Overview of Directives

Angular directives allow us to manipulate the DOM.

Directives can be used to change the appearance, behaviour, or layout of DOM elements.

The directives in Angular are divided into three types :

1. Component Directive.
2. Structural directives.
3. Attribute directives.

Component Directives

Component directives are used for specifying the template/HTML for the Dom Layout.

Component directives are simple classes decorated with the @component decorator.

Structural Directives

A structural directive can change the layout of the DOM by adding or removing elements.

There are three common structural directives:

1. ngFor
2. ngIf
3. ngSwitch

ngFor Structural Directives

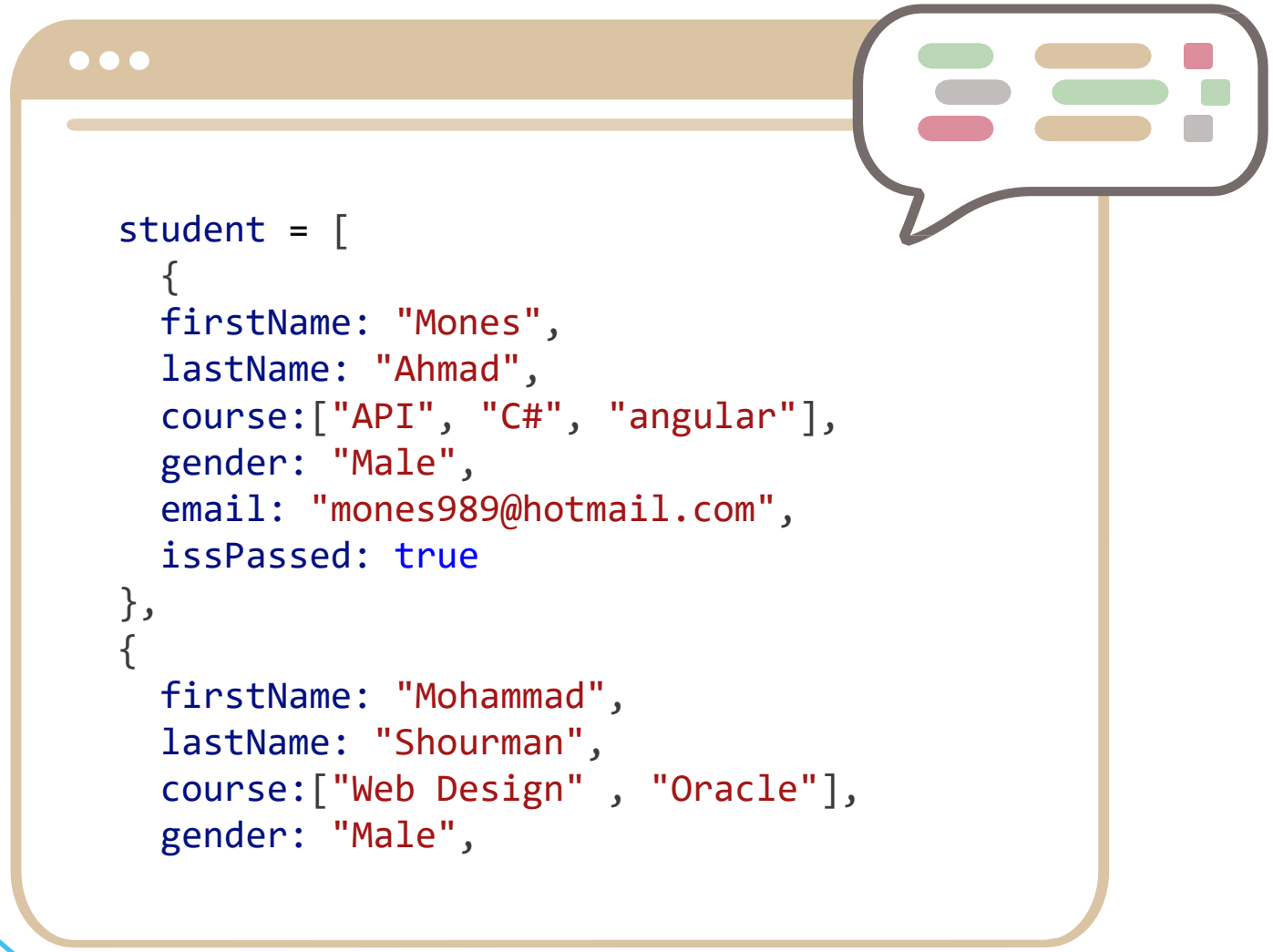
ngFor is an Angular directive that repeats a portion of the HTML template once per iteration of an IEnumerableList (collection).

