Appmodule loads appcomponent

Appcomonent has html template specified in index.html

Main.ts – bootstraps appmodule

dotnet watch run -- command to start the CLI from Command Promp

Problems running ng serve –

(Need to run the npm update after you make changes)

You need to do some changes in package.json

1. Go to package.json and modify "rxjs": "^6.0.0" to "rxjs": "6.0.0"
2. Then run npm update in your project

Every Angular file must have at least one app.module file.

This one is app.module.ts. Inside it’s decorated with @NgModule

App.module is loaded by main.ts file, by way of platformBrowserDynamic().bootstrapModule(AppModule).catch(err => console.log(err));

Angular.json – webpak setup

The only way to see observables is to subscribe to them.

ngOnInit – best place to get data – the constructor is too early

CORS – security for api, who’s allowed to use it, by way of header

\*ngFor – Structural directive, always precede with \*

**Command in terminal to install bootstrap and font-awesome**

npm install bootstrap font-awesome

creating the app with ng new --- a git repository is automatically created for you

ADD USER TO DATABASE

CREATE MIGRATION

dotnet ef migrations add AddedUserEntity

UPDATE MIGRATION

Dotnet ef database update

Why we should use the repository pattern

* Minimizes duplicate query logic
* Decouples application from persistence framework
* All DB queries in the same place
* Promotes testability

------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

In the Startup.cs file we tell the application about the IAuthRepository and AuthRepository classes.

services.AddScoped – service created once per request (this one used)

services.AddTransient – service created each time

services.AddSingleton – service created once.

Inheriting from

ControllerBase – without view support

DTO – Data Transfer Object – an object that carries between processes

(When you're working with a remote interface, such as Remote Facade (388), each call to it is expensive. As a result you need to reduce the number of calls, and that means that you need to transfer more data with each call. One way to do this is to use lots of parameters. However, this is often awkward to program - indeed, it's often impossible with languages such as Java that return only a single value.

The solution is to create a Data Transfer Object that can hold all the data for the call. It needs to be serializable to go across the connection. Usually an assembler is used on the server side to transfer data between the DTO and any domain objects.

<https://martinfowler.com/eaaCatalog/dataTransferObject.html>

)

Data Annotations

[ApiController] allows asp.net.core.mvc to infer where the data is coming from in terms of parameters for a method

Could use if (!ModelState.IsValid) inside to validate the request

**Section 3**

Token Authentications

JWTs – Json Web Tokens

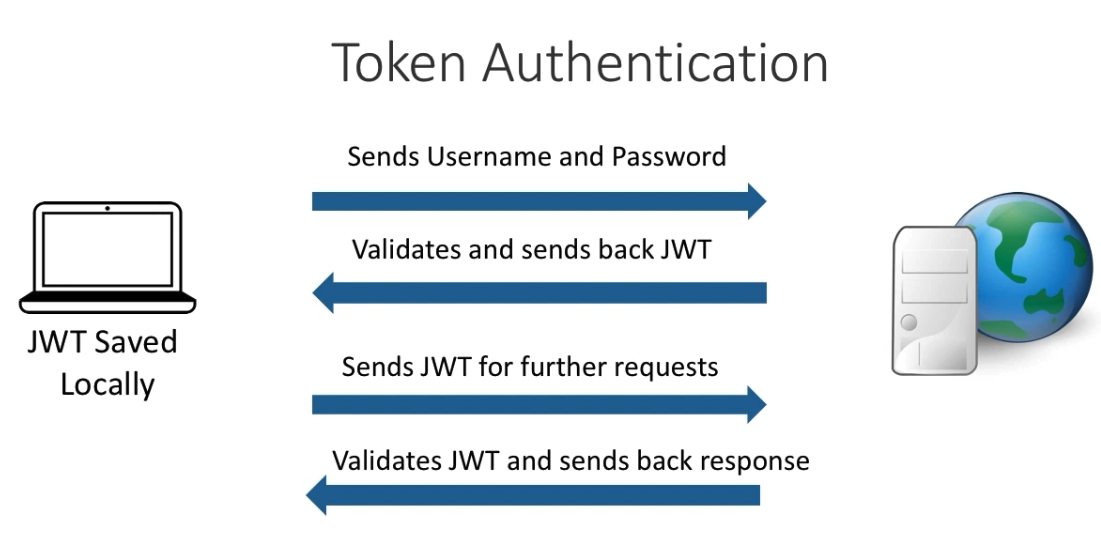
* Industry standard for tokens
* Self-contained and can contain:
  + Credentials
  + Claims
  + Other Information
* 

Nbf – not befor

Exp - expiration

Iat = issued at

Secret used to encode header and payload.



