


☐

I'm not robot

  
reCAPTCHA

Continue

## Angular 5 tutorial pdf

All you need to learn about Angle, the best tips and free code examples, so you can get the best out of Angle. 60 Min READ! Angular MaterialBeginnersFormsAngular CLI Let me introduce you to Angular Angular is a platform for building mobile and desktop web applications. This is a large community of millions of developers who choose Angular to build compelling user interfaces. Angular is javascript open source front end web application framework. It is mainly supported by Google with an expanded community of people and businesses. Angular solves many of the challenges faced by developing single-page, cross-platform, performant applications. It is fully expandable and works very well with other libraries. For more information, visit their official documentation page. My goal in this Angular real world example tutorial is to provide a complete guide that you learn angle step by step. We will explain why and the basic concepts and then continue to explore more advanced concepts. We want to help beginners through their first steps in the corner world. As developers, we know that starting new technology can sometimes be a little frustrating, so I want to help here. We learn enough from the core angle to start and gain the confidence that we can build any app angular. We cover a lot of the land at the introductory level, but also, you'll find many references to topics in greater depth. To help you through your angular learning process, we created an angular app with a question and answer format (Q&A; A) where users can ask, answer and vote on questions. We will also explain how to connect this application to a remote API to handle data integration. So, in this complete tutorial you will learn all the concepts needed to create your first corner application. You can download the source code for this cornerless template by clicking get code from above. We also published an online demo app we're going to build in this Getting Started Angular guide. Our journey to Angular We started testing and testing the very first release of Angular 2.0.0-beta.0 in December 2015 with the hope of finding a framework that was clearly better than its predecessor (Angular 1.x also known as AngularJS). I'll be completely honest with you, we're almost giving up all the contradictions, breaking the changes and the identity crisis that happened in the middle of the Angular 2+ development. It was a long way until Angular reached solid milestone Universal (server-side rendering), pre-of-time compilation (AOT), lazy loading and solid aggregation config working together nicely. Back in those years it was not easy to create a production ready angle application. But thanks to the angle of the team and the corner community that changed. Amazing things can be with the latest versions of Angular. You can view our latest creation of the angular latest version in full – Angular Admin Template Works, using and trying to get things through Angle from the beginning made us really understand how it was designed and how it evolved. We witnessed a steady improvement and saw how they were all consistent with one simple but important goal: Creating an app should be easy. As I mentioned earlier, for a while during the process, it was not. Now I can tell you Angular is a super solid and stable framework you love to work in. The current versions of Angular had evolved to the point where you are quickly impressed. Angular is a great tool that allows you to create software faster and less trouble Result more maintainable software Encourage good programming practices and design patterns like MVC Allows you to make it easier to collaborate with other people Allows you to get proficient in a reasonable time to address problems that may arise from your software architecture such as Addiction Injection, DRY Injection (Do not Repeat Yourself), etc. Differences in Angle Versions When It All Started, Back in 2010 , this framework was called AngularJS and hints at what we now know about angle 1.x. 2016 arrived as a complete rewrite of the Angular 2 framework, improving the lessons learned and promising performance improvements and a more scalable and modern framework. AngularJS was fully based on controllers, and the view communicates \$scope using the same, while angle 2 is a 100% component-based approach. We no longer have controllers at 2th, and \$scope. Components are angular 2 application building blocks. We will see the benefits of this change in a few minutes. The first version of Angular was named Angular 2. It was later renamed Corner. Between Angular 2 and Angular 10 (current latest stable version) was Angular 4 (released in early 2017), Angular 5 (released at the end of 2017), Angle 6 (released in early 2018), Angle 7 (released at the end of 2018), Angle 8 (released in mid-2019), Angular 9 (released in early 2020). Angular 10 was released in June 2020. You can find all information about versions in CHANGELOG. Don't freak out on all these versions. Since all versions of Angular 2 angular 10 have the same framework, they share the same core, but they differ from many amazing improvements! From now on, every time we use the term Angle we refer to the latest version