## Tour of Heroes tutorial:

## Visual Studio 2019 setup:

You can create a Visual Studio solution and create an ASP.NET Core Web application that uses the .NET Core. You can run Angular inside of Visual Studio now.

You can add your solution to the git repository by first initializing the new repo and using the following at the command prompt:

git init

git remote add origin git@github.com/keenanstewart/keenSTEM

The command prompt using git is more informative than using Team Explorer in Visual Studio.

You can go into Branches in Team Explorer to view the branches you created to see what is in and what is being tracked for each branch.

<https://www.talkingdotnet.com/create-an-angular-8-app-with-visual-studio-2019/>

To run a nuget package restore, run the following from the command line or package manager?:

Tools, NuGet Package Manager, Package Manager Settings. Click on General under NuGet Package Manager and both options under Package Restore should be checked.

You can run the following command at the command prompt to restore any packages indicated in the .csproj file:

dotnet restore

### You can use command line to commit the changes:

git init

git add .

git commit -m "first commit"

git remote add origin git@github.com:keenanstewart/keenMVCAngularSolution.git

git push -u origin master

### Issues connecting to git:

Try to ssh to connect to your repository. The example uses ssh, so you can use putty or just at the command prompt where you perform git commands.

Test connection

ssh –T keenanstewart@github.com

Permission denied (public key)

ssh-add –l –E md5

check to see if agent is running :

ssh-add –l

If you receive an unable to start ssh-agent service, error 1058, ensure the OpenSSH Authentication agent is enabled in Windows services.

To generate a public/private key pair, you will need to go to your .ssh directory and run the following command (c:\Users\keenan.stewart\.ssh)

ssh-keygen

Name your RSA key pair by pressing enter to save it as default name

Copy it to your clip board to provide to github:

clip < id\_rsa.pub

Open github.com

Tutorial on:

<https://angular.io/guide/setup-local>

### Node version

Brings back the current version of node installed:

node -v

### Create a new ng app:

ng new keenSTEM

### Start app once it is created:

ng serve --open

### Debugging site:

<http://localhost:4200/>

## NG Serve it up:

cd c:\Work\Projects\Angular\keenSTEM\keenSTEM

ng serve –open

<http://localhost:4200/dashboard>

### Type Script errors:

"compileOnSave": false,

"compilerOptions": {

"experimentalDecorators": true,

"baseUrl": "./"

}

Neat code

Dashboard component

When setting up navigation to a new with an object to display, such as login or heroes, it will only display is there is something in that object so you do not need to test for value in an object as this is handled automatically.

You can remove dead code that is no longer used, but I sometimes leave this code in for review or at least document or check in your changes so you could revert if necessary.

Adding imports to hero-detail.component.ts

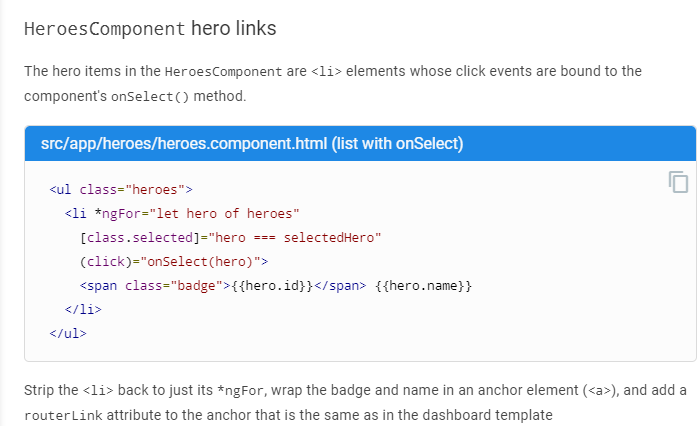
@angular/router contains an ActivatedRoute to help you keep track of your breadcrumbs

Location is an angular service for interacting with the browser.

Need to get HeroComponent and LoginComponent synced.