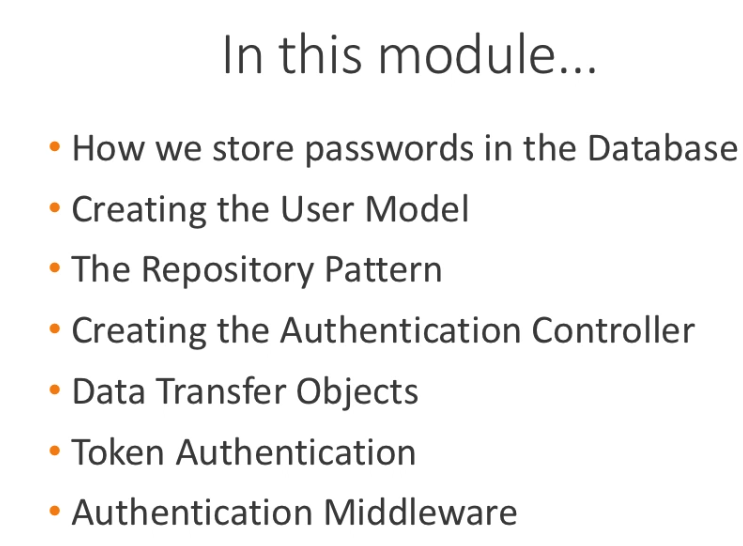
Site where you can check the token:

<https://jwt.io/>

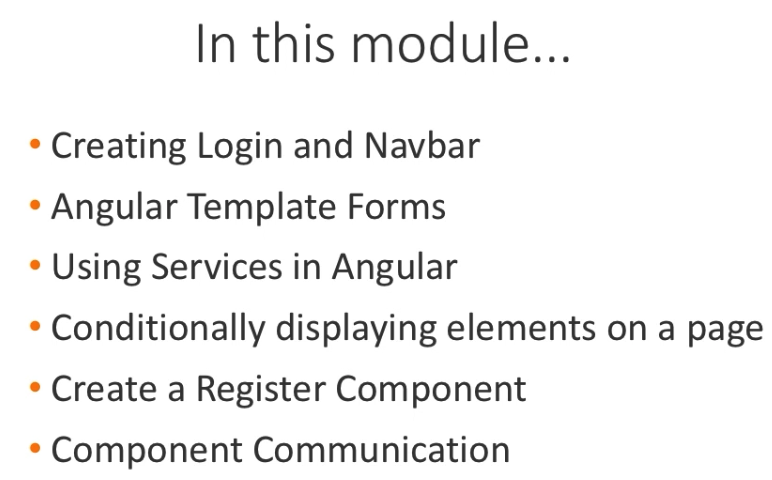
Git –

+ to stage changes, give name. Check √ to commit changes,

Covered in section 3



**Client Login and Register – Section 4**



* HTML forms can be turned into Angular forms.
  + Two types:
    - Template
    - Reactive

Template reference variables, use the # sign then assign the ng directive

#loginForm = “ngForm”

Validation – these were demo’ed, you can see the values as the inputs are entered

<pre>

FormStates angular keeps track of

Form Valid: {{ loginForm.valid}}

Form Touched: {{ loginForm.touched}}

Form Dirty: {{ loginForm.dirty}}

Form Values: {{ loginForm.value | json}}

Username Valid: {{ username.valid}}

Username Touched: {{ username.touched}}

Username Dirty: {{ username.dirty}}

Username Value: {{ username.value}}

Password Valid: {{ password.valid}}

Password Touched: {{ password.touched}}

Password Dirty: {{ password.dirty}}

Password Value: {{ password.value}}

</pre>

Components are injectable by default.

Service not component, so service needs

@Injectible({

providedIn: ‘root’

})

Root is app.module.ts

Add service to providers array

Must always subscribe to observables

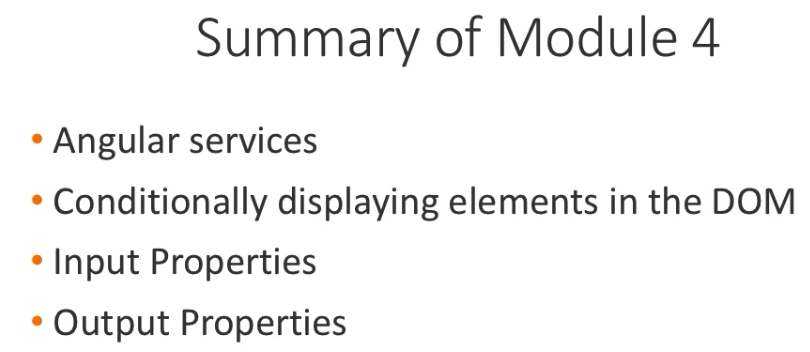
!!token – if value in token return true

\*ngIf – structural directive – directly change the dom (domain object model)