Syntax:

```
ngFor="let obj of collection"
```

ngFor Example:

→ In app.component.ts



```
student = [
  {
  firstName: "Mones",
  lastName: "Ahmad",
  course:["API", "C#", "angular"],
  gender: "Male",
  email: "mones989@hotmail.com",
  issPassed: true
},
{
  firstName: "Mohammad",
  lastName: "Shourman",
  course:["Web Design" , "Oracle"],
  gender: "Male",
```

ngFor Example:

→ In app.component.ts

```
    email: "shurman@meo.com",
    issPassed: false
  },
  {
    firstName: "Mohammad",
    lastName: "Fodeh",
    course:["MVC"],
    gender: "Male",
    email: "mohammad@meo.com",
    issPassed: true
  },
  ]
```

ngFor Example:

→ In app.component.html

```
<table>
        <thead>
            <th>Fiest Name</th>
            <th>Last Name</th>
            <th>Courses</th>
            <th>Gender</th>
            <th>Email</th>
            <th>Status</th>
        </thead>
```

ngFor Example:

→ In app.component.html

```
<tbody>
   <tr *ngFor="let c of student">
       <td>{{c.firstName}}</td>
       <td>{{c.lastName}}</td>
       <td >{{c.course}}</td>
       <td>{{c.gender}}</td>
       <td>{{c.email}}</td>
       <td >{{c.issPassed}}</td>
   </tr>
</tbody>
   </table>
```

ngIf

By using the ngIf Directive, HTML elements can be added or removed based on an expression.

Boolean values must be returned from the expression.

The element is removed if the expression is false, otherwise, the element is inserted.

ngIf Example

In app.component.html

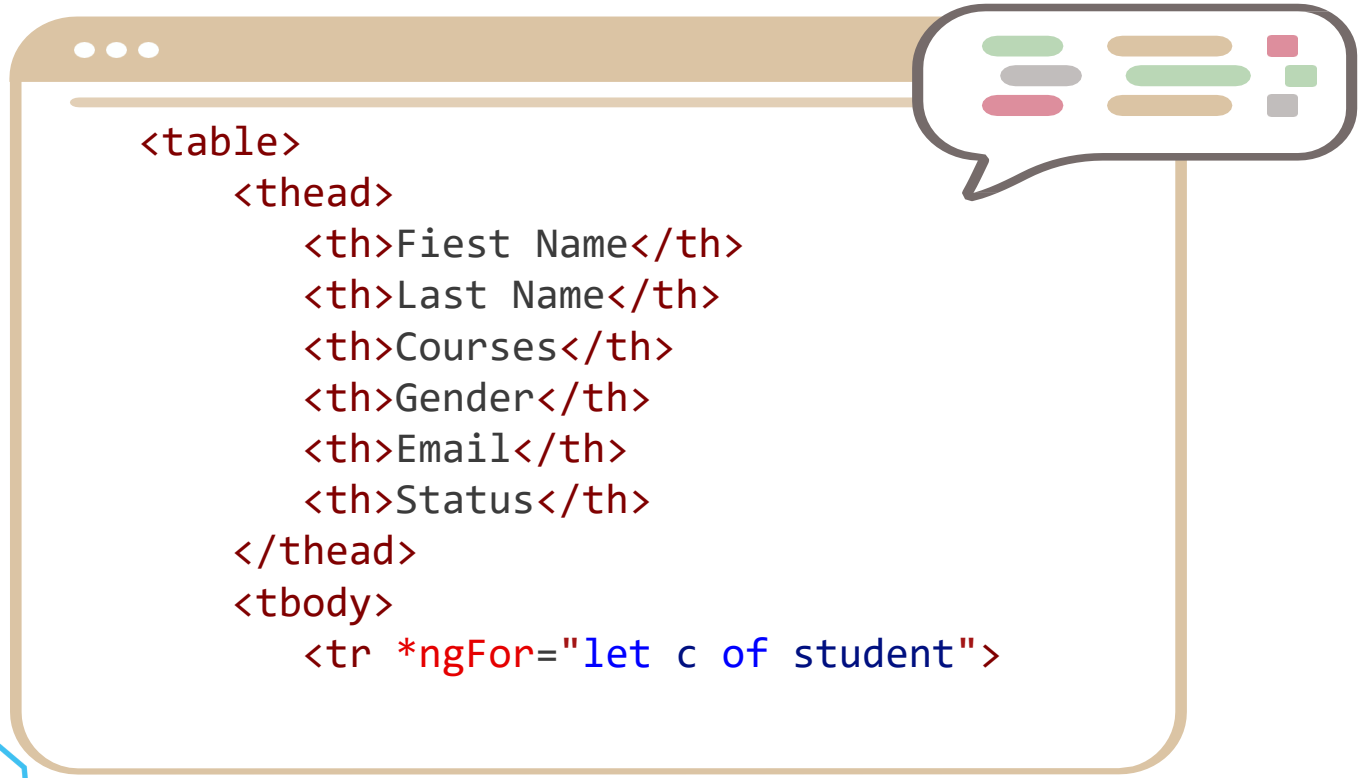
```
<div *ngIf="toggle">
        This is shown if condition is true
 </div>
```

In app.component.ts

```
toggle:boolean=true;
```

ngIf Example:

→ In app.component.html



```
<table>
    <thead>
        <th>Fiest Name</th>
        <th>Last Name</th>
        <th>Courses</th>
        <th>Gender</th>
        <th>Email</th>
        <th>Status</th>
    </thead>
    <tbody>
        <tr *ngFor="let c of student">
```

ngIf Example:

