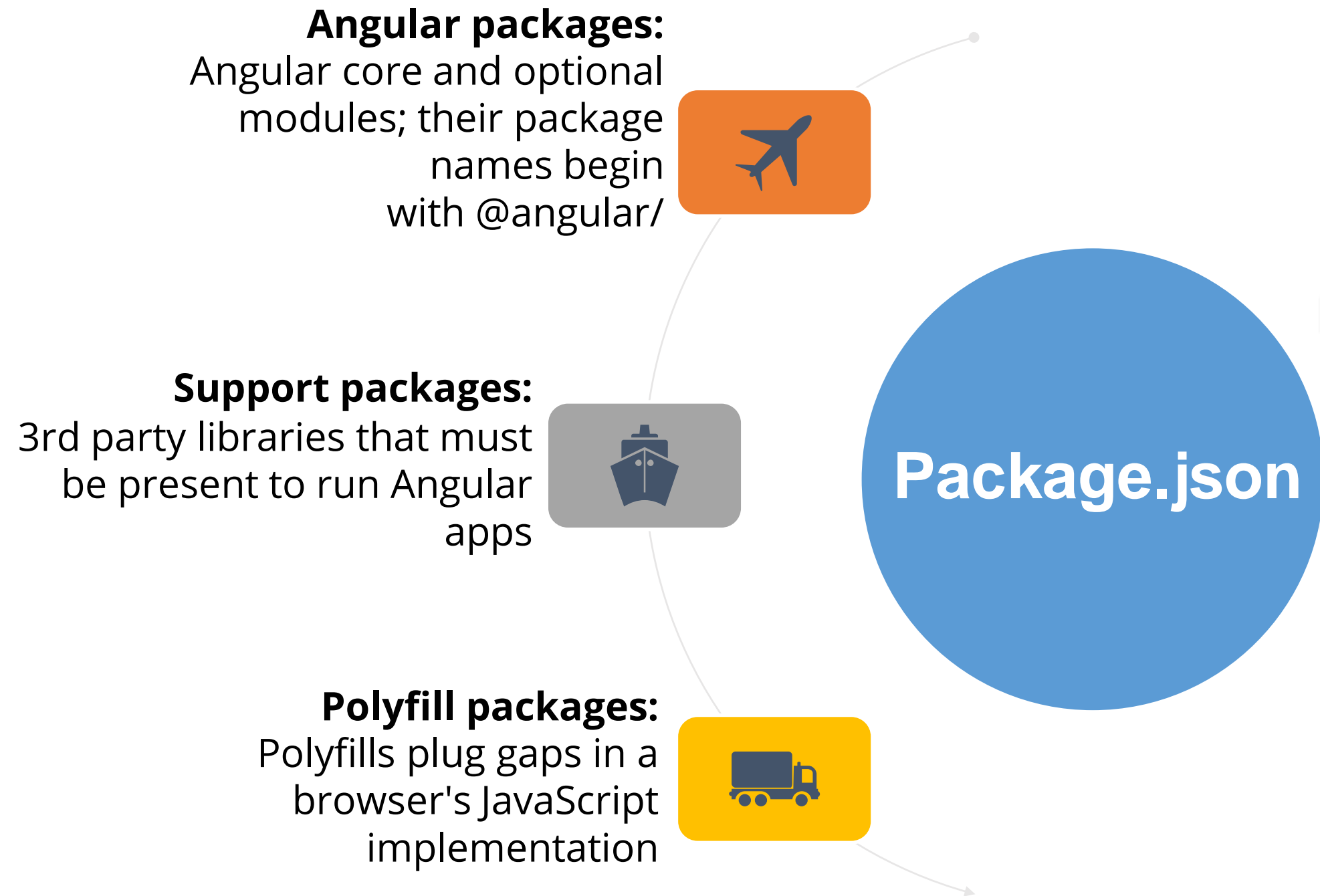
The Angular framework, Angular CLI, and components are packaged as npm packages and distributed via the npm registry.

You can download and install npm packages using **npm CLI client** or use **yarn** as an alternative.