4 parts to Output in ts module

@Output cancelRegister = new EventEmitter();

(this will bring in import{ Component, OnInit, Input, EventEmitter, Output} from ‘@angular/core’;



Section 5 Error Handling – Creating Global exception handler

Properties\launchSettings.json –“environmentVariables”:{

“ASPNETCORE\_ENVIRONMENT”: “Development” (or “Production”

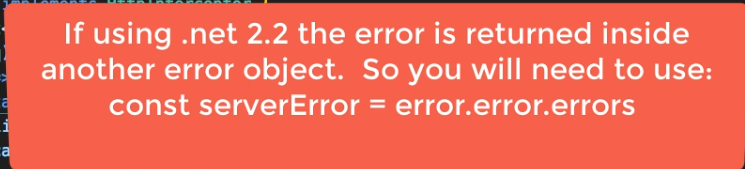
}

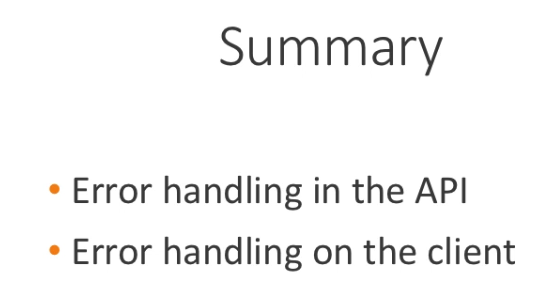
Placing

app.UseExceptionHandler();

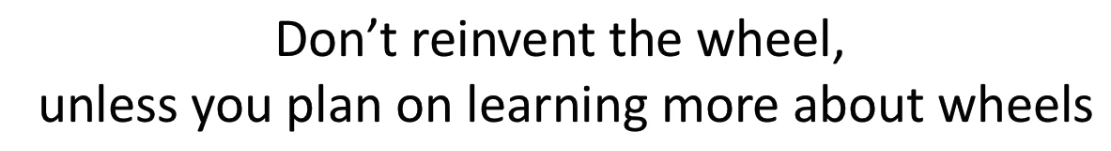
without further configuration in the Configure section of startup.cs, would generate 500 Internal Server errors, but would be handled globally rather than being unhandled.

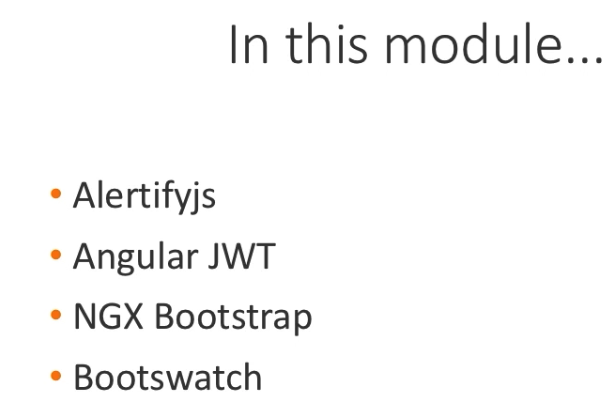
**Handling Errors in Angular**











**AlertifyJS**

Setup alertify as a service.

In DatingApp-Spa, in angular.json file – open “scripts”, add “node\_modules/alertifyjs/build/alerrtify.min.js”

In DatingApp-Spa in styles.css

Add two imports:

@import ‘../node\_modules/alertify/build/css/alertify.min.css’’

@import ‘../node\_modules/alertify/build/css/themes/bootstrap.min.js’;

In in the \_services folder, right click to generate a new service, name it ‘alertify’

In the alerrtify.service.ts file, we declare alertify

Declare let alertify: any;

Then under the constructor add the wrappers

confirm(message: string, okCallback: () => any) {

alertify.confirm(message, function(e) {

if (e)

{

okCallback();

} else {}

});

}

success(message: string) {

alertify.success(message);

}

error(message: string) {

alertify.error(message);

}

warning(message: string) {

alertify.warning(message);

}

message(message: string) {

alertify.message(message);

}

Then add to app.module.ts, under providers.

@NgModule({

declarations: [AppComponent, NavComponent, HomeComponent, RegisterComponent],

imports: [BrowserModule, HttpClientModule, FormsModule],

providers: [AuthService,

ErrorInterceptorProvider,

AlertifyService

],

bootstrap: [AppComponent],

})

Then in the nav component we can inject the service.

