# 1. Introduction to angular.

Angular is a framework for building client applications in html, css, js/ts

Why do we need angular?

* With complex applications, vanilla js becomes hard to maintain
* Need a way to structure the application properly
* Hard to test
* Angular gives applications a loose and clean structure, lots of utility, increases testability

Front End UI: HTML, CSS,TypeScript, Angular, sits on the client’s computer

Backend: On the web server

data and processing from the back-end

Backend includes APIs and databases

We need to install node, before we can start working with angular.

Node is a runtime environment for executing js code outside the browser.

Node provides tools/3rd party libraries, including the very important:

angular CLI, helps us efficiently work with angular.

Files and folders inside the application:

e2e: end to end, where we write end-to-end tests for applications.

Automated tests simulate a real user, for example: write a couple of commands, click some buttons, and then assert that there’s something on the page.

After that, in the file structure, we have node modules: self-explanatory: all node modules/libraries that we’re using in our application.

Here is a layout of what comes after node modules:

src:

apps:

modules: All modules go here

components

assets: static files go here

environments:

// config settings for product environment

// config settings for development environment

favicon: icon displayed in the browser

index.html: HTML file that contains the angular application

// this HTML file doesn’t contain static references, we’ll use dynamic references to css/js code

main file: typescript file, starting point of application, all we’re doing here is bootstrapping the main module

polyfills file: imports scripts required for running angular, angular framework uses features of javascript, not supported by most browsers

styles.css: add global styles for applications, each component can have its own style

Can contain multiple libraries and applications.

A single workspace configuration file called angular.json is created at the top level.

You can either use ng config or edit the angular.json file directly if you want to make major config changes.

Package.json: standard file that every node project has.

Dependencies determine the libraries the application is dependent upon.

protractor.config.js: a tool for running end-to-end tests.

Tsconfig.json: a bunch of settings for ts compiler, the compiler looks at these settings and compiles ts code to js that browsers can understand.

tslint: a static analysis tool for typescript, readability, maintainability, etc.

Anything you change, webpack recompiles applications and refreshes bundles

this is known as hot module replacing

All style sheets are compiled into a javascript bundle.

File structure example:

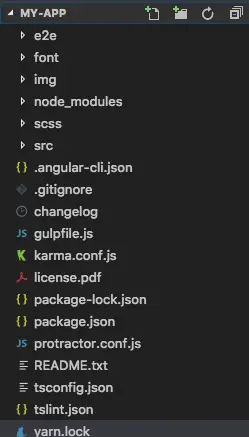


Image Credit: https://mdbcdn.b-cdn.net/wp-content/themes/mdbootstrap4/content/en/angular/tutorials/angular/5/img/01.webp

——————————————————————————————————————————

# 2. Introduction to TypeScript:

Typescript is not an entirely new language, it’s a superset of javascript, it has additional features not supported by most browsers before.

(Strong) typing is optional,

classes, interfaces, constructors, access modifiers, generics etc,

i.e. Object-oriented features

Compilation step is also involved as browsers can’t understand TS.

——————————————————————————————————————————

# 3. Components in angular.

1. Data

2. HTML template

3. Logic

* Reuse components in the same webpage
* Work on smaller, more maintainable, reusable pieces
* Every app has at least one component:
  + App component/root. component

a real-world Angular app is a tree of components starting from the app/root component

app

/ | \

Navbar sidebar courses

|

physics

|

chemistry

|

math

Module: container for a group of related components.

As the application grows, break module into smaller, more maintainable modules.

Modules are less abstract.

So maybe sth like courses, messaging, instructor, admin.

As an analogy, think of a supermarket, with different products, the bigger the supermarket, the more ailes.

A simple application: only one module, as the application grows in functionality, break the module into smaller modules-

bunch of related components.

How to create a component with Angular CLI?

ng generate component <component-name>



——————————————————————————————————————————

# 4. Modules in angular.

Modules are like the building blocks of an angular application. These are logical units that an application can be separated into. For example, if clients and admins work on the same CRUD application, making separate modules for each is probably a good idea. Every component needs to belong to some module. The default module is the App Module that hosts the app component. This is the module that gets bootstrapped by default. This, however, can be changed. To create a module using the angular CLI, use

ng generate module <module-name>.