→ In app.component.html

```
<td>{{c.firstName}}</td>
                <td>{{c.lastName}}</td>
                <td >{{c.course}}</td>
                <td>{{c.gender}}</td>
                <td>{{c.email}}</td>
                <td *ngIf="c.issPassed" >pass</td>
                <td *ngIf="!c.issPassed" >fail</td>
            </tr>
        </tbody>
            </table>
```

ngIf Example

In app.component.ts

```
isStudent = "no"
```

ngIf Example:

→ In app.component.html

```
 <h3>Are you a Student?</h3>
 <select [(ngModel)]= isStudent>
    <option value="yes">Yes</option>
    <option value="no">No</option>
 </select>

 <div *ngIf="isStudent == 'yes'">
<br>
<label>internship start date</label>
 <input type="date">
<br>
 <label>internship end date</label>
<input type="date">
```

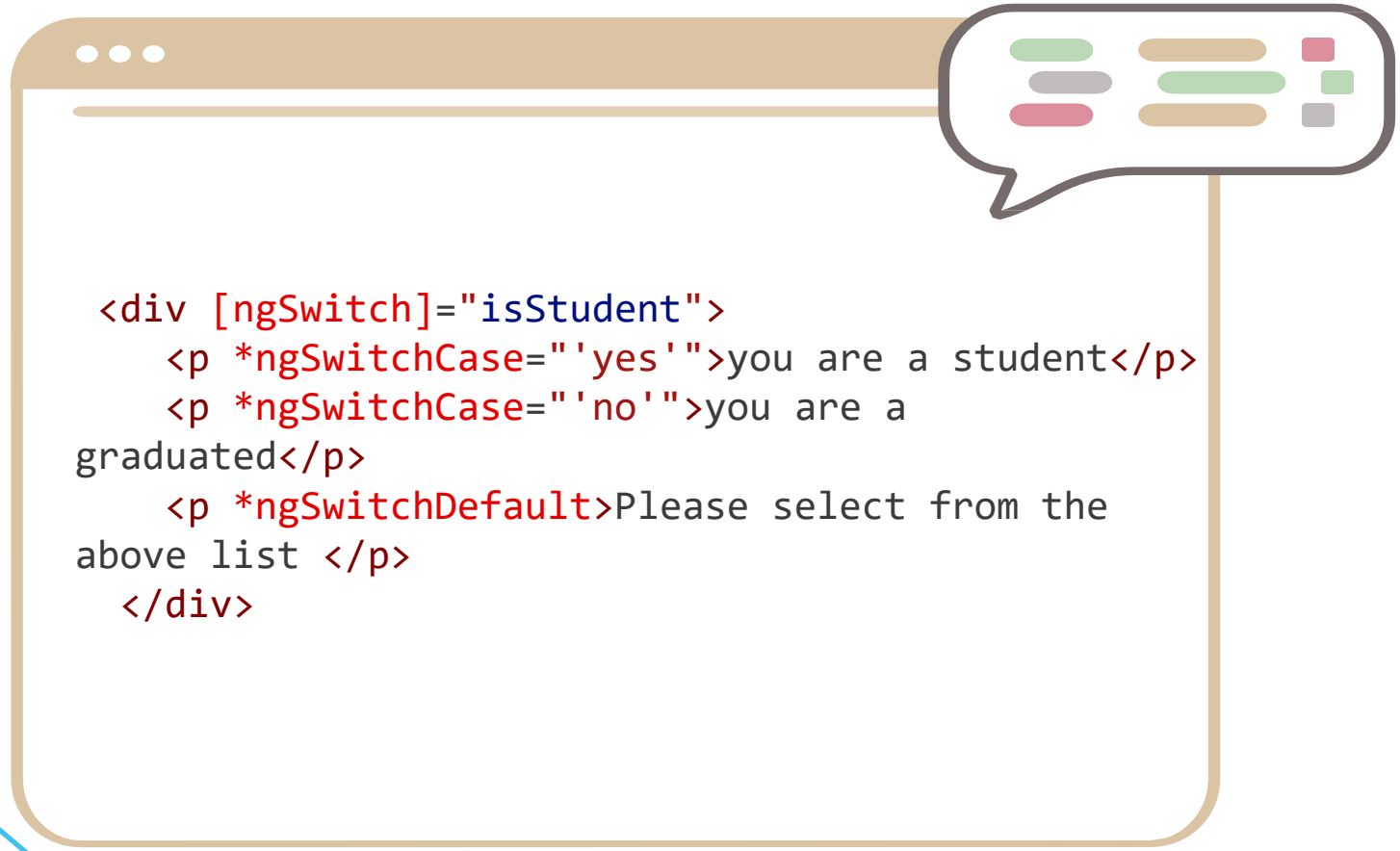
ngSwitch

By using the ngSwitch directive, you can add or remove HTML elements based on match expressions.

ngSwitch directive used with ngSwitchCase and ngSwitchDefault.

ngSwitch Example:

→ In app.component.html



```
<div [ngSwitch]="isStudent">
   <p *ngSwitchCase="'yes'">you are a student</p>
   <p *ngSwitchCase="'no'">you are a
graduated</p>
   <p *ngSwitchDefault>Please select from the
above list </p>
  </div>
```

Attribute Directives

By using a style directive or attribute, we can change how an element appears or behaves.