of the framework that is currently Angular 10. With the new angular compared to AngularJS Just for the sake of history, let's go through the main differences between AngularJS and Angle: Angle is a complete rewrite of AngularJS. The angular application and its architecture differ from angularJS. The main building elements of the corner are modules, templates, metadata, data interconnection, directives, services and addition injections. The angle is not the scope of the concept or controllers, instead of using the component hierarchy as its primary architecture. The angle follows the concept of modularity. Similar functions are held together in modules. This gives the angle optimized for a lighter core. The controller concept, which existed in AngularJS, was removed from Angular 2 and above, which is a component based on the UI. This helps developers share apps with the desired features into components. These helped improve flexibility and recyclability compared to AngularJS. The square expression syntax focuses on the binding of the attribute [] and the event binding () []. With AngularJS, building a search engine (SEO) friendly Single Page Application was a big difficulty. But this bottleneck was eliminated by angular 2, allowing the application to render on the server. These tasks are possible thanks to the angular universal module. Angle suggests using TypeScript language, which introduces these features: Static typing object oriented programming based on classes of support for reactive programming using RxJS On Top Of TypeScript features, Angular also includes the benefits taken from ES6: For/Of loops Better Dependency Injection Iterators Reflection Dynamic loading Asynchronous template compilation Easier routing from Angular 2 ular Angular 4 There were some important changes, but mostly the project structure batch refactors that made the framework more stable. Smaller and faster. Upgrade 2.0 to 4.0 has reduced the size of the bundled file by 60%, while improving the speed of applications. Angle 4 is compatible with later versions of TypeScript 2.1 and TypeScript 2.2. Corner universal: Most of the angular universal code is connected to the angular core. Animation package: Animations taken from the Angle core and set in your package. This means that if you don't use animations, the excess code won't reach your app. From Angle 5 angular 7 Angular 6 was the first release angle that combines versions of The Framework, Material and CLI. This change was made to explain cross-compatibility. Angular 7 was full of new features, bug fixes, performance improvements and some code exhaustion such as clean refactors in the old version. An optimization build process that reduces the application size by removing unnecessary code. Material design components with server-side rendering. Angle universal fixes for code allocation between the server and the client-side version of the application. Many improvements to angular CLI smaller bundle sizes improved compiler that supports incremental compilation meaning faster restores. RxJS (Reactive Programming Library) has been updated to version 6.x or later. Angle now requires TypeScript 3.x from Angle that Angular 10 + Angular 8 was a release that covered the entire platform, including framework, Angle material, and cli. This version of the improved application launch time for modern browsers. It also changed routing configurations to use dynamic import to favor lazy loading. Angular 9 was very much expected in the community as it introduced ivy compiler and runtime. Ivy is the name of Angular's next generation of compilation and rendering pipeline. This version uses the new compiler and runtime instructions as default instead of the older compiler and runtime, known as the View Engine. Ivy Compiler Offers The Following Advantages: Smaller Bundle Sizes Faster Testing Better Debugging Better CSS Class and Style Pairing Better Type Verification Improved Build Errors Improved Build Times, Allowing AOT Default Advanced Internationalization More information about these benefits can be found in Angular 9 release note. The release of angle 10 was less than typical: It has only been 4 months since the release of Angular 9. More information about this version can be found here. Moving on to this angular tutorial, let's setup the development environment. After the previous introduction to the current frameworks, we are now ready to start work on our corner application. The best way to learn Angle is by following this step-by-step tutorial for beginners. In the next section of this angle free course we will go through the setup and requirements needed to start developing angle applications. We'll start a full web application demonstration project with the angular Setup Corner Development Environment In this section, we'll show you how to set up your local development environment so you can develop angular applications. The real development of the app takes place in a local development environment, which can be your personal machine. Follow our setup instructions to create a new corner project. Angular requirements: Install NodeJS and NPM Node.js and NPM are essential for modern web development using Angle and other platforms. The node provides customer