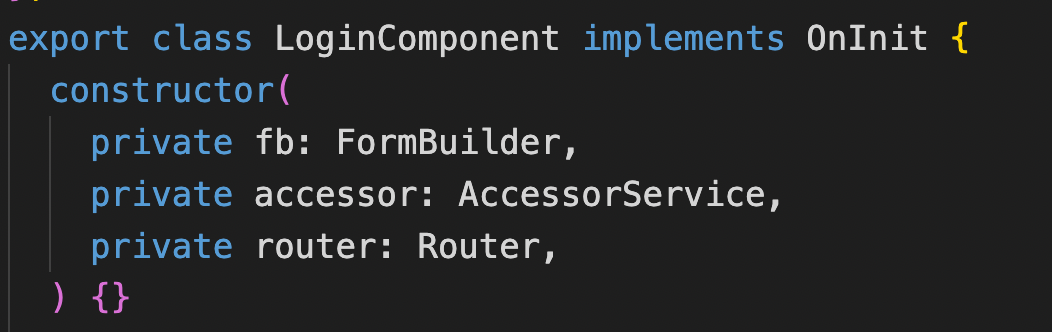
——————————————————————————————————————————

# 5. Services in angular.

In simple words, a service provides functionality to components. Usually components are used for design/view-formatting purposes, whereas services are used to provide some function that can be used by any component.



This is an example of a simple service that makes an HTTP request to get the list of admins. Services are identified by the *@Injectable* decorator. To inject a service into a component, declare it as a private class property in the constructor of the component.



Here, accessor is the service that is injected into the login component.

——————————————————————————————————————————

# 6. Data-Binding, Directives, Pipes

**Data-binding:**

Data-binding happens in 3 ways:

1. Event binding: Listen and respond to user actions such as clicks, scrolls, etc.

For example:



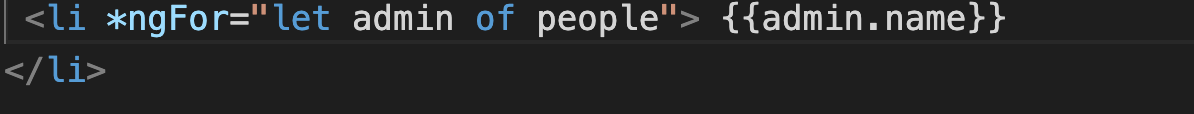
Or in other words, after clicking this button, call the onClickDel method, passing in the id of the admin.

1. Property Binding: Helps set properties of HTML elements. For example:



Here, we’re binding the routerLink property to a path that includes the ID of the admin, given by the class property currentID.

1. Interpolation: When a class property needs to be displayed in the view. Use {{ }} to interpolate class variables.

Example: 

Or in other words, for every admin, display admin.name.

**Directives:** Directives give additional funcitonality to HTML elements.

Example:



Here, we’re using the ngIf directive to only display the error message if the field has been touched and the value is invalid.

**Pipes:** Pipes are used for formatting text and displaying them in a particular layout, usually used for displaying date/time, currency, etc.

——————————————————————————————————————————

# 7. Angular Forms.

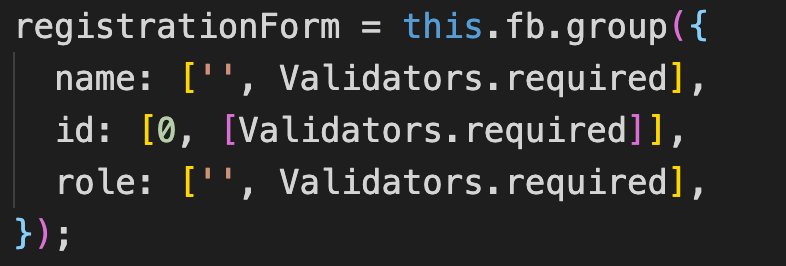
There are two types of forms in angular.

1. Template-driven forms.
2. Reactive forms.

Reactive forms vs Template-driven forms: In reactive forms, form control and validation happen inside the component class, whereas in template-driven forms, form control and validation happen in the template (hence the name template-driven). Although each type of form design has its own benefits, usually reactive forms are used for large-scale applications and come with a lot of built-in functionality.