There are three common Attribute directives

1. ngModel
2. ngClass
3. ngStyle

Attribute Directives

1. ngModel
In order to achieve the two-way data binding, the ngModel directive is used.

2. ngClass
CSS classes are added or removed from HTML elements using the ngClass property.

ngClass Example

In app.component.css

```
.red { color: red;
background-color: gray; }
.size20 { font-size: 20px; }
```

ngClass Example

In app.component.html

```
<div [ngClass]="'red size20'"> Red Text with Size 20px </div>
```

The Result

ngClass Example

→ In app.component.html

```
   <table>
<thead [ngClass]="'tableHead'">
   <th>Fiest Name</th>
   <th>Last Name</th>
   <th>Courses</th>
   <th>Gender</th>
   <th>Email</th>
   <th>Status</th>
</thead>
<tbody>
```

ngClass Example

→ In app.component.html

```
    <tr *ngFor="let c of student"
[ngClass]="{'failStudent' : !c.issPassed}">
        <td>{{c.firstName}}</td>
        <td>{{c.lastName}}</td>
        <td [ngClass]="{'courses' :
c.course.length > 1}">{{c.course}}</td>
        <td [ngClass]="{'male': c.gender == 'Male'
, 'female' : c.gender == 'Female'}
">{{c.gender}}</td>
        <td>{{c.email}}</td>
        <td >{{c.issPassed}}</td>
    </tr>
 </tbody>
    </table>
```

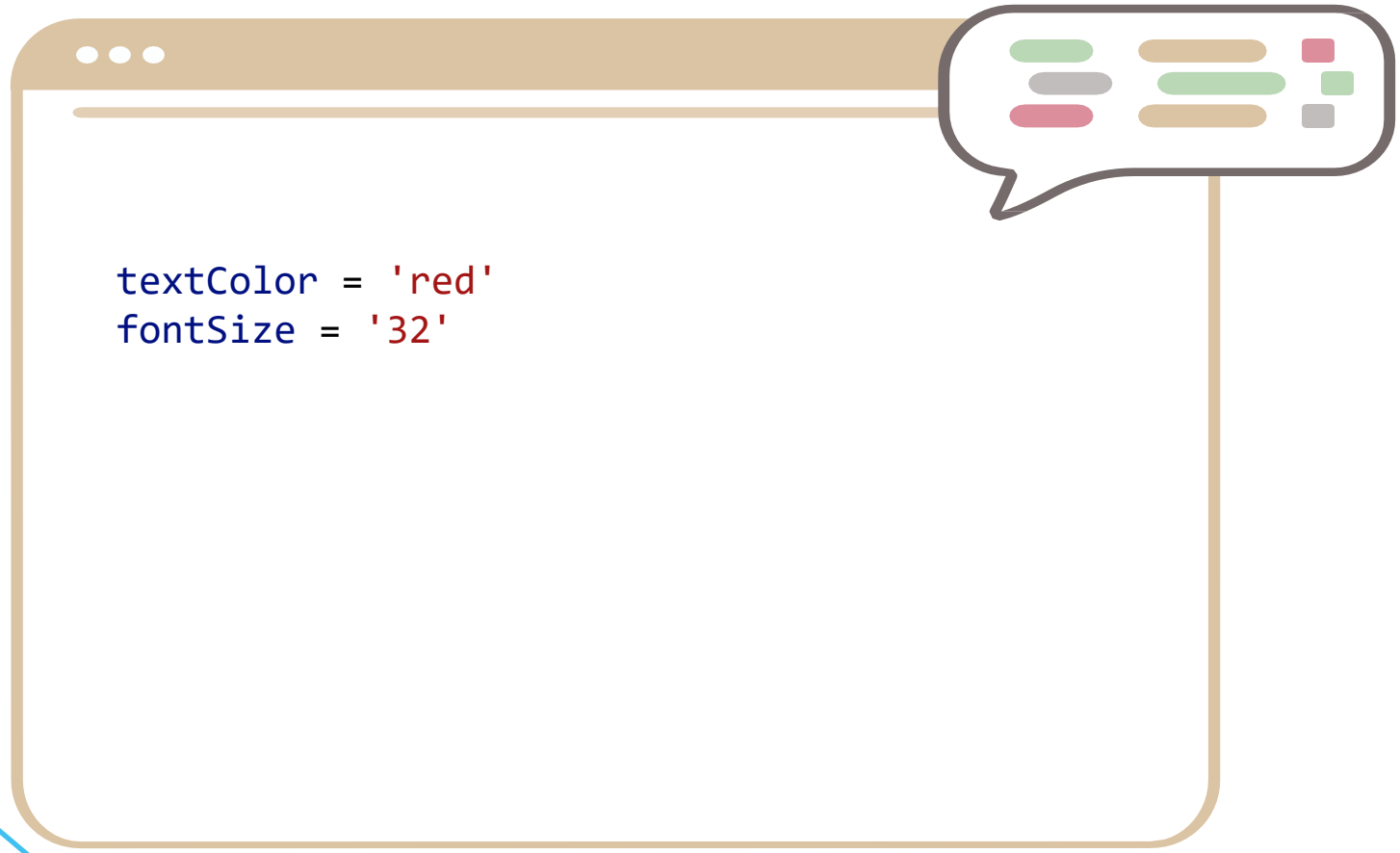
ngClass Example

→ In app.component.css

```
.failStudent {
background-color: red; }
.courses {
    color: blue; }
.tableHead{
        background-color: darkblue;
        color: white; }
    .male {
        background-color: lightblue; }
    .female {
        background-color: lightpink; }
```