Private alertify: AlertifyService

**Whenever you change angular.json you must restart the server**

Installed @auth0/angular-jwt

<https://github.com/auth0/angular2-jwt>

Added JwtHelperService inside of auth.service.ts

This.jwtHelper.isTokenExppired(token) returns true if not found and if is expired so test with !this.jwtHelper.isTokenExpired(token);

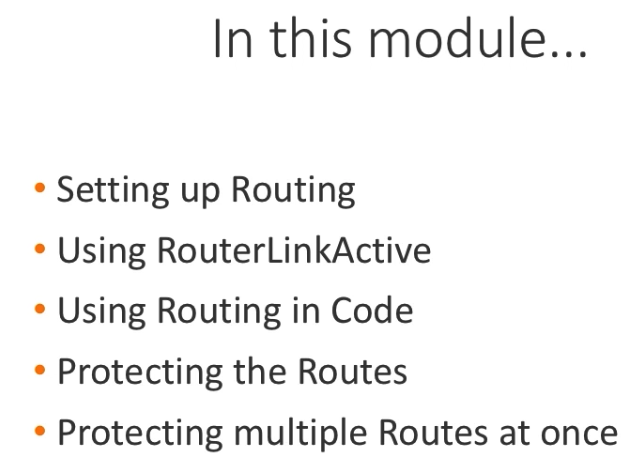
**ngx bootstrap**

Supplements bootsrap css to give app functionality without jquery.

Bootswatch – free themes for Bootstrap







Angular router is first match wins.

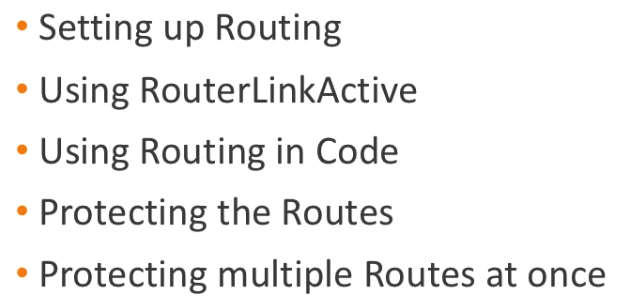
File created in app called routes.ts

This file needs to be added to app.module.ts

canActivate(): Observable<boolean> | Promise<boolean> | boolean

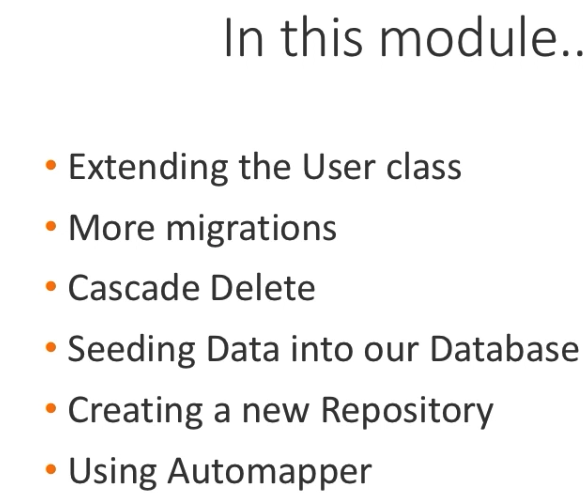
above line -- specifies different return types











Getting out of trouble with migrations

From terminal

dotnet ef migrations add ExtendedUserClass (ExtendedUserClass is name you give the migration)

dotnet ef migrations –help

dotnet ef migrations list

this create table in the migration uses onDelete: ReferentialAction.Restrict.

This would allow deletions from User, while leaving associated photos in the table.

dotnet ef migrations remove

removes the last created migrations that has not had update database performed against it

dotnet ef database update

To go back to earlier migration – you may be able to enter dotnet ef update XXX - where xxx is the last migration you want to go back to.

This may fail if using SQLite due to limitations in SQLite

You can drop the database and recreate it.

dotnet ef database drop

this will ask to confirm that it’s what you want

you can then drop the migration that caused the problem. Then using dotnet ef database update, all the migrations in the file will be applied

Doing this will remove all data from the database.

Defining relationships

Inside the Photo.cs class, add two properties

Public User User {get;set;}

Public int UserID {get; set;}

Then add the migration again. Now the table creation for Photos lists

onDelete: ReferentialAction.Cascade

Database Seeding

Once the seeding is setup, comment out the line in Startup.cs so that it’s not run every time the database is started.