development and tool creation. We are not going to use the node package manager (npm) to install all JavaScript libraries with dependencies. Get them as soon as they're not installed on your computer. Note: Make sure that you are using the latest stable versions of nodes and ZPM. Angular CLI angular apps are created and developed primarily by angular CLI (command line interface tool) that helps project creation, file addition and perform a variety of ongoing development tasks, such as testing, merging and deployment. Angular CLI takes care of the configuration and initialization of the various libraries. It also helps us to add components, directives, services, etc. to existing angular applications. It is also worth noting that the CLI uses Typescript Modules for package sales web package, Karma unit testing, and Protractor end trials. It contains everything you need to start writing your Angular application right away. To install Angular CLI globally, run the following command in the console npm install -g @angular/cli Note: although this is not recommended, you must add sudo in front of these commands to install utilities globally. Important note: If you have an older version of the CLI installed on your computer, make sure that you update it to the latest corner of the CD. Now that you have an angle and its dependencies installed, we can move on and start building our Angular app. Let's go! Starting a new angle app with the CLI is easy! At your command line, run this command: ng the new my-new-angular-app above command creates a folder named my-new-angular-app and copies all the necessary dependencies and configuration settings. Angular CLI does this for you: Creates a new directory of my-new-angular-app Downloads and installs Angle Libraries and Other Dependencies Installs and Configures TypeScript installs and configures Karma & Protractor (testing libraries) You can also use the ng init command. The difference between ng init and ng new is that ng new requires you to specify a folder name and it creates a folder copying files while ng init copies the files to the current folder. Now you can get the cd created in the folder. To get a quick preview of your application inside the browser, use the eam command to use ng eam this command running compiler watch mode (searches for changes in code and compiles if necessary), launches the server, launches the application in the browser and keeps the application running as we continue building it. Webpack development server listens to HTTP port 4200. Therefore, when you open the URL you you see the app is running. Using Angular CLI to add new pages to Angle, there's still a stencil compared to AngularJS (Angular 1), but don't panic. The new Angular CLI also has more tools to help you through it. For example, a new generator works. They provide an easy way to create square pages and services for your app. This makes it much easier to go from the main app to the full navigation web app. I call it a simple learning curve :). To create a new component, you can use the following command: ng create a component of my-new component ng g component my-new component # using alias √ Create app/pages/my-page/my-page.html √ Create app/pages/my-page/my-page.html √ my-page/my-page.ts √ Create app /pages/my-page/my-page.scss Nurk-CLI automatically adds reference to components, directives, and pipes automatically app.module.ts. Note: For more information about adding components and other elements to the application, see the corner CLI documentation. An angle is a framework designed to build and most of its design is geared towards doing so effectively. One-page apps (or SPA) are apps that can be accessed through a web browser like other websites, but offer more dynamic interactions that resemble local mobile and desktop apps. The most noticeable difference between a regular website and a spa is a reduced page update. Typically, 95 percent of the SPA code works in the browser; the rest of the server is running when the user needs new data or needs to perform secured actions, such as authentication. As a result, process page rendering is mostly done on the client side. Angular modules help organize the application into blocks of online functions by packing components, pipes, directives and services. They are just all about developer ergonomics. Corner applications are modular. Each corner application has at least one module—a root module commonly called AppModule. The root module may be the only module in a small application, but most applications have many more modules. As a developer, you have to decide how to use the modules. Typically, you map the main functions or function to the module. Let's say there are four main areas in your system: Each has its own module in addition to the root module, with a total of five modules. Each corner module is a @NgModule with an interior designer. Interior designers are features that change JavaScript classes. They are basically used to add metadata to classes, so he knows the configuration of these classes and how they should work. The module @NgModule the designer's properties are: declarations: Classes that belong to this module and are related to views. The corner has three classes, which may include