Left off here, about to synch LoginComponent:

<https://angular.io/tutorial/toh-pt5>



# Got files synced, about to proceed:

When initializing a class variable, such as a custom Message, you will need to use let (not var):

let messageToLog: Message = {

callingPage: “HeroDetailComponent,

message: message,

};

Can’t get laptop to sync with TFS/GitHub in Team Explorer. May need to explore that tomorrow.

Was able to commit locally, place to start off tomorrow up in Visual Studio.

InMemoryDataService can be used to simulate a database without needing to connect to an actual data store. When accessing the InMemoryDataService, you use syntax as follows:

api/heroes

api/logins

All classes need an id created, it will not work with loginId.

To create an in memory database, you will need to update the genId with the table name, such as:

## Partial <T>

There are times when creating a new object, you may not know the required new id for the new object. You can use the Partial<T> to only provide the name of the hero and the genId will be used to number the next available ID to use. I had to update the AddHero in the hero.service.ts file to use a partial hero as follows:

<code>

/\*\* POST: add a new hero to the server \*/

addHero(hero: Partial<Hero>): Observable<Hero> {

// TODO: logic in adding heroName

this.log(`In addHero: heroeName is: ${hero.name}`);

return this.http.post<Hero>(this.heroesUrl, hero, this.httpOptions).pipe(

tap((newHero: Hero) =>

this.log(`added hero with string id=${newHero.id}`)),

catchError(this.handleError<Hero>("Error in addHero")));

}

</code>

I also updated the heroes.component.ts to call the heroService.ts as follows:

<code>

addHero(hero: string): void {

this.log(`In addHero of HeroesComponent, name is: ${hero}`);

//hero.name = hero.name.trim();

if (!hero)

{

this.log(`In if(!hero.name) of add hero: ${hero}`);

return;

}

this.heroService.addHero({ name: hero } as Partial<Hero>)

.subscribe(hero => {

this.heroes.push(hero);

});

}

</code>

## subscribe

If you do not subscribe to the service as seen in the code above, the service will not send the request to the server. As a rule, an Observable does not do anything until something subscribes to the service.

## ng statements

When using ngFor with observables, use the dollar sign ($) to indicate you are using an Observable, and not an array, such as:

<li \*ngFor=”let hero of heroes$ | async”>

You will need to use the | async so that the statement will subscribe to the Observable automatically so the subscription will not need to be completed in the component class. For Observables, you can use the ngOnInit method to set any needed variables. For a search, you can set a searchTerm private Subject<string>(); and put the search term in a Observable stream.

// Push a search term into the Observable stream.

searchHero(term: string): void {

this.log("In searchHero");

this.searchTerms.next(term);

}

A Subject is both a source of observable values and an Observable itself. You can subscribe to a Subject as you would any Observable.

You can also push values into that Observable by calling its next(value) method as the search() method does.

The event binding to the textbox input event calls the searchHeroes method:

<input #heroSearchBox id="hero-search-box" (input)="heroSearch(heroSearchBox.value)" />

Every time the user types in the textbox, the binding calls search() with the textbox value, a "search term". The searchTerms becomes an Observable emitting a steady stream of search terms.

The searchTerms simply takes the searchTerms and emitting an Observable steady stream of search terms.