ngStyle Example

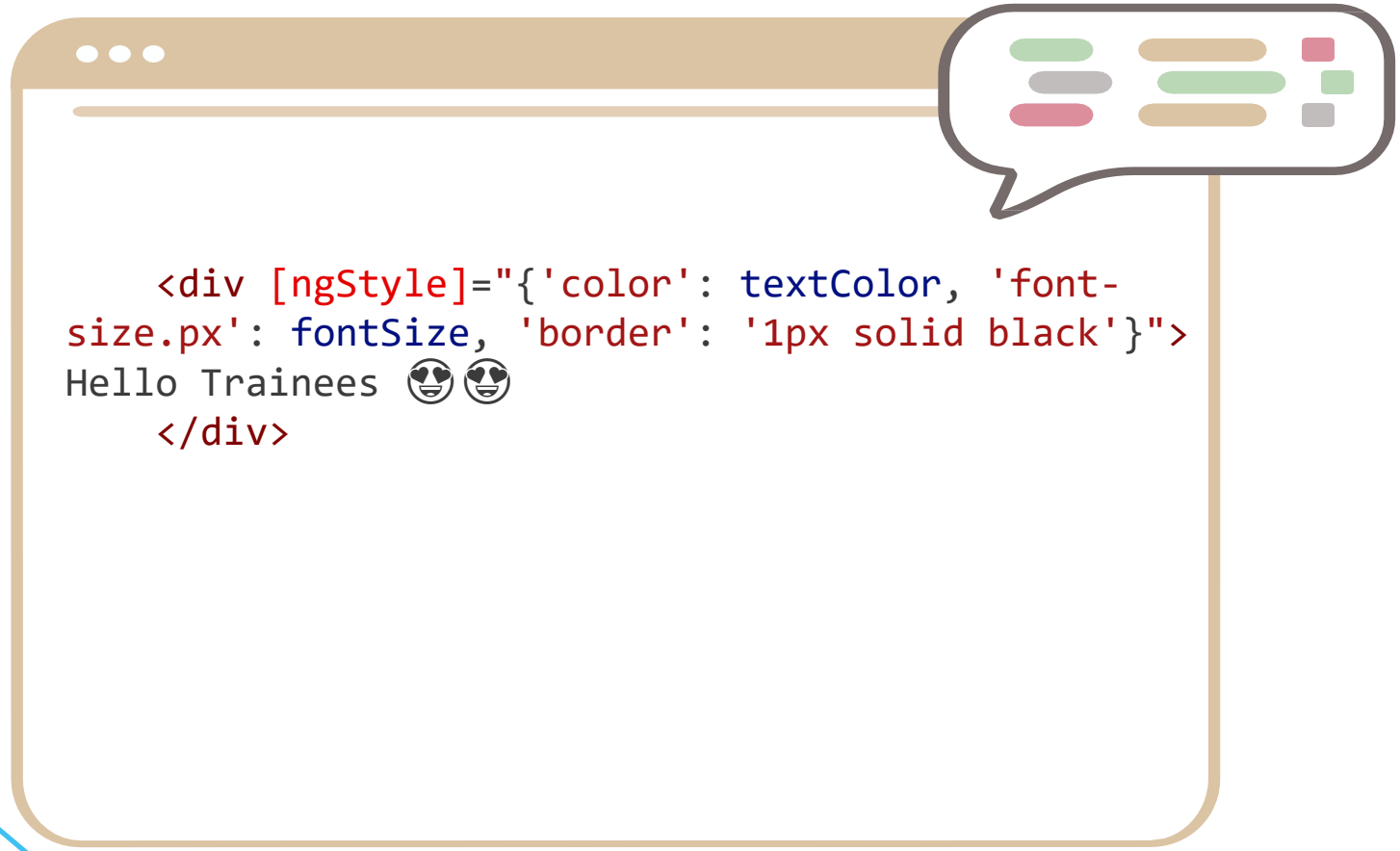
→ In app.component.ts



```
textColor = 'red'
fontSize = '32'
```

ngStyle Example

→ In app.component.html



```
    <div [ngStyle]="{'color': textColor, 'font-
size.px': fontSize, 'border': '1px solid black'}">
Hello Trainees 😍😍
    </div>
```

Module in Angular

Overview of Module

It is a deployment subset within your full Angular application.

Using it, you can divide an application into smaller parts and load each one separately, as well as create libraries of components that can be imported directly into other applications.