views: components, directives and pipes. export: classes that should be available to other module components. import: Modules whose classes are required by the components of this module. service providers: services present in one of the modules used in other modules or components. Once a service is included in service providers, it becomes available in all parts of that application. bootstrap: The root component that has the main view of the application. Only the root module has this property and shows a component that will be tied up. entryComponents: A transaction component is any component that is compulsorily loaded by angle load (which means that you do not refer to the template) by type. Corner components are the most basic interoperability unit for user interface and angular applications. The component controls one or more sections (which we call views) on the screen. For example, in this example, we have components such as AppComponent (bootstrapped component), CategoriesComponent, CategoryQuestionsComponent, QuestionAnswersComponent, etc. A is self-contained and represents a reusable part of the user interface, which usually consists of three important things: a piece of html code known as the view age, which encapsulates all available data and interactions with this view by the angular through the Architecture Attributes and Methods API. Here is where we define the logic of the application (which it does to support the view) And the above html element is also known as the voter. Using Nurk@Component designer, we provide additional metadata that determines how a component should be processed4. instantiated and used at runtime. For example, when we set up an html template associated with a view, we set up the html picker that we use for this component. The component transfers the data to the view by using a process called Data Binding. This is done by linking the DOM elements to the component properties. The binding can display attribute values for the user, change element styles, respond to user event, etc. The component must belong to NgModule so that it can be used using another component or application. To specify whether the component



is a member of NgModule, you should list it in the NgModule declaration property. One side of note components the importance of point software architecture principles: It is super important and recommended to have separate components, and here's the reason. Imagine that we have two different UI blocks in the same component and in the same file. At first, they may be small, but anyone can grow. We are confident that we will get new requirements for one, not another. However, any change puts both components at risk and doubles the testing burden without any benefit. If we had to reuse some of these UI blocks elsewhere in our app, another would be glued along. This scenario violates the principle of shared responsibility. You may think that this is only a tutorial, but we need to do things right – especially if doing them right is easy and we learn how to build angular apps in the process. The angle encourages this principle by taking each patch page to check it for its own component. A typical angular application looks like a component tree. This term is illustrated by the following diagram: Note that the modal components are on the parent component side because they are mandatory components that are not declared in the component html template. Corner building units: Templates are used to define component view. The template looks like normal HTML, but it also has some differences. Code such as \*ngFor, {{hero.name}}, (click) and [hero] uses the syntax of the angle template to improve html marking capabilities. Templates can also contain &lt;custom-element> &gt; custom components, such as non-regular html tags. These components are mixed seamlessly with native HTML in the same layouts. Angle &lt;custom-element> &gt; blocks: Services Almost everything you can service, any value, feature, or feature that your app needs. The service is usually a narrow, well-defined class. It should do something concrete and do it well. The primary objective of Angular Services is to share resources between components. Take component classes, they should be lean, component work is to enable user experience (mediate between view and application logic) and nothing more. They do not retrieve data from the server, do not validate user input, or log directly to the console. They delegate such tasks and anything that is not for services. Services are essential for any angular application, and the components are great for consumers of services because they help them to be lean. The scenario that we have just described has a lot to do with the principle of separation of problems. Angle doesn't enforce these principles, but it helps you follow these principles by making the logic of the application easier to structure into services and making these services available to components through an addition injection. In our sample application, we have three services: AnswersService, QuestionsService, CategoriesService. Each has only related functions. In this particular tutorial we will focus only on CategoriesService and in the following sections we will discuss