Declaring forms using FormBuilder:

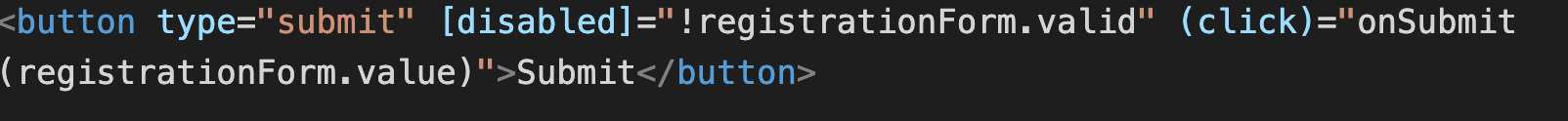
1. Import the reactive forms module.
2. Inject the FormBuilder service.
3. Create the form using the Form Builder’s group method:



1. Link the form to the template using the [formGroup] directive. Example:

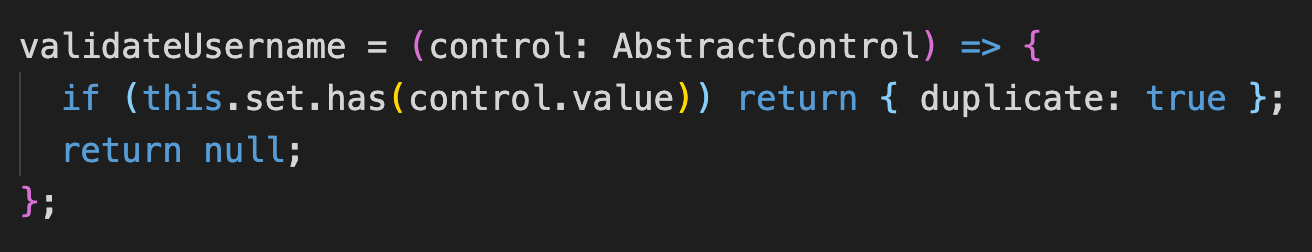


Validation: Validation happens in angular with the help of validator functions. In the image above, all fields have the required validator, which ensures that the field isn’t empty. For a form to be valid, all fields must be valid. If any field is invalid, the entire form is invalid. The submit button can be disabled using the form’s valid property. Example:



Here, registration form is the form group formed by using the form builder service.

Custom Validation Functions: Angular also supports the use of custom validation functions. The output of the function should be null, if the input is valid, else it should return an object that somehow explains what is wrong. For example:



It is always good practice to make **arrow validator functions** in order to avoid undefined errors. Any validator functions go into the array in the form group declaration. To access a particular error, use *this.form.get(‘<name-of-control>’)?.errors?.[‘object-attribute].* Here, for example, we would use this.form.get(‘username’)?.errors?.[‘duplicate’]. It always is good practice to create getter functions for the form controls that return this.form.get(‘control-name’). For example:



This way, getting access to the form control is much simpler and the template is going to be much cleaner.

——————————————————————————————————————————

# 8. Routing

Configuring routes in angular:

Import the router module first. Then follow these steps.

1. Configure the routes in the module.



Here: **path**: the desired path (: indicates a dynamically given link).

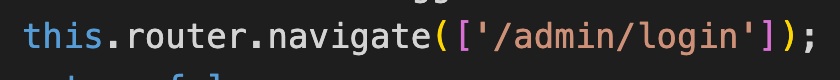
**component**: the required component.

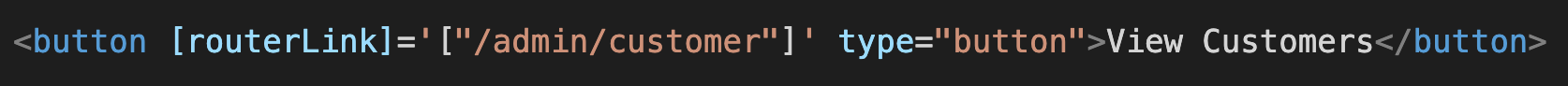
**canActivate:** any guards.

2. Add a router outlet.



3. Add links.





There are 2 ways of linking to a route in angular:

1. In the component: using this.router.navigate(url-object).

url-object: [link, dynamic-variables].