## Injections:

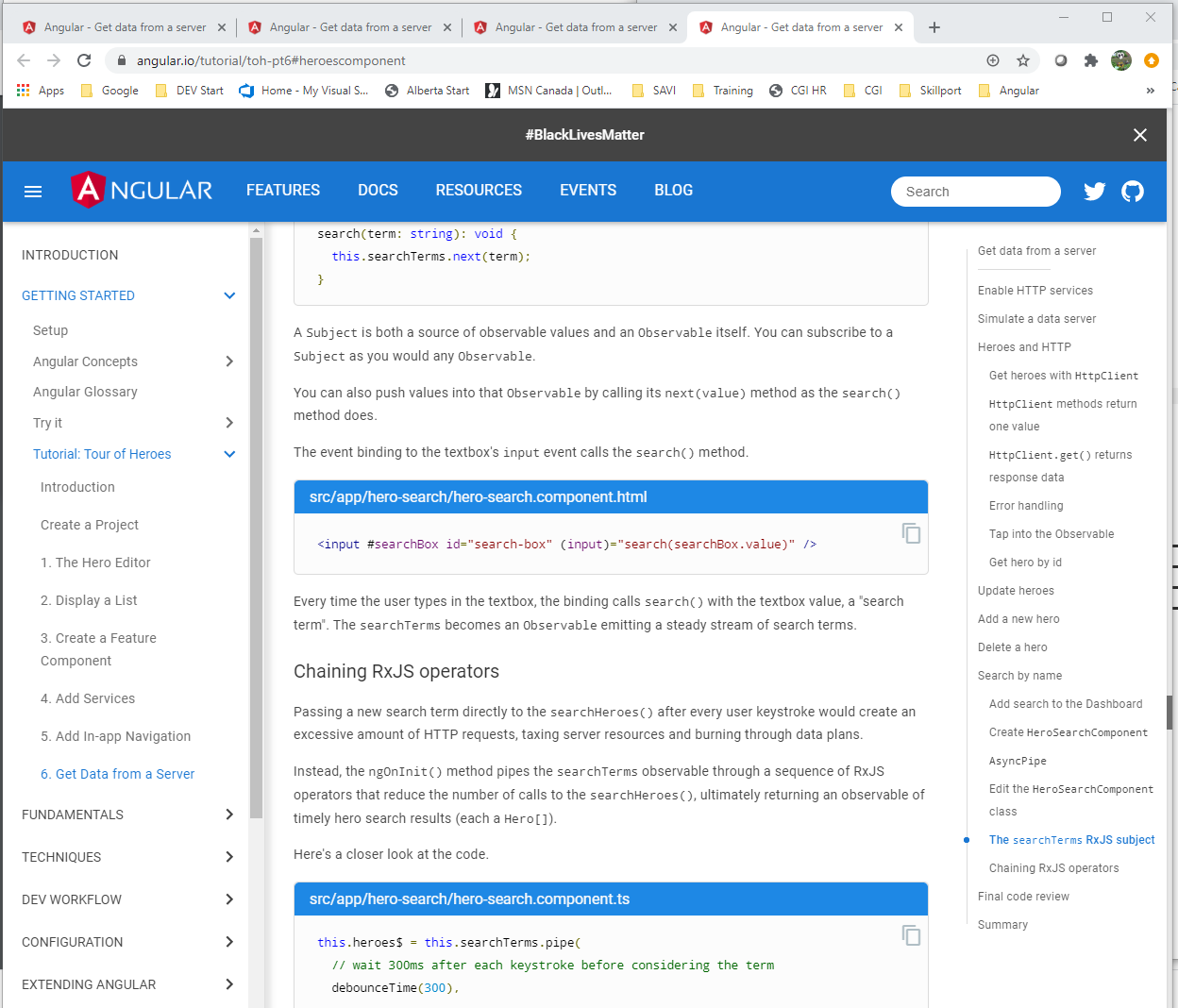
You can inject a class, such as a service into the constructor or the appropriate component by the following steps:

1. Import the services, such as a MessageService
   1. import { MessageService } from "../services/message.service";
2. and then inject the required services being used:
   1. constructor(messageService: MessageService) { }
3. You can now use this message service to log any notifications by creating a method and passing it the appropriate message. I customized my message service to take an argument of the calling page so I can debug it easier. Here is the code I used:
   1. I might be able to add the log method to the message service