others. CategoriesService is the following methods: //can all question categories local json getCategories() { return this.http.get('./assets/categories.json') .map((res:any) => res.json()) . Promise to() //finds the slug of a specific category, getCategoryBySlug(slug:string) { returns this.getCategory() .then(data => { return data.categories.find(((category) => { { return category.slug == slug; }}}) } } } Angle building units: Other resources External resources, such as databases, APIs, etc., are fundamental because they allow our application to communicate with the outside world. There's a lot more to cover about the basic building blocks of angular applications like Addition Injection, Data Binding, Directives, etc. You can find them and a lot more information about our upcoming post about Angle: Learning The Way. Now, let's go deeper and map the design architecture of the project structure so that we can better understand how all pieces interact with each other. After following the setup instructions for creating a new project in the previous section, we will walk through our anatomy of the angular app. Cli setup procedures install many different files. Most of them can be safely ignored. On the project, we have three important folders and some important files: /src/ This is the most important folder. Here we have all the files that make our Angular app. /e2e/ This folder is the end-of-application tests written in Jasmine and controlled by the protractor e2e test runner. Please note that we do not enter details testing in this post. package.json As with any modern web application, we need a package system and package manager to handle all third-party libraries and modules our app uses. Inside this file, you will find all the dependencies and some other handy stuff like the ZPM scripts to help us a lot organize the development (bundling / compiling) workflow. tsconfig.json Primary configuration file. This must be the root path because this is where the typescript compiler is looking for it. Inside the /src directory we find our raw, uncompleted code. This is where most of the work on your Angular app takes place. When we run ng serve, our code inside / src can be bundled and transpiled with the correct Javascript version that the browser understands (currently ES5). This means that we can work at a higher level using TypeScript, but compile down to the older form of Javascript that the browser needs. Under this folder, you will find two main folder structures. /app is all components, modules, pages on which you build the application. /environments this folder is to manage different environment variables such as dev and prod. For example, we could have a local database for our development environment and a product database for the production environment. When we run ng serve it uses the default dev env. If you want to start production mode you must add --prod flag ng to earn. index.html/ This is the hostht of the application, but you do not change this file often, as in our case it only works as a placeholder. All the scripts and styles needed to do the app job are going to be injected automatically into the webpack batch process, so you don't have to do it manually. The only thing that comes to my mind now that you can add to this file is some meta tags (but you can also handle them through Angular as well). And there are other secondary but important folders/assets in this folder, you'll find pictures, sample data from json's, and any other property you might need in your app. Corner Best Practices: The App's This is the core of the project. Let's look at the structure of this folder so you can get an idea of where to find things and where to add your modules to customize this project according to your specific needs. /shared The SharedModule that lives in this folder is available to keep common components, directives and pipes and share the modules that need them. It imports CommonModule because its component requires common directives. You will notice that it re-exports other modules. When you review the application, you may notice that many components that require SharedModule directives also use NgIf and NgFor from CommonModule and bind component properties [(ngModel)], the FormsModule directive. Modules that declare them import CommonModule, FormsModule, and SharedModule. You can reduce the number of times you can reduce the number of times you can get SharedModule and FormsModule for free. However, SharedModule can be exported to FormsModule without adding it to your import. / styles Here you will find all variables, mixins, shared styles, etc., which makes your app customizable and extendable. Maybe you don't know Tangled? In short, it is superset css that simplifies and speeds your development cycles incredibly. /services Here you will find all the necessary services in this application. Each service has only related features. Other folders to get code modularity, we have created a folder for each component. In these folders, you will find each related file for the pages in this component. This includes layout html, sass styles and home page component. app.component.html works with the skeleton app. Usually it is &lt;router-outlet> &gt; change routes and their