1. Using the [routerLink] directive in the template itself.

RouterModule.forRoot() vs forChild:

forRoot for the root module vs for child for any other modules.

forRoot once in the application vs forChild many times.

In each module we’re gonna have a set of routes.

For not found: path, use ‘\*\*’ (wildcard matching) at the bottom of the declaration array, as it matches from top to bottom.

**NEVER USE HREF!**

Why not? The entire page is downloaded and the angular app is reinitialized. Defeats the purpose of the single-page application.

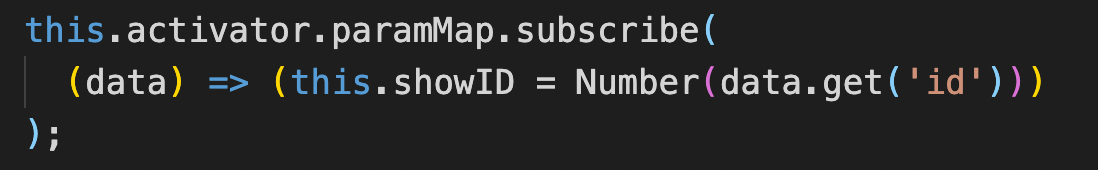
Use directive called routerLink instead of href.

How to get access to route parameters?

Activated route service:

Need to subscribe to paramMap: map of all parameters.

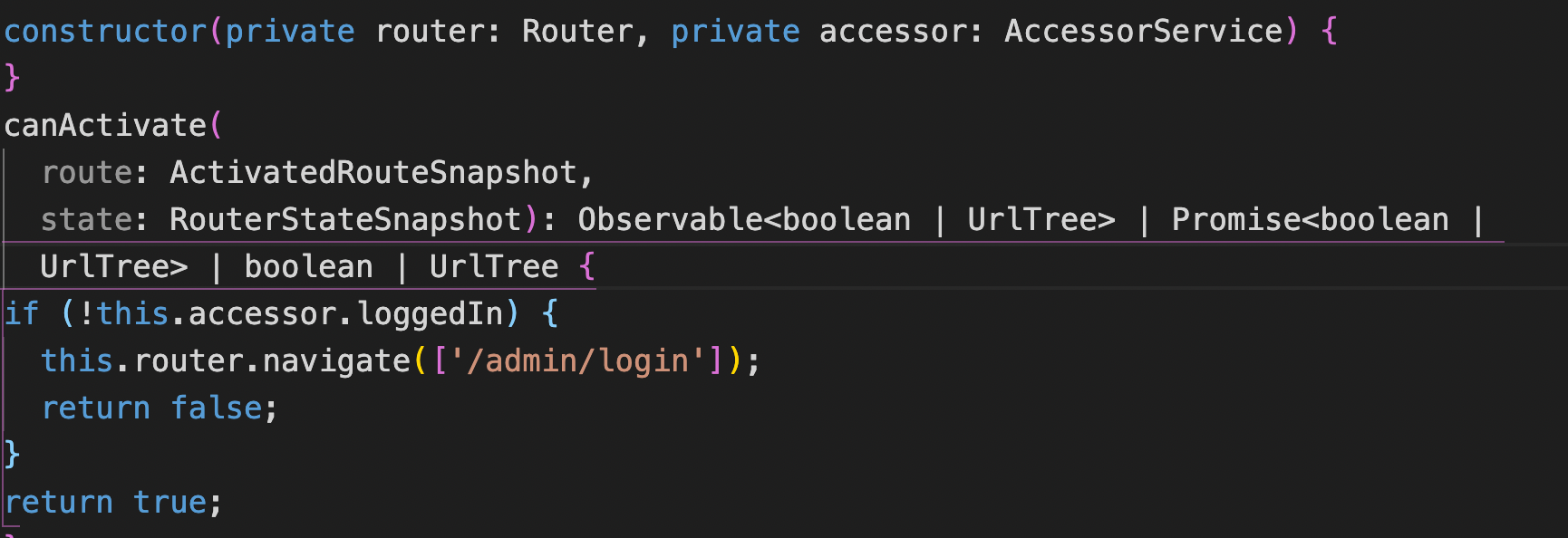
Example:



Clearly, this is an asynchronous operation as we’re subscribing to an observable.