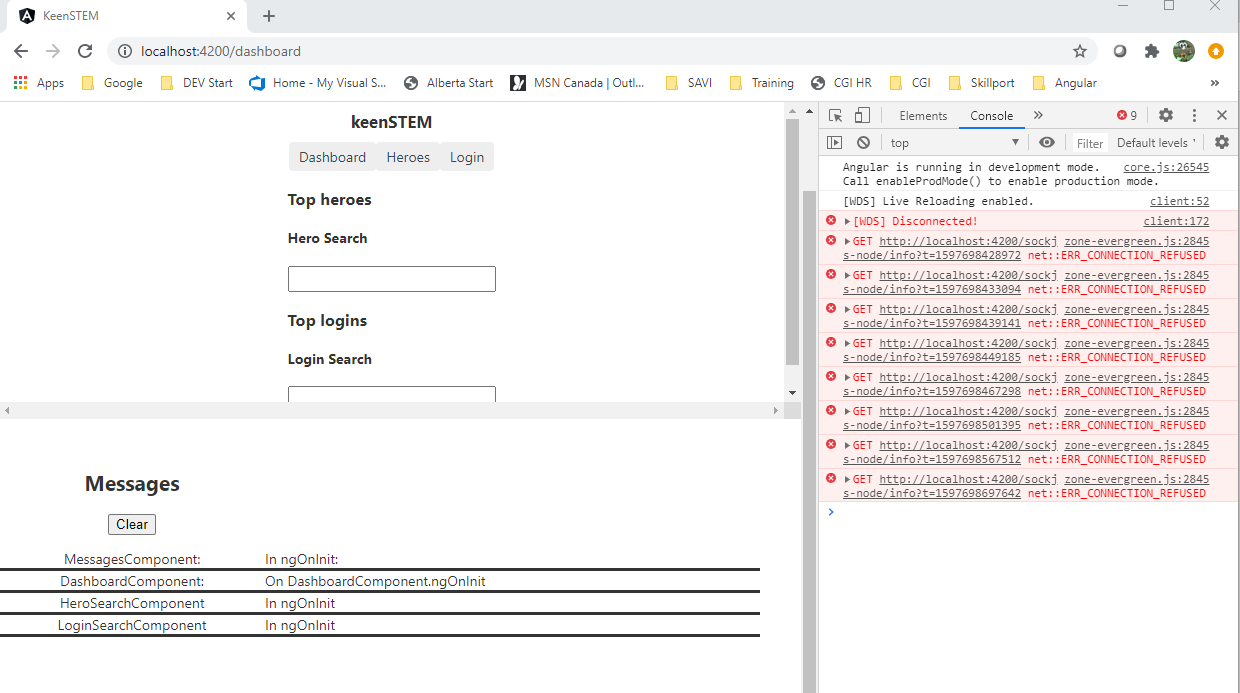
# Left off here:

Open Visual Studio on second monitor as well as this document to continue the Angular course at the following section: Trying to get LoginService to recognize I changed the code in the this.log file, needed to restart.

<https://angular.io/tutorial/toh-pt6#heroescomponent>



Bring up the test site:



## RESTful APIs

# Angular concepts:

Angular is a platform and framework for building single-page client applications using HTML and TypeScript. Angular contains core and optional functionality you can optionally import into your apps.

The basic building blocks of Angular is NgModules. NgModules contain related code into functional sets. Simply put, an Angular app is defined by NgModules with at least a root module that enables bootstrapping, and typically contains more feature modules.

## Components view and services

Components defines views that are used to modify your program logic and data and components use services that can be injected into components as dependencies, making your code modular, reusable, and efficient.

Modules, components, and services are classes that use decorators. These decorators mark their type that allows Angular to read the metadata associated with these decorators on how they can be used.

* The metadata for a component class associates it with a template that defines a view. This template combines ordinary HTML with Angular directives and binding markup that allows Angular to modify the HTML before rendering it for display
* The metadata for a service class provides the information Angular needs to make it available to components through dependency Injection (DI)

To navigate between views in an Angular app, Angular provides the Router service to help you define navigational paths among views. This router provides sophisticated in-browser navigational capabilities.