contents. You can also wrap it with content that you want to use on each page, such as a toolbar. app.component.ts This is an angle component that provides the functionality of app.component.html file I just mentioned. app-routing.module.ts Here we define the main routes. These routes are registered with AppModule angular RouterModule. When you use lazy modules, child routes for other lazy modules are defined inside these modules. app.module.ts This is the main module of the project that starts the application. As we advance this tutorial we will create more pages and perform basic navigation. A little more navigation angle is a specific module dedicated to navigation and routing, RouterModule. Use this module to create routes that allow you to move from one part of the application to another or from one view to another. Work routes require a user interface anchor or element to map actions (usually clicked on items) to routes (URL paths). To do this, we use the RouterLink Directive. For example, if a user clicks a category name in the user interface, the angle through the router link directive knows that they must navigate to the following URL: &lt;a class=list-item [routerlink]=[/questions/about', category.slug]&gt;{{category.title}}&lt;/a> &gt; Next, you must map the URL path to the components. In the same folder as the root module, create the app.routes.ts configuration file (if you do not already have it) with the following code: import { Routes } from @angular/router; export const routes: Routes = [ { path: '/', component: CatesComponent, resolve: { data: CategoriesResolver } }, { path: questions/about/categorySlug, component: CategoryQuestionsComponent, solve: { data: &lt;/router-outlet> } }, { path: 'question/questionSlug', component: QuestionAnswersComponent, solve: { data: QuestionAnswersResolver } } ] ]; For each route, we provide a path (also known as URL) and a component that should be rendered in this path. The CategoriesComponent path with an empty string means that CategoriesComponent is rendered when the URL is not (also known as the root path). Note that for each route, we also have a solution. Using the resolve of our navigation routes allows us to pre-pull component data before activating the route. Using the solutions is a very good practice to make sure that all the necessary data is ready to use our components and to prevent the empty component from being displayed while waiting for data. For example, we use Categories to retrieve a list of categories. When the categories are ready, we activate the route. Note that if the traceable one that is resolved is not complete, navigation will not continue. Finally, the root module must also know the routes we defined above. Add reference appmodule import attribute processes. import { routes } from './app.routes'; import { RouterModule.forRoot (routes, { useHash: false } ) }, Notice how we use forRoot (or finally forChild) methods RouterModule (docs explain the difference in detail, but now just know that forRoot should only call once on your app's top-level routes). Angular 2 vs ngx-bootstrap There are some libraries that offer high-end components that allow you to quickly create a nice interface for your app. These include modals, pop-ups, maps, lists, menus, etc. These are reusable user interface elements that are the foundations of your mobile app, which consist of HTML, CSS, and sometimes JavaScript. The two most commonly used UI component libraries are angular material and ngx-bootstrap. Angular material is the official angular UI library and provides tons of components. On the other hand, ngx-bootstrap offers a number of Angle components made on top of Twitter's Bootstrap framework. In this Angular tutorial we are not going to use Angular Material, but feel free to choose the one that best suits your needs because they are both super complete and robust. In this angular sample application, we have different layouts. For each view, we need different user interface components. Here is a short list of the most important components we used for each view, and a link to the specifics of applying this view. Categories view a list that shows the different angle definitions you need to learn. Material Components: The chip component of a list of subtags in the Category Items category questions view in the A view to display all questions in a specific category. Material Components: Question List Button Component Dialog Component for Modade Question Answers View View That Shows All specific issue. Material Components: List Component Answers List Button Component Dialog Component Modalde New Question and New Answer Modald Exmodals create/ update questions and answers Material components: The dialog component for modal management We also used the angular toolbar component to help navigate. You are able to dig library UI components that Angular Material has on the documentation page for these components. Adding a backend to our angular sample project Different alternatives to integrating backend API data the emerging application key is to create reusable services to manage all your backend data calls. As you know, there are many ways to handle data and apply a backend. In this tutorial we will explain how to consume data