To add devDependencies use the below commands:
npm install -dev <package-name>
Yarn add -dev <package-name>

npm Dependencies



Angular Packages

Packages	Description
@angular/animations	Used primarily for applying animations to the application
@angular/common	Common elements like directives are present here. HttpClientModule is also included in this package
@angular/compiler	Comes into picture mainly when application is compiled
@angular/core	Critical runtime components, such as components, directives, and services are present here
@angular/forms	Used to build template-driven and reactive forms
@angular/platform-browser	Renders the browser components into DOM
@angular/platform-browser-dynamic	Providers and JIT compiler are present here
@angular/router	Includes all the operations of Routing Mechanism

Support, Polyfill Packages, and Browser Support

Packages	Description
rxjs	Maintains JavaScript language standards
zone.js	Helps in detecting the changes in application

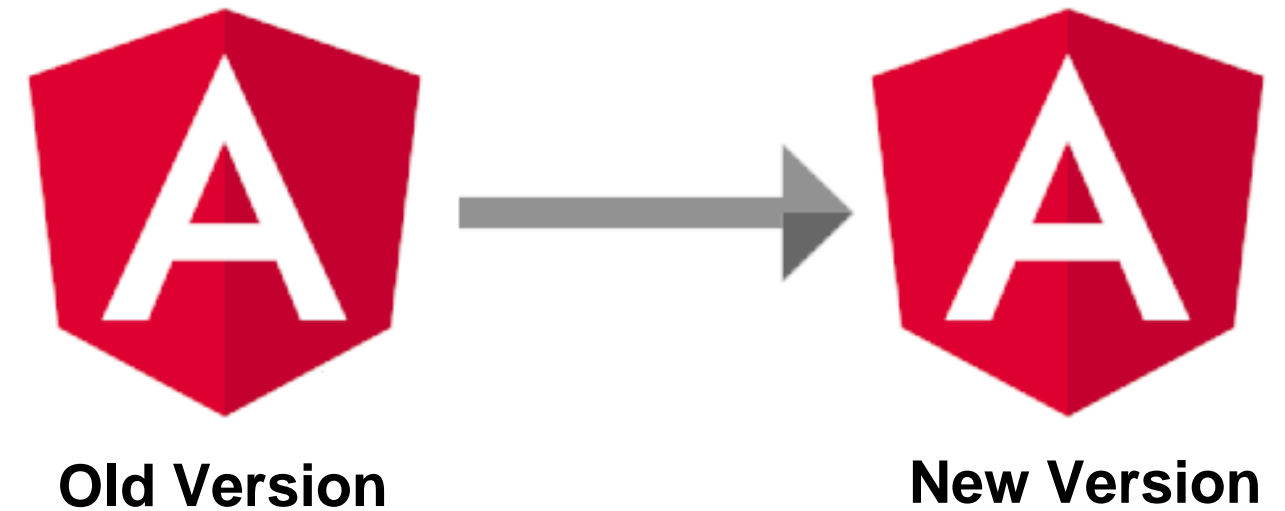
- Polyfill packages deal with supplementing the required adjustments that need to be done for the application to successfully run in the browser.
- Package.json contains core.js which polyfills the missing dependencies required by the applications.
- You can install polyfills using the below command:
- **npm install --save web-animations-js**
- To know more about Polyfills and BrowserSupport, you can refer the below link:
- <https://angular.io/guide/browser-support>