## Modules

Angular NgModules differ from and complement JavaScript (ES2015) modules. An NgModule declares a compilation context for a set of components that is declared to an application domain, a workflow, or a closely related set of capabilities. An NgModule can associate its components with related code, such as services, to form functional units.

Every Angular app contains a root module, conventionally called AppModule, which provides the bootstrap mechanism that launches the application. An app typically contains many function modules.

You can import and export NgModules into other NgModules to extend and use services in other NgModules. For example, to use the router services in your app, you import the Router NgModule.

NgModules allow lazy loading, which is loading modules on demand to minimize the code loaded during startup. This is accomplished by organizing code into distinct functional modules to help manage the development of complex applications, and in designing for reusability.

## Components

Each Angular application will have at least one component, the root component, that connects a component hierarchy with the page document object module (DOM). Each component defines a class that contains application data and logic, and is associated with an HTML template that defines the view to be displayed in the target environment.

The @Component() decorator identifies the class immediately below it as a component, and provides the template and related component-specific metadata.

Decorators are functions that modify JavaScript classes. Angular defines a number of decorators that attach specific kinds of metadata to classes, so that the system knows what those classes mean and how they should work.

## Templates, directives, and data binding

A template combines HTML with Angular markup that can modify HTML elements before they are displayed. Template directives provide program logic, and binding markup connects your application data and the DOM.

* Event binding allows your app to respond to user input in the target environment by updating your application data
* Property binding lets you interpolate values that are computed from your application data into the HTML.

Before a view is displayed, Angular evaluates the directives and resolves the binding syntax in the template to modify the HTML element sand the DOM, according to your program data and logic. Angular supports two-way data binding, meaning that the changes in the DOM, such as user choices, are also reflected in your program data.

Angular pipes can be used to further improve the user experience by transforming values for display. For example, use pipes to display dates and currency values that are appropriate for a user’s locale. Angular provides predefined pipes for common transformations, and you can define your own pipes.

## Services and dependency injection

Fro data or logic that isn’t associated with a specific view, and that you want to share across components, you can create a service class. A service class definition is immediately preceded by the @Injectable() decorator. This decorator provides the metadata that allows other providers to be injected as dependencies in to your class.

Dependency Injections (DI) lets your keep your component classes lean and efficient. They don’t fetch data form the server, validate user input, or log directly to the console, they delegate such tasks to services.

## Routing

The Angular Router NgModule provides a service that lets you define a navigation path among the different application states and view hierarchies in your app. It is modeled on the familiar browser navigational conventions.

* Enter a URL in the address bar and the browser navigates to a corresponding page.
* Click links on the page and the browser navigates to a new page
* Click the browser’s back and forward buttons and the browser navigates backward and forward through the history of pages you’ve seen.

The route maps URL-like paths to views instead of pages. When a user preforms and action, such as clicking a link, that soul d load a new page in the browser, the router intercept the browser’s behaviour, and shows or hides view hierarchies.

If the router determines that the current application state requires particular functionality, and the module that defines it hasn’t been loaded, the router can lazy-load the module on demand.

The router interprets a link URL according to your app’s view navigational rules and data state. You can navigate to new views when the user clicks a button or selects from a drop box, or in response to some other stimulus from any source. The router logs activity in the browser’s history, so the back and forward buttons work as well.

To define navigational rules, you associate navigation paths with your components. A path uses a URL-like syntax that integrates your program data, in much the same way that template syntax integrates your browser with program data. You can then apply program logic to choose which views to show or hide, in response to user input and your own access rules.

Together, a component and a template define an Angular view.

* A decorator on a component class adds the metadata, including a pointer to the associated template.
* Directives and defining markup in a component’s template modify views based on program data and logic.

The dependency injector provides services to a component, such ash the router service that lets you define navigation among views.