<https://www.json-generator.com/>

to generate users for seeding

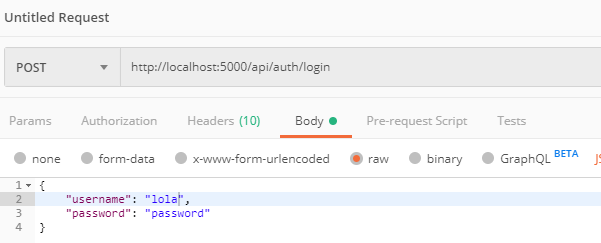
IDatingRepository

Public interface IDatingRepository{

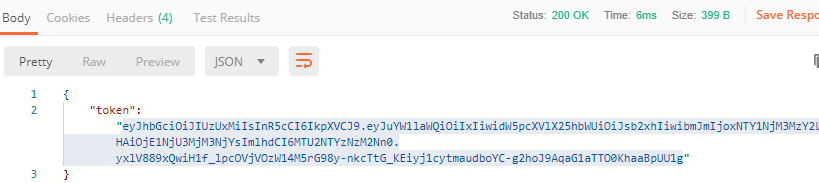
Void Add<T>(T entity) where T: class;

Adds genic entity, either user or photo.

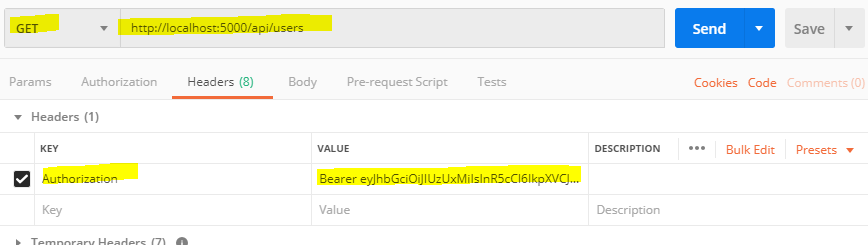
-------------------------------POSTMAN-----------------------------------------



Returns the token



New GET



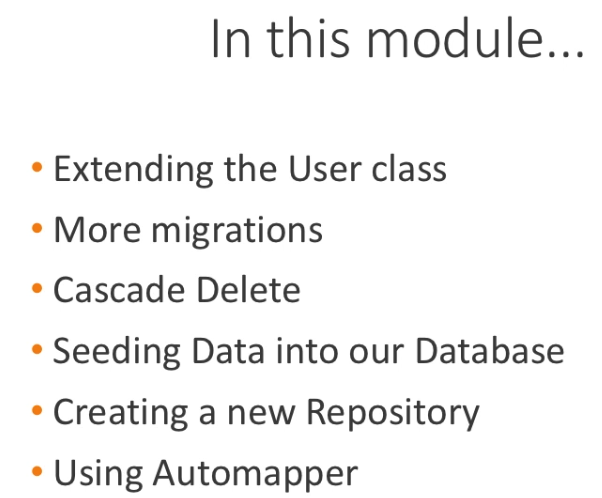
Select Authorization in the Headers, type ‘Bearer’ then paste the token in the value (without quotes) to test

**AutoMapper**

Use NuGet to install package into CORE. If using Visual Studio, NuGet is available. We added and extension Nuget at set-up on CODE.

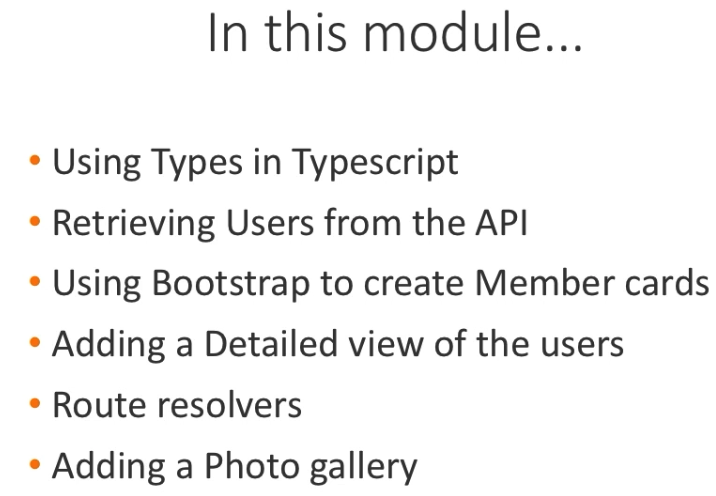
Ctrl + Shift + P for the command shell, search for Nuget: Add Package