Overview of Module

An NgModule is defined by a class decorated with @NgModule().

This decorator is a function that takes an object that describes a module as its single metadata property.

Properties for NgModule

The most important properties of NgModule:

Declarations: The components, directives, and pipes of this module.

Imports: The subset of NgModule declarations that must be visible and useable in other NgModule templates.

Properties for NgModule

Exports: This NgModule depends on other modules whose exported classes are needed by this component template.

Providers: Services created by this NgModule become accessible in all parts of the application since this NgModule contributes to the global collection of services. (It is also possible to specify providers at the component level.)

Properties for NgModule

bootstrap: The main view of the application, called the root component, is where all other views reside. It's only the root NgModule that needs to set the bootstrap property.

Generate a New Module

Use this command in the terminal to generate a new module.
→ ng generate module module _name - -routing
Or
→ ng g m module _name - -routing

Generate a New Module Example

In our project **(The Learning Hub)**, create a new module called **auth,** and for this module generate two components inside it:

login and **register**.

Generate a New Module Example

Generate auth module:

```
PS C:\Users\d.kanaan.ext\Desktop\TheLearningHub> ng g m auth --routing
CREATE src/app/auth/auth-routing.module.ts (247 bytes)
CREATE src/app/auth/auth.module.ts (272 bytes)
PS C:\Users\d.kanaan.ext\Desktop\TheLearningHub>
```

Generate a New Module Example

Generate a login component in auth module.

To determent, these components related to this module, write moduleName/componentsName.

```
PS C:\Users\d.kanaan.ext\Desktop\TheLearningHub> ng g c auth/login
CREATE src/app/auth/login/login.component.html (20 bytes)
CREATE src/app/auth/login/login.component.spec.ts (552 bytes)
CREATE src/app/auth/login/login.component.ts (198 bytes)
CREATE src/app/auth/login/login.component.css (0 bytes)
UPDATE src/app/auth/auth.module.ts (352 bytes)
```

Generate a New Module Example

Generate a Register component in auth module.

```
PS C:\Users\d.kanaan.ext\Desktop\TheLearningHub> ng g c auth/register
CREATE src/app/auth/register/register.component.html (23 bytes)
CREATE src/app/auth/register/register.component.spec.ts (573 bytes)
CREATE src/app/auth/register/register.component.ts (210 bytes)
CREATE src/app/auth/register/register.component.css (0 bytes)
UPDATE src/app/auth/auth.module.ts (442 bytes)
```

Shared Module

Overview of Shared Module

Your code can be organized and streamlined by creating shared modules.

Directives, pipes, and components that are commonly used can be put into this module, which you can then import wherever you need them in your application.

Overview of Shared Module

In order to generate a shared module, the same command to generate any other module will be used.

```
PS C:\Users\d.kanaan.ext\Desktop\TheLearningHub> ng g m shared
CREATE src/app/shared/shared.module.ts (192 bytes)
```

Consider the following

The CommonModule is imported because the module's component needs common directives.

The module declares and exports utility pipes, directives, and components.

It re-exports all modules, components, pipes, and directives that are declared and imported.

```
@NgModule({
  declarations: [],
  imports: [
    CommonModule,
    FormsModule,
    ReactiveFormsModule,
    MatFormFieldModule,
    MatInputModule
  ],
  exports:[
    FormsModule,
    ReactiveFormsModule,
    MatFormFieldModule,
    MatInputModule
  ]
})
```

Note

You can create a template that you will use multiple times, such as navbars, footers, or sidebars, in the shared module.

Exercise

Create a shared module that’s include a navbar and footer components and transfer the template from the app module to the shared module.

Note: Don’t forget to include the shared module in which you want to Use the navbar and footer components.

Routing

Overview of Routing

Routing allows you to move from one part of the application to another part or one View to another View.

Overview of Routing

Since the app component is the first component that is loaded when the project
Has run and if you want to go to another page, we will use routing in the app component.

Add this tag in the app.component.html:

→ `<router-outlet></router-outlet>`

Example of Routing

Generate a new component called about us and another component called contact us in the app module.

Example of Routing

Generate the about us component in the app module:

```
PS C:\Users\d.kanaan.ext\Desktop\TheLearningHub> ng g c aboutUs
CREATE src/app/about-us/about-us.component.html (23 bytes)
CREATE src/app/about-us/about-us.component.spec.ts (567 bytes)
CREATE src/app/about-us/about-us.component.ts (209 bytes)
CREATE src/app/about-us/about-us.component.css (0 bytes)
UPDATE src/app/app.module.ts (819 bytes)
```

Example of Routing

Generate the contact us component in the app module:

```
PS C:\Users\d.kanaan.ext\Desktop\TheLearningHub> ng g c contactUs
CREATE src/app/contact-us/contact-us.component.html (25 bytes)
CREATE src/app/contact-us/contact-us.component.spec.ts (581 bytes)
CREATE src/app/contact-us/contact-us.component.ts (217 bytes)
CREATE src/app/contact-us/contact-us.component.css (0 bytes)
UPDATE src/app/app.module.ts (727 bytes)
```

Overview of Routing

The first page will load **app components** and if you want to go to the about us component in the same module **(App Module)** you should use routing.