## Introduction to modules

<https://angular.io/guide/architecture-modules>

Angular apps are modular and Angular has its own modularity system called NgModule. NgModules are containers for a cohesive block of code dedicate4d to an application domain, a workflow, or a closely related set of capabilities. They can contain components, service providers, and other code files whose scope is defined by the containing NgModule. The can import functionality that is exported from other NgModules, and export selected functionality for use by other NgModules.

Every Angular app contains at least one NgModule, the root module, which is conventionally named AppModule and resides in a file named app.module.ts. You launch your app by bootstrapping the root NgModule.

Wile a small application might only have only one NgModule, most apps have many more feature modules. The root NgModule for an app is so named because it can include a child NgModule in a hierarchy of any depth.

### NgModule metadata

An NgModule is defined by a class decorated with @NgModule(). The @NgModule() decorator is a function that takes a single metadata object, whose properties describe the module. The most important properties are as follows:

* declaration: The components, directives, and pipes that belong to this NgModule.
* exports: The subset of declarations that should be visible and useable in the component templates of other NgModules.
* imports: Other modules whose exported classes are needed by components templates declared in this NgModule
* providers: Creators of services that this NgModule contributes to the global collection of services; they become accessible in all parts of the app. (You can also specify providers at the component level, which is often preferred.)
* bootstrap: The main application view, called the root component, which hosts all other app views. Only the root NgModule should set the bootstrap property.

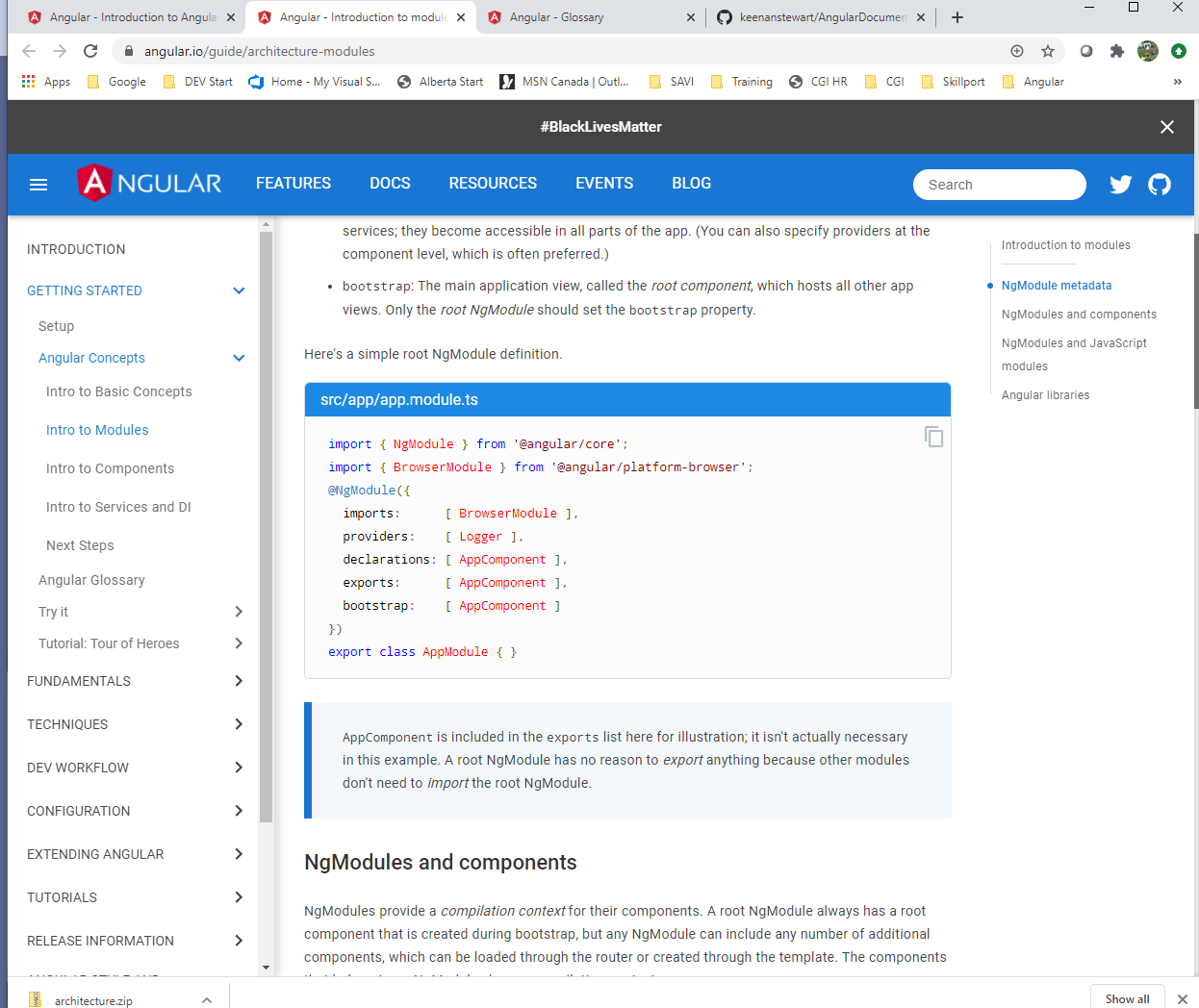
Here’s a simple root NgModule definition.