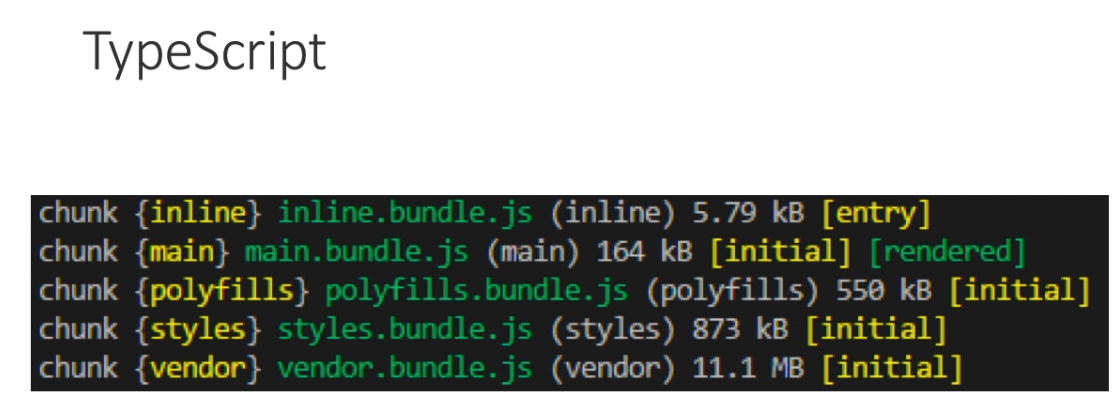








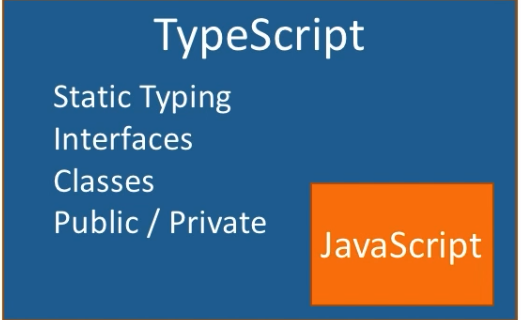


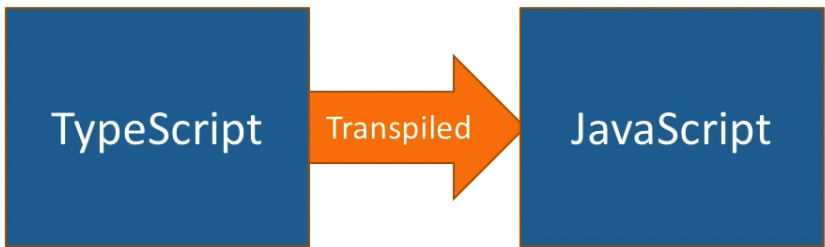


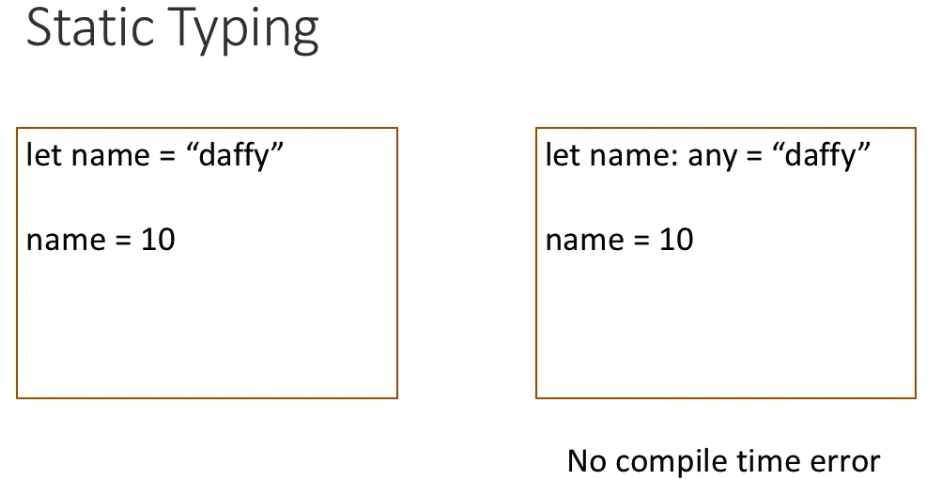
Angular CLI compiles code we write.

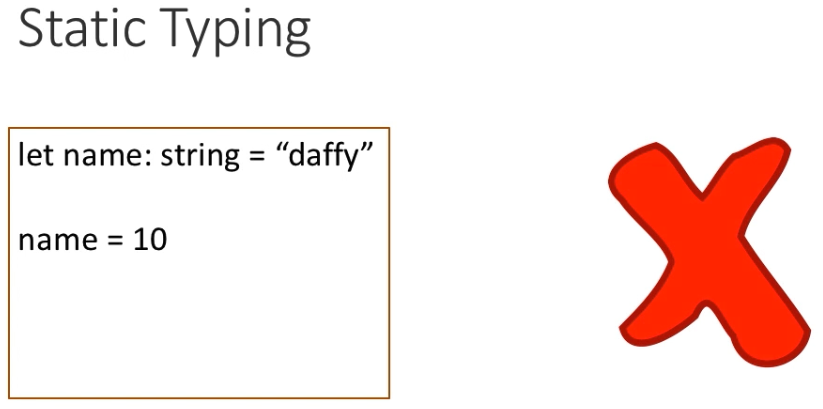
Nothing done in Javascript that can’t be done in TypeScript. The end result will be compile into javascript that the browser can understand.