In-app-routing.module.ts. There is an array called **routes**
this array is used to add the route for all components and other modules.

Overview of Routing

Each route is an object, and each route consists of a path and the component name.

path: where you want to go for example /about us, but you must write the page name without using /.

component: component name which you want to display.

Example of Routing

```
const routes: Routes = [
  {
      path: 'about',
      component: AboutusComponent
  }
];
```

NOTE: Once you write the component name it will import this component like this:

```
import { AboutusComponent } from './aboutus/aboutus.component';
```

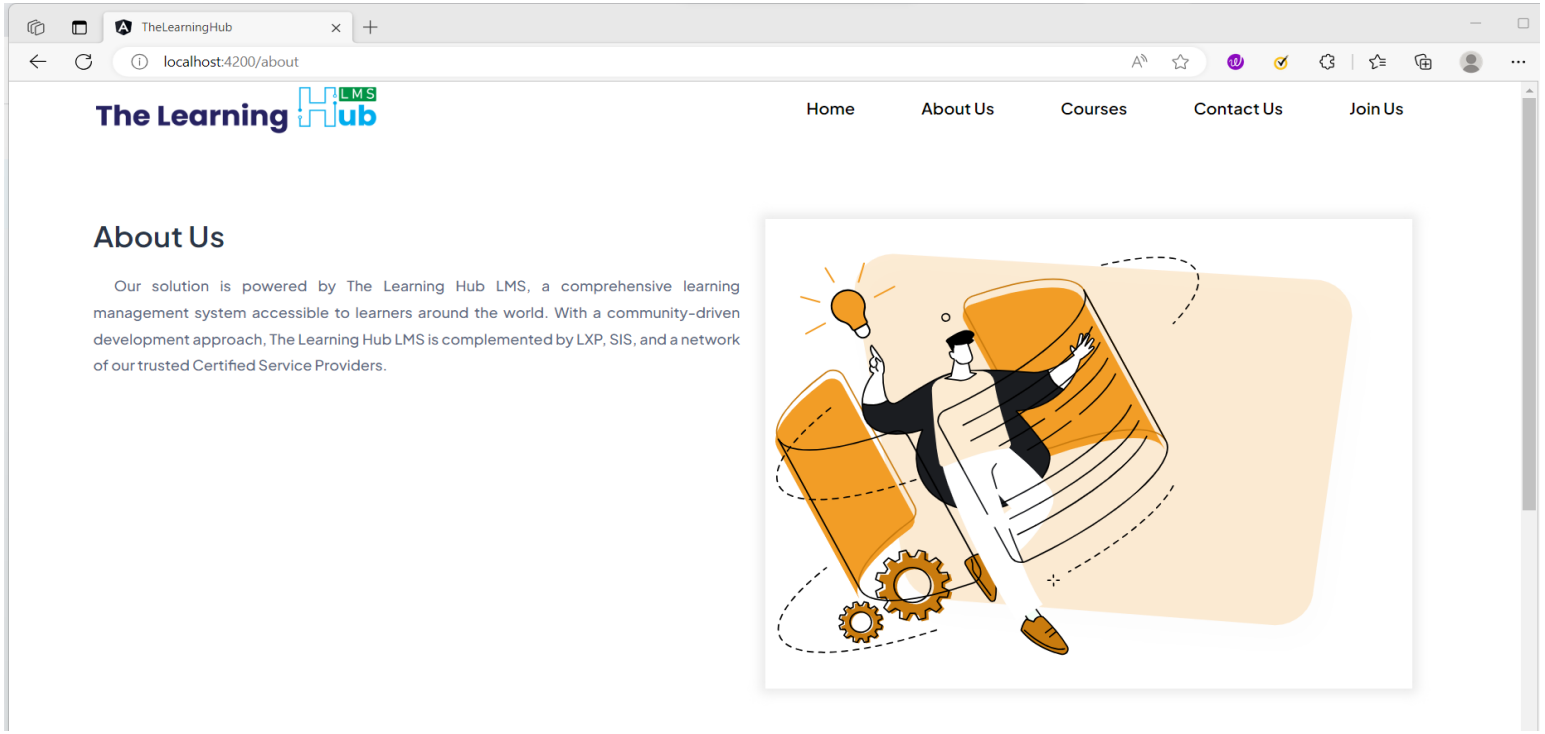
Example of Routing

To add more than one route separate them with a comma.

```
const routes: Routes = [
  {
      path: 'about',
      component: AboutusComponent
  },
  {
      path: 'contact',
      component: ContactusComponent
  }
];
```