scr/app/app.module.ts

import

# Left off here:

<https://angular.io/guide/architecture-modules>



## Angular Glossary

### ahead-of-time (AOT) compilation

During the build phase, the Angular ahead-of-time (AOT) compiler converts Angular HTML and TypeScript code into efficient JavaScript code before the browser downloads and runs that code. This is the best compilation mode for production environments, with decreased load time and increased performance compared to just-in-time (JIT) compilation. By compiling your application using ngc command line tool, you can bootstrap directly into a module factory, so you don’t need to include the Angular compiler in your JavaScript bundle.

### Angular element

An Angular component packaged as a custom element.

## // TODO: define other Angular glossary terms

### Zone

A zone is an execution context for a set of asynchronous tasks and is useful for debugging, profiling, and testing apps that include asynchronous operations, such as event processing, promises, and calls to remote servers. An Angular app runs in a zone where it can respond to asynchronous events by checking for data changes and updating the information it displays by resolving data bindings. A zone client can take action before and after an async operation completes.

# Errors encountered:

Property id is missing. This message is from the in-memory-data.service.ts file:

Adding to this discussion, I had to use Partial as it kept on saying I was missing an id when completing the Tour of Heroes Lesson 06. I used the message service to see how far the code went before it stopped working, so you can ignore the this.log lines, but you can add them to your own project to help with debugging. First, I updated the addHero as follows:

addHero(hero: string): void { this.log(`In addHero of HeroesComponent, name is: ${hero}`);

if (!hero)

{

this.log(`In if(!hero.name) of add hero: ${hero}`);

return;

}

this.heroService.addHero({ name: hero } as Partial<Hero>)

.subscribe(hero => {

this.heroes.push(hero);

});

}

And then I updated the heroes.service.ts file to use the partial hero and allow the code to generate its own id from the in-memory-data.service.ts:

addHero(hero: Partial): Observable { this.log(`In addHero: heroeName is: ${hero.name}`); return this.http.post(this.heroesUrl, hero, this.httpOptions).pipe( tap((newHero: Hero) => this.log(`added hero with string id=${newHero.id}`)), catchError(this.handleError("Error in addHero"))); }

As far as I can see, it is now working as expected. There may be some more code to update, but this should help anyone with the following error message:

ERROR in src/app/heroes/heroes.component.ts:52:30 - error TS2352: Conversion of type '{ name: never; }' to type 'Hero' may be a mistake because neither type sufficiently overlaps with the other. If this was intentional, convert the expression to 'unknown' first. Property 'id' is missing in type '{ name: never; }' but required in type 'Hero'.