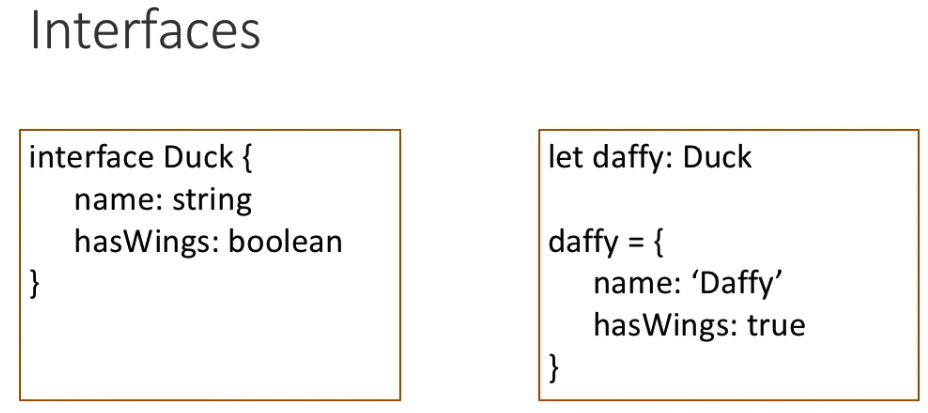
TypeScript is super set of Javascript that provides us with additional features.