**Guards:**

Use ng generate guard <guard-name>. Don’t forget to mention it using the canActivate property in the router declaration.



This simple guard returns true or false depending on whether the user is logged in.

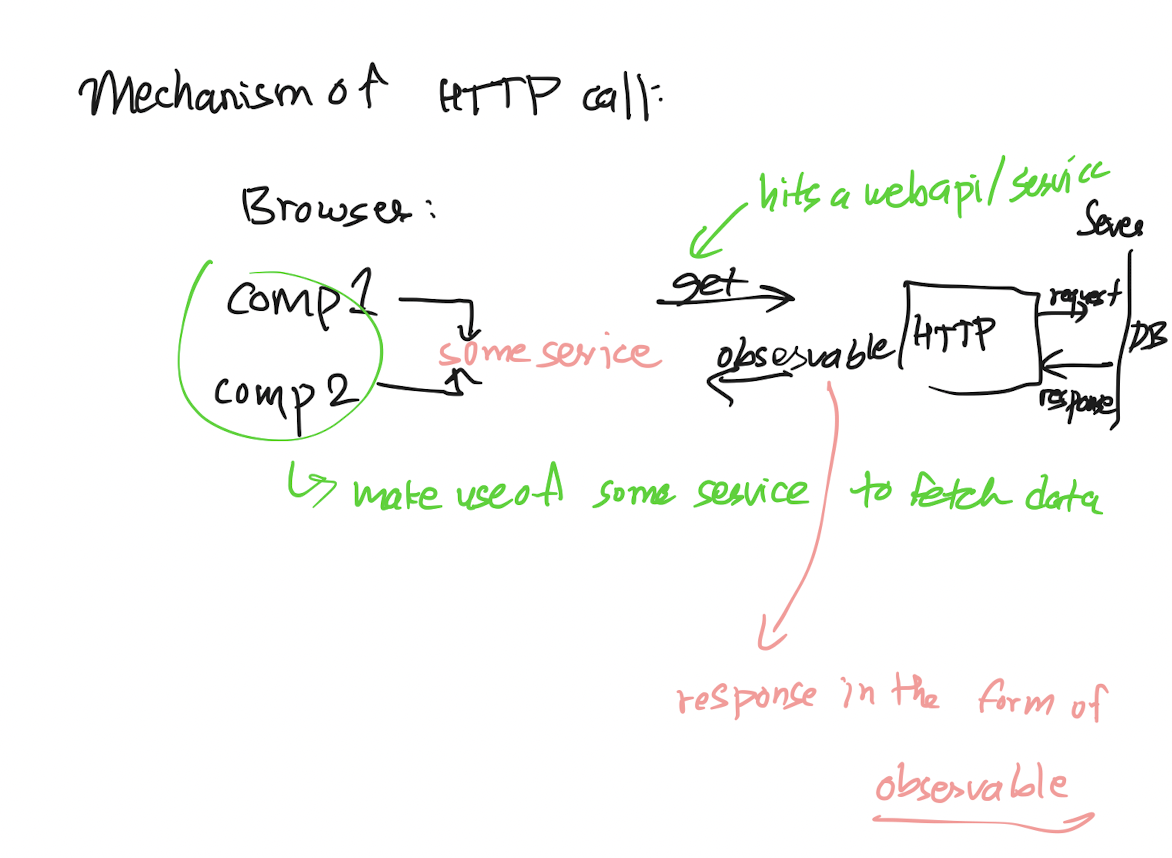
——————————————————————————————————————————