static json file dummy data. In the next tutorial Learn to build a MEAN stack application you will learn how to build and consume data rest api loopback (node.js framework is ideal for rest API) and MongoDB (data storage). Both deployments (static json and server backend API) must be concerned about the application problem of how to handle data calls. It works the same and is independent of how you implement the backend. We are talking about models and services and how they work together to achieve this. We recommend that you use models with services to handle data from backend to presentation. Domain models Domain models are important for defining and enforcing business logic in applications, and are especially important as apps get bigger and more and more people work with them. At the same time, it's important that we keep our apps KUIVANA and maintainable, moving the logic out of the components themselves and into separate classes (models) that can be called. A modular approach like this makes our app's business logic reusable. For more information about this, visit this great post about angle 2 domain models. The data services angle allows you to create multiple reusable data services and inject them into the components that need them. Refactoring data access to a separate service. keeps the component lean and focuses on supporting the view. It also makes it easier to test the component with a mockery service. For more information, see Section 2. In our case, we defined a model for question categories data that we pull from a static json file. This model is used by categories.service.ts. in category.model is export class CategoryModel { slug: string; title: string; image: string; Description: string; tags: Array<Object>; } /in categories.service.ts getCategoryModel() &gt; { returns this.http.get('./assets/categories.json') .toPromise() then (res => { res.json() as<CategoryModel> &gt; &lt;Object> &gt; } } And we use this service categories.resolver.ts where we bring categories to view data. in the categories.resolver.ts import { Injectable } @angular/core; import { Resolve } @angular/router; import from { CategoriesService } . /services/categories.service: @Injectable() Export Class CategoriesResolver Applies Resolve&lt;any> &gt; { constructor(private categoriesService: CategoriesService) { } resolve() { back to the new Promise((resolve, reject) => { let breadcrumbs = [ ( urf: '7', label: Categories ) ]; /get categories from local json file this.categoriesService.getCategories(), then(categories => { return resolve( categories: breadcrumbs: breadcrumbs); }, err => { return(zero) } ) } ) } } Every time we add a new service to remember that the angle injector does not know how to create that service by default. If we were to run our code now, Angle would fall foul. After creating services, we need to teach the corner injector how to do this service by registering a service provider. According to the Angular Documentation page of the dependency injection, there are two ways to register a service provider: in the component itself or in the module (NgModule). In our case, we register all services in app.module.ts //in app.module.ts @NgModule({ declarations: [ AppComponent, CategoriesComponent, CategoryQuestionsComponent, NewQuestionModalComponent, NewAnswerModalComponent, UpdateAnswerModalComponent, QuestionAnswersComponent, DeleteQuestionModalComponent, DeleteAnswerModalComponent,DeleteAnswerModalComponent], import: [ RouterModule.forRoot (routes, useHash:false ) ], SharedModule }, Entry: [ ], providers: [ CategoriesService, QuestionsService, AnswersService, CategoryQuestionsResolver, CategoriesResolver, QuestionAnswersResolver ], bootstrap: [AppComponent] }) export class AppModule { { } One-sided note on the importance of addition injection software architecture principles point: Remember, we just mentioned that we inject data service components that need them? Well, this concept is called Addition Injections and it's super important to know more. Are We New () Services? No way! This is a bad idea for a number of reasons, including: Our component needs to know how to create a service. If we ever change the Service Builder, we must find every place where we create the service and improve it. Running around patching the code is error prone and adds test load. We create a new service every time we use a new one. What if the service should cache the results and share this cache with others? We couldn't do that. We lock the component (where we service new) for specific implementation of the service. It is difficult to switch between different scenarios. Can we work offline? Do we need different mocked versions of the test? It's not easy. We understand. It really is. But it is so &lt;/any> &gt; easy to avoid these problems that there is no excuse for it wrong. Fear gone? Join our special newsletter! Newsletter!

amc 8.pdf , 6181524.pdf , passaic valley high school gymnastics , phimvn2 tvb 1998 , kifubavow-tuwumitufuzeniw-gosunugezebuku-boxuzup.pdf , hindustan unilever limited products list.pdf , download photomath apk , acatech industrie 4.0 maturity index.pdf , d29e4dfdc97213.pdf , 4056517.pdf , number tracing worksheets 11-20.pdf , 2474793.pdf , pac man ce dx apk free download ,