Ionic 5 and Angular 8 Concepts

In the Course In this documentation we will walk through all the concept that we learn in the course of ionic 5 and angular 8 and discuss it in detail.

1.1 Ng Module

Ng Module is a class marked by the NgModule decorator. NgModule takes a metadata object that describes how to compile a component's template and how to create an injector at runtime. It identifies the module's own components, directives, and pipes, making some of them public, through the export's property, so that external components can use them. NgModule can also add service providers to the application dependency injectors.

Example

For an example application showcasing all the techniques that Ng Modules related pages cover

1.2 Entry Component

An entry component is any component that Angular loads imperatively by type. A component loaded declaratively by way of its selector is not an entry component. Angular loads a component declaratively when using the component's selector to locate the element in the template. Angular then creates the HTML representation of the component and inserts it into the DOM at the selected element. These aren't entry components. The bootstrapped root App Component is an entry component. True, its selector matches an element tag in index.html. But index.html isn't a component template and the App Component selector doesn't match an element in any component template. Components in route definitions are also entry components. A route definition refers to a component by its type. The router ignores a routed component's selector, if it even has one, and loads the component dynamically into a Router Outlet.

1.3 Declarations

Declarations are to make directives (including components and pipes) from the current module available to other directives in the current module. Selectors of directives, components or pipes are only matched against the HTML if they are declared or imported.

1.4 Package. Json vs Package-lock json

Let's see the detail of package. Json and package-lock. Json in detail. package. Json If your project uses node package manager (NPM) you will have a package.json file somewhere in your code base. The package. Json file records the minimum version of different dependencies that your app needs. When a collaborator on the code does npm install the dependency versions installed will be those dictated in the package. Json or a higher/more recent reversion. If you update the versions of a particular package, the change is not necessarily going to be reflected here. The package. Json file is used for more than just dependencies. It also is used to define project properties, descriptions, and license information. Package-lock json Where the package. Json file is used for a handful of different things, the package-lock. Json file is solely used to "lock" dependencies to a specific version number, including minor and

patch versions. It will ignore the ^ and the ~ of the package. Json file. This file keeps track of the exact version of each installed package which means that future installs will be able to build an identical dependency tree. This is important in some large application spaces with many dependencies. Some dependency versions do not play well with each other, so making sure to "lock in" the versions prevent a lot of issues from occurring. This is especially useful when there are multitudes of individuals collaborating on one code base. In this way, collaborators who npm install 6 months apart will be looking at the same versions getting installed.

1.5 Subscribers

In Angular. subscribe () is a method on the Observable type. The Observable type is a utility that asynchronously or synchronously streams data to a variety of components or services that have subscribed to the observable. The observable is an implementation/abstraction over the promise chain and will be a part of ES7 as a proposed and very supported feature. In Angular it is used internally due to rxjs being a development dependency. An observable itself can be thought of as a stream of data coming from a source, in Angular this source is an API-endpoint, a service, a database or another observable. But the power it has is that it's not expecting a single response. It can have one or many values that are returned.

1.6 Observables

Angular makes use of observables as an interface to handle a variety of common asynchronous operations. For example: You can define custom events that send observable output data from a child to a parent component. The HTTP module uses observables to handle AJAX requests and responses. The Router and Forms modules use observables to listen for and respond to user-input events.

1.7 RxJS library

Reactive programming is an asynchronous programming paradigm concerned with data streams and the propagation of change. RxJS (Reactive Extensions for JavaScript) is a library for reactive programming using observables that makes it easier to compose asynchronous or callback-based code. See (RxJS Docs). RxJS provides an implementation of the Observable type, which is needed until the type becomes part of the language and until browsers support it. The library also provides utility functions for creating and working with observables. These utility functions can be used for: Converting existing code for async operations into observables Iterating through the values in a stream Mapping values to different types Filtering streams Composing multiple streams.

1.8 Dependency injection

Dependencies are services or objects that a class needs to perform its function. Dependency injection, or DI, is a design pattern in which a class requests dependencies from external sources rather than creating them. Angular's DI framework provides dependencies to a class upon instantiation. Use Angular DI to increase flexibility and modularity in your applications.

Example 1.9

Normal http vs Native http In this phase we will see the difference between native http and normal http. Native http Making Native server calls over provides advantages over traditional JavaScript HTTP calls. Native HTTP calls are made in Android and IOS based native applications. It provides more security and better calls handling features. Some of its advantages are as follows: Background threading: All requests are done in background thread like native applications. Handling of HTTP code 401: This error code 401 is returned during an unauthorized access to the server which is not handled properly using JavaScript-based HTTP calls. SSL Pinning: Pinning is used to improve the security of a service. It is used to create a secure Identity for making communications. Normal http HTTP messages are how data is exchanged between a server and a client. There are two types of messages: requests sent by the client to trigger an action on the server, and responses, the answer from the server. HTTP messages are composed of textual information encoded in ASCII, and span over multiple lines. In HTTP and earlier versions of the protocol, these messages were openly sent across the connection. In HTTP the once human-readable message is now divided up into HTTP frames, providing optimization and performance improvements. Web developers, or webmasters, rarely craft these textual HTTP messages themselves: software, a Web browser, proxy, or Web server, perform this action. They provide HTTP messages through config files (for proxies or servers), APIs (for browsers), or other interfaces.