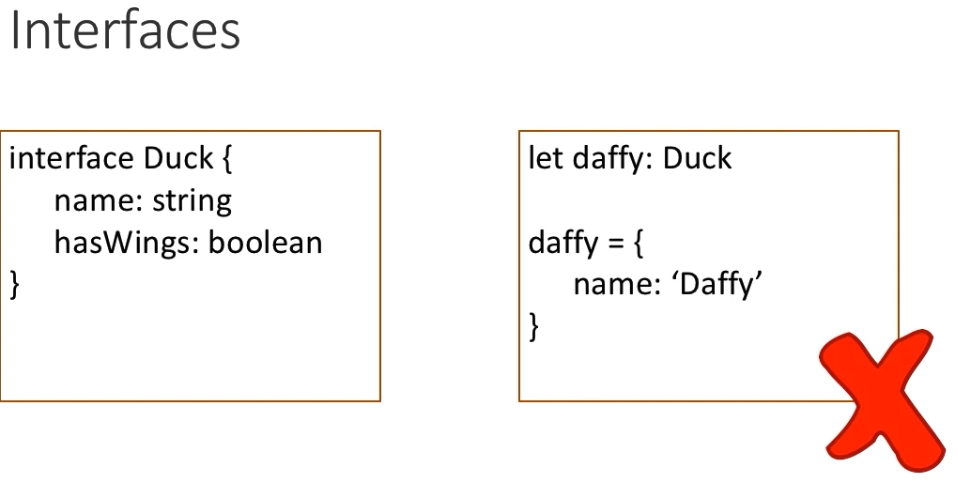


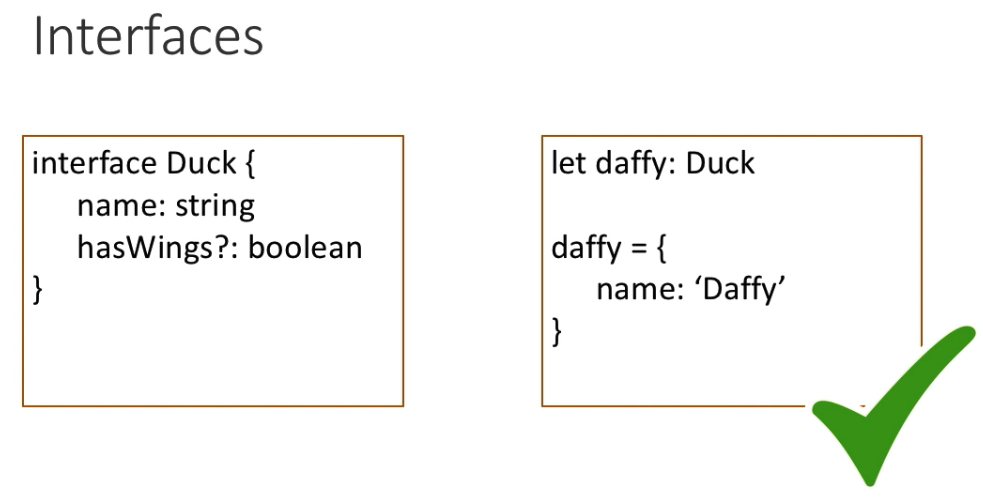


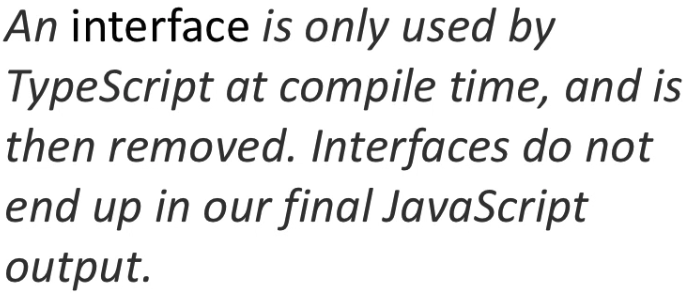


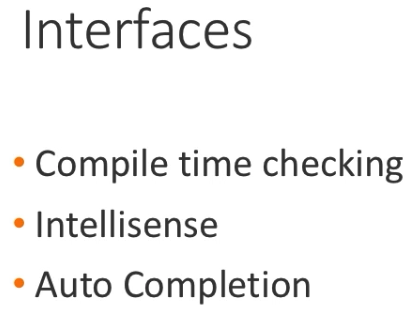












Optional properties contain the elvis (?) opterator – also known as safe navigation operator

interests?: string;

these need to come at the end before the required properties

You can set the baseUrl in the src/environments/environment folder. One for production, one for development.

export const environment = {

production: false,

apiUrl: 'http://localhost:5000/api/'

};

Then set the baseUrl in the service = environment.apiUrl;

----- All services get added to app.module.ts file

Col-lg-2 -- 6 elements on screen

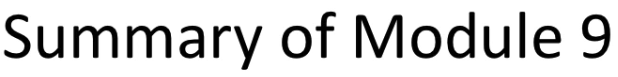
Col-md-3 – 4 elements on screen

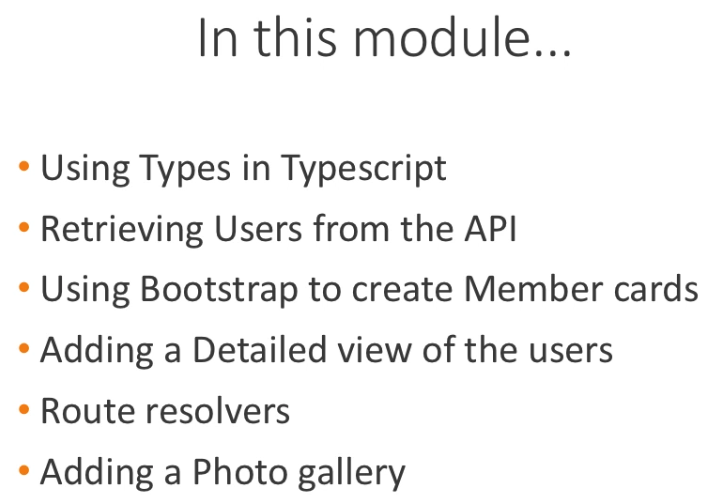
Col-sm-6 – 2 elements on screen

In order to absolutely position an element, the container it’s in needs to be relative.

Ngx-gallery

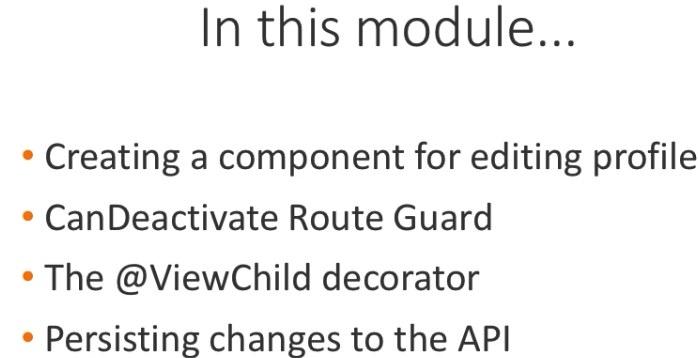
So basically resolver is that intermediate code, which can be executed **when a link has been clicked and before a component is loaded**.











Guards are added to Providers in app.module.ts