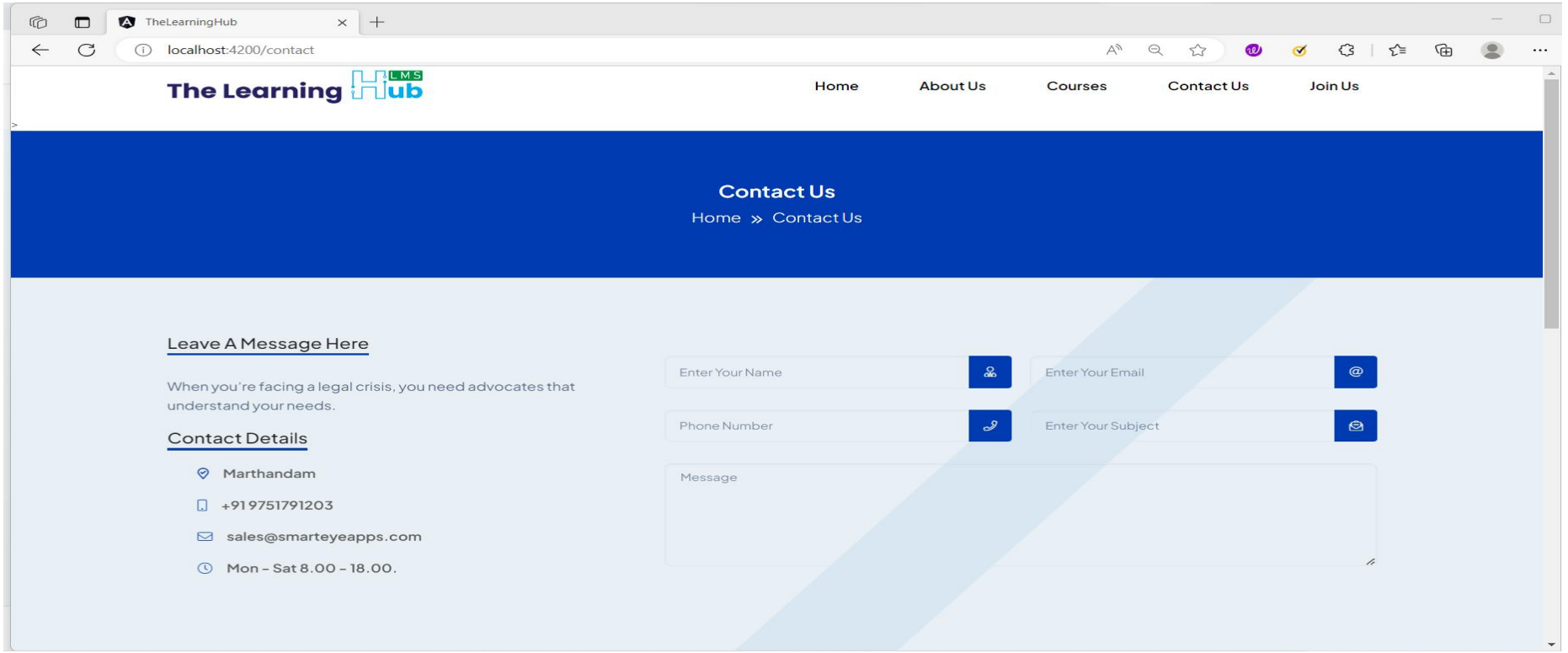
Example of Routing

Because the about us component in the app module use /about in the URL.



Example of Routing

Contact us component.



Default Routing

Overview of Default Routing

The default route is redirected -for example- to the home path.

This means that, when you navigate to the root of your application /, you are redirected to the home path (/home).

The path is empty, indicating the default route.

Overview of Default Routing

The default route of the project is the About Us component.

```
const routes: Routes = [
  {
      path: '',
      component: AboutComponent
  },
]
```

In this way when the project was run the about us component will load.

Eager loading

The eager loading of a resource occurs the moment the code is executed. The eager loading process also involves preloading related entities linked to a resource.

In other words, eager loading refers to loading modules before the application starts.

Lazy Loading

NgModules are eagerly loaded by default, so as soon as an application loads, all its NgModules load too, regardless of whether they are required or not.

It’s a design pattern that loads NgModules as needed.

Lazy loading helps keep initial bundle sizes smaller, which in turn helps decrease load times.

To load a feature module lazily, we need to load it using loadChildren property in route configuration and that feature module must not be imported in application module

Lazy Loading

It is necessary to load a feature module lazily using the **loadChildren** property in route configuration and that feature module must not be imported into the application module.

Exercise

Create a route for auth module to login and register components.

Exercise Solution :

In auth-routing.module.ts:

```
const routes: Routes = [
  {
      path: 'login',
      component: LoginComponent
  },
  {
      path: 'register',
      component: RegisterComponent
  }
];
```

Exercise Solution :

In app-routing.module.ts

```
{
  path: 'security',
      loadChildren : () =>
import('./auth/auth.module')
.then((m) => m.AuthModule)
}
];
```

Example of Default Routing

If you want to make the login page the first page to be loaded.

In-app-routing.module.ts

```
const routes: Routes = [
  {
      path: '',
      loadChildren: () => import('./auth/auth.module').
          then((m) => m.AuthModule)
  },]
```

Example of Default Routing

And In auth-routing.module.ts :

```
const routes: Routes = [
  {
      path: '',
      component: LoginComponent
  },
  {
      path: 'register',
      component: RegisterComponent
  }
];
```