# 9. HTTP Service and Observables:

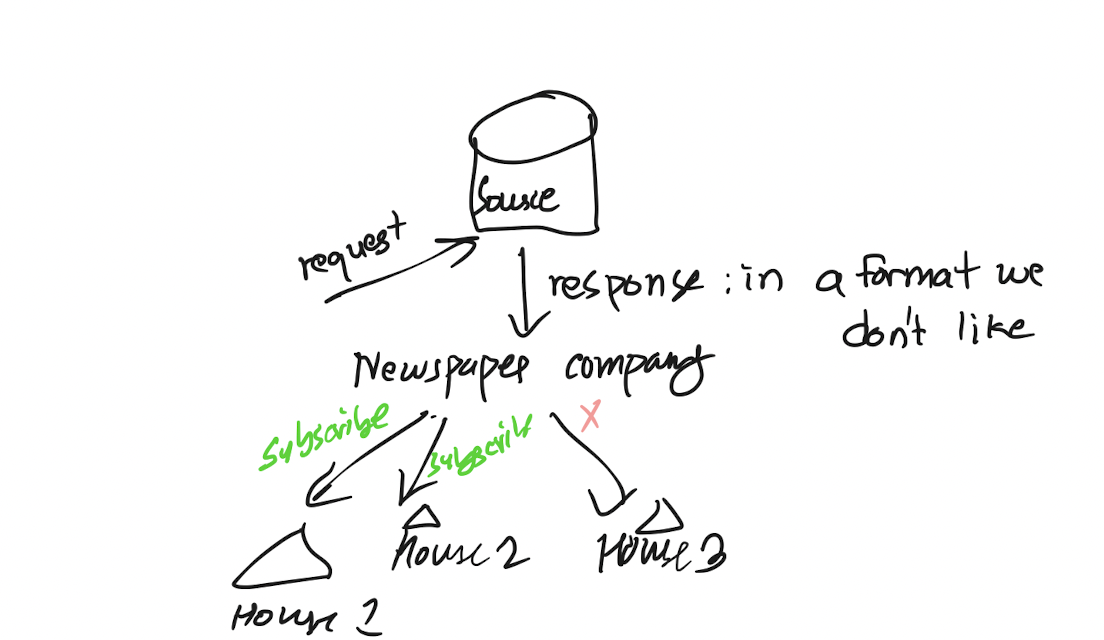
HTTP services:

1. GET
2. POST
3. PUT
4. DELETE

HTTP Calls are made using the **HttpClient** service in angular.



So, what’s an observable? Think of HTTP services using the newspaper company analogy:

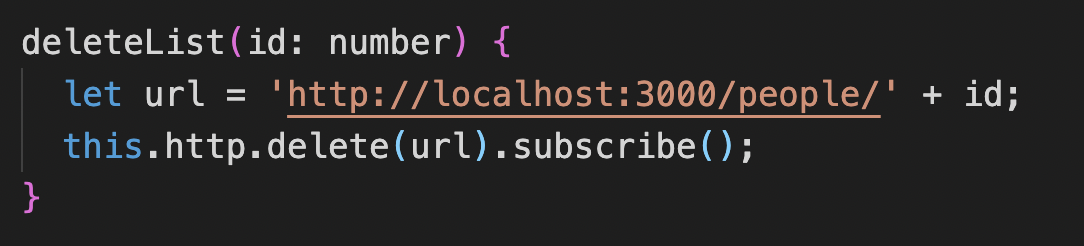


The newspaper company gets news from a source. Obviously, if it sends the news as is, customers won’t be happy. Customers prefer reading the news in nice, well-formatted newspapers. The HTTP response is the raw news that gets sent from the source. Then the newspaper company converts the raw news into newspapers, or in this case, **observables.** It sends the newspapers **only to houses that are subscribed.** So, we need people to subscribe to the news. Else, if no one is subscribed to the company, no newspaper will be sent out.

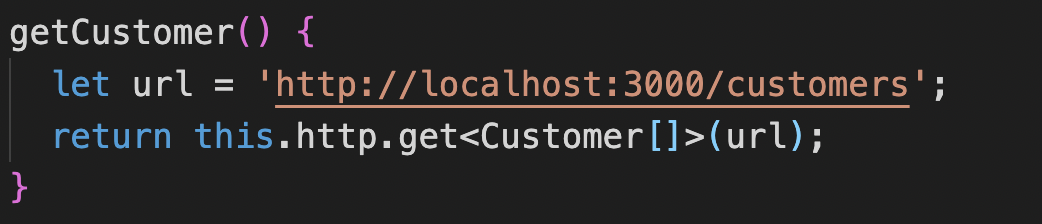
**Hence, whenever we make an HTTP request, we always want to subscribe**.

RXJS (Reactive extension for JS) is a library that helps us work with newspapers in angular.

Subscribing to an observable:



Subscribing to it directly. As mentioned above, subscribing to an observable is asynchronous.



Return the observable and subscribe to it later on in the component class (where we assign the data to a variable).

——————————————————————————————————————————