FULL STACK

Release Management

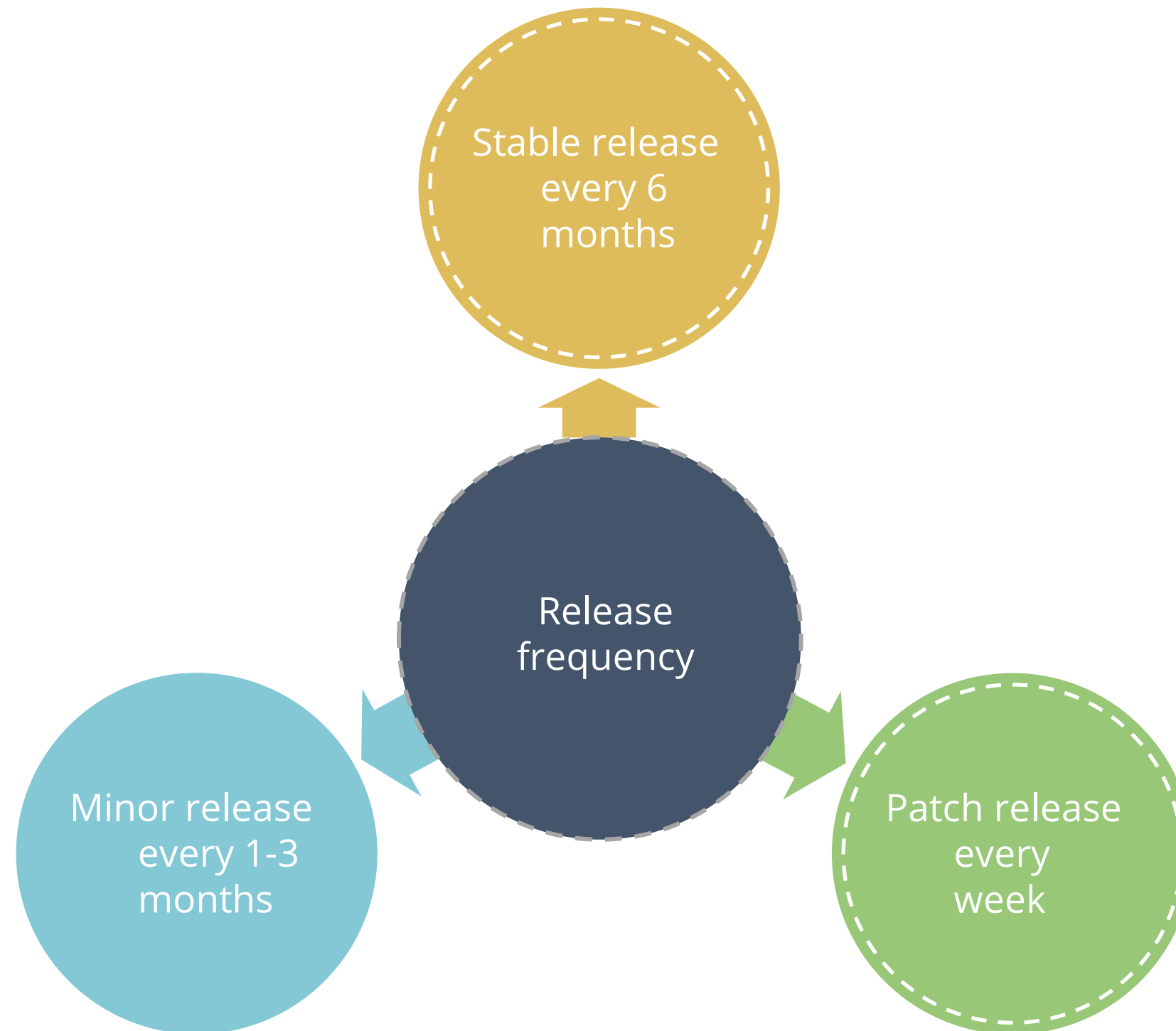
Updating Angular Projects

- One of the major checks that an Angular application should go through is to make sure that it is built upon the latest dependencies and versions.
- Things to remember before release:
 1. Check the Angular version
 2. Major Angular release
 3. Deprecated modules/features
- It is important for enterprises to keep the updates and dependencies incorporated in their product.



Release Practices

Deprecation refers to the removal of some resources/packages, which get upgraded to an improved version.



Deprecation Practices

Deprecation period

- When an API or a feature is deprecated, it will still be present in the next two major releases.
- After that, deprecated APIs and features will be candidates for removal.

Announcement

- Deprecated APIs and features are announced in the change log.
- Deprecated APIs appear in the documentation with Strikethrough and a recommended update path is implemented.



npm dependencies

- npm dependency updates occur only when changes in the app require a major release.
- In minor releases, peer dependencies are updated.

FULL STACK

Deploying an Angular Application on Server

Deploying an Angular Application on Server

When you have created a full-fledged angular application and you want to deploy this application to the server, follow the given steps:

1. Enable production mode
2. Configure your app for testing
3. Build the app
4. Upload the app to the server



Key Takeaways

- Angular is a JavaScript-based, open-source structural framework for dynamic web applications
- AngularJS can be migrated to Angular either by using Gulp or by using ngUpgrade
- The Angular framework, Angular CLI, and components are packaged as npm packages and distributed via the npm registry
- The angular router allows navigation from one view to the another as the user performs application tasks



Build an Angular Application

Duration: 30 min.

Problem Statement:

You are given a project to accomplish the following in your Angular application:

- It should include appropriate routers
- It should have the components for the routers
- It should include bootstrap
- It should include a Sign-Up page
- It should include a Sign-In page



Before the Next Class

Courses: Learn to build Progressive web apps using JavaScript

You should be able to:

- Install node
- Fetch API
- Build a PWA

